Towards Explicit Integration of Knowledge in Expert Systems: An Analysis of MYCIN's Therapy Selection Algorithm

Jack Mostow[*]
Computer Science Department
Rutgers University
Hill Center - Busch Campus
New Brunswick, New Jersey 08903

Bill Swartout
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292

Abstract

The *knowledge integration* problem arises in rule-based expert systems when two or more recommendations made by right-hand sides of rules must be combined. Current expert systems address this problem either by engineering the rule set to avoid it, or by using a single integration technique built into the interpreter, e.g., certainty factor combination. We argue that multiple techniques are needed and that their use -- and underlying assumptions -- should be made explicit. We identify some of the techniques used in MYCIN's therapy selection algorithm to integrate the diverse goals it attempts to satisfy, and suggest how knowledge of such techniques could be used to support construction, explanation, and maintenance of expert systems.

1. Introduction

As expert systems develop and proliferate, researchers have increasingly noticed the serious problems caused by confounding various different kinds of knowledge in an expert system's knowledge base. In particular, [Clancey 83a] focusses on control knowledge, showing how problem-solving strategies are (at best) clumsily encoded in rules, making the rule base difficult to understand and extend. In a sense, the control problem has to do with the relations among the left hand sides of different "if <condition> then <action>" rules -- that is, with deciding which of several apparently relevant rules to fire in a given situation. In contrast, the *integration problem* has to do with the relations among the ***right*** hand sides of rules -- that is, with combining multiple recommendations made by different rules in the same situation. This problem pervades expert systems but tends to get swept under the rug. In this paper we shall try to bring it out in the open and shed some light on it.

Most expert system designers use two basic approaches to integrate the recommendations made by the right hand sides of rules: either they try to finesse the problem by manually compiling it out of the rule set, or they rely on a single uniform integration mechanism. As we shall see, both approaches leave implicit the knowledge and assumptions underlying the integration process. This opens the door to various abuses: leaving knowledge and assumptions implicit makes them easier to violate without noticing.

[*]Much of the work described here was performed at the University of Southern California's Information Sciences Institute, where it was supported in part under DARPA Grant #MDA 903-81-C-0335.

The **compiling out** approach engineers the rule set so as to avoid the need for integrating recommendations at runtime. For example, consider the problem of assessing the likelihood that a patient has some disease D, for which there are two symptoms, A and B. Suppose that either symptom by itself suggests the disease with moderate likelihood (40% or 50%), but the two together are very strong evidence (95%). To compile out the problem of integrating the evidence, we might define three rules:

```
If A and not B then conclude(D, 0.4).
If B and not A then conclude(D, 0.5).
If A and B then conclude(D, 0.95).
```

While this approach could get the estimates right, it has the unfortunate side effect of introducing dependencies among the rules and of making them less understandable. For example, if just symptom A appears and the system is asked to explain its level of belief in disease D, it might give an explanation like the following:

```
"Since A is present and B is not present, there
is moderate evidence that the disease is D."
```

This explanation misleadingly suggests that the absence of B is actually evidence *in favor* of D. The fact that this rule was carefully constructed along with the other two so as to compile out the integration problem is not available to the system for explanation purposes, since it was never more than an intention in the mind of the rule base author. Engineering implicit knowledge into the rule base makes it difficult to understand and extend. For example, suppose that a third symptom, C, provides additional evidence for disease D; adding this knowledge would require splitting each rule listed above into two rules, one for when C is present and and one for when it isn't.

The usual alternative to compiling out the integration problem into the rule base is to build a **uniform integration mechanism** into the rule interpreter for combining recommendations at runtime. For example, MYCIN dynamically computes a "certainty factor" for the recommendation made by the right hand side of a rule, based on the rule's inferential strength, the certainty factors associated with the conditions satisfying its left hand side, and the connectives (like AND and OR) relating those conditions. When two or more rules produce the same recommendation with different certainty factors, MYCIN integrates them by means of a numerical formula. The appropriateness of any such formula depends on certain assumptions. For example, suppose both of the following rules are satisfied:

```
If A then conclude(D, 0.4).
If B then conclude(D, 0.5).
```

If A and B represent independent evidence for D, then the result of combining these two measures using the certainty factor mechanism may be correct, but if B is a special case of A, then that result will be wrong and it may be necessary to revert to the ad hoc "compiling out" approach described earlier.

As this example illustrates, a uniform integration mechanism is based on certain implicit assumptions; in cases where these assumptions are violated, it becomes necessary to program around the mechanism, producing artifacts in the rule base that degrade the quality of the explanations generated by the system.

In summary, most expert systems represent their integration mechanisms procedurally, whether in the rule interpreter or compiled into the rules. In either case, the reasoning that goes into integration is not perspicuously represented and hence is unavailable to the system for explanation purposes. Rather than hide knowledge integration in the design of the rule base or its interpreter, we argue that it should be made explicit so that it can be reasoned about and explained to the user.

One strategy toward this end is to encode integration mechanisms as meta-rules, i.e., rules for interpreting and combining other rules. This strategy has received some attention [Davis 77, Clancey 83b]. However, it is not clear that standard rule-based architectures are well-suited to representing and applying integration mechanisms. We are not saying that rule-based architectures (which would be more explicit) could not in principle support integration, just that in practice, system builders have not seemed to find it convenient to use rules to reason about integration.

An alternative approach to expert system construction that makes integration knowledge explicit was first adopted in the XPLAIN system [Swartout 83] and is undergoing further development in the Explainable Expert Systems project [Neches et al 85]. The approach is based on the observation that usually (sometimes only) the person who created an expert system can give a very good explanation of how it works. Our approach for capturing the knowledge normally lost during expert system construction is to use an automatic program writer to create the expert system from an initial knowledge base. This knowledge base contains both abstract knowledge about the domain and general knowledge about expert system construction, including integration knowledge. As the writer creates the system, it records its reasoning in a development history. Explanation routines use this history to produce explanations of the design rationale incorporated in the constructed system. These explanations are valuable both to end-users interested in understanding the system and to system builders interested in modifying or extending it.

This approach promises other benefits in addition to improved explanations. Because integration knowledge is explicitly separated out, it does not become confounded with problem-solving knowledge as occurred in the example with MYCIN certainty factors above. It allows greater flexibility, because several different techniques for integration can be represented in the system's knowledge base, with each one used only when it is appropriate. Finally, the explicit separation makes the system more modular and easier to extend. For more details on this approach, see [Neches et al 85].

This paper presents some general knowledge integration techniques we have identified and shows how they were incorporated in the therapy selection algorithm used by MYCIN [Buchanan & Shortliffe 84] to prescribe drugs after it had diagnosed which organisms were likely infecting the patient. We found this algorithm to be a rich source of such techniques, since it integrates a set of diverse and conflicting criteria in selecting the best therapy. In fact, the interactions among these criteria made it impractical to implement therapy selection using MYCIN-style rules alone, so a somewhat *ad hoc* procedure [Shortliffe 84] was used. Because the knowledge for therapy was implicitly encoded in a procedure, explanations of therapy decisions could not be given and the code proved difficult to maintain. A subsequent reimplementation by William Clancey [Clancey 84] factored out most of the medical knowledge embedded in the procedure into sets of rules pertaining to various therapeutic factors like drug sensitivity and contra-indications. The reimplemented therapy selection algorithm was considerably more general and invoked these rules to evaluate individual therapeutic factors based on knowledge about specific drugs, organisms, or patients; however, the algorithm itself was responsible for integrating these factors into therapy recommendations. The algorithm makes a good case study because its design is dominated by knowledge integration concerns, rather than medical or computational details.

2. Specification of the therapy selection problem

The therapy selection problem is easy to specify informally -- given a diagnosis (one or more organisms suspected of infecting the patient), choose the therapy (set of drugs) that best satisfies the following medical goals:

1. Maximize drug sensitivity.
2. Maximize drug efficacy.
3. Continue prior therapy.
4. Minimize number of drugs.
5. Give priority to covering likelier organisms.
6. Maximize number of suspected organisms covered.
7. Don't give two drugs from the same general class.
8. Avoid contraindications for the patient.

Suppose we implemented these goals as rules, e.g.:

```
Rule for goal 4:
If therapy x uses fewer drugs than therapy y,
then prefer therapy x over therapy y.

Rule for goal 6:
If therapy x covers fewer suspected organisms
 than therapy y,
then prefer therapy y over therapy x.
```

Therapy selection would require integrating the recommendations made by the rules' right-hand sides. For instance, if therapy A has fewer drugs than therapy B, but covers fewer organisms, the rules would conflict.

Integrating goals requires knowledge about their relative importance. The informal specification above is ill-defined because it omits this knowledge, i.e., it doesn't specify which therapy is "best" where the goals conflict. What's medically "best" (i.e., for the patient, the physician, the hospital, and society at large) depends on information not available with certainty at therapy selection time, e.g., the actual effectiveness of the therapy, whether the benefits of the therapy will turn out to compensate for its side effects, and so forth. That is, this ideal sense of "best" is non-operational [Mostow 81], and we must settle for a heuristic approximation. Indeed, the operational definition of "best" is determined by the decisions made in the course of designing the therapy selection algorithm [Swartout & Balzer 82].

The design of MYCIN's therapy selection algorithm was also influenced by computational concerns and restrictions on the design process itself. A comprehensive analysis of the algorithm would explicitly model these aspects as well, but is outside the scope of this paper.

3. MYCIN's therapy selection algorithm

We now describe in brief how MYCIN performs the therapy selection task informally specified in the previous section. MYCIN's (revised) therapy algorithm begins by considering in turn each of the organisms classified by the diagnosis component as most likely. For each organism, it uses rules to assess each known drug as a first, second, or third choice based on the organism's apparent sensitivity to that drug. For example, a typical rule is

```
If the organism growing from the culture
 appears resistant to the drug,
then classify the drug as a third choice.
```

Since MYCIN uses rules to handle part of the sensitivity criterion, the reasons why a drug is classified as a first, second or third choice are accessible and MYCIN can explain them. However, the reasons for partitioning the drugs into three categories (and why three is an appropriate number of partitions) are implicitly built into the algorithm and MYCIN doesn't explain them.

Once the drugs have been classified, MYCIN proposes various combinations of them as possible recommendations. This is done by a series of fixed "instructions" that express how many of each category of drug to select (see Figure 3-1).

	Number of 1st choice drugs:	Number of 2nd choice drugs:	Number of 3rd choice drugs:
1.	1	0	0
2.	2	0	0
3.	1	1	0
4.	1	0	1
5.	0	1	0

Figure 3-1: MYCIN's Table of Therapy "Instructions"

MYCIN goes through the instructions in order until an acceptable therapy is found. For example, the third instruction specifies that one first choice drug and one second choice drug should be selected. Thus the table of instructions integrates the goals of minimizing the number of drugs to administer and selecting the most effective drugs.

The proposed set of drugs is then subjected to three tests to determine whether or not it is acceptable. First, coverage is tested to see whether the proposed drugs cover for all of the most likely organisms. Then the set of proposed drugs is examined to ensure that all the drugs prescribed are in different drug classes. Drugs in the same class work via the same mechanisms so prescribing a second drug from the same class will not increase the overall effectiveness of therapy. Finally, MYCIN checks for patient-specific contraindications. Since all three tests are performed by sets of rules, MYCIN can explain them, e.g., explain the rejection of a given therapy by describing which test it failed and why.

While some of MYCIN's therapeutic expertise is explicit, its overall therapy selection strategy and its knowledge about how to integrate the various therapy goals are encoded procedurally (and implicitly). In Section 5 we will show how such knowledge could be made explicit. But first we discuss how the therapy goals themselves are formulated.

4. Representations of goals

Our analysis of MYCIN's therapy selection algorithm suggests that many of the crucial decisions in its design dealt with how to formulate (or reformulate) the therapy goals listed in Section 2. A goal like "maximize drug effectiveness" can be represented in several ways. The choice of representation determines what kind of information about the goal can be encoded, how much space it takes to do so, what inferences can be drawn from the representation, and how much time they take. For example, information about drug effectiveness might be expressed as:

- A **set of axioms** describing which drugs are more effective than others and by how much. This representation can encode arbitrary kinds of information, but drawing inferences from it requires a theorem-prover.
- A **partial ordering** represented as a boolean connection matrix or as an acyclic graph with a drug at each node. This representation encodes no information about the magnitude of differences in drug effectiveness. The matrix representation allows two drugs to be compared in constant time but requires quadratic space; the graph representation is smaller but slower.
- A **"preference" ordering**, which is like a partial ordering but allows distinct elements to be considered equivalent.[**] A preference ordering on drugs can be represented as a graph with an equivalence class of drugs at each node, rather than a single drug.
- A **linear ordering** represented as an array of drugs in decreasing order of effectiveness. This representation takes linear space and allows two drugs to be compared in constant time.

[**]Formally, a preference ordering is a reflexive, transitive binary relation; unlike a partial ordering, it need not be anti-symmetric.

However, it imposes a preference between any two drugs.

- A **metric** represented as a table showing a numerical effectiveness score for each drug, or as a procedure or set of rules for computing a drug effectiveness score from other data (e.g. sensitivity and efficacy). Comparing two scores takes constant time. This representation assigns numerical magnitudes to all differences in effectiveness.

- A **partition** of the set of drugs into symbolic categories corresponding to different levels of effectiveness. This representation suppresses differences among elements of the same category. Relationships among the categories themselves can be encoded using any of the representations described here.

- A yes-or-no **predicate** that tells whether a given drug is effective. This representation converts the optimization problem of choosing the most effective drug to the satisficing problem of choosing a drug that is good enough.

These representations are not equivalent: they differ in the kind of information they can express. For example, a partial ordering or preference can represent incomplete knowledge about which of two drugs is more effective, while a metric cannot. However, they represent no information about the magnitude of the difference in effectiveness, while a metric does.

The different representations also vary in their computational costs; generally speaking, a simple representation like a metric is cheaper to store and use than a more precise representation like a preference graph. In general there is a tradeoff between the expressive power and computational efficiency of knowledge representations when it comes to integrating knowledge. Consider the problem of integrating two therapy goals. Representing each one as a set of axioms allows us to express arbitrary knowledge about it, but requires a correspondingly sophisticated inference mechanism to combine the goals, or to compare how well various therapies satisfy each goal. It is much simpler to represent each goal as a metric, even though this representation has less expressive power. The metrics can then be combined by a weighted sum or other numerical formula. It is even simpler to represent each goal as a constraint -- a predicate that a therapy must satisfy to be considered acceptable. The constraints can be combined simply by conjoining them; that is, a therapy achieves the combined goals if it satisfies both constraints. Faced with the task of integrating diverse goals like the therapeutic criteria listed in Section 2, expert system designers often formulate (or reformulate) them into simple representations such as linear orderings, metrics, or constraints.

5. Partial derivation of the algorithm

We now rederive portions of the therapy selection algorithm described in Section 3 by (re-)formulating and integrating the eight medical goals listed in Section 2. Reformulation and integration techniques are highlighted, e.g., EXTEND, and defined as they are introduced. A forthcoming extended version of this paper will present a more complete derivation and list of techniques.

5.1. Combine effectiveness with number of drugs

MYCIN's table of "instructions," described in Section 3, integrates the goals of fewer and more effective drugs. How can this mechanism be explained?

1. Decide how to represent each goal.

2. Extend the preference for effective individual drugs into a preference among therapies (sets of drugs).

3. Combine the two preferences on therapies into a single ordering.

First we must represent the goals to be integrated. The number of drugs defines a **linear ordering**, call it $<_{\text{fewer}}$, on therapies. Individual drug effectiveness is represented as a **metric** on a scale of 100-1000.

To make the table of instructions small enough to construct and store, MYCIN's designers elected to PARTITION the drug effectiveness metric into three categories: "first choice," "second choice," and "third choice."

Definition: A preference ordering is CONDENSED by defining a many-to-one function F such that $F(x) < F(y)$ implies $x < y$. $F(x)$ can be thought of as an abstraction of x. In particular, a **metric** $M(x)$ is PARTITIONed into the **linearly ordered** categories 1, ..., N+1 by splitting the range of the metric into the N+1 intervals defined by N breakpoints given as parameters:

$$\text{PARTITION}(t_1, \dots t_N)\text{:}\ \ M(x) \rightarrow F(x),$$

where F is defined as $\lambda(x)\ (i \mid t_{i-1} \leq M(x) < t_i)$, $t_0 \leq M(x) < t_{N+1}$, and $\rightarrow$ means "is reformulated as."

In the case at hand, M is MYCIN'S drug effectiveness metric, which ranges from $t_0 = 1000$ (most effective) down to $t_3 = 100$ (least effective), there are $N = 2$ breakpoints, $t_1 = 700$ and $t_2 = 700$, and $F(x)$ is the classification of drug x as a 1st, 2nd, or 3rd choice. The PARTITIONed ordering $1 < 2 < 3$ (note that $<$ means "is *more* effective than") is small enough to combine with the preference for fewer drugs by using an ordered table of "instructions"; at runtime MYCIN assigns each drug to one of these categories based on its score, and uses the table to generate therapies in (roughly) best-first order. If the effectiveness metric was not PARTITIONed, the table would be much too big to store, and would have imposed an unreasonable informational burden on the table's designers by requiring them to make distinctions between therapies based on minute differences in drug effectiveness.

In general, CONDENSE assumes that the function F captures all the information required to compare x and y with respect to <. In practice, this assumption is typically violated to some extent, i.e., the original preference is distorted in order to condense it. Also, notice that $F(x) < F(y)$ *usually* -- but not always --

implies a *significant* difference between M(x) and M(y). Exceptions occur when x and y lie close to the breakpoint separating two categories. For example, the difference between drugs in categories 1 and 2 is usually significant, but not when both are rated close to 700.

Next we EXTEND the ordering on drug categories into a **preference ordering** on therapies.

Definition: An ordering on individual items can be EXTENDed into an ordering on bags of items as follows:

1. $\{x\} < \{y\}$ iff $x < y$.

2. If $X < Y$ and $X' \leq Y'$, then $X+X' < Y+Y'$, where + denotes bag union.

For example $1 < 2$ implies $\{1\} < \{2\}$ and $\{1,1\} < \{1,2\}$.

We combine the preference $<_{fewer}$ for fewer drugs with the preference $<_{effective}$ for more effective therapy*** by CONJOINing them.

Definition: The result of CONJOINing two preferences, $<_P$ and $<_Q$, is the preference $<_{P\&Q}$, where:

1. If P and Q both say that x is at least as good as y, so does P&Q: $x \leq_{P\&Q} y$ iff $x \leq_P y$ and $x \leq_Q y$.

2. If in addition one of them says that x is preferable, so does P&Q: $x <_{P\&Q} y$ iff ($x \leq_P y$ and $x <_Q y$) or ($x <_P y$ and $x \leq_Q y$)

The table of instructions shown in Figure 3-1 is largely specified by the preference ordering $<_{fewer\&effective}$. For example, $\{1\} <_{fewer\&effective} \{1,1\} <_{fewer\&effective} \{1,2\} <_{fewer\&effective} \{1,3\}$.

In general, CONJOINing preferences doesn't specify what to do when they conflict, since it makes no assumptions about their relative importance. For instance, $<_{fewer\&effective}$ imposes no ordering between $\{1,1\}$ and $\{2\}$, or between $\{2,2\}$ and $\{1,3\}$.

This partial ordering is next LINEARIZEd into a total ordering which we'll call $<_{better}$.

Definition: Any partial ordering $<_P$ can be embedded into a linear ordering $<_L$. (Unless $<_P$ is total, it will have more than one possible linear embedding.) The LINEARIZEd ordering has the property that $x <_P y \Rightarrow x <_L y$. The converse does not always hold, i.e., $x <_L y$ need not imply $x <_P y$; $<_P$ might specify no ordering relation between x and y, in which case $x <_L y$ is an artifact of the particular embedding $<_L$. That is, an alternative linear embedding, $<_{L'}$, might not satisfy $x <_{L'} y$.

MYCIN's LINEARIZEd sequence of "instructions" incorporates some implicit assumptions about the relative importance of therapy effectiveness and number of drugs. For example, it specifies $\{1,1\} <_{better} \{2\}$ and $\{1,3\} <_{better} \{2,2\}$, but leaves implicit the supporting assumption that "a first choice drug is worth a *lot* more than a second choice drug" [Clancey, personal communication, January 2, 1985]. This assumption is nowhere represented in MYCIN, and as we saw in the discussion of PARTITION, it is not always true. When it is violated, anomalies can arise. For example, MYCIN would propose a combination of two drugs rated 701 before proposing a single drug rated 699, even though the former are really *not* "worth a lot more." While such anomalies may be infrequent and unimportant enough for the expert system designer to tolerate them, a robust expert system ought to recognize when they occur.

It should be noted that MYCIN preserves some distinctions among drugs in the same category by generating them in order of decreasing effectiveness scores, so as to help generate therapies in best-first order. This feature doesn't prevent the anomaly just described, but would make MYCIN propose the drug rated 699 before another "second choice" drug that wasn't rated as highly.

In general, a LINEARIZEd ordering is ambiguous as a representation of the original preference, since $x < y$ doesn't tell whether x is really preferable to y, or if x just happens to precede y as an artifact of how the preference was LINEARIZEd.

Suggestion: An attractive way to LINEARIZE a partial ordering would be to explicitly specify the assumptions that the resulting linear ordering should satisfy. A theorem-proving engine would use these assumptions to fill in the ordering relation and identify cases where the assumptions fail to imply an ordering. The designer could provide additional rules to cover such cases, and the process could continue interactively until the ordering was complete. If feasible, this approach would be better than constructing a table by hand because it would make explicit the assumptions left implicit in such tables, making it possible to distinguish preferences based on genuine domain knowledge from those based only on general principles or computational expedience.

5.2. Combine coverage preferences

The therapy goals listed in Section 2 include maximizing the number of organisms covered and giving priority to those the patient is likelier to have. Let's see how these two goals are integrated:

1. Classify organisms as "most likely" or "less likely."

2. Relax the coverage goal by ignoring "less likely" organisms.

3. Reformulate the coverage goal as the constraint that all the "most likely" organisms be covered.

Organism likelihood is PARTITIONed into only two categories, "most likely" and "less likely." Assuming that the "most likely" organisms are much more likely than the "less likely" organisms, the importance of treating the "most likely" organisms DOMINATEs the importance of treating the "less likely" organisms.

***Note that A $<_{effective}$ B means therapy A is *more* effective than therapy B, i.e., preferable with respect to effectiveness.

Definition: Letting one preference -- call it $<_{primary}$ -- DOMINATE another preference -- call it $<_{secondary}$ -- means using $<_{secondary}$ only to resolve ties with respect to $<_{primary}$. The resulting preference $<_{primary;secondary}$ is defined by

$$x <_{primary} y \text{ or } (x =_{primary} y \text{ and } x <_{secondary} y).$$

Definition: A preference can simply be IGNOREd. For example, ignoring $<_{secondary}$ reduces $<_{primary;secondary}$ to $<_{primary}$. This particular case of IGNORE is appropriate if ties with respect to $<_{primary}$ are too rare to worry about, or if violating $<_{secondary}$ in the event of such a tie wouldn't do much harm.

It is unlikely for two therapies to be equally effective on the likeliest organisms but different on the less likely ones, so it is reasonable to ignore the less likely organisms altogether. A possible rationale for this compromise is a tradeoff between breadth and effectiveness of coverage, based on the assumption that broad-spectrum drugs are less effective than highly specific drugs. "You could generate all of the recommendations in the equivalence class and pick the one covering the most less likely organisms, but this will probably result in choosing drugs that are lower for most likely organisms (within the rankings). For example, choosing a 950 drug for an organism is preferable to choosing a 750 drug, (both are first rank), even if you pick up a less likely organism" [Clancey, personal communication, January 2, 1985]. This is a good example of a design decision rationale of the form "Errors of type X are unlikely to occur and wouldn't do much harm anyway." Here an "error" would consist of proposing one therapy before another one that covers the more likely organisms just as well and also covers for less likely organisms. The net effect of the PARTITION, DOMINATE, and IGNORE steps is to CONDENSE the preference for maximal coverage by ignoring the less likely organisms.

The CONDENSEd preference compares therapies based on the number of "most likely" organisms covered. This preference is now reformulated into a constraint by THRESHOLDing.

Definition: A metric M(x) can be converted to a constraint by the THRESHOLD transformation

$$\text{THRESHOLD}(t_{min}): M(x) \rightarrow \lambda(x)\ (M(x) \geq t_{min}),$$

where the threshold value t_{min} is a parameter of the THRESHOLD transformation. (Variations on this transformation use $>$, $<$, or $\leq$ in place of $\geq$, and t_{max} in place of t_{min}.)

In the case at hand, x is a candidate therapy, the metric M is the number of "most likely" organisms covered by therapy x, and t_{min} is defined to be the total number of organisms considered "most likely." That is, an acceptable therapy must cover *all* the most likely organisms.

This reformulation incorporates the assumption that all the most likely organisms can and must be covered. The implicit rationale for this assumption has to do with the risks of failing to treat for a likely condition.[****]

[****] Such as being sued for malpractice.

As Section 4 pointed out, reformulating a goal as a constraint makes it possible to test whether a given choice satisfies the goal without having to compare it against alternative choices. Thus converting a preference into a constraint reduces an optimization problem (choose the most preferred element) to a satisficing problem (choose an acceptable element). The latter problem can be solved more efficiently, since it is easier to generate candidates one by one, test each one separately, and accept the first one that passes, than it is to generate all the (possibly infinitely many) candidates, compare them, and pick the best one.

6. Applications

Explicit knowledge about the integration techniques used to construct an expert system could be exploited in several ways, which we illustrate by means of hypothetical examples of behavior.

Notice violated assumptions. If the system can test the assumptions on which an integration technique is based, it may be able to detect flaws in its own reasoning.

```
Therapy A, which consists of two 1st-choice drugs, is rated higher than therapy B, which consists of one 1st-choice and one 2nd-choice drug, based on the assumption that 1st-choice drugs are much better than 2nd-choice drugs. However, one of the 1st-choice drugs in therapy A is rated very close to the 2nd-choice drug in therapy B, so this assumption is questionable here.
```

Detect artifacts. If the system can distinguish genuine domain knowledge from the accidental artifacts of reformulation techniques like METRICIZE or LINEARIZE, it may be able to alert the user to spurious preferences in its recommendations.

```
Therapy X is rated higher than therapy Y because the combination of one 1st-choice drug and one 3rd-choice drug comes before the combination of two 2nd-choice drugs in the table of instructions. However, that might be an accident of how the table was constructed, rather than a genuine medical preference.
```

Support maintenance by inferring constraints and goals. To some extent it is possible to guess rationales for how knowledge has been integrated in an expert system. In the absence of explicit rationales, these guesses may still serve to expose constraints and goals left implicit. For example, the following inferences might be made based on the knowledge that MYCIN's table of instructions is a LINEARIZEd form of the preference formed by CONJOINing the preference for fewer drugs with the CONDENSEd preference for more effective therapy:

```
From the fact that an explicit table is used to decide tradeoffs between maximizing therapy effectiveness and minimizing the number of drugs, it appears that the simpler approach of computing a weighted sum was not considered accurate enough to do the job.

It appears that 3 effectiveness ranks are considered sufficient to discriminate among different drugs, at least for the purpose of deciding tradeoffs between therapy effectiveness and number of drugs. Perhaps 2 ranks were not enough.
```

```
Maximizing therapy effectiveness appears more
important than minimizing the number of drugs, in
the sense that increasing therapy effectiveness by
1 rank is considered more desirable than reducing
the number of drugs by 1.
```

This sort of information might be useful to an expert system maintainer who needed to revise the knowledge base.

Support expert system construction. If reformulation and integration techniques like those described in Section 5 are mechanized, they might eventually be used to help automate expert system construction and documentation.

```
... I detect a conflict between the goals of
maximizing therapy effectiveness and minimizing
the number of drugs.  Which of the following
relationships holds between the two goals?

1. One goal is absolutely less important than the
other.  Only use it to resolve ties with respect
to the more important one.  [=> use DOMINATE]

2. One goal is absolutely less important than the
other, and no ties are expected.  [=> use IGNORE]

3. The relative importance of the two goals can
be adequately expressed as coefficients in a
weighted sum.  [=> use WEIGHTED SUM]

4. The importance of the two goals is relative
and cannot be adequately expressed as coefficients
in a weighted sum.  [=> use TABULARIZE]

[User selects option 4; system tries integrating
preferences in a table.]

There are too many combinations of drug
effectiveness scores to list them in a table.  I
would like to use a coarser measure of
effectiveness [i.e., CONDENSE it].  An ideal
therapy would consist of one drug rated 1000.  How
much lower could one drug be rated and still be
better than any therapy consisting of two drugs?

[User says 700; system PARTITIONs effectiveness
into 300-point subranges.]
```

Automating the knowledge integration process would make it easy to record the design choices, techniques, and assumptions used. Once captured, this information would be available for generating explanations to future users and system maintainers.

7. Conclusion

Rational integration of conflicting preferences involves normalizing them relative to a common supergoal. This process requires identifying an appropriate supergoal and using it to analyze the tradeoff among the preferences. For example, in prescribing therapy, it is preferable to minimize the number of drugs and maximize their effectiveness, but the relative importance of these two preferences depends on their ultimate impact on some higher level criterion. Presumably the implicit topmost criterion in medicine is patient welfare.[*****] Determining the tradeoff between the number of drugs and their effectiveness involves balancing the likelihood and urgency of curing the illness against the likelihood and seriousness of unforeseen drug interactions. However, information about these factors is imprecise at best.

[*****]Cynics think it is physician income.

When the knowledge required to integrate preferences on a mathematically rational basis is unavailable, domain experts and expert system designers generally integrate them instead on whatever *ad hoc* basis is cognitively or computationally expedient. In the absence of compelling medical reasons one way or the other, a physician might choose between a one- and two-drug therapy arbitrarily, out of habit, or based on a medically unjustified rule of thumb. While domain experts make such decisions on a case-by-case basis, expert system designers must anticipate the entire class of situations in which such decisions will be needed, and provide general mechanisms for making them. MYCIN's designers chose to PARTITION drug effectiveness into three categories, which enabled them to store the LINEARIZED set of "instructions" in a precompiled table. Presumably, a simpler design alternative would have been to rate each proposed therapy by computing a WEIGHTED SUM of, say, the effectiveness of each drug in the therapy, with a negative term for the number of drugs. Because MYCIN does not explicitly represent the reasons for using a table of instructions, it is not easy to determine why a weighted-sum approach was considered inappropriate, or even whether it was considered at all.

Lest MYCIN's designers regret their generous assistance to us, or the readers of this paper get the incorrect impression that we are attacking MYCIN, we would like to emphasize that MYCIN is not a particularly egregious example of *ad hoc* integration; the problem of distinguishing arbitrary choices from justified ones is endemic among current expert systems. In fact we chose MYCIN's therapy selection algorithm precisely because the task is too complex to fit the single integration method (certainty factors) used in the rest of MYCIN and in many subsequent systems. We found the algorithm to be a rich source of techniques for integrating knowledge, and we expect case studies of other expert systems and problem-solving programs to help identify, clarify, and formalize such techniques.

If the rationale, or lack thereof, for integrating preferences in a particular fashion is left implicit in the design of an expert system, the artifacts of arbitrary design choices cannot be distinguished from *bona fide* domain knowledge. That is, when the expert system recommends one alternative over another -- for example, when MYCIN prefers a therapy consisting of one 1st-choice and one 3rd-choice drug over a therapy consisting of two 2nd-choice drugs -- we cannot always tell if the recommendation is based on real domain knowledge or is simply the result of some arbitrarily chosen integration scheme.

It is important to distinguish between *justifying* and *explaining* a conclusion made by an expert system. Justification is based on knowledge (or assumptions) about the domain, e.g., "therapy A is rated over therapy B because it's medically more effective." This kind of information is important to the user. In contrast, explanation can refer to computational or design expediency, e.g., "therapy A is rated over therapy B as an artifact of condensing metrics for computational efficiency, and the designers figured it wasn't important enough to bother fixing." This kind of information can be important to the expert system maintainer.

In building an expert system it is expedient to use various knowledge integration techniques, some more justified by domain knowledge than others. The ultimate goal of this research is to create a framework for building expert systems that would support the representation of such integration techniques and the assumptions and tradeoffs involved in using them. Before that goal can be reached, much remains to be done. We must better understand how to formalize the techniques and represent the situations they apply to. We must also develop mechanisms for applying them and for reasoning about which technique to use in a given situation. Finally, the entire knowledge integration process must be recorded in a machine-understandable fashion for subsequent use in generating explanations. Such formalization will impose considerable overhead on the design process (though it should be somewhat offset by automating some of the techniques). However, we argue that an expert system ought to be able to explain its knowledge integration techniques and their underlying assumptions, both to help the user evaluate its recommendations, and to guide the expert system maintainer in adding new knowledge. In the long run, these enhanced capabilities should justify the overhead required to support them.

Acknowledgements

We thank Bill Clancey and Ted Shortliffe for their patient explanations of MYCIN, and Tom Dietterich, Jim Bennett, and Rich Keller for their comments on earlier drafts. Of course any errors are our own.

References

[Buchanan & Shortliffe 84]
B. G. Buchanan and E. H. Shortliffe (Editors).
Rule-Based Expert Systems.
Addison-Wesley, 1984.

[Clancey 83a] W. J. Clancey.
The epistemology of a rule-based expert system: A framework for explanation.
Artificial Intelligence 20(3):215-251, 1983.

[Clancey 83b] William J. Clancey.
The advantages of abstract control knowledge in expert system design.
In *AAAI83*, pages 74-78. Washington, DC, 1983.

[Clancey 84] William J. Clancey.
Details of the Revised Therapy Algorithm.
In B. G. Buchanan and E. H. Shortliffe (editors), *Rule-Based Expert Systems.* Addison-Wesley, 1984.

[Davis 77] R. Davis.
Interactive transfer of expertise: Acquisition of new inference rules.
In *IJCAI-5*, pages 321-328. Cambridge, MA, 1977.

[Mostow 81] D. J. Mostow.
Mechanical Transformation of Task Heuristics into Operational Procedures.
PhD thesis, Carnegie-Mellon University, 1981.
Technical Report CMU-CS-81-113.

[Neches et al 85]
R. Neches, W. Swartout, and J. Moore.
Enhanced maintenance and explanation of expert systems through explicit models of their development.
IEEE Transactions on Software Engineering SE-11(11):1337-1351, November, 1985.

[Shortliffe 84] Edward H. Shortliffe.
Details of the Consultation System.
In B. G. Buchanan and E. H. Shortliffe (editors), *Rule-Based Expert Systems.* Addison-Wesley, 1984.

[Swartout 83] Swartout, W.
XPLAIN: A system for creating and explaining expert consulting systems.
Artificial Intelligence 21(3):285-325, September, 1983.
Also available from USC Information Sciences Institute as ISI/RS-83-4.

[Swartout & Balzer 82]
Swartout, W., and Balzer, R.
On the inevitable intertwining of specification and implementation.
CACM 25(7):438-440, July, 1982.