Representation and Control of Knowledge Bases for Support of Multiple Tasks

Young-Tack Park and David C. Wilkins
Department of Computer Science
405 North Mathews
University of Illinois
Urbana, Illinois 61801

Abstract
This paper discusses the MINERVA approach to a knowledge-based system, KBS, development. An important design goal for building the MINERVA architecture is to represent meta-level knowledge abstractly in a manner that facilitates learning as well as problem-solving. Since there is a synergistic relationship between a KBS and a learning program, the structure of MINERVA supports the learning program, thereby reduces the difficulties in knowledge-base refinement process. In the ODYSSEUS learning program, we explored the advantages and limitations of MINERVA's structure as a knowledge representation that supports a learning program. To resolve the difficulties based on the experience of the ODYSSEUS apprenticeship learning program, we defined and implemented functional capabilities of MINERVA that allow the multiple use of strategy knowledge, namely, (i) modular, layered, and explicit representation of knowledge according to the functional roles, (ii) more declarative representation of strategy knowledge that facilitates the diverse tasks, (iii) opportunistic control of strategy knowledge, and (iv) explicit representation of reasoning process.

1 Introduction

Human experts use their expertise to do more than just solve a problem. In addition, they have the capabilities to explain the reasons for their problem-solving steps and explain the observed problem-solving behavior of another expert. Human experts use the same knowledge to solve a problem, explain their problem-solving, teach a student, and learn by watching another expert. Since expert systems are intended to be models of human experts, they should be able to exhibit such diverse capabilities in their domain of expertise.

Expert systems research has been developed to produce a paradigm where different types of knowledge are cleanly separated and are represented more declaratively [Clancey and Bock, 1985], with the intention of facilitating various tasks based on the same knowledge base. It is obvious that declarative and explicit representation is useful for more than a single possible use of knowledge [Winograd, 1976].

A common agreement in AI is that knowledge representation is closely related to the application of the knowledge. An appropriate representation can greatly affect how easy it is to do a task with it. Both learning and explanation do not happen in a vacuum. The well-suited structure of expert systems greatly affects the design and efficiency of both the learning [Minton et al., 1989] and the explanation [Neches et al., 1985] systems. And, how knowledge is represented has a close relationship to how the knowledge will be used [Bylander and Chandrasekaran, 1987].

One major problem in building expert systems is the knowledge base refinement problem [Ginsberg et al., 1988]. However, few expert systems are designed to facilitate the automatic knowledge base refinement. Because of this lack of support from expert systems, the traditional machine learning approach to refine knowledge base, such as ID3 [Quinlan, 1986], INDUCE [Michalski, 1983], and SEEK2/Ginsberg et al., 1988], cannot facilitate the synergistic relationship between the expert system and the learning systems. Therefore, when designing the expert systems, the issues involved in knowledge base refinement should be addressed.

A source of human expertise is the strategy knowledge about what to do next. Experts apply the strategy knowledge to solve a problem and generate comprehensive explanations of their problem-solving behavior. And, when they watch the problem-solving behavior of another expert, they explain observed actions using their strategy knowledge. Hence, the representation of the strategy knowledge in a generic task is important to expert systems research.

Our goal is to develop a use-independent knowledge structure at the domain and strategy level, and to enable a knowledge-based system to exhibit diverse dimensions of expertise, such as problem-solving, explanation, and learning, using the knowledge base. With the objective of constructing the use-independent knowledge base, the domain and meta-level knowledge [Davis and Buchanan, 1977] is represented explicitly and declaratively. And the control of the meta-level strategy knowledge becomes more flexible and oppor-
The outline of this paper is as follows: We begin in the next section with the ODYSSEUS apprenticeship learning [Wilkins, 1988] that motivates this research. We then go on to describe the MINERVA expert system shell that supports the diverse tasks. And, we demonstrate how this mechanism shows the needed capabilities.

2 The ODYSSEUS Apprenticeship Learning

The ODYSSEUS apprenticeship learning uses a learning by watching approach to acquire new knowledge from expert's behavior [Wilkins, 1988]. The ODYSSEUS learning program attempts to use the strategy knowledge that the HERACLES expert system shell [Clancey and Bock, 1985] uses in problem-solving to explain expert's problem-solving steps. The major concern here is how ODYSSEUS generates explanations of external agent's problem-solving steps and what were the challenges with using the strategy knowledge of the HERACLES expert system shell. [Wilkins, 1988] describes details of global and local credit assignment process of the ODYSSEUS apprenticeship learning to refine a knowledge base.

Each time an human problem solver performs a problem-solving action, the action is input to the ODYSSEUS learning program. The ODYSSEUS learning program produces explanations of the human's action. An explanation is a sequence of strategy metarules that connects the observed action to a high-level problem-solving goal. While the HERACLES problem-solver runs metarules forwards to achieve a goal, from a goal to an action, ODYSSEUS runs meta-rules backwards, from an observed action of an expert to plausible higher goals of the action.

2.1 ODYSSEUS' Explanation Generation Method

An explanation in ODYSSEUS is a proof that demonstrates how an expert's problem-solving step is a logical consequence of the current problem state, the domain and strategy knowledge, and one of the current high-level strategy goals. An explanation is created by backchaining the metarules until a metarule is reached whose head represents a high-level problem-solving goal.

Consider the declarative metarules in Fig 1, written in PROLOG form. A metarule has a goal, conditions, and a subgoal. To backchain a metarule requires unification of the conditions with domain and problem state knowledge. Suppose ODYSSEUS observes the expert asking if the patient has diplopia, then G2 of ask subgoal in findout metarule (see Fig 1) is bound to the observed action, diplopia. If the body, not derivable(diplopia) and askable(diplopia), of the metarule is satisfied, then ODYSSEUS assumes that the expert asks diplopia because he or she cannot infer the value. Hence, the intermediate explanation of ask(diplopia) becomes findout(diplopia).

Using this intermediate candidate, ODYSSEUS continues to backchain the clarify FINDING metarule since its subgoal unifies with the candidate, findout(diplopia). The backward interpreter tries to find the binding value of G1 of clarify FINDING metarule in Fig 1 that satisfies the given bindings and the conditions of the metarule. Suppose, only more specific(diplopia, focalsigns) exists in the knowledge base and focalsigns satisfies the clauses in the premise, such as always get specifics and redflag, then ODYSSEUS assumes a plausible explanation of intermediate action, findout(diplopia), is clarify FINDING(focalsigns). The clarify FINDING metarule is one of high-level problem-solving goals and the metarule backchain process stops. And the line of reasoning for the expert's action, asking diplopia, becomes a metarule chain, clarify FINDING(focalsigns) → findout(diplopia) → ask(diplopia), meaning that the expert intends to clarify new datum, focalsigns, as a result he or she tries to find out diplopia which is a specific fact of focalsigns, and asks the value of diplopia because it cannot be derivable from the given problem-solving state and the domain knowledge.

2.2 Lessons Learned

The task and metarule representation of HERACLES is not well-designed to handle procedures required by the metarule backchaining method. Since the knowledge representation of HERACLES imposes limitations to the ODYSSEUS learning system, ODYSSEUS had to rely on hand-coded second version of each metarule. HERACLES uses procedural attachments and flags in the premise clauses of a metarule and in control blocks for the metarules. Both features make the learning program difficult to explain observed actions of experts. The procedural attachment attaches a procedure to a statement to specify the handling of that statement. This procedural representation does not allow the metarule backchain method to find the binding values that satisfy the premise clauses with an observed action. Flags imposes disadvantages on the backward interpreter. Flags are side effects and don't have clear semantics. The backward interpreter cannot reason about the flags to drive the clause back-
wards.

To solve this irreversible problem, ODYSSEUS has to use special version of metarules. A clause with procedural attachment is rewritten in a declarative form that is almost equivalent to the clause. However, flags make the reconstruction process difficult. Some clauses should be ignored because of vague semantics of flags. Hence, the reconstructed premises are more general than the original ones. Furthermore, due to the fact that ODYSSEUS has to use more general premises to explain observed behaviors, it suffers from multiple explanations. Intuitively, the mapping from behavior to explanation is not always unique. There may be several plausible explanations whose actions are indistinguishable. This problem is aggravated by the use of more general premises.

3 The MINERVA Shell for Multiple Use of Knowledge

MINERVA is a rule-based expert system shell that has evolved from the HERACLES shell, based on the experience gained in creating the ODYSSEUS [Wilkins, 1988] apprenticeship learning program for HERACLES [Clancey and Bock, 1985]. HERACLES is itself a refinement of EMYCIN, based on the experience gained in creating the GUIDON case-based tutoring program [Clancey, 1987] for EMYCIN. These shells use a problem-solving method called heuristic classification, which is the process of selecting a solution out of a pre-enumerated solution set, using heuristic techniques [Clancey, 1985].

In designing the MINERVA shell, there are several major goals in which the system seeks to achieve. First, the strategy knowledge should be use-independent. Both the performance system, MINERVA, and the learning program, ODYSSEUS, should be able to use the same knowledge base for problem-solving and learning, respectively. Second, the control of strategy knowledge should be explicit and opportunistic. To achieve this goal, meta-level knowledge is further modularized into the scheduler knowledge and the strategy knowledge according to their roles.

3.1 Architecture of MINERVA

The system architecture for MINERVA is illustrated in Fig 2. Both forward and backward interpreters access the same meta-level strategy and scheduler knowledge for problem-solving and apprenticeship learning, respectively. MINERVA has 19 types of strategy knowledge and 54 meta-rules that embody the strategy knowledge. The reversible representation of strategy knowledge allows both the problem-solver and the learning program to use the same knowledge base.

A maintenance system supports both the problem-solving and the apprenticeship learning. The problem-solver needs to explain its problem-solving steps. This requires recording the inferences made by the forward interpreter. Advanced explanations, such as why, how, and why-not, can be possible by tracking down the explicit justification structure created by the maintenance system. The backward interpretation tries to find a set of binding values that is consistent with the conditions of metarule chains. This process involves many backtrakings and can be efficient by reducing redundant backtracking using the maintenance system.

![Figure 2: A MINERVA-based expert system](chart)

Domain knowledge is declaratively represented. Domain rules are Horn clauses that associate a set of conditions to a hypothesis. An example of domain rule in Horn clause format is `conclude(meningitis, &): :- finding(seizures, yes), finding(tense.fontanel,yes),` meaning "conclude with a degree of certainty of .8 that the patient has meningitis if the patient has seizures and the patient's anterior fontanel is abnormally tense or bulging." The domain structural knowledge is a set of predicates that describe the frame knowledge, such as taxonomy of hypotheses, properties of findings and rules, and subsumption relations between findings. An example of frame knowledge is `parent(meningitis, meningitis),` meaning "hypothesis meningitis is a parent of the hypothesis meningitis." `Red.flag(seizures)` and `trigger_rule(rule2000)` represent the finding, seizure, and the domain rule, rule 2000, are the serious ones to deserve to clarify first. The knowledge base for diagnosis of meningitis cases contains approximately 400 rules and 2000 facts.

3.2 Explicit Representation of Strategy Knowledge

The achievement of multiple use of strategy knowledge appears to be highly dependent on making the knowledge in the expert system as explicit, declar
tive and modular as possible. The strategy knowledge of MINERVA improves that of the HERACLES shell in several ways. First, metarules are restricted to be Horn clauses without cuts. No clause is attached to complex procedural attachment that puts the limitation of multiple use of knowledge. Second, premises of metarules are declarative. They do not change the state of the system and do not call other tasks. This side-effect free representation facilitates explanation and learning. Third, more of the problem-solving knowledge are explicitly represented. MINERVA attempts to represent parts of domain rule interpreter in metarules as much as possible. Fourth, MINERVA attempts to represent parts of domain rule the control of strategy knowledge is explicitly represented. Implicit representation of control proves to be inadequate for opportunistic problem-solving and robust apprenticeship learning.

The representation of metarules is the most significant difference between HERACLES and MINERVA. While the metarules in HERACLES are frame structures encoded in MRS [Clancey and Bock, 1985], those in MINERVA are Horn clauses. The following HERACLES metarule and its MINERVA translation in Fig 3 serve to illustrate representational difference between the shells. The example is a metarule from the task process datum. This rule causes the system to find out more detailed findings for a finding that is known to the system. For example, if headache is being processed, the user will be prompted for headache.duration and headache.severity.

In HERACLES, metarule premises consist of a conjunction of propositions. In Fig 3 (a), metarule premises are composed of two propositions. In addition, a relation may be a composite inferred from rules, which is called a metarule premise relation, in this example clarify-question is such the relation. Since the domain knowledge of NEOMYCIN has not been translated to predicate calculus, procedural attachment is used. The questions ($QPARAM) in the metarule premise relation in Fig 3 (a) are stored as the PROCESSQ property of the finding. Therefore, it needs a procedure to search the property list. Similarly, TRACEDP is a function that examines a property list to see if a question has been asked before. This procedural attachment makes it difficult to run metarules backwards to explain an observed action of another expert and to explain its own problem-solving steps to user.

While HERACLES uses procedural attachments, MINERVA represents strategy knowledge declaratively based on the declarative domain knowledge. In Fig 3 (b), the clarifying relation between findings is represented by the domain predicate, such as clarified_by(headache.duration, headache). Unknown predicate tests the problem-state knowledge that is generated during the execution of MINERVA to see if a finding is known to the system or not. Using these predicates, MINERVA can collect all the findings that satisfy the conditions without using complex procedure. Findings that satisfy the premises of the metarule invoke findout subgoal to infer the value or ask for a user.

3.3 Flexible Control of Strategy Knowledge

In HERACLES, strategy knowledge is represented as task procedures and metarules. A task that consists of metarules is a procedure for accomplishing some well-defined problem-solving subgoal. Examples of tasks are test a hypothesis, refine a hypothesis, and clarify a finding. However, HERACLES uses a hard-wired control to invoke tasks. A task interpreter invokes tasks in a pre-determined way. For instance, MAKE-DIAGNOSIS task in Fig 4 implicitly orders the sequence of execution of its subtasks without considering problem state. Hence, only after the task explores the first subtask, IDENTIFY-PROBLEM, completely, it invokes the second subtask, REVIEW-DIFFERENTIAL. The interpreter cannot invoke a promising task with respect to the current problem-solving state opportunistically but it has to wait until the task is called by a metarule controlled by the implicit and hard-wired structure. Because of this hard-wired control, the rationale behind the control decisions of tasks is implicitly embedded in the task control blocks. This implicit representation puts the limitation to opportunistic problem-solving, comprehensible explanations, and flexible apprenticeship learning.

Each action within a task procedure for achieving the task procedure is called a metarule. The metarules are repeatedly applied by the interpreter. However, the sequence of execution of metarules is determined by the hard-wired order of metarules. Interpreter
invokes metarules without considering the problem-solving state but always activates them in the same way as it is ordered implicitly.

(PUTPROP RULE384
PREMISE:  T
ACTION:  (DO-ALL
(TASK IDENTIFY-PROBLEM)
(TASK REVIEW-DIFFERENTIAL)
(TASK COLLECT-INFO))
TASK:  MAKE-DIAGNOSIS)

Figure 4: Implicit control of tasks in HERACLES

One of major goals in MINERVA is to represent scheduler knowledge of strategy meta-rules explicitly. In MINERVA, the meta interpreter uses a blackboard agenda mechanism [Hayes-Roth, 1985] to decide which action to execute next. MINERVA deliberates the problem-solving strategy metarules based on the recently changes on the blackboard. Unlike the hard-wired metarules in HERACLES, the strategy metarules in MINERVA are separate, modular, and are only triggered by the changes on the blackboard opportunistically. When a triggered strategy metarule satisfies its conditions, it suggests a feasible action to do. Using the explicit scheduler knowledge, the meta interpreter selects an opportune action at the point among the set of feasible actions. This explicit schedule also allows MINERVA to explain schedule decisions as well as the strategy and domain level reasonings. In MINERVA, the scheduler knowledge does not determine which metarules to evaluate but it selects which action to perform next based on the current problem-solving state.

4 Multiple Use of Knowledge

In this section, we will describe how the modular, declarative, and more explicit meta-level knowledge supports multiple inferences.

4.1 Problem-Solving

MINERVA encodes most of the problem-solving strategy using the explicit and declarative metarules. Strategy knowledge represents the heuristic classification problem-solving method. At each point in the problem-solving process, several metarules are applicable and suggest multiple actions. The scheduler knowledge encodes the expertise to determine which action to perform next based on the current problem-solving state.

diagnose(Case)

Initialization

\[
\text{A} \leftarrow \emptyset \text{ % blackboard agenda}
\]
\[
\text{C} = \{c_i \mid c_i \in \text{chief_complaints}\}
\]
repeat

Collect metarules that are triggered by the changes on the blackboard

\[
\text{T} = \{t_j \mid t_j \in \text{metarules}, \text{trigger_part}(t_j) \text{ unifies with } c_i \in \text{C}\}
\]

Deliberate the metarules

\[
\text{M} = \{m_i \mid m_i \in \text{T}, \text{satisfy(premise_part}(m_i))\}
\]

Update the blackboard agenda

\[
\text{A} = \text{A} \cup \{a_i \mid a_i = \text{action_part}(m_i)\}
\]

Evaluate scheduler metarules

\[
\text{D} = \{(d_i, p_i) \mid d_i \text{ a desirable action, } p_i \text{ a preference value, } p_i \geq p_j \text{ if } i \geq j\}
\]

Select most opportune action

\[
\text{L} = \{l_m \mid l_m \in \text{A}, l_m \text{ unifies with } l_m\}
\]

\[
\text{Action} = l_m, m \text{ is the largest index in } L
\]

Evaluate the selected action

\[
\text{exec(Action)}
\]

Monitor the blackboard change

\[
C \leftarrow \text{monitor}\text{(exec(Action))}
\]

until \(C = \emptyset \land A = \emptyset\)

print results, history, and maintenance records.

Figure 5: A forward interpretation to solve a problem

MINERVA solves a problem in three steps: deliberation, schedule, and execution. Given any
problem-solving state, the meta-interpreter deliberates problem-solving metarules. Each metarule is responsible for knowing the trigger condition under which it can contribute to a solution. Each metarule has premises that indicate the conditions that must exist on the problem-solving state and the domain knowledge before the action part is activated. When the premise is satisfied, the action part of the metarule is inserted into the agenda. The agenda contains actions of applicable metarules. The next step is to decide which action in the agenda to take next. Various kinds of expertise is necessary to resolve the control problem. The scheduler knowledge uses the problem-solving state to determine the focus of attention which indicates the next action to be processed. Since the scheduler knowledge is represented explicitly, this endows MINERVA with the capability to reason about which action to perform opportunistically (see section 3.3). The next step is to execute the selected action. An example of the action is to fire a domain rule. The interpreter finds the conditions of the domain rule, tests the conditions, and changes the belief value of the conclusion of the domain rule. Each task at this level is also represented explicitly using the metarules. MINERVA attempts to represent much of domain interpreters in the explicit and declarative way. The problem-solving state is then changed by the result of the execution. In this example, a change may be generation of a new hypothesis or an active hypothesis. Similarly, this change triggers problem-solving strategy metarules and the cycle loops again. Figure 5 describes the deliberation, schedule, and execution steps in MINERVA.

4.2 Explanation Generation of Observed Actions

The declarative representation of metarules allows the backward interpreter to use the same metarules that are used by the problem-solver. There are many advantages to this reversible metarules. First, it is not necessary to write specialized code for each metarule that runs the metarules backward. Second, ODYSSEUS can generate only explanations that are equivalent to MINERVA. When ODYSSEUS uses the HERACLES shell, it generates more explanations due to the general version of metarules.

The backward interpretation starts with metarules that contain ask(Finding) action, since this is the place where MINERVA performs observable actions. Finding unifies with an observed action. ODYSSEUS then tries to find bindings for variables in the premise of the metarule that are consistent with earlier bindings. Using the binding value of Finding, it finds the binding values of variables in the last clause of the metarule, Cn. It continues to find binding values of Cn-1 that satisfy the earlier bindings. Based on a set of binding values that satisfy the premise, ODYSSEUS can instantiate the head of the metarule, goal(Focus). Next, it retrieves a metarule whose action part unifies with goal(Focus). And, it applies the same method to get the binding values and instantiate the head of the metarule. This process continues until it reaches one of higher problem-solving strategy goals. In this way,

\[
\text{explain(ObservedAction)}
\]

\[
\text{LORE} \leftarrow \emptyset, \% \text{Line Of Reasoning Explanations} \\
\text{Explanation} \leftarrow \{\text{ObservedAction}\}
\]

repeat

\[
\text{pick } c \text{ in Explanation} \\
\text{Explanation} \leftarrow \text{Explanation} - e \\
\text{Action} \leftarrow \text{unexplained_goal_of}(c) \\
\text{Goals} \leftarrow \text{backchain}(\text{Action})
\]

if Goals  \neq \emptyset then

for each element \( g \in \text{Goals} \)

\[
\text{explanation} \leftarrow \text{expand-metarule-backchain-one-level}(e, g)
\]

if \( g \in \text{High-level Goal} \) then

\[
\text{LORE} \leftarrow \text{LORE} \cup \text{explanation}
\]

else \text{Explanation} \leftarrow \text{Explanation} \cup \text{explanation}

until Explanation = \emptyset

\text{print LORE}

\text{backchain(Action)}

\[
\text{Goals} \leftarrow \emptyset, \\
\text{mr} = \{m|\text{m} \in \text{metarules, unify(action_part(mr), Action, } \theta)\}
\]

for each mr do

\[
\text{Condition} \leftarrow \text{condition_part(mr)} \cdot \theta; \\
\% \theta \text{ is a substitution}
\]

if \( \exists \theta' \text{ that satisfies Condition} \) then

\[
\text{Goals} \leftarrow \text{Goals} \cup \text{head_part(mr)} \cdot (\theta \cup \theta')
\]

return(Goals)

Figure 6: Explanation generation by metarule backchaining

for each observed action of the expert, ODYSSEUS can generate a set of line of reasoning explanations. Each explanation is the sequence of metarules, such as problem-solving state change \( \rightarrow \text{goal} \cdot (\text{focus}) \rightarrow \ldots \text{goal}_n \cdot (\text{focus}_n) \rightarrow \text{observed action} \). Figure 6 describes the application of the use-independent knowledge to the explanation generation in ODYSSEUS.

4.3 Explanations

Explanations provided by MINERVA take advantage of its functional representation of knowledge. This explicit, modular, and declarative knowledge representation at the domain and meta level enables MINERVA to generate more detailed explanation.

MINERVA uses the assumption-based maintenance system [de Kleer, 1986] to explicitly save the domain and meta level inferences. The maintenance system, also, records the success and failure of metarules and the history of domain-level interpretation. This enables MINERVA to provide rich explanations to user and knowledge acquisition program, thereby makes it easy to debug the knowledge base, both the domain and meta level, when MINERVA behaves incorrectly. MINERVA can generate multi-level explanations to explain why it takes a particular action. At the high-

56
est level of explanation, it describes why MINERVA prefers a particular strategy metarule to other activated metarules at the point. Suppose there are two activated metarules, clarify_finding(headache) and differentiate_hypotheses(meningitis, migraine), and the scheduler knowledge prefers the latter because enough active hypotheses are already generated and two hypotheses have high belief values. This explicit scheduler knowledge selects an action opportunistically based on the problem-solving state. In addition, the opportunistic decision of the scheduler is explicitly recorded. Hence, the explanation program can provide detailed explanations such as why MINERVA takes a particular metarule at the problem-solving state and does not take other activated metarules. At the middle level of explanation, MINERVA shows a sequence of meta-rules that achieve the goal of the selected strategy. In this example, to differentiate two hypotheses, MINERVA finds a finding that can distinguish a hypothesis from another and applies the domain rules related to the finding to change the belief values of the hypotheses. This is done by applying metarules in sequence and this sequence of metarules becomes the detailed explanation about why a particular action is taken. At the bottom level, MINERVA explains what metarule invokes the observable action. This multi-level explanation is quite useful for user to understand problem solving rationale and for the knowledge engineer or a failure-driven learning program to reason about the failure of MINERVA to debug the knowledge base.

5 Conclusion

Expert systems that are intended to be models of human experts should be able to exhibit diverse capabilities, such as problem-solving, learning, and explanations, in their domain of expertise. In this paper, we have shown the functional capabilities of such advanced generic expert systems with respect to representation and control of problem-solving strategy knowledge. In this approach, the domain and meta level knowledge is represented in a declarative, explicit, and modular way. This improved representation at the strategy and domain level enables the performance system, MINERVA, and the learning program, ODYSSEUS, to use the same knowledge base. Explicit representation of scheduler knowledge enables MINERVA to solve a problem opportunistically and to generate multi-level explanations of its own problem-solving.

Acknowledgments

We would like to express our deep gratitude to Lawrence Chachere, Ziad Najem, and Kok-Wah Tan for their major role in the design and implementation of the MINERVA shell.

References


