KNOWLEDGE REPRESENTATION AND PROBLEM SOLVING IN MDS.

by

Chitoor V. Srinivasan

Jan. 1981

DCS-TR-89

Department of Computer Science
Hill Center, Busch Campus,
Rutgers University, New Brunswick, N.J. 08903

Knowledge Representation and Problem Solving in MDS.

by

Chitoor. V. Srinivasan

January 1981

TABLE OF CONTENTS

KNOWLEDGE REPRESENTATION AND PROBLEM SOLVING IN MDS.

by

Chitoor V. Srinivasan

JANUARY, 1981

ABSTRACT

This work presents a new approach for using a first order theory to generate procedures for solving goal satisfaction problems without using general theorem proving. The core of the problem solving system has three basic components: an inferencing mechanism based on residues, a control structure for "means-end" analysis that uses natural deduction, and a generalization scheme that is based on the structure of statements in the domain theory itself.

The work represents a beginning in the development of knowledge based systems that can generate their own problem solving programs, evolve with experience and adapt to a changing domain theory.

1. Introduction

We propose here a new approach for using a first order theory to generate procedures for solving goal satisfaction problems without using general theorem proving. The theory, which is a part of the "domain knowledge" used by a computer, is represented in the computer using the knowledge representation system called the Meta Description System (MDS) [Srinivasan,1973,'77] [Sridharan,1976,'77,'78] [Schmidt,1978]. The core of the problem solving system has three basic components: an inferencing mechanism based on residues [Srinivasan,1973,'77,'80], a control structure for doing "means-end" analysis using natural deduction, and a generalization scheme that is based on the structure of statements in the domain theory itself. These make it possible to use the declarative theory to construct procedures and solve problems using methods that are simpler than general theorem proving. Also, the organization is very sympathetic to accepting externally specified (declaratively stated) focus rules for guiding search. The basic concepts are introduced in sections 3 and 4 in the context of a simple stylized example.

The complexity and size of problems that we can reasonably expect to solve using the new techniques presented here are yet to be determined. We are now involved in this process. We have chosen a simple experimental domain, one where one can progressively increase the complexity of the problems in order to test the capabilities of the system and identify the nature of the expanding search problems. The kinds of issues we are focussing on are briefly discussed in section 5.

There is now a version of an MDS based system called AIMDS [Sridharan,1978] that is being used to model legal reasoning [McCarty,1980] and to model psychological experiments in interpersonal communication [Schmidt,1978]. This system has only limited capabilities to do reasoning and planning using residues. The focus in the work reported here is on reasoning with residues: we show how residues are used to reason about problem solving and how in the context of residues the "means-end" analysis process itself can be viewed as a problem solving task in which "domain knowledge" is used to guide search.

The significance of this work and the principal concepts that distinguish this work from other approaches to knowledge representation and use are briefly discussed in the next section.

2. General Comments on Knowledge Representation.

One may view domain knowledge as consisting of three kinds of information: (1). facts that are true in a given state of the domain (usually represented in a computer in some form in a database); (2). a set of statements (constraints) that are known to hold true in all the states of the domain, and that also limit the kinds of transitions that can occur between the states in the domain (we shall refer to this set of statements as the domain theory); and (3). methods (control structures and procedures) for using the facts and the domain theory to state and solve problems in the domain.

Research on "knowledge based systems" has generally focussed on two areas: one is the formulation of "knowledge representation schemes" [Srinivasan,1973,'77] [Woods,1975] [Newell,1973] [Minsky,1975] [Brachman,1978] [Bobrow,1977] [Davis,1980] [Schank,1977] [Stallman,1977] [Roberts,1977]; and the other is implementation of "expert systems" based on the schemes, systems that work reasonably well in given domains on given classes of problems [Bobrow,1977] [Sridharan,1978] [Shortliff,1976] [Weiss,1978] [Carbonell,1978] [Stallman,1977] [Cullingford,1977] [Charniak,1975]

[Goldstein,1977]. The phrase "knowledge engineering" [Feigenbaum,1977] has been used to refer to this technology. The representation schemes that have so far been proposed and used fall into two distinct groups:

[A]. Program Based Representations: In these schemes the specification of domain knowledge requires one to also specify the methods for using the knowledge. These methods are usually flexibly specified using "procedural attachment" [Bobrow,1977] to components of "frames" [Minsky,1975] [Bobrow,1977] or "scripts" [Schank,1977] or "structural inheritance nets" [Brachman,1978]. In systems like these it is not possible to clearly identify the "domain theory" (item 2 above). This effectively robs such a system of the ability to reflect about its own domain theory, and use the theory to respond to unanticipated situations. To realize such systems it is necessary for systems to have the ability to develop their own programs based on a statement of a domain theory. The work proposed here is a beginning in this direction.

[B]. Theory Based Representations: In these schemes the domain theory is explicitly represented in a system as a declarative theory. The meta theory of systems of this kind will usually specify the "control structures" (methods) that are needed to use the declarative theory to state and solve problems in the domain. Thus in a first order logic system its meta-theory gives us the theorem proving methods. MDS partly uses theory based representations. The meta-theory of MDS-like systems is discussed in [Srinivasan,1980].

A promising variant of the first order system for knowledge representation is the "Logic Programming" scheme of Kowalski [Kowalski,1979]. By using "Horn clauses" as basic representational devices, Kowalski proposes a new approach to knowledge representation that beautifully combines both procedural and declarative aspects of stating and using "domain knowledge". However, in this scheme the essential aspects of search control are not directly addressed. Also, a user has the total responsibility to make sure that a specification is complete (i.e. all the information needed for theorem proving is present in the specification). Even for small domains it is difficult to ascertain this.

MDS proposes another approach to representing and using a declarative first order theory. A significant difference between the MDS approach and the "logic programming" approach is that in MDS the domain theory is interpreted over a model space of the domain in a three valued logical system of T (true), ? (unknown) and F (false), and the interpretation process is used as the basis for problem solving, as is shown below. Also, the system brings to bear a separate body of domain knowledge, called focus rules (which are again declarative statements) to reduce its search effort in "means-end" analysis. The information provided by the focus rules is quite distinct from the information provided by the domain theory itself. The focus rules have at times the flavor of meta-rules, at times they are second (and higher order) statements in the domain theory and at times they refer to the attributes and states of the problem solving control structure itself. As may be noted, the system has intrinsically a greater tolerance to incomplete information (missing facts in the database or missing steps in a procedure) than systems that are based on general theorem proving.

Only the principal representational concepts and problem solving schemes are introduced in this paper. The nature and properties of the focus rules themselves and the organization of search in means-end analysis are discussed elsewhere [Srinivasan,1981].

3. The Meta Description System.

3.1. The Basic Concepts.

We shall use a simple domain to illustrate the concepts. Let us call the domain, the domain of simple physical reality (SPR). The domain is simply the following: It contains PEOPLE, PLACES and OBJECTS. Every PERSON and OBJECT is constrained to be exactly at one PLACE. PEOPLE can hold OBJECTS and other PEOPLE. However, if a PERSON holds something then the PERSON and the thing being held should both be at the same PLACE. Let us use the following relation names to describe the states of SPR:

VOCABULARY: {is-at, location-of, holding, heldby,
PERSON, PLACE, OBJECT}

Thus one may say that in a given state, S, of SPR the following relation instances are true:

S = {(John is-at Princeton) (Bicycle is-at Washington)
(PERSON John) (OBJECT Bicycle) (PLACE Princeton)
(PLACE Washington)} (1)

(We shall hereafter omit mentioning the unary relations, whenever it is clear from the context what unary relations are true in a given state). The concept of a domain in MDS may then be defined as follows:

DOMAIN: A domain is a set of states. Each state is represented in a computer by a finite set of relation instances. The arguments of the relation instances that occur in the representation of a state can range only over the constants that exist in the domain (the name of a state itself cannot occur as an argument in a relation instance). We shall say that each such set of relation instances is a snap-shot of the state that it represents. The representation of a domain, hereafter called the domain, is a set of snap-shots.

A snap-shot can be incomplete in the sense that it may not store all the relation instances that are true (false) in a given state. Thus in SPR one may have a state like,

S1 = {(John is-at Princeton) (John holding Bicycle)}. (2)

Here the location of the bicycle is not given in S1. But MDS can use its interpretation process to infer the location of the bicycle from the information in S1 and the domain theory. The inferencing ability of MDS is much more limited than the general inferencing ability of a theorem prover in first order logic, i.e. there can be relation instances that do not occur in a snap-shot and for which the truth values cannot be inferred by MDS using its interpretation processes, even though it is logically deducible from the available information. We shall show below in section 4 how the limited inferencing ability of MDS can nevertheless be used to solve interesting classes of goal satisfaction problems in a domain.

We shall say that a given relation instance is True (T) if it is in the snap-shot, False (F or NIL) if its negation is in the snap-shot, and is Unknown (?) otherwise. Thus the truth value of "(John holding Bicycle)" is unknown in the snap-shot S above. So also the truth value of "(Bicycle is-at Princeton)" is unknown in S1. No relation instance can be both true and false in a snap-shot.

DOMAIN THEORY: This will consist of two parts, a static theory and a dynamic theory. The static theory is simply a set of constraints stated in a first order language. The constraints are interpreted over the snap-shots using the standard first order interpretation [*] in the three valued logical system: T > ? > F, ¬T = F, ¬? = ?, and ¬F = T. Let PHI be the semantic function that evaluates the constraints over given snap-shots: For a constraint, C, and a snap-shot S, PHI(C S) = T, ? or F.

The static theory will characterize the snap-shots that belong to the domain in the following sense: A snap-shot, S, is in the domain if and only if Phi(C S) = T for all the constraints, C, in the theory. If there is a constraint C such that Phi(C S) = ? then it is unknown whether S is in the domain or not. If Phi(C S) = F for some constraint in the theory then the snap-shot is definitely not in the theory. In the case of SPR the static theory will simply consist of the following two constraints:

```
((x) (PERSON(x) V OBJECT(x)) =>
     ((EXISTS unique y)PLACE(y) & (x is-at y))),                    (3)

((x) (y) (PERSON(x) & (PERSON(y) V OBJECT(y)) &
       (x holding y)) => ((EXISTS z) (PLACE(z) &
                          (x is-at z) & (y is-at z)))).             (4)
```

We shall write constraints of this kind in the following form, using the classes like PERSON, PLACE and OBJECT over which the variables range as a part of the quantifiers themselves:

```
((PERSON | OBJECT x)
  (EXISTS unique PLACE y) (x is-at y))                              (5)

((PERSON x) (PERSON | OBJECT y) (x holding y) =>
        ((EXISTS PLACE z) (x is-at z) (y is-at z)))                 (6)
```

where we use "|" within quantifiers to denote a disjunction of classes and assume implicit conjunction between parenthesized expressions. It should be noted that none of the constraints in a static theory may refer to the snap-shots themselves as variables. The static theory is a declarative first order theory of the domain.

Besides the domain constraints like the ones given above the static theory in MDS will also contain definitions of classes of objects that exist in a domain. Thus in SPR one will have the classes PERSON, PLACE and OBJECT. Classes are defined in MDS by specifying the relational forms in which they can occur. This is done as follows:

```
{(PERSON is-at PLACE) (OBJECT is-at PLACE) (PERSON held-by (PERSON))
 (PLACE location-of (PERSON | OBJECT)) (OBJECT held-by PERSON)
  (PERSON holding (OBJECT | PERSON))}                               (7)
```

Relational forms like the ones in (7) above are called dimensions. The dimension "(PERSON is-at PLACE)" is interpreted as

---

* ((ALL x)P(x)) <=> (P(c1) & P(c2) &... & P(cn)), and
((EXISTS x)P(x) <=> (P(c1) V P(c2) V ... V P(cn)),
where c1, c2, ..., cn are the constants that occur in a snap-shot.

Every PERSON can be in at most only one PLACE.

Notice that what (5) says is stronger than what the dimensions say: (5) says that there has to be exactly one PLACE for every OBJECT and PERSON, whereas the dimensions say only that the number of such PLACEs cannot be greater than 1. The dimension "(PLACE location-of (PEOPLE | OBJECTS))" is interpreted as

"every PLACE can have an arbitrary number of PERSONs and OBJECTs such that the PLACE is their location".

These interpretations of dimensions are made a part of the semantic function PHI and the instantiation function, ADD (see discussion below). Thus, if PHI((John is-at Princeton) S) is true then since a PERSON can be only at one PLACE as per dimension given above, MDS will return the truth value false for all PHI((John is-at x) S), where x is not "Princeton". Similarly, if one added the relation instance "(John is-at Washington)" to the snap-shot, in view of the above dimensions, the system will automatically remove the instance "(John is-at Princeton)" from S (for more details on the theory of dimensions see [Srinivasan,1980]).

The second part of the domain theory is the dynamic theory. The dynamic theory will specify constraints on possible state transitions that may occur in a domain. Thus in SPR one may have a constraint on how a PERSON may go from one PLACE to another. If our domain contained VEHICLES, and we had notions of PEOPLE and OBJECTS being in VEHICLEs, then one may have a constraint which in effect specifies that in order to go from one PLACE to another a PERSON has to use a VEHICLE and in order to use a VEHICLE he/she has to get into it. Constraints like this are stated as transformation rules in MDS. The transformation rule for the above dynamic constraint might read as follows:

```
[[(PERSON x) (PLACE y) (PLACE z)                    <<the dimension
  (x is-at y) (GOAL (x is-at z))]                     of the rule>>
 [(EXISTS VEHICLE v) (ASSERT(x is-in v))            <<its body>>
                     (ASSERT(v is-at z))
                     (ASSERT¬(x is-in v))]]                          (8)
```

The dimension of the rule is a declaration of what the rule does. In the rule above it says that the rule pertains to any PERSON, x, and any two PLACEs y and z such that x is initially at y and x has the GOAL of going to z (if y = z then the GOAL will be satisfied without executing the body of the rule). Whenever a problem of this kind arises MDS will invoke the above rule by "matching" the problem with the rule dimension and execute the body of the invoked rule to achieve the goal.

It may be noted that the SPR with the above augmentation will contain besides (5) and (6) also the following additional constraint:

```
((PERSON | OBJECT x) (VEHICLE y) (x is-in y) =>
     ((EXISTS PLACE z) (x is-at z) (y is-at z)))                     (9)
```

This static constraint will allow x to get into v in rule (8) above only if v and x are at the same PLACE. The language in which rules like (8) are stated is the procedural language that MDS uses to describe its solutions to the problems that it solves. In all our discussions in this report we shall assume for simplicity, that our domain SPR does not have any dynamic constraints, since our primary focus here is on methods of obtaining procedures by using a declarative theory (See

[Srinivasan,1973] for a discussion of how MDS will use rules like (8) above to solve goal satisfaction problems).

3.2. Types of Problems.

UPDATING PROBLEMS: Here the problem is to assert into a current snap-shot, S, a set of relation instances, A: (ASSERT A). A can contain both positive and negative relation instances. The objective here is to add the relation instances in A into S and check whether the new S' so obtained is in the domain or not. If there are no transformation rules associated with the relation instances in A, then the above requirement can be stated as follows using the semantic function:

Let C[A] be the set of all constraints in the static theory of a domain whose truth values in S are likely to be affected by the addition of the relation instances in A into S. Let (ADD A) be the function that adds the relation instances in A to the snap-shot S producing the new snap-shot S', making sure that for every relation instance, p, in S' the negation of p, namely ¬p, is not in S'. Then one can state the updating problem as one of finding the truth value of (ASSERT A), where (ASSERT A) is defined as,

$$\text{(ASSERT A)} \iff \neg(\text{PHI}(\text{C[A]}\ (\text{ADD A})) = \text{F}). \qquad (10)$$

Here C[A] is interpreted as the conjunction of all the constraints in C[A]. Thus, (ASSERT A) will succeed only if (ADD A) does not contradict any of the constraints in the static theory of the domain. For example, if one asserted "(John holding Bicycle)" into the snap-shot S given in (1) the assert will not succeed because it will violate the domain constraint given in (6).

GOAL SATISFACTION PROBLEMS: These problems are stated in the form "(GOAL <logical expression>)". In the simplest case, if the logical expression is a conjunction of relation instances occurring in a set, say A, then the solution to the GOAL problem might simply be (ASSERT A). But if (ASSERT A) fails then one may still be able to achieve the goal by performing appropriate secondary changes to the new snap-shot. Thus for example, in S above (see (1)) it is possible to achieve the GOAL, "(John holding Bicycle)" by either bringing John to where the Bicycle is or by bringing the Bicycle to where John is, or by taking both John and Bicycle to a common new place. In this case a solution to the (GOAL (John holding Bicycle)) problem is, for example,

```
[(ASSERT (John is-at Washington))
        (ASSERT (John holding Bicycle))].          (11)
```

In the above solution the second ASSERT will be attempted only if the first one succeeds. In general a solution to a GOAL problem will consist of a sequence of assertions. The final snap-shot that is obtained by executing the sequence of assertions will satisfy the goal condition.

We shall use the following basic forms to state procedures in MDS: (<EXP> (ASSERT A)) and (<EXP> (GOAL p)), where p is a logical expression. Here the ASSERT (or GOAL) will be attempted only if the <EXP> (which can be any first order statement or any procedure) is true (succeeds) in the current snap-shot. These forms give the basic sequential control that is needed to state procedures. For a discussion of the semantics of this procedural language see [Srinivasan,1980].

Let us now review how MDS solves a goal satisfaction problem in the SPR domain using the declarative domain theory of SPR.

4. The Goal Satisfaction Problem: An Example.

4.1. The Logic of Residues.

Suppose we had the snap-shot, S, shown in (1) above and the following problem was encountered:

(GOAL (John holding Bicycle)). (12)

The dimension of this problem is first determined. The dimension is

[(PERSON j) (OBJECT b) (GOAL (j holding b))].
[j <- John][b <- Bicycle]. (13)

A dimension of this kind is constructed by first identifying the classes to which the constants in the given problem belong and creating for each such constant a dummy variable of the appropriate kind. In the above case, this causes (PERSON j) and (OBJECT b) to be created. The problem itself is then expressed in terms of the created variables. The particular constants supplied with the problem are then assigned as the bindings for the created variables.

The direct solution is first attempted by executing,

(ASSERT (j holding b)).

This will cause (ADD (John holding Bicycle)) to be performed on S producing the new snap-shot,

S' = {(John is-at Princeton)
(Bicycle is-at Washington) (John holding Bicycle)}. (14)

and causing constraint (6) to be evaluated. This constraint will evaluate to false. Thus the ASSERT will fail. Let us take a brief look at how this evaluation takes place, and how the system extracts from constraint (6) the reasons for the failure of this assertion.

To understand the evaluation process, let us first expand the constraint (6) to its equivalent propositional expression in S', namely the following:

((John holding Bicycle) =>
((John is-at Princeton) & (Bicycle is-at Princeton) V
(John is-at Washington) & (Bicycle is-at Washington))).

This proposition is equivalent to the proposition Q below:

Q = (¬(John holding Bicycle) V
((John is-at Princeton) & (Bicycle is-at Princeton) V
(John is-at Washington) & (Bicycle is-at Washington))) (15)

Since the semantic function operates in a three valued logical system, while evaluating Q over S', in general, one can partition the proposition Q into three

subexpressions, one called the true part of Q, TP[Q], the second called the unknown part of Q, ?P[Q], and finally the third called the false part of Q, FP[Q]. Roughly, these various parts are obtained as follows (see [Srinivasan,1973,'77] for precise definitions): The true part is obtained by simply deleting from Q all the relation instances that are either false or unknown. Similarly, the unknown part is obtained by deleting all the relation instances that are either true or false, and the false part is obtained by deleting all the relation instances that are either true or unknown. In our case above, PHI(Q S') = F, and we have the following partition of Q:

TP[Q] = [(John is-at Princeton) V (Bicycle is-at Washington)],
?P[Q] = NULL, and
FP[Q] = [¬(John holding Bicycle) V (John is-at Washington) V
(Bicycle is-at Princeton)]. (16)

For PHI(Q S') = F the false part of Q is called the false residue of Q in S', FR[Q]. These partitions are actually represented in MDS in terms of the dummy variables that were used in the problem dimension. This will cause the false residue FR[Q] = FP[Q] to appear as follows:

FR[Q] = [¬(j holding b) V (j is-at Washington) V
(b is-at Princeton)] (17)

FR[Q] is the subexpression that caused Q to evaluate to false in S'. It is now noted that the subexpression

[(j is-at Washington) V (b is-at Princeton)] (18)

in the above false residue is obtained from the parent expression

[(EXISTS z)(j is-at z) & (b is-at z)], (19)

and (19) is viewed as the generalization of the expression in (18). It is also noted that (18) is obtained from (19) as the false part of the expansion of (19) produced by the bindings shown below:

FP([(EXISTS z)(j is-at z) & (b is-at z)]
[z <- (Washington Princeton)]). (20)

It may be noted that (20) is obtained directly as part of the process of getting (15) from (6) and evaluating (15). In (20) the expression "[z <- (Washington Princeton)]" is called the bindings of z in S', denoted by the functional notation "bindings(z S')". Now, since "¬(j holding b)" is the negation of the goal condition this is removed from the false residue in (17), resulting in the following final FR[Q] and its generalization GFR[Q],

FR[Q] = FP([(EXISTS z)(j is-at z)(b is-at z)] [bindings(z S')]),
bindings(z S') = (Washington Princeton), (21)

GFR[Q] = [(EXISTS z)(j is-at z) & (b is-at z)]. (22)

The conjunction of the TP (true part) and the ?P (unknown part) in (16) is called the Not-False-Part of Q, NFP[Q], which in this case is the same as TP[Q],

NFP[Q] = TP([(EXISTS z)(j is-at z)(b is-at z)] [bindings(z S')])
bindings(z S') = (Washington Princeton). (23)

There are several advantages to storing the residues and other parts of a constraint in this manner: First of all, the size of the expressions does not increase as the number of constants in a snap-shot itself increases (if the residues are stored as propositional expressions they will get enormous even for reasonably moderate sized snap-shots). Secondly, it facilitates the evaluation of the residues and other subexpressions in different environments (for different bindings) as the snap-shot itself is changed, as in (24) below.

The false residue (21) and the not-false-part (23) have the following property (this is obtained from the meta-theory of MDS):

> There will exist a snap-shot, S'', in which the goal is true iff in that snap-shot S'' both NFP[Q] and FR[Q] are true for the new bindings, namely bindings(z S'').

This may be written as follows:

```
[(EXISTS S'')
 ((S'' is-in SPR) & (PHI((John holding Bicycle) S'') = T))
      <=> ((PHI(NFP[Q](S'') S'') = T) &
                (PHI(FR[Q](S'') S'') = T))],                    (24)
```

where NFP[Q](S'') and FR[Q](S'') indicate that these expressions are evaluated for the bindings in S''. Property (24) provides the basis for doing means-end analysis to achieve our goal (for a more general statement of this property see [Srinivasan,1973]). Since NFP[Q] is already the conjunction of the true and unknown parts, the expression for which one might seek to change the truth value (from false to true) is the false residue FR[Q]. This is done in such a manner that in the new S'' (24) will be true (i.e. while making FR[Q] true one should take care that NFP[Q] is not made false).

4.2. Doing Means-end Analysis Using Residues.

Let us first consider the false residue at its lowest level of generalization (i.e. the most specific false residue), namely expression (18) above (this is obtained by evaluating (21) in S' for the indicated bindings). Let [Q1] denote this expression in (18). The system's objective now is to make [Q1] true. The following new goal is set up to reach this objective and achieve the initial goal given in (12):

```
[[¬[Q1] (GOAL [Q1])] (ASSERT (j holding b))]].                  (25)
```

In the above problem statement "(GOAL [Q1])" is to be attempted only if ¬[Q1] is true in the current snap-shot. Notice that since [Q1] is the false residue in S', ¬[Q1] is true in S'. To achieve this new goal [Q1] is analyzed using the method of Natural Deduction, as follows.

First [Q1] is represented as a sequent (see [Wang,1960] [Kanger,1963]), whose left hand side contains the current snap-shot (the symbol "<SS>" below represents this current snap-shot). The left hand side of a sequent is always interpreted as a conjunction, and its right side is interpreted as a disjunction. The form of goal [Q1] as a sequent is shown below:

```
<SS> -> (John is-at Washington), (Bicycle is-at Princeton);     (26)
```

The system's objective now is to modify the snap-shot <S5> in a manner that will cause the above sequent to be "blocked" (a sequent is said to be blocked if the left side of the sequent contains a relation instance that is identical to one on its right side). This blocking is to be done making sure that NFP[Q] (see (23)) will be true in the resulting new snap-shot. We have here a very simple case (we discuss a slightly more complicated case in section 4.3 below). To block the above sequent MDS can now attempt to introduce into the snap-shot either one (or both) of the following two relation instances:

(John is-at Washington), or
(Bicycle is-at Princeton). (27)

It has thus three choices. At this point it has no criteria for choosing one of these three in preference to others. The system does not even know that if both the above relation instances are introduced into the snap-shot it will again encounter a contradiction with constraint (6) (we shall see in section 4.3 how this problem can be avoided by using the generalization GFR[Q]). To make choices in situations like this MDS uses devices called the <u>focus rules</u>. A focus rule may contain meta statements of the kind,

"choose the one that causes minimum change in the snap-shot",

or it may contain domain dependent advise of the form

"choose to move PERSONs in preference to OBJECTs".

A preference of this kind is expressed in MDS as a focus rule, as follows:

((PERSON is-at) > (OBJECT is-at)), (28)

where "(PERSON is-at)" and "(OBJECT is-at)" refer to the PLACEs where they are located. These are called <u>anchors</u> in MDS. Focus rules like these may also have conditions associated with them (stated in a first order language). MDS will then use the specified focus rule only if the associated condition is true. Let us assume that in the above case rule (28) is used. This will result in

(ASSERT (j is-at Washington)) (29)

being executed first, and then

(ASSERT (j holding b)), (30)

thereby achieving the goal and yielding the procedure

[[¬[Q1] (ASSERT (j is-at Washington))] (ASSERT (j holding b))]. (31)

A generalization of this procedure is now constructed as follows.

It is first noted that "Washington" is not one of the constants that was given as part of problem (12) (it is a dependent variable). An attempt is therefore made to determine the dependency between "Washington" and the input constants "John" and "Bicycle". This will result in the recognition that "Washington" is equal to "(Bicycle is-at)" (we use the convention that "(x r)" is equal to $\{y \mid (x\ r\ y)\}$). Using this piece of information (29) is now replaced by the following,

(ASSERT (j is-at (b is-at)). (32)

Condition ¬[Q1] is now generalized producing the following safe (but not very interesting) generalization:

<Q2> = [((EXISTS PLACE x) (EXISTS PLACE y)
¬(x = y) & ¬((j is-at x) V (b is-at y))]. (33)

Using this generalization the following procedure is obtained:

[[(PERSON j) (OBJECT b) (GOAL (j holding b)] <<dimension>>
[[<Q2> (ASSERT (j is-at (b is-at)))]
(ASSERT (j holding b))]. <<body>> (34)

Clearly, this is not quite the procedure that one wants. But this is the simplest and quickest generalization of (31) that one can obtain. We shall see in the next section an important variation of the above analysis, one that produces a more interesting solution not by generalizing a procedure after it has been constructed, but by choosing the right generalization of the false residue for means-end analysis.

Procedures like (34) may be used as follows: When the system encounters a new problem which matches with the dimension of rule (34) it can invoke this rule directly and execute its body to achieve the goal. In doing this MDS will bind the variables j and b to new constants and evaluate <Q2> in the environment of the new snap-shot. If <Q2> happened to be true in this new environment then the first ASSERT will automatically get executed, otherwise the control will skip directly to the second ASSERT. If the first or second ASSERT now happened to fail for some reason in the new environment, then the new false residue obtained from this failure will be used to set up new subgoals, and the results obtained from this will be used to update the procedure.

the use of generalizations of residues for means-end analysis is discussed in the next section.

4.3. Using Generalization of Residues to do Means-End Analysis.

Suppose for some reason John could not be moved from Princeton to Washington. Then the system can try to move the Bicycle from Washington to Princeton. Suppose even this was not possible. Then the system can try to move both of them simultaneously. This also will not succeed in the above case. We will show here how by choosing to use the generalization GFR[Q] of [Q1] (see (22)) the system can avoid considering this third alternative and also produce a more general solution than (34). The following sequence of analysis occurs if GFR[Q] is considered instead of (26) above. Initially the problem statement and the sequent are,

[[¬(GFR[Q]) (GOAL GFR[Q])] (ASSERT (j holding b)],

and

<SS> -> [(EXISTS z) (j is-at z) (b is-at z)]; (35)

Using Existential generalization the following new sequent is obtained from (35):

<SS> -> GFR[Q], ((John is-at u) & (Bicycle is-at u)); (36)

where u is a new variable. Now, by eliminating the "&" in (36) one gets,

```
<SS> -> GFR[Q], (John is-at u);
<SS> -> GFR[Q], (Bicycle is-at u);                                    (37)
```

At this point an attempt is made to find a way of blocking both the sequents in (37), by finding a possible binding for the variable u from among the PLACEs that exist in the snap-shot and asserting the relation instances in (37) into the snap-shot for the chosen bindings. Here again focus rules are used to make choices. Whatever the choices are, in this case John and the Bicycle will always be moved to a common place. Also, if places other than Washington and Princeton exist in the snap-shot the system can try to move the Bicycle and John to a common new place and then make John take hold of the Bicycle.

If the focus rule (28) is used then the solution will be:

```
[[(PERSON j) (OBJECT b) (GOAL (j holding b))]   <<dimension>>
 [[¬(GFR[Q])] (ASSERT (j is-at (b is-at)))]
          (ASSERT (j holding b))]]               <<body>>          (39)
```

Clearly then, the choice of the level of generalization chosen for the false residue that is used for setting up the subgoal can have a profound influence on the search efficiency and the nature of the resultant solution.

It may be noted that if no focus rules are available then MDS can try all the possible choices that are available and develop a focus rule of its own for future use based on the success/failure information obtained through its trials. We shall not here discuss the details of this process.

It should also be noted that the natural deduction system that we have used in means-end analysis above is not strictly a theorem proving system. In a theorem proving system one cannot hypothesize new relation instances and introduce them within a proof (as we did above to "block" the sequents), unless of course such relation instances themselves are derivable from given premises. In the above analysis we are using the theorem proving framework of the natural deduction system to hypothesize the candidate relation instances that might be appropriate to reach a goal. Since the hypothesized relation instances are accepted by the snap-shot only if they do not falsify any of the constraints in the domain theory, soundness is guaranteed (one may say that we use theorem proving with model evaluation to justify soundness). Completeness is however not guaranteed since we do not know whether the domain theory itself is complete or not (we assume that it is consistent).

As the reader may note MDS has here learned a significant amount of new knowledge about the SPR domain. Using its interpretation scheme and a primitive form of natural deduction it learned how it can use one of the domain constraints to solve a problem. It seems one could use a method like this to create a system that can learn from its problem solving experience in a domain. The system might function in the following mode:

> Every time the system is presented with a problem, P, it will construct a procedure to solve P using the relevant constraints in its domain theory and appropriate generalizations of the residues obtained from the constraints. This procedure is then added to its library of procedures.. As new problems are presented the system will first attempt to use its library of known procedures to solve them. Every time it solves a new problem using a procedure that is not in the library, or a modification of a

> procedure that is in the library, it will add this procedure to its library or update its existing procedure. As more problems are solved the system's library will progressively expand and get enriched. The expanded library will give the system the potential to perform better in the domain, thus exhibiting a capability to evolve with experience.

This sounds very much like what STRIPS [Fikes,1971] attempted to do. The differences are the following: In STRIPS the possible ADD/DELETE-lists are given a priori to the system. MDS infers the necessary ADD/DELETE lists by analyzing the residues. STRIPS uses general theorem proving to check consistency. MDS uses model evaluation. STRIPS cannot tolerate missing information in its database. MDS can hypothesize the missing information. STRIPS could not use the structure of its axioms to guide itself while generalizing the procedures it constructed. MDS uses the structure of its domain constraints themselves to choose the generalizations.

There is another significant difference between the procedures that we construct in MDS and the procedures constructed in STRIPS and other program construction experiments [Manna,1977] [Barstow,1979]. The semantics of MDS procedures is dependent on the domain theory in whose context they are executed. For example, if it were possible for John at Princeton to directly take hold of his Bicycle in Washington in a strange world in which he lived (i.e. if constraint (6) did not exist in the domain theory), then (ASSERT (John holding Bicycle)) will succeed in this world. Thus, unlike an assignment statement in a programming language, the ASSERT statement does not have any intrinsic meaning. It acquires a meaning only in the context of a domain theory. The work that is done to select constraints and either to check them or find ways of satisfying them is itself not a part of the procedure. In conventional programming, one does not have this kind of separation between a domain theory and the steps within a program. In other words, in conventional programming the program semantics is independent of the domain in which the program works. All the information that is needed for a program to work "correctly" within a domain should be represented within the program itself. This is not so in MDS.

Because of this, the procedures that MDS constructs are much simpler objects than procedures in a conventional programming language and we are able to use simpler methods to construct the procedures. A procedure in MDS merely specifies the major tasks that are to be performed and the order in which they should be done, and leaves the details to the constraint satisfaction system. It should nevertheless be noted that procedure construction and generalization processes in MDS are in general not as simple as those presented above. Complexities can arise in various ways. The nature of some of the complexities are discussed in the next section.

5. The Nature of Complexities that arise in Reasoning with Residues: Need for experimentation.

The kinds of complications that can arise in the above procedure construction process are briefly outlined in this section. The list below is not intended to be comprehensive, but is representative of the kinds of problems on which we are focussing our experimentation.

[A]. The sequents are usually more complicated and in general one needs extensions to snap-shots.

Generally, for a relation instance, p, C[p] will contain more than one constraint (C[p] is the set of all constraints affected by the relation instance p). If (ASSERT p) fails then one or more of the constraints in C[p] might have false residues. Also, in this case the not-false-part will be more complicated. Initially we will have the following two sequents:

```
<SS> -> GFR[C[p]];
<SS> -> GNFP[C[p]];                                        (40)
```

where GNFP is a generalization of the not-false-part of C[p] (notice that in the example above C[p] = Q and GFNP[Q] = GFR[Q]). The sizes of the expressions on the right hand side of these sequents will clearly add to the complexity of the means-end analysis process. There is another reason for complication to arise.

In the example above the constraint (6) was directly evaluated on the snap-shot. This may not in general be possible. Consider, for example, the snap-shot S1 shown in (2). For this snap-shot constraint (6) evaluates to ? (unknown) and the unknown part of it will then become its unknown residue, namely

```
?R[<constraint (6)>](S1) = ?P[<constraint (6)>](S1)
                         = (Bicycle is-at Princeton)       (41)
```

This unknown residue may be interpreted as follows:

```
"(Bicycle is-at Princeton) should be true
 if S1 is to belong to the SPR domain".
```

Unknown residues like this may be viewed as hypotheses under which a given snap-shot like S1 can remain within a domain (i.e. satisfy all the domain constraints). In simple cases like the one above, or in general if a unknown residue is simply a conjunction of relation instances, then one may directly assert the relation instances into a snap-shot and make the snap-shot contain all the relation instances that are implied by the hypotheses. But there are several reasons why one might not want to (or be able to) do this: There are situations where it is necessary to keep such unknown residues explicitly as hypotheses and not assert them into a snap-shot.

For example, if John in S1 keeps changing his place often, then it might be easier to do the updates if the following generalization of the above unknown residue,

```
(Bicycle is-at (John is-at)),                              (42)
```

is retained as a hypothesis associated with S1, instead of every time asserting it into the snap-shot.

Another reason for retaining hypotheses is that it may not be possible to find a unique set of relation instances that will guarantee that a given snap-shot remains within a domain. This kind of a situation will arise if the unknown residues contain disjunctions. In cases like this there will be many ways for keeping a snap-shot within a domain and it may be desirable to retain all the options.

The set of all the hypotheses associated with a snap-shot is called the extension of the snap-shot. Each element in the extension will be either a propositional expression or an expression of the form

?R[<constraint> <bindings>] (43)

(namely the unknown residue of a <constraint> for given <bindings>). Let EXT[SS] denote the set of all such hypotheses. Then, in general, the sequents used for means-end analysis will be of the form [*]:

<SS>, EXT[<SS>] -> GFR[C[p]];
<SS>, EXT[<SS>] -> GNFP[C[p]]; (44)

The use of such extensions make it possible to do means-end analysis in the context of incompletely specified snap-shots. But it also complicates the means-end analysis process considerably. To do means-end analysis using sequents of the form (44) the analysis process should be carefully guided. One approach to this is to specify "abstractions" [Sacerdoti,1977] and do means-end analysis as a series of progressive refinements of solutions in abstraction spaces. Another approach is through the use of "focus rules". A third approach is to classify the constants in a domain into a hierarchy of classes (like, for example the "kind-of" and "type-of" hierarchies that are often used in knowledge representation systems) and view solutions as refinements over such hierarchies.

In MDS we have yet another option: Residues in MDS are associated with anchors, which are units of the form "(x r)" where x is an object and r is a relation name. The unknown residue at anchor (x r) may be interpreted in MDS as the condition that must me made true in order to make a given set of relations of the form "(x r y)" true in the snap-shot at the anchor (x r). One may form a hiererchy in the means-end analysis process by imposing an order on the forms of relations which the system attempts to make true: thus for example, one may say that

[(TRANSPORTATION mode) (PASSENGR location) (VEHICLE location)]

is the preferred order in which the system should attempt to satisfy the constraints in a means-end analysis process in a transportation problem. All the above four methods are feasible in MDS [Srinivasan,1981].

[B]. There will usually be several choices for generalizations of false residue and not-true-part.

In the example in section 4 we had only two cases: The most specific residue, namely, the residue in (18), and its only generalization, namely GFR[Q] in (22). The kinds of generalizations that are available for a residue will in general depend on the structure of the constraint that produced the residue. One can, of course, always start with the most specific residue and proceed upwards in ones analysis. But in practice it will be necessary to specify some criteria for the choice of generalizations in order to keep the problem solving search within manageable limits. Such criteria are likely to be highly domain dependent. Further experimentation is necessary to understand the problems here.

[C]. Procedures can get complex. It is necessary to have methods to analyze and simplify procedures.

---

* In general <SS> and EXT[<SS>] in (44) will be subsets of the actual snap-shot and its extension, respectively.

The case by case generation of procedures, illustrated in section 4, is likely to produce procedures that are unweildy and unnecessarily complex. When a procedure gets large and complex one should have ways for breaking it apart into component parts and expressing the larger procedure in terms of its component parts.

Also, it is necessary to recognize situations, where by using iterations (or recursion) one can simplify the statement of a procedure. It is possible to identify such situations using the notion of the "dimension" of a procedure and the problem that it is attempting to solve, and notions of dimensions of residual problems, namely the problems that remain unsolved at a given stage of procedure execution. We shall not discuss here the details of this analysis.

These are the principal kinds of concerns that motivate our current experimentation. We have chosen SPR itself as our experimental domain. This domain can be made progressively more complex to consider transportation problems of increasing complexity. For example, one can introduce various modes of transportation, associate costs with them and ask for cost minimization, classify PLACEs into a hierarchy (at the lowest level of the hierarchy a PLACE can be the location of only one entity and at higher levels of the hierarchy one PLACE can contain other PLACEs as a part of it; thus, a PLACE at a higher level of the hierarchy can be the location of more than one entity), introduce "reachability criteria" for something to "hold" something else, etc. We intend to use transportation problems of increasing complexity (and variable constraints) to test the practical feasibility of above kind of procedure construction. We want to find out whether the kinds of processes that MDS uses for procedure construction can be used as the basis to create new kinds of "knowledge representation systems", systems that can evolve with experience, show certain capabilities to adapt to a changing domain theory and themselves take over some of the programming tasks that are currently associated with the creation of such systems.

5.1. Current Work and future Plans.

We are proceeding with our implementation in stages. The first stage contains a simplified version of MDS, consisting of the following subsystems (these are essential for our experiment) an associative relational database (to store snap-shots), the semantic function, PHI, the instantiation function, ADD, a simplified system for constructing extensions of snap-shots using residues, system for doing "means-end analysis" using the natural deduction scheme, and system for procedure formation and interpretation. This stage can solve problems of the kind discussed in section 4 above.

The major implementations that have to be further developed are those of the last three, namely residue extraction, MDS-procedure construction and procedure interpretation, and use of "focus rules" to guide search. The experiments we plan to conduct may be broadly classified into three groups: (a). those that increase the number of options available to the system in reaching a "goal" (this may be achieved by increasing the number of available modes of transportation, the alternate routes that may be taken and the number of items that needs to be transported from one PLACE to another.), (b). those that increase the levels of generalizations that are feasible for the residues (this may be done by classifying PLACEs, OBJECTs, PERSONs, VEHICLEs and relations themselves into hierarchies) and (c). those that increase the complexity of the problems themselves (this may be done by requiring that the solutions satisfy given overall optimality criteria, like minimizing cost or time). We have a broad spectrum of options available to us to design the experiments. The

particular experiments that we choose will depend much on the performance that we are able to achieve in our implementation.

We are not attempting to build a system that can be used in a variety of different domains. We have chosen one domain, the domain of Simple Physical Reality, to test the complexity of problems for which the above method can realistically be used.

There are several practical limitations that we can already anticipate. These pertain to the complexity of constraints (in terms of the number of quantifiers they have) that we can efficiently evaluate, the size of the snap-shots (in terms of the number of constants they have) over which constraint evaluation can be done and the extent of interaction among the constraints (in terms of the size of C[p] for given p and the way different C[p]'s intersect), that can be reasonably analyzed. The constraint evaluation and constraint satisfaction problems are intrinsically hard (NP-hard) problems. We do not expect to produce an implementation that solves this problem in general on existing machines. Our objective is to find out whether problems of this kind can be efficiently solved in given domains by properly organizing the "domain knowledge" in the domain.

The advantage that MDS organization gives over other proposed ones is that it enables one to use basically less powerful techniques (interpretation and restricted forms of theorem proving) to perform tasks that previously required one either to use general theorem proving or explicitly writing the procedures that performed the tasks. Also, organization of "knowledge" in MDS allows one to define various kinds of domain hierarchies and domain specific rules (focus rules) to guide search. Most importantly, MDS makes it possible to function effectively in the context of incomplete information in a well defined and logically sound manner. All of this is done within a framework in which a system can by itself use declaratively stated "domain knowledge" to construct procedures in the domain. These are the promising features of the MDS-organization that make it worthwhile to experiment with it. Their practical utility remains yet to be established.

All the computing is done on the LCSR (Laboratory for Computer Science Research) DEC-20 computer.

6. Bibliography.

[Barstow,1979] Barstow, David R., Knowledge-Based Program Construction, Programming Languages Series, (Ed). Thomas E. Cheatham, The computer Science Library, North Holland, New York, 1979.

[Bobrow,1977] Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H. and Winograd, T., "GUS, A Frame-Driven Dialog System", AI 8, 2, pp. 155-173, 1977.

[Brachman,1978] Brachman, R. J., "A Structural Paradigm for Representing Knowledge", Ph.D. thesis, Harvard University, 1977, also BBN technical report 3605, Bolt, Beranek and Newman Inc., Cambridge, Mass., 1978.

[Carbonell,1978] Carbonell, J. G. Jr., "POLITICS: Automated Ideological Reasoning", Cognitive Science 2, pp. 27-51, 1978.

[Charniak,1975] Charniak, E., "Organization and Inference in a Frame-like System of Common Knowledge", in Theoretical Issues in Natural Language Processing,

R. C. Schank and B. L. Nash-Webber (eds.), Mathematical Social Sciences Board, Cambridge, Ma., pp. 46-55, 1975.

[Cullingford,1977] Cullingford, R., Script Application: Computer Understanding of Newspaper Stories, Ph.D. thesis, Research Report 116, Yale University, 1977.

[Davis,1980] Davis, R. and Lenat, D., Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill, New York, in press, 1980.

[Feigenbaum,1977] Feigenbaum, E. A., "The Art of Artificial Intelligence: I. Themes and case studies of knowledge engineering", IJCAI-5, pp. 1014-1029, 1977.

[Fikes,1971] Fikes, R.E., and Nilsson, N.J. [1971] STRIPS: a new approach to the application of theorem proving to problem solving, AI JI, 2 (3/4), 189:208, 1971

[Goldstein,1977] Goldstein, I. P. and Roberts, R. B., "Nudge, a Knowledge-Based Scheduling Program", IJCAI-5, MIT, Cambridge, Mass., pp. 257-263, 1977.

[Kanger,1963] Kanger, Stig, "A simplified proof method for elementary logic", in Computer Programming and Formal Systems, Braffort & Hirschberg (Eds.), North Holland, Amsterdam, 1963.

[Kowalski,1979] Kowalski, R. A., Logic for Problem Solving, North Holland, Amsterdam, 1979.

[Manna,1977] Manna, Z. & Waldinger, R., "Synthesis: dreams => programs", SRI International, Technical Note 156, Nov. 1977.

[McCarty,1980] McCarty, Thorne L. & Sridharan, N. S., "The Representation of an Evolving System of Legal Concepts: I. Logical Templates", LCSR Technical Report, LRP-TR-7, Laboratory for Computer Science Research, Rutgers University, N.J. 08903, April 1980.

[McCarty,1980] McCarty, Thorne L., "Some Notes on the MAP formalism of TAXMAN II, with Applications to Eisner v. Macomber," LRP-TR-6, Laboratory for Computer Science Research, Rutgers University, New Brunswick, N.J. 08903, May 1980.

[Minsky,1975] Minsky, M., "A Framework for Representing Knowledge", in The Psychology of Computer Vision, P. H. Winston (ed.), McGraw-Hill, New York, pp. 211-277, 1975.

[Newell,1973] Newell, A., "Production Systems: Models of Control Structures", in Visual Information Processing, W. G. Chase (ed.), Academic Press, New York, 1973.

[Roberts,1977] Roberts, R. B. and Goldstein, I. P., FRL Users' Manual, MIT AI Laboratory, Memo No. 408, Cambridge, Mass., 1977.

[Sacerdoti,1977] Sacerdoti, E. D., A Structure for Plans and Behavior, Elsevier, New York, 1977.

[Schank,1977] Schank, R. C. and Abelson, R. P., Scripts, Goals, Plans, and Understanding, Lawrence Erlbaum, Hillsdale, N.J., 1977.

[Schmidt,1978] Schmidt, C. F., Sridharan, N. S. and Goodson, J. L., "The Plan

Recognition Problem: An Intersection of Psychology and Artificial Intelligence", AI 11, 1 and 2, pp. 45-83, 1978.

[Shortliffe,1976] Shortliffe, E. H., MYCIN: Computer-Based Medical Consultation, American Elsevier, New York, 1976.

[Sridharan,1976] Sridharan, N. S., "The Frame and Focus Problems in AI: discussion in relation to BELIEVER system", Proc. Conf. Artificial Intelligence and Simulation of Behavior, 322-333, 1976.

[Sridharan,1977] Sridharan, N. S., "Representation of Actions that have side-effects", IJCAI-77, 265-266, 1977.

[Sridharan,1978] Sridharan, N. S., AIMDS User Manual - Version 2, CBM-TR-89, Department of Computer Science, Rutgers University, New Brunswick, N.J., 1978.

[Srinivasan,1973] Srinivasan, C. V., "The Architecture of Coherent Information Systems: A General Problem Solving System", IJCAI-3, pp. 618-628, (also in IEEE Trans. on Computers, vol. C-25, 4, pp. 390-402, 1976), 1973.

[Srinivasan,1977] Srinivasan, C. V., "The Meta-Description System: A System to Generate Intelligent Information Systems, Part I, The Model Space", Tech. Report SOSAP-20A, Dept. of Computer Science, Rutgers University, New Brunswick, N.J., 1977.

[Srinivasan,1980] Srinivasan, C. V., "Knowledge Based Learning Systems, An Introduction to the Meta Theory", DCS-TR-89, Department of Computer Science, Rutgers University, New Brunswick, N.J. 08903, May 1980.

[Srinivasan,1980] Srinivasan, C. V., "Knowledge Based Learning, Logical Foundations", DCS-TR-89, Department of Computer Science, Rutgers University, New Brunswick, N.J. 08903, May 1980.

[Srinivasan,1981] Srinivasan, C. V., "The Use of Focus-rules to Guide Search in Means-end Analysis in MDS", In preparation.

[Stallman,1977] Stallman, R. M. and Sussman, G. J., "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", AI 9, 2, pp. 135-196, 1977.

[Wang,1960] Wang, Hao, "Toward Mechanical Mathematics", IBM Journal of Research and Development, pp 2-22, Jan 1960.

[Woods,1975] Woods, W. A., "What's in a Link: Foundations for Semantic Networks", in Representation and Understanding, D. G. Bobrow and A. Collins (eds.), Academic Press, New York, pp. 35-82, 1975.