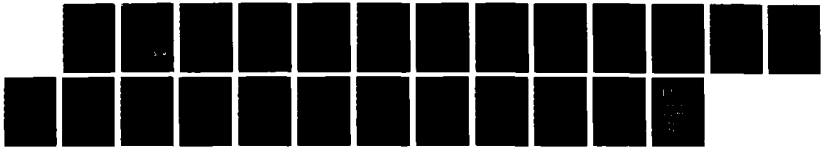
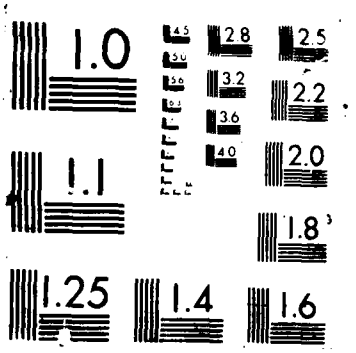
AD-A187 502 KNOWLEDGE BASE REFINEMENT BY MONITORING ABSTRACT CONTROL KNOWLEDGE REVISI..(U) STANFORD UNIV CA DEPT OF COMPUTER SCIENCE D C WILKINS ET AL. AUG 87 1/1

UNCLASSIFIED STAN-CS-87-1182-REV-1 N00014-85-K-0305 F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

August 1987

Report No. STAN-CS-87-1182
Also numbered KSL-87-01

Knowledge Base Refinement by Monitoring Abstract Control Knowledge

by

D. C. Wilkins, W. J. Clancey, and B. G. Buchanan

Department of Computer Science

Stanford University
Stanford, CA 94305

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS AD-A187 502
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S) ONR TECHNICAL REPORT #
6a NAME OF PERFORMING ORGANIZATION STANFORD KNOWLEDGE SYSTEMS LABORATORY 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION PERSONNEL AND TRAINING RESEARCH PROGRAMS
6c ADDRESS (City, State, and ZIP Code) COMPUTER SCIENCE DEPARTMENT 701 WELCH ROAD, BUILDING C PALO ALTO, CA 94304	7b ADDRESS (City, State, and ZIP Code) OFFICE OF NAVAL RESEARCH (CODE 1142PT) 800 NORTH QUINCY STREET ARLINGTON, VA 22217-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-85K-0305
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
61153N	RR04206	0C	NR702-003

11 TITLE (Include Security Classification)
Knowledge Base Refinement by Monitoring Abstract Control Knowledge

12 PERSONAL AUTHOR(S)
David C. Wilkins, William J. Clancey, and Bruce G. Buchanan

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
TECHNICAL	FROM _____ TO _____	August 1987	20

16 SUPPLEMENTARY NOTATION
Also, Knowledge Systems Lab Report KSL-87-01

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
05	09		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

An explicit representation of the problem solving method of an expert system shell as abstract control knowledge provides a powerful foundation for learning. This paper describes the abstract control knowledge of the Heracles expert system shell for heuristic classification problems, and describes how the Odysseus apprenticeship learning program uses this representation to automate "end-game" knowledge acquisition. Particular emphasis is given to showing how abstract control knowledge facilitates the use of underlying domain theories by a learning program.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
DR. SUSAN CHIPMAN	(202) 696-4318	ONR 1142PT

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Knowledge Systems Laboratory
KSL Report No. KSL-87-01

January 1987
Rev. 1: August 1987

Knowledge Base Refinement by Monitoring Abstract Control Knowledge

David C. Wilkins, William J. Clancey and Bruce G. Buchanan

Department of Computer Science
Stanford University
Stanford, CA 94305

To appear in:

Knowledge Acquisition for Knowledge Based Systems, J. Boose and B. Gaines, editors, Academic Press, 1987

and

International Journal of Man-Machine Studies, 1987

Knowledge Base Refinement by
Monitoring Abstract Control Knowledge

David C. Wilkins, William J. Clancey, and Bruce G. Buchanan

Knowledge Systems Laboratory
Department of Computer Science
Stanford University
Stanford, CA 94305

Abstract

An explicit representation of the problem solving method of an expert system shell as abstract control knowledge provides a powerful foundation for learning. This paper describes the abstract control knowledge of the HERACLES expert system shell for heuristic classification problems, and describes how the ODYSSEUS apprenticeship learning program uses this representation to semi-automate "end-game" knowledge acquisition. The problem solving method of HERACLES is represented explicitly as domain-independent *tasks* and *metarules*. Metarules locate and apply domain knowledge to achieve problem solving subgoals, such as testing, refining, or differentiating between hypothesis; and asking general or clarifying questions.

We show how monitoring abstract control knowledge for metarule premise failures provides a means of detecting gaps in the knowledge base. A knowledge base gap will almost always cause a metarule premise failure. We also show how abstract control knowledge plays a crucial role in using underlying domain theories for learning, especially weak domain theories. The construction of abstract control knowledge requires that the different types of knowledge that enter into problem solving be represented in different knowledge relations. This provides a foundation for the integration of underlying domain theories into a learning system, because justification of different types of new knowledge usually requires different ways of using an underlying domain theory. We advocate the construction of a definitional constraint for each knowledge relation that specifies how the relation is defined and justified in terms of underlying domain theories.

1 Introduction

An apprenticeship period is the most effective means that human problem solvers use to refine domain-specific problem solving knowledge in expert domains. This provides motivation to give apprenticeship learning abilities to knowledge-based expert systems, since they derive their power from the quality and quantity of their domain-specific knowledge. By definition, apprentice learning programs improve an expert system in the course of *normal problem solving* and derive their power from the use of *underlying domain theories* (Mitchell et al., 1985).

There are two principal apprenticeship learning scenarios used by human problem solvers in knowledge-intensive domains such as medicine and engineering. In the first scenario, an apprentice problem solver learns in the course of observing the problem solving behavior of another problem solver. A learning opportunity occurs when the apprentice fails to explain an observed problem solving action. At this point, the apprentice can often use the problem solving context and underlying domain theories to identify missing or wrong problem solving knowledge, or at worse be able to ask a pointed question that will isolate the knowledge discrepancy. Our past research focused on this type of scenario: the ODYSSEUS learning program improves a HERACLES-based expert system in the course of watching a human expert solve problems (Wilkins et al., 1986; Clancey, 1986a).

In the second apprenticeship learning scenario, an apprentice problem solver learns in the course of solving problems and monitoring his or her own problem solving failures. This paper describes how the ODYSSEUS learning apprentice can perform this type of learning; the ODYSSEUS learning apprentice improves a HERACLES-based apprentice expert system by having ODYSSEUS monitor the expert system's normal problem solving.

This paper is organized as follows. Section 2 briefly describes the problem solving architecture of the HERACLES expert system shell. The key aspects of HERACLES that are crucial for learning are a separation of the domain knowledge from control knowledge and an explicit representation of the control knowledge using tasks and metarules. Section 3 describes the learning method used by ODYSSEUS,

provides two learning examples, and discusses the generality and limitations of the learning approach. Section 4 covers related research, and Section 5 summarizes the contributions of this paper.

2 Heracles' Problem Solving Architecture

HERACLES is an expert system shell for solving problems using the heuristic classification method; it provides the user with a vocabulary of knowledge relations for encoding domain knowledge, and a domain-independent body of control knowledge that solves problems using this domain knowledge. In HERACLES, control knowledge is represented as *task procedures* and *metarules*, which are invoked by a *task interpreter* (Clancey, 1986b).

A *task* is a procedure for accomplishing some well-defined problem-solving subgoal. Examples of tasks are to test a hypothesis, group and differentiate hypotheses, refine a hypothesis, forward reason, ask general questions, and process hard data. Each action within a task procedure for achieving the task procedure subgoal is called a *metarule*. Metarules, which might more precisely be called "inference procedure rules", do not contain domain knowledge; they index the domain knowledge using a relational language.

The domain knowledge in HERACLES consists of MYCIN-like rules and facts and is encoded using the MRS relational language (Russell, 1985). This knowledge is accessed when metarules premises are unified with domain knowledge relations. There are approximately 120 knowledge relations, such as `subsumes($parm1, $parm2)`[1], `trigger($rule)`, and `evidence.for($parm, $hypothesis, $rule, $cf)`[2]. Tasks and metarules can be viewed as orchestrating the domain knowledge: they piece the domain knowledge together in order to achieve a problem solving goal. Examples of metarules are shown in Section 3. Currently HERACLES

[1]Throughout this paper, all variables start with a "$".

[2]This last relation means that **$parm** contributes evidence for **$hypothesis** in **$rule** and the certainty factor or strength of this rule is **$cf**. If a rule has several parameters in the premise, an **evidence.for** tuple is constructed for each of them.

contains approximately thirty task procedures and eighty metarules.

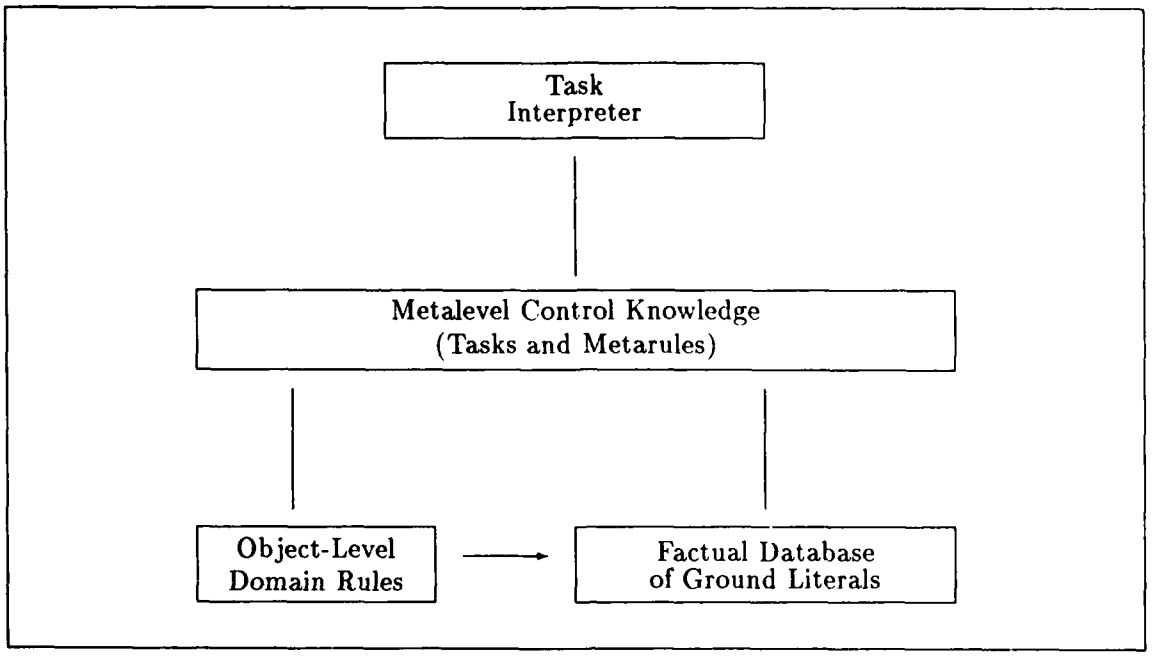


Figure 1: Heracles Problem Solving Architecture. The meta-level (middle layer) is declaratively specified and encodes knowledge of the problem solving method. The object-level (bottom layer) is also declaratively specified and encodes domain-specific knowledge.

The three main levels of organization in HERACLES are shown in Figure 1. The bottom level of organization includes all domain-specific knowledge of the expert domain, such as medical or engineering knowledge. The middle layer contains meta-level control knowledge, which encodes a problem-solving method such as heuristic classification or constraint propagation. Earlier shells such as EMYCIN did not have the middle layer of abstract control knowledge; rather, this knowledge was imbedded in the interpreter and the domain rules.

In the examples in this paper, the domain knowledge base to be refined is the NEOMYCIN knowledge base for diagnosing meningitis and neurological problems (Clancey, 1984). The NEOMYCIN knowledge base is a reorganization and extension of the MYCIN knowledge base, in which distinctions are made between different types

of problem solving knowledge, and the control knowledge is more completely separated from the domain knowledge. The described HERACLES system was actually created by removing the domain knowledge from NEOMYCIN. Patient cases created for the NEOMYCIN domain are used as input (Clancey, 1984). The ODYSSEUS induction theory uses the MYCIN library of solved patient cases (Buchanan and Shortliffe, 1984).

HERACLES metarules have the responsibility for locating and applying all domain knowledge. The form of the metarule provides a way to determine whether the premise of the rule is true by accessing dynamic state information and referencing (and retrieving information from) the domain knowledge base. ODYSSEUS monitors HERACLES metarule premises for failures. If the cause of the failure is missing domain knowledge, ODYSSEUS attempts to create this knowledge using underlying theories of the domain. If ODYSSEUS succeeds in finding the desired domain knowledge, the domain knowledge base in the expert system shell is automatically refined. The metarule for achieving a problem solving subgoal can now be successfully applied.

3 Odysseus' Learning Method

An overview of the learning method to be described is shown in Figure 2. The first major task facing the learning system is global credit assignment, which is the determination of whether there is a potential gap in the knowledge base. The gap can be either a lack of factual or rule knowledge. The use of a relational language for all knowledge, including rules, provides a uniform approach to discovering both types of deficiencies. A gap in the knowledge base is suspected whenever the premise of a metarule fails. Given a failed metarule premise, the learning program checks to see which conjuncts of the premise failed. If the failed conjunct indexes dynamic state information or is used to control the meta-level reasoning, then there is no learning opportunity, as there is no corresponding underlying domain theory. However, if the failed conjunct is the type that accesses the domain knowledge base, then this could be a learning opportunity.

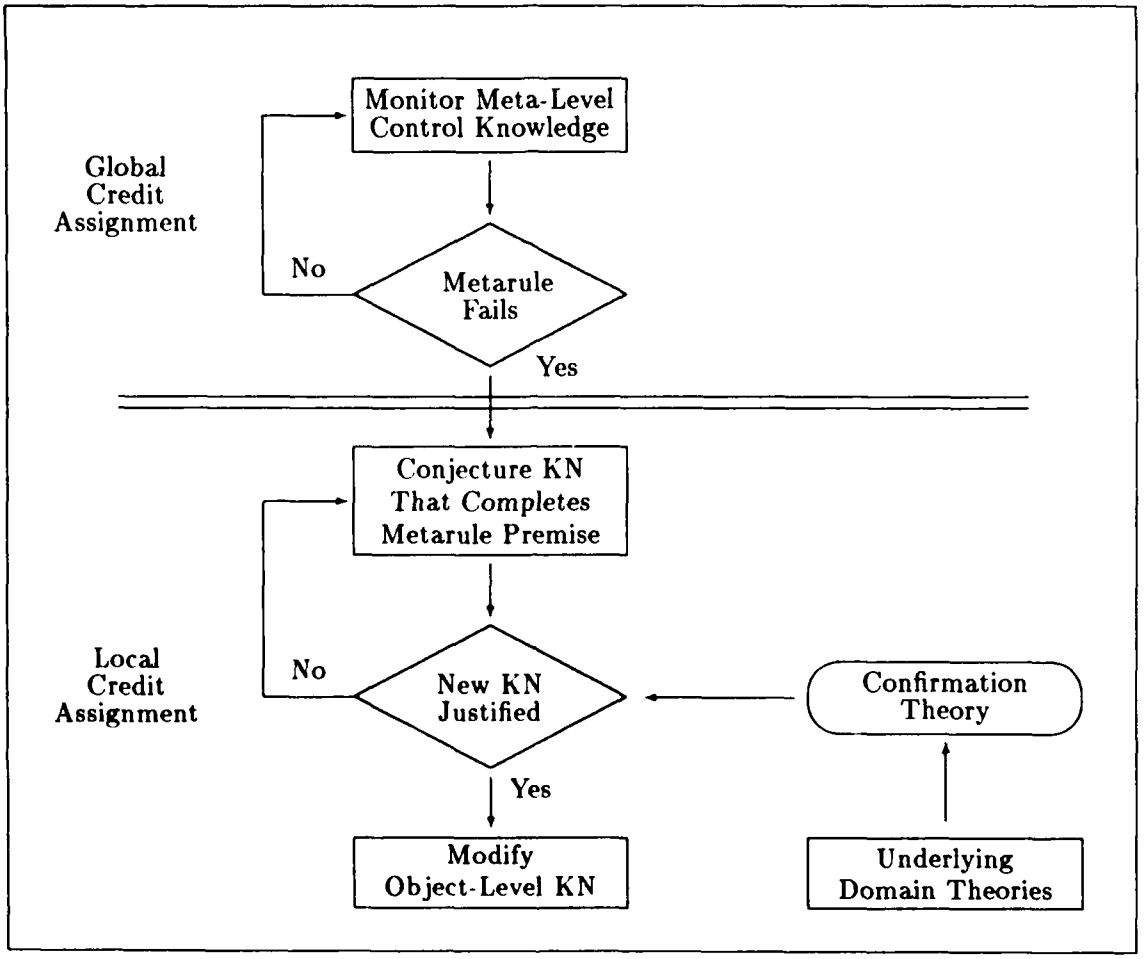


Figure 2: Overview of Odysseus' learning methodology when observing problem solving behavior of an expert system

After detecting the existence of a gap in the knowledge base, the next task is to pinpoint the gap; this is the local credit assignment problem. In our approach, there are two major parts to local credit assignment: generation of potential repairs and the testing of these repairs for validity.

The input to the ODYSSEUS candidate repair generator is the metarule that failed, the known bindings for variables in the clauses of the metarule premise that have been determined outside of the scope of the metarule, and a knowledge of

the range of values that each variable in a metarule clause is allowed to assume. For example, the value of the variable `$finding` can be any finding in the domain vocabulary. The candidate repair generator focuses on the knowledge relations in the metarule and generates all allowable variable bindings for these relations. These instantiated relations are then passed on to the ODYSSEUS candidate tester.

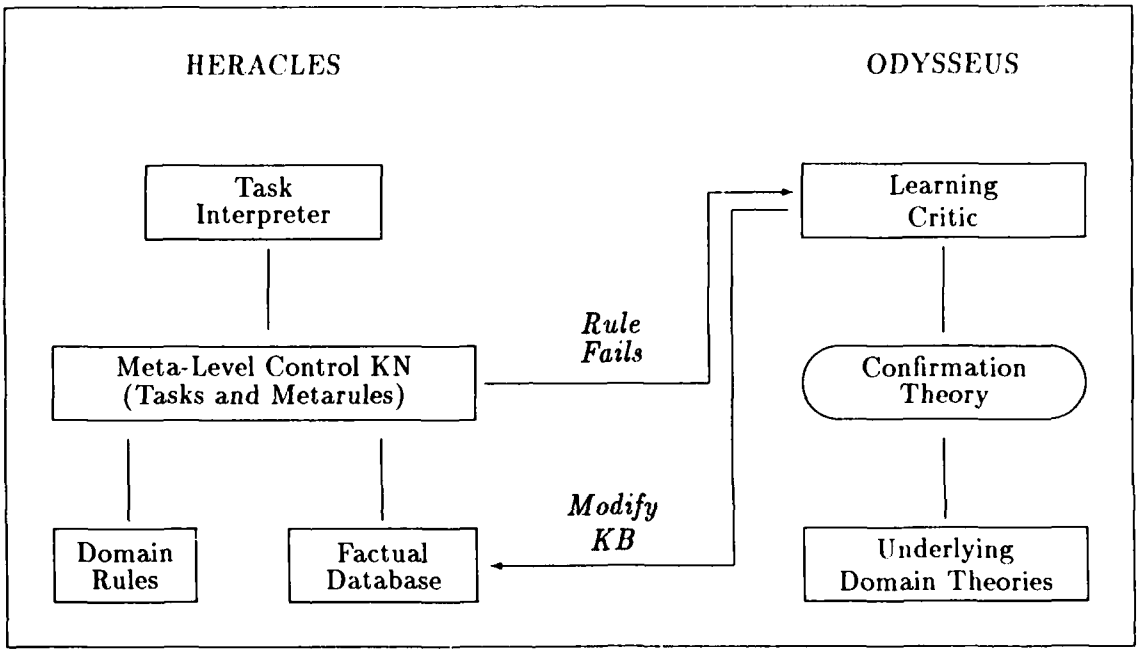


Figure 3: Odysseus monitors Heracles' metarule failures

The input to the ODYSSEUS candidate tester is a knowledge relation instance, such as `subsumes(visual-problems, double-vision)`. In order to test this candidate, two things are necessary. First, ODYSSEUS must have in hand a definition of all the constraints (empirical or otherwise) that determine whether an arbitrary instance of this knowledge relation is valid. Second, the learning program must have underlying theories of the domain that are capable of determining whether the constraints are satisfied, and hence whether the knowledge relation instance is valid. ODYSSEUS tests contains two underlying domain theories for testing of new knowledge: a strategy theory of heuristic classification problem solving and an induction theory based on analysis of past cases.

In the remainder of this section, two learning examples will be described in detail to demonstrate the approach we are advocating. The first example, given in Section 3.1, illustrates the learning of factual knowledge for the knowledge relation `clarifying.questions`, using the ODYSSEUS strategy theory as the underlying domain theory. The second example, given in Section 3.2, illustrates the learning of rule knowledge for the knowledge relation `evidence.for`, using an induction theory based on analysis of past cases as the underlying domain theory. These examples are based on the NEOMYCIN knowledge base, the MYCIN case library, and an actual medical case. Both sections assume that a metarule failure has occurred and that candidate repairs have been generated; they concentrate on the third stage of learning, wherein candidate repairs are tested.

3.1 Learning Factual Knowledge

The focus of this example is the `clarifying.questions` knowledge relation in the `clarify.questions` metarule presented below. As an example of its use, suppose the doctor discovers that the patient has a `headache`. The `headache` finding is associated with many diagnostic hypotheses, so many that it is generally wise to narrow down this set of hypotheses by determining the severity and duration of the headache before pursuing a specific hypothesis. This is the process of *clarifying the finding*, and the questions about various subtypes of this finding (e.g., headache-duration, headache-severity) are called *clarifying questions*. In the HERACLES system, this is implemented by invoking the `clarify.finding` task whenever a new finding is derived by the system or provided by the user. In turn, the `clarify.finding` task invokes the `clarify.questions` metarule.

MetaRule 1: Clarify.questions

```
IF:     goal(clarify.finding $finding1)  ∧
        clarifying.questions($finding1 $finding2)  ∧
        not(value-known $finding2)
```

```
THEN:    goal(findout $finding2)

ENGLISH: If the current goal is to clarify finding1
         and finding1 can be clarified by finding2
         and finding2 is currently unknown
         then try to find out the value of finding2.
```

Only one of the premise conjuncts of Rule 1 accesses domain knowledge, namely `clarifying.questions($finding1 $finding2)`. The first conjunct is for control purposes and the third conjunct checks the value of dynamic state knowledge.

The situation when learning may occur is when Rule 1 is passed a value for the variable `$finding1`, say 'headache', but Rule 1's premise fails because no bindings can be found for `$finding2`. In this situation, `$finding2` is a free variable at the time of failure. ODYSSEUS begins the learning process by invoking the candidate repair generator, which generates every possible candidate binding for `$finding2`. Using information regarding the domain of `$finding2`, the learning critic is able to generate about 300 candidate relations.

In order to be able to validate candidate new domain knowledge for a particular knowledge relation, two steps must be taken beforehand. First, a *justification* for the knowledge relation must be constructed that specifies all the constraints that an instance of the knowledge relation must satisfy in order to be valid. In our example, this requires constructing a precise definition that captures the constraints on an instance of the `clarifying.questions` relation. Second, a way must be found to test these constraints using underlying theories of the domain. This two-step method contrasts with the current manual method of refining the NEOMYCIN knowledge base, which consists of asking physicians what clarifying questions to use.

Let us begin by giving an informal justification of `clarifying.questions`. One reasonable justification for asking clarifying questions is cognitive economy with respect to efficient diagnosis. Much of diagnosis involves the testing of specific hypotheses; however, sometimes a new piece of information is discovered that

suggests a very large number of hypotheses. To reduce the number of relevant hypotheses, it is helpful to ask several clarifying questions that will add confirming or disconfirming evidence to many of the hypotheses associated with the new piece of information. After asking these questions, only a few of the numerous potential hypotheses will now be consistent with what is known.

We can now give a precise description of the constraints operating on `clarifying.questions`. This first-principles interpretation of a clarifying question is as follows: if a question is associated with many hypotheses, say more than six, and there exists a question that provides positive or negative evidence to many of these hypotheses, say between one-third and two-thirds, then always ask this question as a clarifying question. This can be formalized as follows.

Definition 1.

For any finding f, let H_f be the set of all hypotheses h such that relatesTo(f, h) is true. Let f_1 and f_2 be distinct findings, such that `subsumes`(f_1, f_2) is in the knowledge base. Let n be an empirically determined threshold indicating the minimum number of hypotheses that a finding must relate to in order to require the use of clarifying questions. Then

$$\texttt{clarifying.questions}(f_1, f_2) \longleftrightarrow [(\|H_{f_1}\| \geq n) \ \wedge \ (\frac{1}{3}n \leq \|H_{f_1} \cap H_{f_2}\| \leq \frac{2}{3}n)].$$

$\Diamond$

The relatesTo() relation is not part of the domain knowledge base; it is computed on the fly when a new piece of knowledge is validated, using a method which we will now describe. ODYSSEUS has two underlying domain theories that together can be used to check whether a new piece of knowledge satisfies all aspects of Definition 1. One underlying theory is a strategy theory for heuristic classification problem solving. A component of this theory is a line of reasoning explanation generator. Given a finding, all paths from that finding to reasonable possible diagnostic hypotheses via metarule applications can be determined. The generator can enumerate all the reasons that a question could possibly be asked, given the strategy and domain knowledge in HERACLES. The line of reasoning generator al-

lows determination of all the hypotheses that are associated with any one question either directly or indirectly; it is used to compute relatesTo(f, h).

We now describe the results of encoding Definition 1 and implementing our approach for the NEOMYCIN knowledge base. Currently, there are two clarifying questions for headache in the NEOMYCIN knowledge base: headache duration and headache severity. Our implemented metarule critic for the `clarify.questions` metarule considered the effect of all headache-related questions on the set of hypotheses associated with headache, and determined that one more clarifying question met the above described constraints: headache progression (i.e., is the headache getting better or worse). ODYSSEUS automatically modified a slot value under headache in the knowledge base to include this clarifying question; in the future, this question will always be asked when the patient complains of a headache.

3.2 Learning Rule Knowledge

All rule knowledge is represented within HERACLES using knowledge relations. This means that rules can be learned much as factual knowledge is learned. The example in this section involves learning an instance of the `evidence.for` relation in the `Split.Active.Hypotheses` metarule. This rule is one of three invoked by the task `Group.And.Differentiate.Hypotheses`. This metarule is useful during diagnosis when there are currently a large number of strong diagnostic hypotheses. The `Split.Active.Hypotheses` metarule searches for a finding to ask about that will simultaneously provide strong positive evidence for some active hypotheses and strong negative evidence against other active hypotheses.

MetaRule 2: Split.Active.Hypotheses

```
IF:     goal(group.and.differentiate.hyps $active.hypotheses) ∧
        member($hypothesis1 $active.hypotheses) ∧
        member($hypothesis2 $active.hypotheses) ∧
        not(equal($hypothesis1 $hypothesis2)) ∧
```

```
        evidence.for($finding $hypothesis1 $rule1 $cf1) ∧
        evidence.for($finding $hypothesis2 $rule2 $cf2) ∧
        greater($cf1 .2) ∧
        less($cf2 -.2)

THEN:   goal(findout $finding)

ENGLISH: If the current goal is to group and differentiate a
         list of active hypotheses and a single finding provides
         positive evidence for one of the hypotheses and
         negative evidence for another of the hypotheses
         then try to find out the value of this finding.
```

The metarule is passed a value for the variable `$active.hypotheses`. The interpreter attempts to find a unifier for all the clauses such that `$hypothesis1` is bound to one member in `$active.hypotheses`, `$hypothesis2` is bound to a different member of `$active.hypotheses`, and there is a single finding in the premise of a metarule that concludes that `$hypothesis1` is probably present and is also in the premise of a rule that concludes that `$hypothesis2` is probably absent. That is, a finding is asked that simultaneously provides evidence against some of the hypotheses and evidence for other hypotheses. Even though the NEOMYCIN knowledge base has been under development for several years, the `Split.Hypothesis.List` metarule is rarely invoked on any of the patient cases in the NEOMYCIN case library. Therefore implementing a learning critic for this metarule is useful.

In
the example in which our learning critic was called into play, `$active.hypotheses` consisted of seven hypotheses: `AV malformation`, `mycobacterium TB meningitis`, `viral meningitis`, `acute bacterial meningitis`, `brain aneurysm`, `partially treated bacterial meningitis` and `fungal meningitis`. The metarule fails because a binding for `$finding` cannot be found in the two relations `positive.evidence.for` and `negative.evidence.for`. Other clauses establish bindings for `$hypothesis1` and `$hypothesis2`. Using information regarding the domain of `$finding`, the learning critic conjectures many potential missing rules.

The number of conjectures can be quite large. For 300 findings and seven active hypotheses, this number is $7 \times 6 \times 300$.

Given these conjectures, a confirmation theory determines whether any of them is true. This requires the use of a formal definition for each relation. In this case we need a formal definition of `$evidence.for`.

Definition 2.

Let r be a justifiable domain rule. Let f be a finding that appears in the premise of r, and let h be a hypothesis that appears in the conclusion of r. Let s be the certainty factor strength of r, normalized to lie between ± 1. Then

$$\texttt{evidence.for}(f, h, r, s).$$

$\Diamond$

To actually determine whether a domain rule is justifiable requires the use of an underlying domain theory. ODYSSEUS uses induction over a case library to determine whether the conjectured rule is valid. That is, ODYSSEUS does a statistical analysis of the cases and determines whether the rule has good generality, specificity, and economy, and satisfies other measures of rule fitness[3].

The confirmation theory using the ODYSSEUS induction system found five rules that divide the list of active hypotheses, including:

Object-Level Rule 1.

IF: `duration.of.symptoms` $\leq$ `1 day` $\wedge$
`evidence.for(meningitis)` $\geq$ `.6`

[3]The library of test cases that we used to generate rules is the MYCIN case library (Buchanan and Shortliffe, 1984). Because diseases are defined in the Neomycin knowledge base that are not defined in the Mycin system (in this case, AV malformation, partially treated bacterial meningitis, and brain aneurysm), the values of the certainty factors (CFs) for some rules will be slightly inaccurate.

```
THEN:   suggests fungal.meningitis (cf = -.8) ∧
        suggests mycobacterium.tb.meningitis (cf = -.8) ∧
        suggests acute.bacterial.meningitis (cf = .7)
```

Upon being accepted, this rule is added to the object-level rule set; it is also re-represented as knowledge relations and these are added to the factual database

3.3 Comparing Apprentice Scenarios

Table 1 contrasts the two different ODYSSEUS apprenticeship learning scenarios of watching another problem solver and watching one's own problem solving. Table 1 compares the way the two scenarios accomplish the three major learning tasks faced by an apprenticeship learning system: the realization that knowledge is missing, the generation of candidate repairs, and the testing of those repairs. Note that the latter two tasks, i.e., the local credit assignment process that involves the use of underlying domain theories and the construction of definitional constraints, are identical in the two scenarios. On the other hand, the global credit assignment process is easier when watching oneself, because there is none of the uncertainty connected with inferring another agent's line of reasoning. Generating repairs is also easier when watching oneself, as there is no uncertainty as to exactly which metarule and hence which knowledge relation is responsible for the failure.

Compared to watching another problem solver, one can learn from watching one's own problem solving earlier in the knowledge acquisition "end-game". When watching another problem solver, a relatively large knowledge base is required; otherwise it is impossible to follow the line of reasoning of an expert most of the time, which is a requirement of this scenario.

A disadvantage of watching oneself is a large number of false alarms. Metarules fail most of the time, and it is not clear what the failure rate would be for a really good knowledge base. Perhaps it would only be a little lower than with a fairly incomplete knowledge base. More experimentation is required to answer these questions.

	Scenario 1: Watching Other Problem Solving	Scenario 2: Watching Own Problem Solving
Global Credit Assignment	Attempt to construct an explanation of observed action fails	Meta-level control rule fails
Local Credit Assignment: Generate Repairs	Generate domain KN element that completes an explanation	Generate domain KN element that allows rule to succeed
Local Credit Assignment: Test Repairs	Check constraints on KN relation using underlying domain theories	Check constraints on KN relation using underlying domain theories

Table 1: Comparing Apprenticeship Scenarios

4 Discussion

Monitoring abstract control knowledge appears to be a very promising lever for aiding apprenticeship learning. In showing two examples of the leverage obtained by this approach, we have only scratched the surface of the topic. This section discusses some of the remaining open issues.

As described in Sections 3.1 and 3.2, we have begun to implement constraint definitions to link knowledge relations to underlying theories. A key question that needs investigation is the reusability of these constraint definitions: are there sets of knowledge relations that can use the same or similar constraint definitions? As there are scores of different knowledge relations in the NEOMYCIN system, reuse of definitions could significantly reduce the amount of effort needed to create metarule critics for all metarules in the expert system shell. Further, it is not yet known whether all types of knowledge relations will be amenable to formal constraint definitions.

The best method of gauging the improvement produced by the addition of new knowledge is another open question. The heuristic knowledge that the examples of Section 3 added to the knowledge base is clearly helpful for the example cases, because it allows several hypotheses to be confirmed or disconfirmed with a single question. However, a complete validation should show improvement in performance on a validation set of cases. The measure of performance should be diagnostic accuracy and efficiency.

Another issue involves the control of the learning process. When should this type of learning be invoked? Not every metarule failure signals missing knowledge: how can learning opportunities be distinguished from routine failures?

Another open problem relates to the quantity of new knowledge introduced into the system. For example, in Section 3.2 five new rules were found that would divide the current hypothesis list. More generally, an open problem in the induction of rule bases is how to adequately *bias* the selection of rules (Fu and Buchanan, 1985; Michalski et al., 1983). There may be very many good candidate rules, but having too many rules is injurious to an expert system - efficiency is decreased, debugging is complicated, and explanations of actions become harder to follow. Of course, learning knowledge in the context of normal problem solving increases the likelihood that the rules produced by the induction system are going to be useful for problem solving. Only adding rules that are needed by the metarules of the inference procedure is a good step towards introducing a sufficient bias on rule selection.

5 Related Work

Two major apprenticeship learning systems are LEAP and DIPMETER ADVISOR (Mitchell et al., 1985; Smith et al., 1985). In both of these systems there is a single *type* of knowledge. In LEAP, all knowledge is implementation rules. In DIPMETER ADVISOR all knowledge is heuristic rules. In contrast, there are dozens of types of knowledge in HERACLES - each knowledge relation corresponds to a differ

ent type of knowledge. The key to automatic learning seems to be the definition of constraints to tie each knowledge relation individually to one or more underlying domain theories.

There has been a great deal of research on failure driven learning that monitors control and planning knowledge (Mitchell et al., 1983; Korf, 1985; Minton, 1985). The goal of these research efforts is to create better control knowledge so as to speed up problem solving, rather than to learn domain-specific factual knowledge. This compliments our approach, as we do not address the learning of abstract control knowledge for a problem-solving method; in other words, we do not learn tasks and metarules.

ODYSSEUS has a separate definitional constraint for each knowledge relation. This allows it to determine whether the candidate new knowledge relation instance is valid. This is reminiscent of the approach taken in AM (Lenat, 1976), where each slot of a concept has a a set of associated heuristic rules that can be used to validate the contents of the slot.

6 Summary

It is well known that expert systems derive much of their power from the quality and quantity of their domain specific knowledge. The method described in this paper provides a method of partially automating the acquisition of some of this knowledge.

The construction of expert system shells for generic tasks has become a common practice. There is a growing awareness that the power of a knowledge acquisition system for an expert system shell is bounded by the complexity and explicitness of the inference procedure (Eshelman and McDermott, 1986; Kahn et al., 1985). There is also a growing awareness that *automated* knowledge acquisition must be grounded in underlying domain theories (Mitchell et al., 1985; Smith et al., 1985). Using the HERACLES expert system shell and the ODYSSEUS apprenticeship learning program, we have demonstrated how underlying theories of a problem solving

domain can be effectively used by a learning method centered around an explicit representation (i.e., tasks and metarules) of the problem solving method.

The learning method described in this paper has three stages. The first stage is global credit assignment, the process of determining that there is a gap in the knowledge base. This is accomplished by monitoring metarule premise failures in the expert system shell, since all knowledge base gaps cause these. The second stage of learning is generating candidate repairs. Candidate repairs are generated by locating the knowledge relation in the failed metarule premise, and generating all values of the relation for the free variables in the relation. The last stage of learning is evaluation of candidate repairs. The ODYSSEUS method involves constructing a constraint definition for each different type of knowledge, to describe how an underlying domain theory can be used to validate the repair. In the described experiments, we used the NEOMYCIN knowledge base for the HERACLES expert system shell. The underlying domain theories are a strategy theory and an induction theory based on analysis of past cases.

A major open question is to determine how many of the knowledge relations in the expert system shell can be grounded in underlying theories of the domain. In particular, we are investigating the extent to which the different knowledge relations can be grounded in the two underlying theories that are part of ODYSSEUS. However, for certain types of domain knowledge used in the metarules, such as definitional and causal knowledge, we currently have no underlying theory; construction of such theories to allow *automated* knowledge acquisition will be difficult and perhaps impossible.

The type of learning demonstrated in this paper is more powerful than most forms of failure-driven learning, because the definition of failure is weaker. Failure to solve the overall problem is not necessary; rather, failure to satisfy a metarule premise for achieving a problem solving subgoal is sufficient for learning to take place.

7 Acknowledgments

We express our gratitude for helpful comments provided by Haym Hirsh and Marianne Winslett for several draft versions of this paper.

This work was supported in part by NSF grant MCS-83-12148, ONR/ARI contract N00014-79C-0302, Advanced Research Projects Agency (Contract DARPA N00039-83-C-0136), the National Institute of Health (Grant NIH RR-00785-11), National Aeronautics and Space Administration (Grant NAG-5-261), and Boeing (Grant W266875). We are grateful for the computer time provided by the Intelligent Systems Lab of Xerox PARC and SUMEX-AIM.

8 References

Buchanan, B. G. and Shortliffe, E. H. (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, Mass.: Addison-Wesley.

Clancey, W. J. (1984). NEOMYCIN: reconfiguring a rule-based system with application to teaching. In Clancey, W. J. and Shortliffe, E. H., editors, *Readings in Medical Artificial Intelligence*, chapter 15, pages 361–381, Reading, Mass.: Addison-Wesley.

Clancey, W. J. (1986a). From GUIDON to NEOMYCIN to HERACLES in twenty short lessons. *AI Magazine*, 7:40–60.

Clancey, W. J. (1986b). Representing control knowledge as abstract tasks and metarules. In Coombs, M. and Bole, L., editors, *Computer Expert Systems*, Springer Verlag. Also, Knowledge Systems Lab Report KSL-85-16, Stanford University, April 1985.

Eshelman, L. and McDermott, J. (1986). MOLE: a knowledge acquisition tool that uses its head. In *Proceedings of the 1986 National Conference on Artificial Intelligence.*

Fu, L. and Buchanan, B. G. (1985). *Inductive knowledge acquisition for rule based expert systems.* Technical Report KSL 85-42, Stanford University, Computer

Science Dept.

Kahn, G., Nowlan, S., and McDermott, J. (1985). MORE: an intelligent knowledge acquisition tool. In *Proceedings of the 1985 IJCAI*, pages 573–580.

Korf, R. (1985). *Learning to solve problems by searching for macro-operators.* Marshfield, Mass: Pitman.

Lenat, D. B. (1976). *AM: An artificial intelligence approach to discovery in mathematics as heuristic search.* PhD thesis, Stanford University.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors (1983). *Machine Learning: An Artificial Intelligence Approach.* Tioga Press.

Minton, S. (1985). Selectively generalizing plans for problem solving. In *Proceedings of the 1985 IJCAI*, pages 596–599.

Mitchell, T., Utgoff, P. E., and Banerji, R. S. (1983). Learning by experimentation: acquiring and refining problem-solving heuristics. In Michalski, T. M., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach.* pages 163–190, Palo Alto: Tioga Press.

Mitchell, T. M., Mahadevan, S., and Steinberg, L. I. (1985). LEAP: a learning apprentice for VLSI design. In *Proceedings of the 1985 IJCAI*, pages 573–580.

Russell, S. (1985). *The Compleat Guide to MRS.* Technical Report KSL-85-108, Stanford University.

Smith, R. G., Winston, H. A., Mitchell, T. M., and Buchanan, B. G. (1985). Representation and use of explicit justifications for knowledge base refinement. In *Proceedings of the 1985 IJCAI*, pages 673–680.

Wilkins, D. C., Clancey, W. J., and Buchanan, B. G. (1986). An overview of the ODYSSEUS learning apprentice. In Mitchell, T. M., Michalski, R. S., and Carbonell, J. G., editors, *Machine Learning: A Guide to Current Research*, pages 332–340, New York: Kluwer Academic Press.

END

DATE

FILMED

FEB.

1988