

OPERATIONALIZING HEURISTICS: SOME AI METHODS FOR ASSISTING AI PROGRAMMING

Jack Mostow
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Frederick Hayes-Roth
Information Sciences Department
The Rand Corporation
1700 Main Street
Santa Monica, California 90406

Machine-aided heuristic programming is a paradigm for incorporating domain knowledge in intelligent task performance programs. In this paradigm, a system interactively assimilates a natural language description of a task, advice on how to perform it, and definitions of the domain concepts in terms of which the advice is expressed. The system translates this input into an internal representation, operationalizes the assimilated knowledge to make it effective, integrates different pieces of advice, and applies them to the performance of the task. A typed applicative LISP-like language is used for the internal representation of domain knowledge. Operationalization is defined in terms of transforming well-defined but non-effective expressions into effectively executable ones. Several techniques for performing this process mechanically are presented and applied to an example drawn from the domain of the card game Hearts. A system to operationalize Hearts advice is currently being implemented as an instantiation of the advocated paradigm.

1. Introduction

In a longer version of this paper [4], we advocate machine-aided heuristic programming as a paradigm for training computers to do new tasks, and formulate a set of research problems involved in building a machine-aided heuristic programming system (hereafter abbreviated "mahps") capable of assimilating conceptually diverse knowledge and putting it to effective use. In this paradigm, an expert interactively gives the mahps advice about how to do a task, plus definitions of the concepts in terms of which that advice is expressed. For example, in training a mahps to play the card game Hearts,** the expert might advise it to avoid taking points. Such advice is *parsed* into a syntax tree which is *interpreted* into an internal representation. Advice may then be *operationalized*, either *statically* -- independent of a particular game state -- or *dynamically* -- exploiting current information arising from an actual game. Once operationalized, the advice can be applied to produce recommendations relevant to the decision at hand.

Diverse advice is *integrated* by combining these recommendations, either rationally or heuristically, and *applied* to the system's game decisions and to its interpretations of its opponents' behavior. At any point, the system can pose questions to the expert using the rule

If x is an important question which has resisted solution, explain it to the user (preferably with a small set of multiple choice answers) and ask for the answer or a piece of relevant advice.

The expert monitors the system's behavior and improves it by providing additional advice.

In this paper, we focus on the automatic operationalization of advice. We assume that the advice has been interpreted into an unambiguous internal representation of its meaning, and suggest some approaches to the problem of transforming that representation so that it can be effectively applied to the task.

2. Representation of Conceptual and Heuristic Knowledge

Our knowledge representation is a typed lambda-calculus combining an applicative LISP-like language [3] with a schema-like structure. The basic unit of representation is the *concept*, which is a named lambda-expression. Schematic structure associated with each concept is described in [4]. Briefly, it provides type information about the concept and its lambda-variables, and defines inheritance relations between general and specific concepts.

*This work was supported in part by the National Science Foundation, RCN 6669, Grant No. MCS77-03273 to the Rand Corporation, in part by the Graduate Fellowship program support to the first author, and in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33815-78-C-1551. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

**The card game Hearts is used throughout this paper for purposes of illustration; the rules of the game are given in [4].

A type is simply a class concept. Since one class may be a subclass of another, an entity may belong to more than one type. For instance, ME (the machine's card-playing persona) is both a PLAYER and a COMPUTER. Two types are said to *match* if they overlap. Thus PLAYER matches HUMAN since it is possible to be both human and a player.

Although there is no assignment operation in this representation, every expression is implicitly a state function, i.e., its interpretation depends on the state of the world it describes. Actions are represented as functions from a state to an *event*. A *history* is an actual or hypothetical sequence of states and events. When an action is performed, the history of what happens during its execution is recorded so it can be referred to subsequently. Thus at any point in time one can refer to the PAST -- the history of events that have already occurred, the PRESENT -- the set of events that have begun and not yet ended, and the FUTURE -- the as yet undetermined history about to occur.

2.1. Meta-functions

Our representation provides a limited number of built-in meta-functions, characterized by their lack of strict type (some of them take a type as a parameter) and their need to examine some of their arguments without evaluating them. These meta-functions include the following (optional parameters and their default values appear in brackets):

(EXISTS x S P) = there exists x in S such that P

(SET-OF x S P) = the set of x in S such that P

(THE x S P) = the unique x in S such that P

(SOME x S [P]) = any x in S [such that P]; any expression containing this construct is potentially non-deterministic

(ALL e) = set of possible values for the non-deterministic expression e; thus (ALL (SOME x S)) = S and (SOME x (ALL e)) = e

(SUM V x S [P]) = the summation of V(x) for x in S [such that P(x)]

(* S) = the number of elements in the set S

(GIVEN P x) = the value of x given that P is true

(PR P) = probability that P is true

(PR P Q) = conditional probability of P given that Q is true; thus (PR P Q) = (PR (GIVEN Q P)) = (GIVEN Q (PR P))

(CASE (P₁ V₁) ... (P_k V_k) [V]) = the unique (GIVEN P_i V_i) such that P_i is true [otherwise V]

(IF P₁ V₁ ... P_k V_k V) = the first (GIVEN P_i V_i) such that P_i is true, otherwise V

(SET x₁ ... x_k) = the set {x₁, ..., x_k}

(SCENARIO x₁ ... x_k) = the event sequence x₁ ... x_k; each x_i can be either a single event or a scenario

(EACH x L A) = (SCENARIO A(x₁) ... A(x_k)) where L = (LIST x₁ ... x_k)

(DO scenario state) = simulate the effects the scenario would have on the specified state

(ACTIONS-OF agent [state = PRESENT]) = the set of possible actions the agent can do at present [or in a specified state]

(DURING history event₁ ... event_k) = true iff the scenario event₁ ... event_k occurs during history

(CURRENT event) = history of the current incarnation of a generic event such as TRICK

(CHANGE P₁ P₂ state) = altered state produced by making P₁ false and P₂ true

(AFTER event) = state following a specified event

2.2. Example concept definitions

An essential feature of this representation is its *extensibility*. A mahps must be able to assimilate new concepts defined by the expert in terms of existing ones. The following definitions of concepts used in later examples illustrate the representation.

```
(DE TRICK NIL
  (SCENARIO (EACH P (ORDER-OF-PLAY)
    (PLAY-CARD P))
    (TAKE-TRICK (TRICK-WINNER))))

(DE PLAY-CARD (P)
  (PLAY P (SOME C (LEGAL-CARDS-OF P))))

(DE CARD-OF (P)
  (THE C CARDS
    (DURING (CURRENT TRICK)
      (PLAY P C))))

(DE LEGAL-CARDS-OF (P)
  (SET-OF C CARDS (LEGAL P C)))

(DE LEGAL (P C)
  (AND [HOLDS P C]
    [CASE ((FOLLOWING P)
      (OR [VOID P (SUIT-LED)]
        [= (SUIT-OF C)
          (SUIT-LED)])])
    ((LEADING P)
      (OR [CAN-LEAD-HEARTS P]
        [INEQ (SUIT-OF C) H])]))]
```

```

(DE TRICK-WINNER NIL
 (THE P PLAYERS
  (DURING (CURRENT TRICK)
   (PLAY P (WINNING-CARD)))))

(DE TAKE-TRICK (P)
 (EACH C (CARDS-IN-POT) (TAKE P C)))

(DE TAKE (PLAYER CARD)
 (MOVE PLAYER CARD POT (PILE-OF PLAYER)))

(DE PLAY (PLAYER CARD)
 (MOVE PLAYER CARD (HAND-OF PLAYER)
  POT))

(DE MOVE (AGT OBJ ORIG DEST)
 (CAUSE AGT
  (CHANGE (AT OBJ ORIG)
   (AT OBJ DEST)))))

(DE CAUSE (AGENT EVENT)
 (SOME ACT (ACTIONS-OF AGENT)
  (LEAD-TO ACT EVENT)))

(DE POINTS (CARD)
 (CASE ((= CARD QS) 13)
  ((= (SUIT-OF CARD) H) 1)
  ((= CARD JD) -10)
  0))

(DE AVOID (AGENT EVENT)
 (SOME ACT (ACTIONS-OF AGENT)
  (SMALL (PR-LEAD-TO ACT EVENT)))))

(DE SMALL (X) (< X 0.1))

(DE PR-LEAD-TO (EVENT1 EVENT2 (H = FUTURE))
 (PR (LEAD-TO EVENT1 EVENT2 H)))

(DE LEAD-TO (EVENT1 EVENT2 (H = FUTURE))
 (GIVEN (DURING PRESENT EVENT1)
  (DURING H EVENT2)))

```

Heuristics are concepts too. A goal like "avoid taking points" can be represented as a predicate to be satisfied by an agent:

```

(DE AVOID-TAKING-POINTS (P)
 (AVOID P (TAKE-POINTS P)))

```

2.3. Discussion

A key feature of our approach is the separation of representation from implementation. The definition of a concept encodes only enough information to specify its meaning precisely. The task of converting this definition into an effective implementation is factored out into the operationalization process. Essentially we are retaining LISP-like semantics but replacing EVAL with a collection

of heuristic methods for evaluating expressions. We feel that the traditional popularity of LISP as an implementation language has obscured its potential as a powerful, concise representation of knowledge.

3. Operationalization

This section suggests several methods for operationalization, which we define as transforming well-defined but non-effective expressions to make them effectively executable. The expressions used in the examples in this section use the representation described in Section 2. An expression may be non-effective in any of several ways:

- It may depend on non-observable information that can be deduced, or on uncertain data that can be predicted only probabilistically.
- It may depend on events that have not yet occurred but can be reasoned about.
- It may be defined intensionally over an infinite or combinatorially large domain and hence infeasible to compute systematically.
- It may be defined in terms of a function which can only be computed by special-case reasoning because it has no known general formula.
- It may be too complex to evaluate exactly but may still be amenable to approximation.

Operationalization seeks to resolve such difficulties by combining appropriate methods: means-ends rules, special-case reasoning, simplifying assumptions, search, etc. It involves two important subproblems:

1. *Mapping expressions to methods:* Match an expression to a suitable method for operationalizing it, and decide how to instantiate the premises of the method in terms of the expression.
2. *Representation of AI methods:* Represent inference methods in a form that can readily apply to novel problems. This requires formalization of techniques which, although widely used, have not previously been characterized with sufficient precision to permit automatic application to a wide range of tasks.

Given an internal representation of what "avoid taking points" means, the mahps must *operationalize* that concept, i.e., figure out how to avoid taking points.

3.1. Goal-specific operationalization rules

The simplest way is to use some method provided by the expert, such as

To avoid taking points, play a low card.

This method can be represented by a means-end rule that essentially ignores the underlying conceptual structure of "avoid taking points" and simply treats it as a goal symbol. Such rules enable user-provided domain-specific operationalization of concepts.

3.2. Systematic evaluation

Systematic evaluation can be used as an operationalization technique. In this approach, concepts are treated as functions, and expressions are systematically expanded until known quantities are reached. Functions are applied to arguments by the rule

To operationalize an expression $(F\ e_1 \dots e_n)$ where F has no known operationalization, substitute $e_1 \dots e_n$ for the corresponding lambda-variables in the functional definition of F .

Thus

AVOID-TAKING-POINTS

```
= (AVOID ME (TAKE-POINTS ME))  
  
= (SOME ACT (ACTIONS-OF ME)  
  (SMALL (PR-LEAD-TO ACT  
           (TAKE-POINTS ME))))
```

The subexpression

(PR-LEAD-TO ACT (TAKE-POINTS ME))

can be expanded to

```
(PR (DURING FUTURE (TAKE-POINTS ME))  
  (DURING PRESENT ACT))  
  
= (GIVEN (DURING PRESENT ACT)  
  (PR (DURING FUTURE (TAKE-POINTS ME))))
```

Systematic evaluation of this expression is precluded by the lack of a general definition that can be substituted for PR. In general, this approach encounters difficulties when there is inadequate knowledge to evaluate an expression, or when systematic evaluation is computationally exorbitant. Heuristics proposed elsewhere [4] for dealing with such difficulties apply special-case knowledge about partially-defined functions or introduce simplifying assumptions that permit approximate solutions.

3.3. General operationalization methods

(This section uses fault tree analysis as an example of a general operationalization method and shows how it can be used to operationalize "avoid taking points." The purpose of this exercise is to suggest the feasibility of performing such reasoning automatically by applying simple search and manipulation operations to a knowledge base of user-defined concepts.)

A reasonable way to operationalize "avoid taking points" is to apply knowledge about how to avoid things in general. For instance, one method for avoiding an event is fault tree analysis: figure out what conditions can cause the event, and prevent them. Analysis of the game rules shows that taking points only occurs by winning a trick in which points have been played. Fault tree analysis might suggest trying to lose the current trick if it is likely to have points.*

Note what reasoning is required to do this sort of operationalization. The idea of using fault tree analysis can be suggested by instantiating a rule like

If your goal is (operationalize (avoid x))
then find a necessary condition for x and make
its negation a goal.

The analysis itself might be performed as follows. (This description omits numerous fruitless branches taken along the way; the intermediate steps of the derivation given below require some search to find.) First the definition of TAKE-POINTS is expanded in terms of TAKE:

```
(TAKE-POINTS ME)  
  
= (TAKE ME (SOME X CARDS (HAS-POINTS X)))
```

Next the mahps looks for EVENT concepts in which TAKE occurs. (This simply requires the ability to iterate over the set of defined concepts.) The only such concept is

```
(DE TAKE-TRICK (P)  
  (EACH C (CARDS-IN-POT) (TAKE P C)))
```

Thus TAKE-TRICK is the only way the mahps knows about of taking a card. In particular,
(TAKE ME (SOME X CARDS (HAS-POINTS X))) implies

```
(AND [TAKE-TRICK ME]  
  [EXISTS C (CARDS-IN-POT)  
    (HAS-POINTS CARD)])
```

*A sophisticated player will save low cards to use during the latter part of the game, when the risk of taking points normally increases. It would be rather difficult for a mahps to infer this subtle tactic from the advice "avoid taking points"; it is much more reasonable for the expert to suggest it explicitly.

The conjunct

```
(EXISTS C (CARDS-IN-POT)
  (HAS-POINTS CARD))
```

In this expression is the definition of TRICK-HAS-POINTS. Similarly, the only known case of TAKE-TRICK occurs in

```
(DE TRICK NIL
  (SCENARIO (EACH P (ORDER-OF-PLAY)
    (PLAY-CARD P))
    (TAKE-TRICK (TRICK-WINNER))))
```

so (TAKE-TRICK ME) implies (= ME (TRICK-WINNER)) In short, (TAKE-POINTS ME) occurs iff

```
(AND [= ME (TRICK-WINNER)
  (TRICK-HAS-POINTS)])
```

To avoid (TAKE-POINTS ME), the mahps can add to its goals the proposition

```
(IF (TRICK-HAS-POINTS)
  (NOT (= ME (TRICK-WINNER))))
```

Each step of this derivation can be accounted for by a simple operation, such as lambda-substitution or unification matching. It appears quite feasible to automate this sort of analysis as, say, a set of rules for when to perform each operation.

3.4. Weak methods

(This section uses heuristic search as an example of a standard AI "weak method" [5] and applies it to the operationalization of "avoid taking points." It is argued that the reasoning involved in so doing might feasibly be performed mechanically. Some of the problems involved in applying a general method to a given problem are discussed, and some techniques for solving them are proposed.)

AI provides several weak methods applicable to the operationalization problem. One such method is heuristic search (hereafter abbreviated "HSM"), which can be used to find a sequence of elements satisfying a given property [5]. Figure 1 shows (in quasi-SAIL syntax) a generic "procedural" for HSM which yields an effective search procedure when the underlined components are instantiated by the components of a particular search problem. Figure 2 gives a schematic description of these components. For example, it specifies "operator" as the type consisting of all functions from a node to a node.

Let us see how the mahps might operationalize "avoid taking points" in terms of heuristic search. Suppose it has encountered difficulties in operationalizing

```
(SMALL (PR-LEAD-TO ACT (TAKE-POINTS ME)))
```

and applies

If you're having trouble operationalizing a predicate P, find a non-trivial predicate Q that implies P, and operationalize Q as an approximation for P.

with P = (SMALL (PR x)) and Q = (NOT x). The mahps is now left with the problem of operationalizing

```
(NOT (LEAD-TO ACT (TAKE-POINTS ME)))
```

```
= (NOT (GIVEN (DURING PRESENT ACT)
  (DURING FUTURE
    (TAKE-POINTS ME))))
```

```
= (NOT (EXISTS H (ALL FUTURE)
  (DURING H ACT TAKE-POINTS ME)))
```

```
node sequence procedure HSM-search =
begin "search"
  node n, n';
  operator op;
  search-space := {inode};
  repeat
    if (n := select-node(search-space)) = NIL
    then return failure
    ! search-space exhausted
  else if prune-test(n)
    then remove n from search-space
    ! prune nodes not worth considering
  else begin "consider node"
    if (op := select-op(n)) = NIL
    then remove n from search-space
    ! no operators left to apply to n
    else begin "expand node"
      n' := op(n);
      Link(n, n');
      if solution-test(n')
      then return Path-to(n');*
      ! (inode, ..., n') is solution
      if retain-test(n')
      ! prune if not worth retaining
      then put n' in search-space;
      end "expand node";
    end "consider node";
  end "search";
```

Figure 1: Procedural for heuristic search

*Path-to(n') finds the path (inode, ..., n') by tracing back along the links established by Link(n, n').

To decide whether such a scenario H exists, the mahps can use heuristic search. This method might be suggested by partial-matching the sub-expression

(EXISTS H (ALL FUTURE)
(DURING H ACT (TAKE-POINTS ME)))

HSM	type METHOD
problem	type PROBLEM def (FIND S (SEQUENCES-OF node) (ADMISSIBLE S))*
node	type TYPE
search-space	type node set
inode	type node
operator	type TYPE def function with arg: node, result: node
operators-on	type function with arg: node, result: operator set
test	type TYPE def predicate with arg: node
solution-test	type test
prune-test	type test
retain-test	type test
select-node	type generator** from search-space
select-op	type generator from operators-on(node) with arg: node
solution	type node sequence def (HSM-search)

Figure 2: Schema for heuristic search

*Defined by

```
(DE ADMISSIBLE (S)
  (AND (= S1 inode)
    [solution-test S1(LENGTH S)]
    [FORALL I (BETWEEN 2 (LENGTH S))
      (EXISTS OP (operators-on S1I-1)
        (= (OP S1I-1 S1)))]))
```

**A generator is a procedure that returns an element of its choice set each time it is called. The choice set may be expanded by other processes between calls to the generator.

against the HSM problem description

(FIND S (SEQUENCES-OF node) (ADMISSIBLE S))

The facets of HSM can then be instantiated by identifying corresponding residuals in the partial match [1], using

To operationalize an expression E in terms of a method M, partial-match E against M's problem description and instantiate the components of M with the corresponding components of the expression.

For instance, the correspondence between (ALL FUTURE) and (SEQUENCES-OF node) suggests using game states as nodes in the search, since each future is a sequence of game states. However, additional information, not explicit in the expression, is needed to formulate an effective search. For instance, what are the operators?

The answer to this question is not explicit in the expression, but the HSM schema gives a clue: it specifies the operators as functions from a node to a node, i.e., from a state to a state. By examining the logical types of the concepts it knows about, the mahps can find that actions are functions from a state to an event. By applying the rule

To view a concept c as an entity of type X, find a known function F whose domain contains c and whose range matches X, and view c as (F c).

the mahps can view any action (A x₁ ... x_k) as the composite function (AFTER (A x₁ ... x_k)) from a before state to an after state.

Next the mahps can decide which actions should be used as operators in the search. An adequate set of operators in this case is given by (PLAY (PLAYER) C) for C in (LEGAL-CARDS (PLAYER)), where (PLAYER) denotes the player whose turn it is to play in the current state. A more efficient search might use abstract operators like

(OVERPLAY (PLAYER) ME)

```
= (PLAY (PLAYER)
  (SOME C CARDS
    (AND [HIGHER C (CARD-OF ME)]
      [= (SUIT-OF C)
        (SUIT-LED)])))
```

(SLUFF-POINTS (PLAYER))

```
= (PLAY (PLAYER)
  (SOME C CARDS
    (AND [HAS-POINTS C]
      [NEQ (SUIT-OF C)
        (SUIT-LED)])))
```

as suggested by

To reduce a search space, suppress operator details that are irrelevant to the search criterion, and use the resulting abstractions in place of the original operators.

Such abstract operators have been used to increase planning efficiency in problem-solving systems [6, 7].

Other information absent in the expression includes the functions select-node and select-op. The simplest solution to this problem is to use random selection functions. However, this would result in an inefficient search. A better choice can be made by applying

When selecting nodes and operators in a search, select first those most likely to satisfy the search criterion.

Thus when assigning successive values to the non-deterministic expression (SOME C (LEGAL-CARDS P)), the mahps should choose first those cards most likely to satisfy (TAKE-POINTS ME), namely cards that underplay its own choice or that break suit. This ordering preference can be solicited from the expert by or derived by applying fault tree analysis to the search criterion. If (TAKE-POINTS ME) occurs in some plausible scenario (i.e., a scenario consistent with ME's knowledge about legal play and the distribution of unplayed cards), this would be detected early in the search. If it doesn't, however, the whole combinatorial space would be searched. This could be avoided in a couple of ways. First, the mahps might simply prune branches of the search considered unlikely to lead to (TAKE-POINTS ME), using

If a search node is unlikely to lead to a solution, prune it.

Second, it could apply the general rule

If you search for something and look at the most likely candidates without finding it, its probability of being there is small.

Thus an incomplete, unsuccessful heuristic search for

```
(SOME H (ALL FUTURE)
  (DURING H ACT (TAKE-POINTS ME)))
```

could itself serve as evidence for the hypothesis (SMALL (PR-LEAD-TO ACT (TAKE-POINTS ME))).

3.5. Dynamic operationalization

(This section illustrates how information arising from an actual task situation can be used to simplify the problem of operationalizing a generally applicable concept or heuristic. A hypothetical game situation is described, and a line of reasoning a human player might use in that situation is presented. The problem of automatically generating a similar line of reasoning is examined, and some general heuristics useful in solving it are proposed.)

Let's see how a mahps might operationalize "avoid taking points" in the context of a specific game situation such as the following: The order of play is Joe, then ME (the machine), then Ann. Joe has just led the 15th trick of the round with the King of diamonds. The machine has two diamonds (the 10 and the Ace) and knows that the only outstanding diamond is the Jack, and that the only point card still out is the Queen of spades. Clearly it behooves the machine to compare carefully the consequences of playing the 10 or the Ace.

A person in this situation might reason as follows: "I can certainly avoid taking points by playing the 10, since then Joe will take the trick. If I play the Ace and Ann has the Jack, she'll have to play it, so I won't take points. But if she doesn't have the Jack and she does have the Queen of spades, she might stick me with it. Otherwise I'm safe."

How might a similar line of reasoning be generated automatically? To start with, the mahps expands the definition of AVOID-TAKING-POINTS to

```
(SOME ACT (ACTIONS-OF ME)
  (SMALL (PR-LEAD-TO ACT
    (TAKE-POINTS ME))))
```

This generates the goal of evaluating (SMALL (PR-LEAD-TO ACT (TAKE-POINTS ME))) for each ACT in (ACTIONS-OF ME). In the situation at hand, (ACTIONS-OF ME) consists of (PLAY ME 10D) and (PLAY ME AD). It is easy to evaluate (PR-LEAD-TO (PLAY ME 10D) (TAKE-POINTS ME)) as zero by applying fault tree analysis (3.3): (NEQ 10D (WINNING-CARD)) follows from (LOWER 10D KD) and the definition

```
(DE WINNING-CARD NIL
  (HIGHEST (SET-OF C (CARDS-IN-POT)
    (= (SUIT-OF C)
      (SUIT-LED)))))
```

It is trickier to evaluate (PR-LEAD-TO (PLAY ME AD) (TAKE-POINTS ME)). Expanding the definition of PR-LEAD-TO and substituting (CURRENT trick) for FUTURE yields

```
(PR (DURING (CURRENT TRICK)
  (TAKE-POINTS ME))
  (DURING PRESENT (PLAY ME AD)))
```

```
= (PR (DURING (SCENARIO (PLAY JOE KD)
  (PLAY ME AD)
  (PLAY ANN
    (SOME C
      (LEGAL-CARDS ANN))))
  (TAKE-TRICK (TRICK-WINNER)))
  (TAKE-POINTS ME)))
```

Consider the sub-expression (TAKE-POINTS ME) in this expression. (TAKE-POINTS ME) occurs iff (AND (= (TRICK-WINNER) ME) (TRICK-HAS-POINTS)). (PLAY ME AD) implies (= (TRICK-WINNER) ME) since no higher card can be played, so (TAKE-POINTS ME) reduces to (TRICK-HAS-POINTS) in the case at hand. Furthermore, since (CARDS-IN-TRICK) = (SET KD AD (CARD-OF ANN)), (TRICK-HAS-POINTS) reduces to (PLAY ANN (SOME C CARDS (HAS-POINTS C))). This expression instantiates the definition

```
(DE UNLOAD-POINTS NIL (P)
  (PLAY P
    (SOME C CARDS
      (AND (HAS-POINTS C)
        (NEQ C (WINNING-CARD))))))
```

since (= (TRICK-WINNER) ME) precludes (PLAY ANN (WINNING-CARD)). Thus (TAKE-POINTS ME) reduces to (UNLOAD-POINTS ANN).

Now consider the sub-expression (SCENARIO ...). Evaluating it requires determining (LEGAL-CARDS ANN). Since (NOT (LEADING ANN)), (LEGAL-CARDS ANN) simplifies to

```
(SET-OF C (HAND-OF ANN)
  (OR (VOID ANN DIAMONDS)
    (= (SUIT-OF C) DIAMONDS)))
```

Since JD is the only outstanding diamond, (= (SUIT-OF C) DIAMONDS) reduces to (= C JD) and (VOID ANN DIAMONDS) reduces to (NOT (HOLDS ANN JD)). Thus (LEGAL-CARDS ANN) reduces to (IF (HOLDS ANN JD) (SET JD) (HAND-OF ANN)).

The resulting expression

```
(PR
  (DURING (SCENARIO (PLAY JOE KD)
    (PLAY ME AD)
    (PLAY ANN
      (IF (HOLDS ANN JD)
        JD
        (SOME C (HAND-OF ANN))))
    (TAKE-TRICK ME))
    (UNLOAD-POINTS ANN)))
```

can be simplified to

```
(PR (DURING (PLAY ANN
  (IF (HOLDS ANN JD)
    JD
    (SOME C (HAND-OF ANN))))
  (UNLOAD-POINTS ANN)))
```

since (UNLOAD-POINTS ANN) can only occur as part of (PLAY ANN ...). The IF construct can be factored out of this expression, yielding

```
(IF (HOLDS ANN JD)
  (PR (DURING (PLAY ANN JD)
    (UNLOAD-POINTS ANN)))
  (PR (DURING (PLAY ANN
    (SOME C (HAND-OF ANN)))
    (UNLOAD-POINTS ANN)))))
```

The falsity of

```
(DURING (PLAY ANN JD)
  (UNLOAD-POINTS ANN))
```

follows from (= (POINTS JD) -10). We still have the problem of evaluating

```
(PR (DURING (PLAY ANN (SOME C (HAND-OF ANN)))
  (UNLOAD-POINTS ANN)))
```

This expression can be simplified by worst-case analysis using the rule

```
(PR (DURING (f ... (SOME x S) ...) bad-event))~
  (PR (EXISTS x S (DURING (f... x ...) bad-event)))
```

Applying this rule yields

```
(PR (EXISTS C (HAND-OF ANN)
  (DURING (PLAY ANN C)
    (UNLOAD-POINTS ANN))))
```

which reduces to

```
(PR (EXISTS C (HAND-OF ANN)
  (HAS-POINTS C)))
```

```
= (PR (HOLDS-POINTS ANN))
```

In other words, worst-case analysis assumes

```
(GIVEN (HOLDS-POINTS ANN)
  (DURING (CURRENT TRICK)
    (UNLOAD-POINTS ANN)))
```

Combining the above simplifications, we have (PR-LEAD-TO (PLAY ME AD) (TAKE-POINTS ME)) = (IF (HOLDS ANN JD) 0 percent (PR (HOLDS-POINTS ANN))).

Of course, the rules of the game prevent the mahps from inspecting Ann's hand in order to evaluate (HOLDS ANN JD) and (HOLDS-POINTS ANN). This illustrates a particular type of operationalization problem: evaluation of well-defined expressions based on unobservable data. There are various ways to cope with this problem. Sometimes an expression can be evaluated indirectly by deduction from data that are observable. For instance, if the mahps remembers that Ann has shown void in diamonds, it can infer that (HOLDS ANN JD) is

false. Another way to cope with unobservable data is to treat it probabilistically. This approach is discussed in some detail in [4].

3.6. Mapping expressions to methods

This section has described several methods which might be used to operationalize the expression (AVOID ME (TAKE-POINTS ME)). An important issue in operationalization is mapping expressions to methods. This mapping involves not only finding a method appropriate to a given expression but also determining the correspondence between the elements of the expression and the elements of the method: the mahps must figure out how to satisfy the informational requirements of the method given the information contained in the expression or otherwise inferable. The examples in this section involved a variety of techniques for doing this mapping. These include goal-specific means-end rules (3.1), expanding expressions in terms of lower-level concepts (3.2), instantiation of general rules (3.3), partial-matching expressions against problem descriptions associated with known methods (3.4), viewing concepts as related concepts (3.4), and introduction of simplifying approximations (3.5).

Intuitively, the further removed the method from the expression, the weaker the method, since it exploits less specific knowledge about the expression. Thus fault tree analysis can be expected to produce a better (more efficient) operationalization of "avoid taking points" than heuristic search, since the fault tree solution operates at the level of scenarios, while the heuristic search solution operates one conceptual step lower at the level of sequences.

4. Current and Future Work

The body of previous research related to this work is too large to cite here, but is discussed elsewhere [2, 4].

The first author's dissertation (in progress) focuses on operationalization. We have operationalized several examples of Hearts advice by hand and are formulating general rules to account for the derivations. A working program generates such derivations interactively. At each step it determines which rules are applicable and applies the one ranked highest according to some simple criteria. In case of a tie, the user selects from among the best-ranked candidates. In addition we have defined a procedural for heuristic search and are working towards a formalization of fault tree analysis. The representation used in this paper has been implemented, including most of the meta-functions described in Section 2.1. An interpreter capable of executing this representation has been implemented in LISP.

Acknowledgments

This work has benefited from discussions with our colleagues at the Rand Corporation and Carnegie-Mellon University. Stan Rosenschein was especially helpful in clarifying many knowledge representation issues.

References

- [1] Hayes-Roth, F. "The Role of Partial and Best Matches in Knowledge Systems." In D. A. Waterman and F. Hayes-Roth (eds), *Pattern-Directed Inference Systems*. Academic Press: New York, 1978, pp. 557-574.
- [2] Hayes-Roth, F., Klahr, P., Burge, J., and Mostow, D.J. "Machine Methods for Acquiring, Learning, and Applying Knowledge." Technical Report P-6241, The Rand Corporation, Santa Monica, Calif., 1978.
- [3] McCarthy, J., et al. *LISP 1.5 Programmer's Manual*. MIT Press: Cambridge, Mass., 1963.
- [4] Mostow, D.J., and Hayes-Roth, F. "Machine-Aided Heuristic Programming: A Paradigm for Knowledge Engineering." Technical Report N-1007, The Rand Corporation, Santa Monica, Calif., 1979.
- [5] Newell, A. "Heuristic Programming: Ill-Structured Problems." In J. Aronofsky (ed.), *Progress in Operations Research*. Wiley: New York, 1969, pp. 363-414.
- [6] Newell, A., and Simon, H.A. *Human Problem Solving*. Prentice-Hall: Englewood Cliffs, N. J., 1972.
- [7] Sacerdoti, E.D. "Planning in a Hierarchy of Abstraction Spaces." *Artificial Intelligence*. 5, (1974) 115-135.