

Correcting Imperfect Domain Theories: A Knowledge-Level Analysis*

Scott B. Huffman **Douglas J. Pearson**
John E. Laird

Artificial Intelligence Laboratory
The University of Michigan
1101 Beal Ave.
Ann Arbor, Michigan 48109-2122

March 8, 1996

Abstract

Explanation-Based Learning [Mitchell *et al.*, 1986; DeJong and Mooney, 1986] has shown promise as a powerful analytical learning technique. However, EBL is severely hampered by the requirement of a complete and correct domain theory for successful learning to occur. Clearly, in non-trivial domains, developing such a domain theory is a nearly impossible task. Therefore, much research has been devoted to understanding how an imperfect domain theory can be corrected and extended during system performance. In this paper, we present a characterization of this problem, and use it to analyze past research in the area. Past characterizations of the problem (e.g, [Mitchell *et al.*, 1986; Rajamoney and DeJong, 1987]) have viewed the types of performance errors *caused* by a faulty domain theory as primary. In contrast, we focus primarily on the types of knowledge deficiencies present in the theory, and from these derive the types of performance errors that can result. Correcting the theory can be viewed as a search through the space of possible domain theories, with a variety of knowledge sources that can be used to guide the search. We examine the types of knowledge used by a variety of past systems for this purpose. The hope is that this analysis will indicate the need for a “universal weak method” of domain theory correction, in which different sources of knowledge for theory correction can be freely and flexibly combined.

To appear in *Machine Learning: Induction, Analogy and Discovery*, edited by Susan Chipman and Alan Meyrowitz, Kluwer Academic Press, 1992.

*Sponsored in part by NASA and the Office of Naval Research under contract NCC 2-517, and by a University of Michigan Research Partnership Fellowship.

Contents

1	Introduction	1
2	Performance Tasks Of EBL Systems	1
3	Domain Theory Problems	3
3.1	Types of Domain Theory Imperfections	3
3.2	Observed Types of Failures	4
3.3	Mapping Knowledge Deficiencies to Failures	5
4	Knowledge Sources for Correcting Domain Theories	7
4.1	Internal Knowledge	9
4.1.1	Complete Underlying Domain Theory	9
4.1.2	Other Internal Knowledge	10
4.2	Teacher	12
4.2.1	Teacher provided examples	12
4.2.2	Advice	13
4.3	External World	14
4.3.1	Active Use of the External World	14
4.3.2	Passive Observation of the External World	15
5	Analysis Summary	17
6	Relationship to Other Frameworks	18

1 Introduction

Much recent research in machine learning has centered around analytical learning techniques. In particular, *Explanation-Based Learning* (EBL) [Mitchell *et al.*, 1986; DeJong and Mooney, 1986] has emerged as an important approach to using prior domain knowledge to improve performance. In its classic conception, EBL involves using a *domain theory* to construct an *explanation* (or proof) of why a given *training example* is an instance of some *goal concept*. By analyzing the dependency structure of the explanation, the learning system can construct a generalized, *operational* rule which directly recognizes the training example as an instance of the goal concept. EBL has taken a variety of slightly different forms, such as knowledge compilation [Anderson, 1986], chunking [Laird *et al.*, 1986], operationalization [Mostow, 1981], and schema construction [Mooney, 1990], and has proven useful in domains from recognizing simple concepts to learning reactive plans for mobile robots [Mitchell, 1990; Laird and Rosenbloom, 1990].

This technique is severely hampered, however, by the requirement of a complete and correct domain theory. Even for simple domains, a perfect domain theory is extremely difficult to construct. Therefore, to make EBL a plausible learning technique, it must be augmented by methods for correcting and extending incomplete domain theories.

There have been a number of different methods advanced to attack this problem. Researchers have focussed on examining a variety of *mechanisms* for extending and correcting domain theories. In this paper, however, we will focus primarily on the *knowledge* used to extend and correct a domain theory. Understanding the knowledge available is the first step towards the long-term goal of developing a general, domain-independent mechanism for improving domain theories, which can draw on any of the available knowledge sources in *combination*. The ability to flexibly draw upon a combination of knowledge sources should greatly enhance a system's ability to improve its domain theory. Thus, our long-term goal can be viewed as the development of a "universal weak method" [Laird and Newell, 1983] for domain theory refinement.

In this paper, we will present a framework that lays out the space of possible approaches to extending and correcting domain theories. In order to understand how various knowledge sources are used to refine domain theories, we must begin by analyzing the refinement task itself. We will examine the kinds of tasks EBL systems perform, the types of domain theory errors that can be present, and the ways in which they manifest themselves. Next, we will describe the domain theory correction process, casting it as a heuristic search through the space of possible domain theories. Knowledge that is used in the correction process can be then be viewed as controlling this search. In analyzing past work in the area (including our own) in these terms, we find that few approaches have demonstrated the ability to draw from a variety of knowledge sources in refining an imperfect domain theory.

2 Performance Tasks Of EBL Systems

Two complementary tasks have been performed by EBL systems. The first type, what we will call *analysis* tasks, involve explaining or understanding some observed example. This category includes plan recognition [Mooney, 1990], in which the system tries to explain the actions of characters in a narrative, and apprenticeship learning systems (e.g., [Wilkins, 1988; Chien, 1989; VanLehn, 1987]), in which the system observes an expert carrying out some

task which it must learn to perform. The second type, what we will call *generation* tasks, involve constructing (as opposed to observing) a plan to reach some goal from an initial state. This category includes typical planning tasks, such as planning a robot's actions (e.g., [Gupta, 1987; Laird *et al.*, 1989; Bennett, 1990]), and standard concept recognition tasks (e.g., [Ourston and Mooney, 1990]), where the sequence of inferences needed to recognize an example of the concept is considered the "plan" that is generated.

These two types of tasks are similar, in that they require similar types of knowledge to be performed. Both require a knowledge of the operations performable in the domain, their applicability and their effects. For analysis tasks, the system observes a sequence of states ending at a goal, and must infer a consistent sequence of operations that would produce the sequence of states. For generation tasks, the system is given only the initial and final states, and must produce a sequence of operations leading from one to the other. Thus the analysis task is a more constrained version of the generation task. In addition, generation systems may require knowledge of how to actually carry out the operations in the external world. Many systems (e.g., [Bennett, 1990; Chien, 1989]) perform both types of problems. In the remainder of this paper we will cast our discussion from the point of view of generation tasks, but the analysis is equally applicable to analysis tasks. Where the different tasks lead to differences in the analysis, we will highlight the differences.

We can cast the performance task faced by an EBL system in generic terms using the standard AI concept of problem space search. Search in a problem space involves finding a path from an initial state to a goal state. The system has a set of *operators* which move between states in the space. The task is to find a sequence of operators which lead from the initial state to the goal state. This sequence is a *plan* in the generation case (or an execution trace if the system is executing operators before a complete plan is formed). In the analysis case, an observation must be explained. The sequence forms an explanation of the observation. Throughout the paper we will refer to the reasoning process as planning, and its result as plans.

Given this characterization of the problem, the knowledge that comprises the system's domain theory becomes clear. Simply, the system's knowledge of operators - their preconditions and effects - makes up the domain theory. In pure inference tasks, such as the cups domain [Winston *et al.*, 1983], each inference can be viewed as an operator. Note that this simple definition of the domain theory is a result of how we have cast the problem. A system may or may not actually have its knowledge partitioned into distinct, discrete operators with explicit pre- and postconditions. Clearly, knowledge can be applicable to many different operators (for example, general "frame axioms"). However, conceptually the knowledge can be viewed as a set of operators, with distinct knowledge about each.

There may be other knowledge in the system, of course, besides what can be viewed as operator knowledge. This additional knowledge is used to *guide* search through the states of the problem space. There may be standard *search control* knowledge, which express preferences between operators, and/or reasoning strategy knowledge, such as knowledge allowing the system to perform an abstract or hierarchical search.

We do not consider such search guiding knowledge to be a part of the domain theory proper of a system. The reason is that search control is intended to affect problem solving's *efficiency*, not its *correctness*. Search control may make an intractable theory tractable, but the theory is not incorrect in either case. If a system has no search control, or incorrect search control, it will have to explore more of the search space to solve the problem, but

the lack of search control will not preclude solving the problem. EBL systems that have learned search control (e.g., PRODIGY [Minton *et al.*, 1989] and Soar [Laird *et al.*, 1987]) did not extend or correct domain theories by doing so. However, if resource constraints can affect the correctness of the solution, then the solution may not be independent of the search process. In this case, knowledge about resource constraints must be incorporated into the domain theory.

3 Domain Theory Problems

Almost every domain theory is actually an approximation. This is due to the frame problem: the preconditions and effects of actions are extremely difficult to describe fully except in limited domains. For example, knowing the preconditions of actions in the real world is extremely difficult; this is known as the *qualification problem* [Ginsberg and Smith, 1987]. Exceptions can be generated almost ad infinitum. Consider an operator for going through a door. The most obvious precondition is that the door is opened. What if the door is open but there are steel bars across the opening? Or, maybe a transparent glass plate? Or an unseen magnetic force field? We could go on and on. Similarly, in an environment with other complex processes or agents, it is usually impossible to have a complete domain theory that will predict all the activity of the other processes and agents. This explains AI's fondness for the closed world assumption, for without it, our domain theories are always incomplete, if not incorrect.

When a system's domain theory is imperfect, errors may arise, either during planning or execution. In this section we analyze the types of domain theory imperfections that may arise, and the errors they can lead to. Note that correcting a domain theory in response to failures is different from learning search heuristics from search failures, which can be done using standard EBL methods and does not alter the domain theory [Mostow and Bhatnager, 1987; Minton *et al.*, 1989].

3.1 Types of Domain Theory Imperfections

We have cast the problem facing an EBL system as that of finding a sequence of operators. Therefore, the domain theory consists of the system's knowledge of the preconditions and postconditions of its operators. This gives rise to the set of possible problems with the domain theory:

- **Overgeneral Preconditions.**
An operator precondition is missing, or an overgeneral test is used (e.g. "fruit" instead of "banana").
- **Overspecific Preconditions.**
An extra, unnecessary precondition is present, overly restricting the set of situations in which an operator is applicable.
- **Incomplete Postconditions.**
The planner is unaware of some effect of an operator.

- Extraneous Postconditions.

The planner incorrectly believes that an operator will produce some effect which it does not.

- Missing Operators.

An entire operator is absent from the domain theory.

Note that when a postcondition is simply “incorrect” this can be viewed as a case of both an extraneous postcondition (the current effect) and a missing postcondition (the correct effect).

3.2 Observed Types of Failures

As a result of an imperfect domain theory, an EBL system might encounter a failure either during planning or during execution of the plan. These two categories are further delineated below:

1. Planning Failures

(a) Incomplete Plan

In this case, the planner is unable to form a complete sequence of steps that it believes will lead from the initial state to the goal. For analysis tasks, this corresponds to being unable to construct a complete explanation of the observed training example. Several systems attempt to construct a parse of training examples by working both bottom-up and top-down [VanLehn, 1987; Hall, 1986]. A failure to complete the parse represents an incomplete plan failure.

(b) Multiple Inconsistent Plans

Rajamoney and DeJong [1988] examine cases in which it is known that only one plan should be formed, but multiple plans are found. In many domains, multiple paths to the same goal are possible, so forming multiple plans does not necessarily constitute an error. Even if it is known that there is only one legal path to the goal, detecting this error requires that the planner actually attempt to construct multiple plans, which is not common.

(c) Reaching an impossible state

During planning, the system will use its knowledge of operators to search through the states of the domain. If the domain theory is incorrect, it may be possible to reach a world state which is known to be *impossible* to achieve. For example, if during planning in a STRIPS domain the robot is found to be in two different rooms at once, this indicates a domain theory problem. Note that to detect this kind of error during planning, the system must have explicit knowledge about states that are impossible.

2. Execution Failures

Execution failures occur when the system completes a plan but is unable to successfully execute it. Failure occurs when the system’s expectations are not met in the external world. Note, however, that the operator being executed when the error is detected is

not necessarily the one for which the domain theory is in error. The incorrect operator may have occurred anywhere before the current operator in the plan.

The failure may be detected in a number of ways:

- (a) Preconditions of the next operator to be executed are not achieved.

Here an operator is slated to be executed but cannot be because one of its preconditions does not hold in the external world. This means that some earlier operator either was expected to achieve the precondition but did not, or that some earlier operator was expected to preserve the precondition but clobbered it [Carbonell and Gil, 1987].

- (b) Postconditions of the operator just executed are not achieved.

The operator does not cause all of the effects that the planner predicted. Carbonell and Gil [1987] break this down further into two cases: either all postconditions are unachieved, or only some subset are unachieved. If all postconditions are unachieved, it probably indicates that the operator didn't apply at all. The operator apparently had preconditions for its application that the planner was unaware of. If only a subset of the operator's expected effects are unachieved, then the operator did apply but didn't do all it was expected to. This indicates that the unachieved effects are either incorrect or somehow conditional on the situation in an unexpected way.

- (c) Failure is explicitly detected.

In some cases, the system may have rules indicating states of the world which should not be reached. This is essentially a "theory of failure." For example, Gupta's system [1987] has a rule indicating that if the robot's gripper melts, that is a failure. In classification tasks, execution corresponds to classifying an example, and incorrectly classifying an example corresponds to an explicit detection of failure.

It should be noted that not all execution failures are due to an erroneous domain theory. The domain theory is a theory of the processes that can operate on the world. To operate correctly, the domain theory depends on the agent having a correct knowledge of the *state* of the world as well. This is the standard process/data distinction in computer science. Here we are considering imperfections in the agent's *process* knowledge; there might also be imperfections in the agent's *data* about the state of the world. These imperfections can result from incomplete or noisy sensing.

3.3 Mapping Knowledge Deficiencies to Failures

Each class of errors in the domain theory can produce a subset of the types of observed failures in planning or execution. The six types of failures are plotted against the five types of domain theory deficiencies in Figure 1. Many of the dots in the figure are obvious: for instance, it is clear that a missing operator can lead to incomplete plans, and that extraneous postconditions can lead to postconditions which are not met during execution.

The figure reveals some explainable patterns, with various rows containing the same set of dots. Missing operators and overspecific preconditions, for example, both lead only

	Planning Failures			Execution Failures		
	Incomplete	Multiple	Impossible State	Prec. Unmet	Postc. Unmet	Explicit Detection
Overgen. Preconds.		•	•		•	•
Overspec. Preconds.	•					
Missing Postconds.	•	•	•	•	•	•
Extra Postconds.	•	•	•	•	•	•
Missing Operator	•					

Figure 1: Domain theory error types and their manifestations.

to incomplete plan failures. This is because both overly restrict the selection of a needed operator. This can eliminate correct plans from consideration, but it cannot cause incorrect plans to be formed.

Missing and extra postcondition errors both lead to all six types of failures. This is not surprising, since both types of errors result in the planner having an incorrect model of the world after applying the operator. Clearly, missing some necessary postcondition can preclude any plan being formed; similarly, some extraneous postcondition can wrongly clobber the preconditions of a necessary future operator. Either a missing or an extra postcondition can allow incorrect plans to be formed: missing postconditions, because the plans would have been clobbered had the postcondition been known; extra postconditions, by enabling other operators that are actually not possible in reality. These incorrect plans might be detected during planning, as a multiple plan failure, or during execution, as either unmet pre- or postconditions. Again, here the planner’s model of the world does not match reality. If the system has explicit knowledge of what reality is supposed to be like, it may detect the reaching of an impossible state during planning, or an explicit failure during execution.

This leaves the overgeneral preconditions category. Overgeneral preconditions cannot preclude a plan being formed, since they simply loosen the restrictions on choosing the operator. However, they may allow incorrect plans to be formed in addition to the correct plan. These incorrect plans might lead to a state that is known to be impossible to reach, or (if executed) to a state that can be explicitly detected as an execution failure. Overgeneral preconditions can lead to the faulty operator not being applicable at execution time. However, this is detected not as a precondition failure (the system thinks the operator’s preconditions are met), but as a postcondition failure, when it is detected that none of the operator’s postconditions have been achieved. Overgeneral preconditions cannot lead to unmet preconditions for some future operator during execution, because if the faulty operator is able to be executed, the system’s model of the world will be correct (the effects of the operator are correct); if the faulty operator is unable to be executed, this will be detected before any further operators are attempted.

Finally, we can briefly examine the columns of the table. The impossible state and explicit detection columns are identical, because in both cases the domain theory error leads to a world state that is explicitly known to be a failure. The dots in the multiple plans column correspond to the union of the dots in all three execution failure columns. An inconsistent plan may be detected during planning, if multiple plans are considered, or it may be detected when executed, by leading to any of the three types of execution failures.

4 Knowledge Sources for Correcting Domain Theories

The previous section described the ways in which various types of domain theory errors might manifest themselves during planning or execution. But once an error is detected, what is to be done about it?

There is a tradeoff between simply fixing the *error*, and fixing the domain theory that led to the error. In some cases, fixing the domain theory may not be cost effective. For example, consider an execution error in which a slippery block slips out of a robot arm's gripper. If the block only slips one in a thousand times, it is probably enough for the system to simply grab the block again when it falls. However, if the block slips ninety percent of the time, it is probably better to fix the underlying domain theory (for instance, to squeeze harder for suspected slippery objects). This tradeoff has not been examined in detail, and we will not discuss it further here.

Let us assume, then, that in response to an error, the system will attempt to improve its domain theory. The problem can be viewed as a search through the space of domain theories. The goal is to alter the current domain theory, usually in the smallest amount possible, so that the new domain theory will not commit the error.

This search can be formalized as follows. Let us call the original domain theory T . T is a set of operators. We will denote the preconditions of an operator op as $Pre(op)$, and the effects of op as $Post(op)$.

We assume that the pre- and postconditions of operators may be generalized or specialized in known ways. For a pre- or postcondition p , $Gen(p)$ returns the set of nearest generalizations of p , and $Spec(p)$ returns the set of nearest specializations.

As an example of how Gen and $Spec$ might work, consider the case in which all possible predicates in the domain are organized into a generalization hierarchy. The hierarchy specifies a set of generalizations and specializations for each predicate. For example, the condition $isa(?X,banana)$ might have the (nearest) generalization $isa(?X,fruit)$. If such a hierarchy is present, it is easy to define $Gen(p)$ and $Spec(p)$: each simply traces either up or down the generalization hierarchy, for each of the predicates appearing in p . However, note that errors in the generalization hierarchy could preclude correcting the domain theory error.

The search for a correct domain theory is a heuristic search through the space of possible domain theories, starting at the current theory. The states of the space are full-fledged domain theories, and the operators alter a domain theory to produce a new one. These *theory revision operators* either alter an existing operator of the domain theory or create a new one:

- **ReplacePre**(op, p) - Replaces the preconditions of operator op with preconditions p .
- **ReplacePost**(op, p) - Replaces the postconditions of operator op with postconditions p .
- **CreateNewOp** - Creates a new, empty operator and adds it to the domain theory. The new operator has no preconditions or effects.

At each step in the search for a new domain theory, we may either generalize or specialize a precondition or effect of an operator, or add a new, empty operator. Thus, the set of possible operators that could be applied to alter a domain theory at each step is:

$$\begin{aligned}
& \forall op \in T[\forall p \in Gen(Pre(op))\mathbf{R}eplacePre(op, p)] \cup \\
& \forall op \in T[\forall p \in Spec(Pre(op))\mathbf{R}eplacePre(op, p)] \cup \\
& \forall op \in T[\forall p \in Gen(Post(op))\mathbf{R}eplacePost(op, p)] \cup \\
& \forall op \in T[\forall p \in Spec(Post(op))\mathbf{R}eplacePost(op, p)] \cup \\
& \mathbf{C}reate\mathbf{N}ew\mathbf{O}p
\end{aligned}$$

Clearly, this is an infinite search space, including all possible domain theories that can be constructed out of the domain’s predicates. To make the problem tractable requires strong biases and knowledge to control the search. In the rest of this section, we attempt to classify the types of biases and knowledge sources that are used.

Nearly all research on correcting domain theories has biased the search in domain theory space by using some type of hill-climbing. That is, only a single “node” of the search tree - a single variant domain theory - is stored and considered for further modification. In addition, most assume even stronger biases; for example, it is typical to assume that the domain theory errs in only a single operator (or is missing only a single operator). For example, if an execution error occurs, it might be assumed that the error was caused by the most recent operator being executed. This type of assumption greatly reduces the search space. Some systems make assumptions about which types of conditions to consider altering. For example, Gil [1991a,1991b] use heuristics such as considering only properties of objects being directly manipulated. Finally, it is often assumed that a single operator will only be altered in a single pre- or postcondition, thus reducing the depth of the search to a constant.

The search being done to alter a domain theory must solve a form of the credit assignment problem. Credit assignment is the problem of determining which of a possible set of causes is responsible for some observed effect. Successfully correcting the domain theory involves determining which knowledge (or lack of knowledge) was responsible for the failure, and altering the theory accordingly. In general, credit assignment is a very difficult problem. It is not unique to correcting domain theories, but shows up any time an effect must be attributed to one or more of a set of possible causes. This is a ubiquitous problem in machine learning. It is faced, for example, by concept learning programs, which must determine which combinations of a set of features determine category membership. It is also faced by reinforcement learning systems (e.g., [Sutton, 1990; Holland, 1986; Rumelhart and McClelland, 1986]) that learn which actions will achieve which effects in the world, by receiving positive feedback from the environment when some goal is met.

There are three basic sources of knowledge that a system can use to guide the search for a corrected domain theory:

1. **Internal Knowledge.** The system may use knowledge that it already has. For example, the system might have a complete (but intractable) deductive theory of the domain, or it might have knowledge of past cases of failures which can be modified to account for the current case.
2. **Teacher.** The system may receive knowledge from outside itself, from a teacher. This knowledge might take the form of direct advice (for example, indicating which knowledge in the domain theory is faulty), or perhaps carefully selected examples that will guide the system to the proper revision of the theory.
3. **External World.** The system might make use of observations of the external world; that is, additional examples. These examples might be randomly chosen (such as those

given to a typical concept learner). Alternatively, the examples might be *generated* by the system, in order to distinguish between alternative theory revisions.

These sources of knowledge can overlap and interact. For example, if a teacher gives the system a key example, the example consists of knowledge from the external world that has been wisely selected by the teacher.

Next, we examine each source of knowledge in more detail.

4.1 Internal Knowledge

By internal knowledge, we mean knowledge that the system already contains, which provides guidance in the search through the space of theory revisions. We have already mentioned one general form of internal knowledge that limits the search for a corrected theory - namely, the assumptions and biases that the system uses in performing the search. Many systems contain heuristic biases that limit the range of theory revisions considered. For instance, Gil [1991a,1991b] discusses heuristics such as assuming immediate feedback for operator effects. This allows only the most recently executed operator to be considered for revision upon execution failure, greatly reducing the credit assignment problem.

Another type of internal knowledge, which is available to all systems, is the original domain theory itself. Although the domain theory is imperfect, it is generally still useful in narrowing the set of possible alterations needed to fix the theory. For example, in analysis tasks, many systems have dealt with the problem of not being able to fully explain an example (corresponding to our “incomplete plan” category). Often these systems will form a maximal *partial explanation* using its incomplete domain theory. The parts of the example which are left unexplained can then be focussed on, to generate a set of hypotheses for the lack of knowledge within the domain theory [Pazzani, 1988; VanLehn, 1987; Hall, 1986]. Empirical techniques are often used to discriminate these hypotheses (discussed further below).

In addition to the domain theory used to perform the task, the system may have other knowledge which it can use to analyze and understand its failure. Below, we investigate additional types of internal knowledge the system may have.

4.1.1 Complete Underlying Domain Theory

In some domains, it is possible to write a domain theory which is complete, but intractable. For example, it is easy to write down the rules of chess completely and correctly, but clearly intractable to use those rules to reason completely and correctly (and thus play perfect chess). In such cases, one way to reason tractably on the task is to use an *approximate* domain theory. For example, a novice player’s domain theory for chess might contain the rule, “If you can capture the opponent’s queen, always do so.”

The approximate domain theory, used by the EBL system in trying to perform its task, will be imperfect, and errors will occur when using it. In such cases, it may be possible to use the intractable domain theory to analyze and correct the error in the approximate theory.

If the theory was originally intractable, why might it be tractable to use it once an error arises? The reason is that explaining a specific error is much more constrained than planning to avoid any *possible* error. Consider the situation in chess. It is much easier to explain why your queen was taken in a particular game, after a particular sequence of moves, than it is to *plan* a move that will truly minimize the possibility of losing your queen later on. In the

first case, the full sequence of moves has been chosen; it simply needs to be analyzed. In the second case, we must consider every possible move that might be taken from this point in the game on.

In terms of search through the space of theory revisions, using a complete underlying domain theory to explain a failure corresponds to reasoning to pinpoint exactly the correct theory revision operator(s) to use in correcting the domain theory. Since the underlying theory is complete and correct, it is able to completely solve the credit assignment problem.

Gupta [1987] describes a scheme in which an complete underlying domain theory is used to explain an execution-time error. In Gupta's system, the domain theory is approximated by not considering certain goal interactions. Once a plan is produced, errors are detected by explicit failure rules, which monitor a plan's simulated execution and fire when failures occur. For example, a failure occurs if the robot's gripper melts when trying to grasp a part. An explanation of the conditions leading to the failure is then produced, using the underlying complete domain theory. This explanation pinpoints the preconditions that should be added to the operators in the approximate domain theory that is used for planning.

Other researchers have forwarded similar schemes, approximating a complete domain theory in various ways. Chien [1989] approximates a complete domain theory by initially reasoning only with the *direct effects* of operators, and assuming persistence for all other facts. When plans fail (or unexpectedly succeed) during execution, the inferred effects of operators are reasoned with to explain the failure. Here again, the intractable theory is made tractable when it is used to explain a particular interaction between operators, as opposed to planning for every possible contingency. Tadepalli [1989] approximates a domain theory for chess by considering only a limited subset of possible moves from each board position (those moves the system has learned in observing past games). Bennett [1987] deals with intractability in mathematical reasoning by approximating mathematical formulas. Doyle [1986] presents a system given a set of domain theories at multiple abstraction levels, which reasons with the most abstract first, and falls back on more and more concrete theories as failures arise. FAILSAFE-2 [Bhatnager and Mostow, 1990] learns overgeneral search heuristics during planning by assuming the most recent operator applied is to blame for search failures, but later specializes these heuristics when all search paths are eliminated. Bennett's [1990] GRASPER system approximates how far a gripper should be opened to pick up an object, and upon failure, uses a theory of tuning continuous approximations to alter the opening width. Ellman [1988] describes a methodology for choosing good simplifying assumptions for certain types of intractable theories. Flann [1990] approximates a domain theory by assuming a limited number of objects will be present.

Other work on approximating a domain theory to make it tractable to reason with has appeared under a different guise, namely abstraction planning. ABSTRIPS [Sacerdoti, 1974] is the classic example, showing that search could be greatly reduced by abstracting out preconditions of operators during planning. More recently, Unruh and Rosenbloom [1989] and Knoblock [1990,1991] have demonstrated methods that automatically abstract preconditions from operators, producing abstract versions of problem spaces to improve efficiency.

4.1.2 Other Internal Knowledge

Other types of internal knowledge besides a complete domain theory may be used in correcting imperfect domain theories as well. Some examples include using *analogy* to relate the error to knowledge in related domains; using *past cases* in which similar errors have been

encountered and corrected; and using the original domain theory *abductively* to determine possible causes of an error. Not as much research has been done using these knowledge sources as using a complete but intractable domain theory, however.

Work in using analogy to repair domain theories has thus far been limited to making simple analogies to other operators within the faulty theory. An unknown operator or an operator being corrected can be compared with known operators that have similar preconditions or effects. Differences indicate revisions to the operator being corrected that may prove useful. Thus in analogy, the system makes use of previous knowledge of operators to guide the search for theory revisions.

Gil [1991a,1991b] compares an operator being revised with other operators which are structurally similar, identifying differences between them. This process is used to filter the set of possible revisions to the operator. The remaining revisions are chosen between via experimentation (discussed further below). Genest *et al.* [1990] try to understand an unknown goal concept (`borrow(Person, amount)`) by explaining examples of it as if they were examples of a known goal concept (here, `withdraw(Person, amount)`). These explanations are incomplete, and are completed using an abductive process (see below). The explanation structure produced using the known concept can then be analogized as similar to the structure of the explanation for the unknown concept, allowing the unknown concept to be learned. Kodratoff and Tecuci [1987] present a system which proposes new domain theory rules by analogizing the conditions of a newly learned rule to similar substructures within a semantic network of domain predicates.

Case-based approaches rely on storing complete past cases, and altering these cases to apply to new situations. CHEF [Hammond, 1986] uses a complete causal theory to explain and repair execution-time errors, but then stores the repaired plan in its case library, indexed under the type of error repaired. When a similar error is predicted in the future, the system can recall the past case and alter it to apply to the new situation. Redmond [1989] discusses using a case-based approach to explain unexpected situations.

Abduction involves reasoning from effects to causes. Logically, it can be viewed as using implication rules “backwards”. For example, suppose we know $LightswitchPosition(off) \supset Darkness$. Then, if the room becomes dark, we might use this rule backwards, positing that the reason for the darkness is that the lights were turned off. If there are multiple conditions that imply darkness, we may have to use additional knowledge to choose between them. Thus abduction can be viewed as a way of limiting the possible theory revisions being considered. OCCAM [Pazzani, 1989; Pazzani *et al.*, 1987] uses an abductive method to fill in missing knowledge in its domain theory. When some aspect of an observation cannot be explained, OCCAM looks for a domain theory rule that contains that aspect as its result. This rule’s conditions are then generalized so that the rule covers the current case. This is a dangerous step, however, because overgeneralizations can easily result. OCCAM does not deal with the situation where multiple rules have the unexplained aspect as their result, and additional knowledge must be used to choose between them. Genest *et al.* [1990] also use abduction to complete partial explanations, but do not extend the domain theory using the abductive conclusions.

Determinations [Davies and Russell, 1987] are a type of internal knowledge that provide capabilities similar to abduction. A determination makes explicit the functional dependencies between attributes within the domain. It is similar to an implication, but not as strong: if A, B, and C *determine* D, this means that for given values of A, B, and C, D will always

have the same value. If A, B, and C *imply* D, then given values of A, B, and C, we know what the value of D *is*, not simply that it is fixed. Determinations are thus “incomplete,” but in a very limited way. Given examples, we can directly turn a determination into a set of implications. This kind of domain theory completion is demonstrated by Mahadevan [1989]. Widmer [1989] uses partial determinations, which guide the construction of “plausible” explanations that may then be confirmed or disconfirmed by a teacher. (Widmer’s system allows additional types of domain theory incompleteness as well, and incorporates inductive learning techniques). A related method of constructing plausible explanations from incomplete knowledge is described by Oblinger and DeJong [1991].

4.2 Teacher

In addition to internal knowledge, knowledge from outside the system itself may be called upon in correcting the domain theory. By knowledge from a teacher, we mean knowledge given by an informed source, meant to direct the system to the right correction of its domain theory. This is as opposed to knowledge from the external world which is either selected randomly, or selected by the system (experimentation).

A number of systems use a teacher as a *passive* source of knowledge. In these systems, the teacher simply verifies that a domain theory revision is or is not correct. The system uses some other knowledge or bias to generate hypotheses about how to fix the domain theory, and then these are validated or invalidated by the teacher. Examples of systems using this kind of passive teaching include [Kodratoff and Tecuci, 1987; Widmer, 1989].

A more interesting use of knowledge from a teacher are cases where the teacher can actively guide the domain theory repair process. Two main categories of teaching have been used. First, the teacher may provide examples to the system. These could be complete plans for the system to “observe,” or unsolved problems for the system to solve. The examples are chosen to help the system repair its domain theory by causing it to concentrate on the right aspects of the situation. The second category of teaching is where the teacher provides advice to the system directly.

4.2.1 Teacher provided examples

Examples provided by a teacher can direct the system to faulty knowledge in its domain theory, and can be used to guide the correction process. In some systems, assumptions about the structure of the examples provides a strong bias that allows the system to solve the credit assignment and revision problems. One such system is VanLehn’s SIERRA [VanLehn, 1987]. SIERRA learns to perform multi-column subtraction by explaining examples of worked out problems given by a teacher. SIERRA receives its examples organized into groups called *lessons*. The key assumption is that the examples in each lesson will differ from those seen in the past by a single disjunct (or operator in our terminology); this is the “one-disjunct-per-lesson” constraint. This constraint limits the search in the space of theory revisions to considering revisions in a single operator. SIERRA learns by completing partial explanations of examples. The one-disjunct constraint greatly reduces SIERRA’s search space when completing the explanations, which could be completed in a number of different ways. Since every example within a lesson is known to be incompletely explained due to the lack of the same disjunct, the “holes” in their explanations can be intersected to hone in on the missing disjunct.

Hall [1986] relies on a teacher to supply paired examples of structures that are equivalent. The system attempts to explain why the two structures are equivalent using rules that can transform one into the other. When a complete explanation cannot be found, the system creates a new transformation rule equating the two parts of the examples which were unable to be explained. Roy and Mostow [1988] use a similar approach.

Other systems make use of teacher-provided problems that are less structured. For example, in Tadepalli's [1989] system, the teacher appears to provide examples simply to "break" the domain theory - examples the teacher knows the system will not perform on correctly. Presumably such examples would eventually come up if examples were randomly selected. Here the teacher simply speeds up that process. Likewise, Redmond's [1989] system takes worked-out examples from a teacher, but these examples are not necessarily structured in any particular way.

4.2.2 Advice

Learning from advice (or "learning by being told") is a rather broad category. Advice could conceivably be given to help at any stage of a problem. In the context of recovering from an error caused by an incorrect domain theory, advice could specify how to recover, and/or how to correct the domain theory itself. This could be more or less direct; for instance, the advice could specify exactly which operator was incorrect, or could only indicate which features of the example caused the error.

There has not been much work on correcting imperfect domain theories by taking advice. Mostow's FOO [1981,1983] was a key early system. Mostow's focus was on making advice *operational* - that is, transforming it into a form the system could use directly. FOO took advice for the card game Hearts, such as "Avoid taking points." The operationalization process involved transforming the advice using a large number of both domain-independent and domain-specific transformation rules. However, which transformation to apply had to be manually selected. The result of operationalizing a piece of advice was a set of heuristics for the game.

FOO's initial domain theory of hearts contained the basic rules of the game. In a sense, the learning FOO did can be viewed as using advice to learn a tractable, approximate theory from an intractable one. However, advice was not used to correct imperfections in the tractable domain theory (the one used to actually play the game).

Martin and Firby [1991] have begun developing a system that learns to correct execution-time failures by being told. The approach is a combination of Martin's DMAP parser [Martin, 1989] and Firby's RAPs system [Firby, 1987]. Upon an execution failure, the system takes advice in natural language indicating how to correctly perform the task. Comprehending the language input is aided by specific expectations which are set up by the failure. The advice allows the system to select an operator to complete performance of the task. The system learns to perform the task correctly next time. However, the issue of how to properly generalize the applicability of the advice is finessed. The system does not attempt to identify the relevant aspects of the state that led to the failure; rather, it seems to assume that the entire state is relevant. This may result in overgeneralizations in some cases.

Another example of using information given by a teacher to correct an imperfect domain theory is the Robo-Soar system [Laird and Rosenbloom, 1990; Laird *et al.*, 1990]. Robo-Soar's domain theory for picking up blocks is incorrect, because it does not realize that for certain pyramid-shaped blocks, the orientation of the gripper must match the orientation

of the block (otherwise the gripper fingers slide off the sides of the pyramid, and the block cannot be picked up). When an execution failure occurs (the block isn't picked up), RoboSoar prepares to search to discover which operator was at fault and what conditions of the state caused the error. Advice from the user guides this search. The guidance is extremely direct - the user indicates exactly which feature (here, block orientation) caused the failure, and which operator should be used to correct it (here, `rotate-gripper`). A similar approach is used to extend domain knowledge, in the case where the `rotate-gripper` operator is entirely missing from the original domain theory. Here, the advisor indicates the pre- and postconditions of the missing operator, and its implementation as an external command.

We are working on extending this work in two directions. The first is allowing more flexible types of instruction and advice, given in natural language. This again raises the issue of operationalization: how can the advice be translated into a form the system can use and learn from in a general way? Secondly, we are working on experimentation techniques, that will allow the system to learn when advice is not available. Instead of performing all the experiments that would be needed to exactly identify the domain theory, the system will use inductive techniques in an attempt to generalize the information gleaned from each experiment.

4.3 External World

In addition to internal knowledge or knowledge provided by a teacher, the external world - the execution environment - may be viewed as a knowledge source for correcting imperfect domain theories. This knowledge source has the property of being reliable and necessarily "correct". The system is faced with the problem of finding a domain theory which agrees with this knowledge source.

Specific observations of the external world can be used to constrain the search space of possible domain theories. These observations might be randomly selected; for instance, the system might be given a set of randomly chosen examples in a concept formation task. Alternatively, the observations might be selected by an informed source. A teacher may be able to give the system examples that will highlight the faulty knowledge in the domain theory, and its proper repair (as discussed above). Without a teacher's input, the system may be able to generate *experiments* with the intent of eliminating theories from the space or of confirming the predictions made by a particular theory. Correcting a domain theory using knowledge gleaned from experimentation is an *active* use of the external world; using randomly selected observations is a *passive* use.

4.3.1 Active Use of the External World

Consider the case where a domain theory for a STRIPS-like domain has the operator **Go-thru-door(door)**, lacking the precondition **Open(door)**. Upon execution of some plan, the robot executes **Go-thru-door** but finds its postconditions unmet. Even if it is possible to localize the error to the preconditions of the **Go-thru-door** operator and make use of some internal knowledge to prune the set of possibilities, the system may still be left with a number of hypothetical corrections to the domain theory. For example, the system might hypothesize that either **Open(door)** or **-Inroom(box, room)** (there are no boxes in the room the robot starts from) could be added as a new precondition. At this point, the system might use the external world as a knowledge source by experimenting to select the correct theory

update from the set of candidates. In this case, the robot could be moved through another door which happens to be open (and succeed), or might remove all the boxes from the room it is in, and attempt **Go-thru-door** again (and fail, if the door were still closed). By carefully selecting experiments to perform, the system is able to directly prune hypotheses for domain theory correction from consideration.

Since the number of domain theory alteration operators which may be applied to the incorrect domain theory is quite large, and experimentation is fairly expensive, typically experimentation systems employ various biases and heuristics to limit the set of theory changes to be differentiated with experiments. The idea is to limit the set of hypotheses as much as possible before performing experiments. Rajamoney and DeJong [1988], for example, appear to use a form of abduction to complete an incomplete explanation, leading to a set of multiple inconsistent explanations. Experimentation is used to choose among the various abductive assumptions and correct the domain theory. Gil [1991a,1991b] employs a number of heuristics, such as locality of action, analogy to similar operators, and generalization of previous experiences where an operator was successful to restrict the set of possible alterations to the preconditions of the operator. In addition, immediate feedback from actions is assumed, restricting the domain theory error to the most recently executed operator. Carbonell and Gil [1987] use heuristics based on the type of failure observed to guide the theory correction process. For example, if an operator was applied but none of its postconditions were met, it is likely that the preconditions of the operator are overgeneral (as opposed to, say, that all of its postconditions are wrong). Previous cases where the operator was successful and internal searches for other operators which may have clobbered the failing operator are used to limit the number and type of experiments to be run by the system.

Once the set of hypotheses has been sufficiently pruned, a system can formulate experiments to select between them. A single experiment may eliminate multiple hypotheses from consideration; thus, if experimentation is expensive, it may be worthwhile to do some analysis to determine the minimal number of experiments which are guaranteed to fully distinguish the hypothesis. This is a “minimal set-covering” problem, similar to that done to find a minimal set of test vectors for a digital integrated circuit.

4.3.2 Passive Observation of the External World

In some domains, it is not possible to actively control the set of examples the system receives. Rather, the major source of corrective knowledge here is a set of randomly chosen examples. The systems which rely on this type of knowledge typically make use of inductive techniques (such as version spaces [Mitchell, 1982] or information-theory approaches [Quinlan, 1986]) to determine a set of changes to the domain theory that will be consistent with all (or most) examples. Since the discriminating knowledge that can be gleaned from each example is less concentrated than for experimentation approaches, more examples are required for these systems to correct a faulty domain theory.

Pure inductive systems, such as ID3[Quinlan, 1986], can be viewed as learning an entire domain theory from scratch. This is the degenerate case of an “incomplete domain theory.” In this paper, the focus is on correcting an already existing domain theory that has proven faulty. Closely related are techniques in which an existing domain theory is used to bias an inductive learner, allowing induction to converge more quickly on correct concept definitions [Flann and Dietterich, 1989; Towell *et al.*, 1990; Bergadano and Giordana, 1988; Danyluk, 1987].

Consider our example of the previous section. In the absence of additional examples, the system might initially add *both* preconditions - `Open(door)` and `-Inroom(box,room)` - to the **Go-thru-door** operator. However, these preconditions may be marked as tentative. Later, when the system is able to go through an open door from a room with boxes in it, it can generalize the preconditions of **Go-thru-door** by removing `-Inroom(box,room)`.

Ali [1989] presents a system that uses a similar technique to learn a new operator. In this system, an overspecific tentative rule is induced to cover an unexplained concept. This rule is produced by conjoining all “related” predicates in the partial proof, where related predicates are those that share arguments. When future failures occur, tentative rules formed from the partial explanation of the new example are intersected with the original tentative rule, generalizing it. Eventually the rule stabilizes. This technique thus learns a “missing operator” - a new rule that the system didn’t have before. The approach is similar to VanLehn’s technique of intersecting partial explanations, but here there are no assumptions about the form of the examples. Thus the rules formed are sometimes faulty.

OCCAM [Pazzani, 1988] also uses an empirical approach to learn missing domain theory rules. Part of the knowledge OCCAM uses to build explanations is a very general set of rules describing causal relationships. These rules derive an abstract “explanation pattern” that is then verified by more specific domain knowledge. When an example cannot be fully explained, OCCAM uses the explanation pattern to pinpoint which step in the causal chain was not verified, and induces a new domain theory rule to verify the causal link. On subsequent examples, the new rule is generalized by an intersection technique, as in Ali and VanLehn’s work. Thus the causal knowledge (internal knowledge) constrains the domain theory corrections that are considered, and induction over examples selects the correct one from amongst them. OCCAM goes a step further by maintaining strengths for empirically induced rules, and eliminating rules that prove incorrect on future examples. This protects the system from overgeneralizations; it might be viewed as a composite “specialization” operator.

Danyluk [1989] discusses a related approach in which a similarity-based learner is used to induce missing domain theory rules. Here, the partial explanation is used to bias the priorities of features for the similarity-based learner, providing a “context” for induction. Widmer [1989] allows similarity-based “explanations” as part of the full explanation of an example. Some feature of the example that cannot be explained either deductively or via determinations (see the discussion of Other Internal Knowledge) can be explained by searching for similarities between the example and previous examples which share the feature. Similarly, Wilkins’ ODYSSEUS [1988] proposes relationships between predicates which are needed to complete incomplete explanations, and then uses induction over examples to verify the proposed relationship and learn a specific rule relating the predicates. In all these systems, as in the experimentation systems, we see some internal knowledge or heuristics being used to initially constrain the set of domain theory revisions that is considered, and then external world knowledge (examples) being used to make (or verify) the final decision.

One of the more comprehensive attempts to use inductive methods to correct imperfect theories in a classification task is presented by Ourston and Mooney [1990]. Their system, EITHER, identifies two types of errors: overgeneral preconditions, which lead to incorrect classifications (an explicit detection of failure), and overspecific preconditions (or missing rules), which prevent an example from being classified (an incomplete plan). If examples cannot be classified, EITHER constructs partial proofs of the examples, and then determines

	Internal K		Teacher		
	Complete DT	Other	Examples	Advice	
Overgen. Preconds.	Gupta [87] Chien [89] Bennett [90]	Gil [91] (biases)		Robo-Soar Martin&Firby [91]	Ou
Overspec. Preconds.		OCCAM (abduction) Widmer [89] (determinations)			Ou
Missing Postconds.	Chien [89] (Gupta [87])			Robo-Soar	
Extra Postconds.					
Missing Operator		OCCAM (causal schemas) Wikins [88] (conf. theory) Widmer [89] (determinations) Genest+ [90] (analogy)	Hall[86] VanLehn [87] (Wilkins [88])	Robo-Soar	Ou

Figure 2: Error type versus corrective knowledge source

a minimal set of assumptions (new rules) needed to complete the proofs (using a greedy covering algorithm). If examples are incorrectly classified, EITHER uses its covering algorithm to find a minimal set of rules deemed responsible for the error. However, simply adding or removing rules may introduce new domain theory errors. If this is the case, then EITHER invokes an inductive learner (ID3) to *specialize* either the new rules being proposed or the old rules which caused incorrect classification.

5 Analysis Summary

We have presented classes of domain theory error types, the types of performance failures that can result, and knowledge sources for correcting errors. By crossing these categorizations with one another, we can examine which types of errors have been addressed using which types of knowledge for theory correction. Since this is a knowledge-level analysis, we are interested in examining which types of knowledge have been applied to which types of theory errors. For each knowledge source and error type, there may be a number of *mechanisms* that can be used in correcting the domain theory; we do not consider that aspect here.

We have chosen fifteen representative systems to classify (two or three for each type of corrective knowledge source). Figure 2 plots the space of error types versus corrective knowledge sources; Figure 3 plots out performance failure types versus knowledge sources. Into each square in these grids, we have placed references to those systems that have addressed applying a particular type of knowledge to correct a particular error type, or errors detected in a particular way.

One key thing to notice about these tables is the presence of empty squares, and the clumping of research within other squares in each column. Research using each type of knowledge source has tended to focus on correcting only a subset of the types of errors (cor-

	Internal K		Teacher		
	Complete DT	Other	Examples	Advice	
Incomplete Plan		OCCAM (causal schemas) Wikins [88] (conf. theory) Widmer [89] (determinations) Genest+ [90] (analogy)	Hall [86] VanLehn [87] (Wilkins [88])		V V Ourst
Multiple Plans					
Imposs. State					
Unmet Preconds.	(Chien [89])				
Unmet Postconds.	(Chien [89]) Bennett [90]			Robo-Soar Martin&Firby[91]	
Expl. Detection	Gupta [87] (Chien [89])				Ourst

Figure 3: Performance failure type versus corrective knowledge source

respondingly, errors detected because of only a subset of the performance failure types). This may be because particular sources of knowledge are most amenable to correcting particular types of errors. However, there does not appear to be an a-priori reason why each knowledge source might not be applied to correcting all of the different kinds of errors.

A second thing to notice is that not many systems appear in multiple columns. That is, most systems have used a single knowledge source for domain theory correction, or have combined sources only in fairly simple ways (although there are exceptions). We feel an important area for future research is combining the different types of knowledge available for correction. Different types of knowledge might be available at different times and in different situations. A truly flexible system would be able to employ any and all of the various types of knowledge successfully, whenever such knowledge is available.

6 Relationship to Other Frameworks

Mitchell *et al.* [1986] made the first attempt to describe the basic ways in which a domain theory may be faulty. In their classification there are three categories. First, the theory may be *incomplete*, meaning that no explanation can be constructed for some examples. Second, the theory may be *inconsistent*, in which case conflicting statements can be proved from the theory. Third, the theory may be *intractable*, meaning that it is computationally prohibitive to construct explanations using it.

However, we have shown that a single type of underlying domain theory error may lead to more than one of these categories. For example, a domain theory containing an operator with missing postconditions may result in incompleteness (no plans) or inconsistency (multiple plans). In addition, we have seen that an intractable domain theory is typically not employed for reasoning. If there is a complete but intractable theory available it cannot be employed directly; instead, it is a knowledge source which may be employed to correct approximations and assumptions made in the actual domain theory used for reasoning. Therefore an intractable theory typically gives rise to either an incomplete or inconsistent theory.

Rajamoney and DeJong [1987] further subdivided the classes that Mitchell *et al.* had

defined. They distinguish incompleteness due to a lack of relevant knowledge, as opposed to incompleteness due to a lack of sufficient detail. A lack of relevant knowledge gives rise to incomplete plans; a lack of sufficient detail can lead the system to make assumptions, leading to multiple plans. However, in our view, making assumptions is part of the process of recovering from domain theory imperfections - a way of “generating” possible fixes. The assumptions can then be tested to determine which are correct, perhaps by using experimentation to distinguish between the multiple plans that arise from them. This is the method used by Rajamoney and DeJong [1988]. Similarly, inconsistency problems are subdivided into inconsistency due to incorrect knowledge and inconsistency due to missing knowledge that would have defeated the inconsistent deductions.

Mitchell *et al.*'s classification focuses on the *performance* of the domain theory: does it lead to incomplete explanations? Does it lead to inconsistent proofs? Rajamoney and DeJong's classification moves closer to the classification of domain theory problems we have proposed here, in that there is more of a focus on the *knowledge* within the domain theory. However, performance is still the primary division, but subdivisions are attempted based on the type of knowledge deficiency giving rise to the performance failure.

In developing our classification, we have instead used the *type of knowledge deficiency* within the domain theory as the primary distinction. The goal was to cleanly separate the type of domain theory imperfection from the way in which it manifests itself in performance. Starting from the types of knowledge deficiency, we were able to derive the types of performance failures that each deficiency can lead to. This was found to be a one-to-many mapping: the various knowledge deficiencies can give rise to multiple types of performance failures. This one-to-many mapping has caused a confounding of error type (knowledge) and error detection (performance) in previous classifications.

Our analysis has revealed that although a number of useful mechanisms for correcting domain theories have been developed, none have yet demonstrated the ability to flexibly make use of any of the possible knowledge sources that might be available to guide the theory revision process. Theory revision is a complex process, taking place in a potentially huge search space. Any knowledge that is available should be utilized to guide the search. Therefore, our future goal is the development of a “universal weak method” of theory revision that is able to flexibly combine whatever knowledge is available to correct a faulty domain theory.

Acknowledgments

Thanks to Greg Weber, Craig Miller, and the other members of the Michigan Soar group for useful comments and discussion of earlier drafts of this chapter.

References

- [Ali, 1989] Kamal M. Ali. Augmenting domain theory for explanation-based learning. In *Proceedings of the International Workshop on Machine Learning*, pages 40–42, 1989.
- [Anderson, 1986] J. R. Anderson. Knowledge compilation: The general learning mechanism. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach, Volume II*, pages 289–310. Morgan Kaufmann, 1986.
- [Bennett, 1987] Scott W. Bennett. Approximation in mathematical domains. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 239–241, August 1987.
- [Bennett, 1990] Scott W. Bennett. Reducing real-world failures of approximate explanation-based rules. In *Proceedings of the International Conference on Machine Learning*, pages 226–234, 1990.
- [Bergadano and Giordana, 1988] Francesco Bergadano and Attilio Giordana. A knowledge intensive approach to concept induction. In *Proceedings of the International Conference on Machine Learning*, pages 305–317, 1988.
- [Bhatnager and Mostow, 1990] Neeraj Bhatnager and Jack Mostow. Adaptive search by explanation-based learning of heuristic sensors. In *Proceedings of the National Conference on Artificial Intelligence*, pages 895–901, 1990.
- [Carbonell and Gil, 1987] Jaime G. Carbonell and Yolanda Gil. Learning by experimentation. In *Proceedings of the International Workshop on Machine Learning*, pages 256–265, 1987.
- [Chien, 1989] Steve A. Chien. Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 590–595, 1989.
- [Danyluk, 1987] Andrea Pohoreckyj Danyluk. The use of explanations in similarity-based learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 274–276, 1987.
- [Danyluk, 1989] Andrea Pohoreckyj Danyluk. Finding new rules for incomplete theories: Explicit biases for induction with contextual information. In *Proceedings of the International Workshop on Machine Learning*, pages 34–36, 1989.
- [Davies and Russell, 1987] Todd R. Davies and Stuart J. Russell. A logical approach to reasoning by analogy. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 264–270, 1987.

- [DeJong and Mooney, 1986] Gerald F. DeJong and Raymond J. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.
- [Doyle, 1986] Richard J. Doyle. Constructing and refining causal explanations from an inconsistent domain theory. In *Proceedings of the National Conference on Artificial Intelligence*, pages 538–544, 1986.
- [Ellman, 1988] Thomas Ellman. Approximate theory formation: An explanation-based approach. In *Proceedings of the National Conference on Artificial Intelligence*, pages 570–574, August 1988.
- [Firby, 1987] R. James Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 202–206, August 1987.
- [Flann and Dietterich, 1989] Nicholas S. Flann and Thomas G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187–226, 1989.
- [Flann, 1990] Nicholas S. Flann. Applying abstraction and simplification to learn in intractable domains. In *Proceedings of the International Machine Learning Conference*, pages 277–285, 1990.
- [Genest *et al.*, 1990] Jean Genest, Stan Matwin, and Boris Plante. Explanation-based learning with incomplete theories: A three-step approach. In *Proceedings of the International Conference on Machine Learning*, pages 286–294, 1990.
- [Gil, 1991a] Yolanda Gil. Acquiring domain knowledge for planning by experimentation. Thesis Proposal, Carnegie Mellon University, School of Computer Science, 1991.
- [Gil, 1991b] Yolanda Gil. A domain-independent framework for effective experimentation in planning. In *Proceedings of the International Machine Learning Workshop*, pages 13–17, 1991.
- [Ginsberg and Smith, 1987] Matthew L. Ginsberg and David E. Smith. Reasoning about action I: A possible worlds approach. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, pages 233–258. Morgan Kaufmann, 1987.
- [Gupta, 1987] Ajay Gupta. Explanation-based failure recovery. In *Proceedings of the National Conference on Artificial Intelligence*, pages 606–610, 1987.
- [Hall, 1986] Robert J. Hall. Learning by failing to explain. In *Proceedings of the National Conference on Artificial Intelligence*, pages 568–572, 1986.
- [Hammond, 1986] Kristian J. Hammond. Learning to anticipate and avoid planning problems through the explanation of failures. In *Proceedings of the National Conference on Artificial Intelligence*, pages 556–560, 1986.

- [Holland, 1986] John H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach, Volume II*. Morgan Kaufmann, 1986.
- [Knoblock, 1990] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 923–928, 1990.
- [Knoblock, 1991] Craig A. Knoblock. Search reduction in hierarchical problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 686–691, July 1991.
- [Kodratoff and Tecuci, 1987] Yves Kodratoff and Gheorghe Tecuci. DISCIPLE-1: Interactive apprentice system in weak theory fields. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 271–273, August 1987.
- [Laird and Newell, 1983] John E. Laird and Allen Newell. A universal weak method: Summary of results. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 771–773, 1983.
- [Laird and Rosenbloom, 1990] John E. Laird and Paul S. Rosenbloom. Integrating execution, planning, and learning in Soar for external environments. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1022–1029, Boston, Mass., 1990. AAAI Press.
- [Laird *et al.*, 1986] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Chunking in Soar: The anatomy of a general learning mechanism. *Maching Learning*, 1(1):11–46, 1986.
- [Laird *et al.*, 1987] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.
- [Laird *et al.*, 1989] John E. Laird, Eric S. Yager, Christopher M. Tuck, and Michael Hucka. Learning in tele-autonomous systems using Soar. In *Proceedings of the NASA Conference on Space Telerobotics*, 1989.
- [Laird *et al.*, 1990] John E. Laird, Michael Hucka, Eric S. Yager, and Christopher M. Tuck. Correcting and extending domain knowledge using outside guidance. In *Proceedings of the International Machine Learning Conference*, pages 235–241, 1990.
- [Mahadevan, 1989] Sridhar Mahadevan. Using determinations in EBL: A solution to the incomplete theory problem. In *Proceedings of the International Workshop on Machine Learning*, pages 320–325, 1989.
- [Martin and Firby, 1991] Charles E. Martin and R. James Firby. Generating natural language expectations from a reactive execution system. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 811–815, August 1991.
- [Martin, 1989] Charles E. Martin. Pragmatic interpretation and ambiguity. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 474–481, August 1989.

- [Minton *et al.*, 1989] Steven Minton, Jaime G. Carbonell, Craig A. Knoblock, Daniel R. Kuokka, Oren Etzioni, and Yolanda Gil. Explanation-based learning: A problem-solving perspective. *Artificial Intelligence*, 40:63–118, 1989.
- [Mitchell *et al.*, 1986] Tom M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1, 1986.
- [Mitchell, 1982] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
- [Mitchell, 1990] Tom M. Mitchell. Becoming increasingly reactive. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1051–1058, Boston, Mass., 1990. AAAI Press.
- [Mooney, 1990] Raymond J. Mooney. Learning plan schemata from observation: Explanation-based learning for plan recognition. *Cognitive Science*, 14:483–509, 1990.
- [Mostow and Bhatnager, 1987] Jack Mostow and Neeraj Bhatnager. Failsafe - a floor planner that uses EBG to learn from its failures. In *Proceedings of IJCAI-87*, pages 249–255, 1987.
- [Mostow, 1981] David J. Mostow. *Mechanical transformation of task heuristics into operational procedures*. PhD thesis, Carnegie-Mellon University, Department of Computer Science, April 1981.
- [Mostow, 1983] D. J. Mostow. Learning by being told: Machine transformation of advice into a heuristic search procedure. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An artificial intelligence approach*. Morgan Kaufmann, 1983.
- [Oblinger and DeJong, 1991] Daniel Oblinger and Gerald DeJong. An alternative to deduction. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 837–841, 1991.
- [Ourston and Mooney, 1990] Dirk Ourston and Raymond J. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the National Conference on Artificial Intelligence*, pages 815–820, 1990.
- [Pazzani *et al.*, 1987] Michael Pazzani, Michael Dyer, and Margot Flowers. Using prior learning to facilitate the learning of new causal theories. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 277–279, 1987.
- [Pazzani, 1988] Michael J. Pazzani. Integrated learning with incorrect and incomplete theories. In *Proceedings of the International Machine Learning Conference*, pages 291–297, 1988.
- [Pazzani, 1989] Michael Pazzani. Detecting and correcting errors of omission after explanation-based learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 713–718, 1989.
- [Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

- [Rajamoney and DeJong, 1987] Shankar Rajamoney and Gerald DeJong. The classification, detection and handling of imperfect theory problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 205–207, 1987.
- [Rajamoney and DeJong, 1988] Shankar A. Rajamoney and Gerald F. DeJong. Active explanation reduction: An approach to the multiple explanations problem. In *Proceedings of the International Machine Learning Conference*, pages 242–255, 1988.
- [Redmond, 1989] Michael Redmond. Combining case-based reasoning, explanation-based learning and learning from instruction. In *Proceedings of the International Workshop on Machine Learning*, pages 20–22, 1989.
- [Roy and Mostow, 1988] Subrata Roy and Jack Mostow. Parsing to learn fine grained rules. In *Proceedings of the National Conference on Artificial Intelligence*, pages 547–551, 1988.
- [Rumelhart and McClelland, 1986] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing*. MIT Press, Cambridge, MA, 1986.
- [Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.
- [Sutton, 1990] Richard S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, 1990.
- [Tadepalli, 1989] Prasad Tadepalli. Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 694–700, 1989.
- [Towell *et al.*, 1990] Geoffrey G. Towell, Jude W. Shavlik, and Michiel O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the National Conference on Artificial Intelligence*, pages 861–866, 1990.
- [Unruh and Rosenbloom, 1989] Amy Unruh and Paul S. Rosenbloom. Abstraction in problem solving and learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1989.
- [VanLehn, 1987] Kurt VanLehn. Learning one subprocedure per lesson. *Artificial Intelligence*, 31(1):1–40, 1987.
- [Widmer, 1989] Gerhard Widmer. A tight integration of deductive and inductive learning. In *Proceedings of the International Workshop on Machine Learning*, pages 11–13, 1989.
- [Wilkins, 1988] David C. Wilkins. Knowledge base refinement using apprenticeship learning techniques. In *Proceedings of the National Conference on Artificial Intelligence*, pages 646–651, 1988.
- [Winston *et al.*, 1983] P. H. Winston, T. O. Binford, B. Katz, and M. Lowry. Learning physical descriptions from functional descriptions, examples, and precedents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 433–439, 1983.