# Expert Systems in the Micro-electronic Age

EDITED BY DONALD MICHIE

FOR EDINBURGH

UNIVERSITY

PRESS

1979

Solving Mechanics Problems using Meta-level Inference 50
A. Bundy, L. Byrd, G. Luger, C. Mellish and M. Palmer
*Department of Artificial Intelligence, University of Edinburgh*

# SOLVING MECHANICS PROBLEMS USING META-LEVEL INFERENCE

Alan Bundy, Lawrence Byrd, George Luger, Chris Mellish & Martha Palmer

In this paper we shall describe a program (MECHO), written in Prolog [14], which solves a wide range of mechanics problems from statements in both predicate calculus and English. MECHO uses the technique of meta-level inference to control search in natural language understanding, common sense inference, model formation and algebraic manipulation. We argue that this is a powerful technique for controlling search while retaining the modularity of a declarative knowledge representation.

*Keywords:* Natural Language, Mathematical Reasoning, Search Control, Meta-level Inference, Predicate Calculus, Mechanics.

This work was supported by SRC grant number B/RG 94493 and an SRC research studentship for Chris Mellish. This paper was also delivered as an invited talk to the 6th IJCAI held in Tokyo in August 1979.

## 1. INTRODUCTION

The work described in this paper addresses the question of how it is possible to get a formal representation of a problem from an English statement, and how it is then possible to use this representation in order to solve the problem. Our purpose in studying natural language understanding in conjunction with problem solving is to bring together the constraints of what formal representation can actually be obtained with the question of what knowledge is required in order to solve a wide range of problems in a semantically rich domain. We believe that these issues cannot be sensibly tackled in isolation. In practical terms we have had the benefits of an increased awareness of common problems in both areas and a realisation that some of our techniques are applicable to both the control of inference and the control of parsing.

Early work on solving mathematical problems stated in natural language was done by Bobrow (STUDENT – [1]) and Charniak (CARPS – [5]). However, the rudimentary parsing and simple semantic structures used by Bobrow and Charniak are inadequate for any but the easiest problems. Our intention has been to build on advances in natural language processing (eg [18] ) in order to study parsing and problem solving in a domain which requires sophisticated knowledge about the world. The domain we have been working in is that of mechanics problems, which deal with idealised objects such as smooth planes, light inextensible strings, frictionless pulleys etc. The idealised nature of this domain made it feasible to consider building an expert inferential system which would be able to cope with a wide range of problems. To date, our program has tackled problems in the areas of: pulley problems, statics problems, motion on smooth complex paths and motion under constant acceleration. Our intention is to continue ex-

panding this in order to force generality into our solutions. In recent years a lot of similar work has been in progress on Physics-type domains such as ours. (eg [13], [7], [15], [11] ). We have been concerned to adopt methods developed by these workers into MECHO, and to solve mechanics problems tackled by them.

## 2. DESCRIPTION OF THE PROGRAM

The block diagram (fig 1) gives a very general overview of the structure of the MECHO program. Each block represents a closely related collection of Prolog clauses (procedures), the arrows between blocks indicate invocation/communication links. (For practical reasons MECHO is split into three separate modules, but this is irrelevant to the overall structure). The accompanying diagram (fig 2) tries to capture the changes in representation and the various types of knowledge required during the execution of the program. The following discussion will elaborate on these.

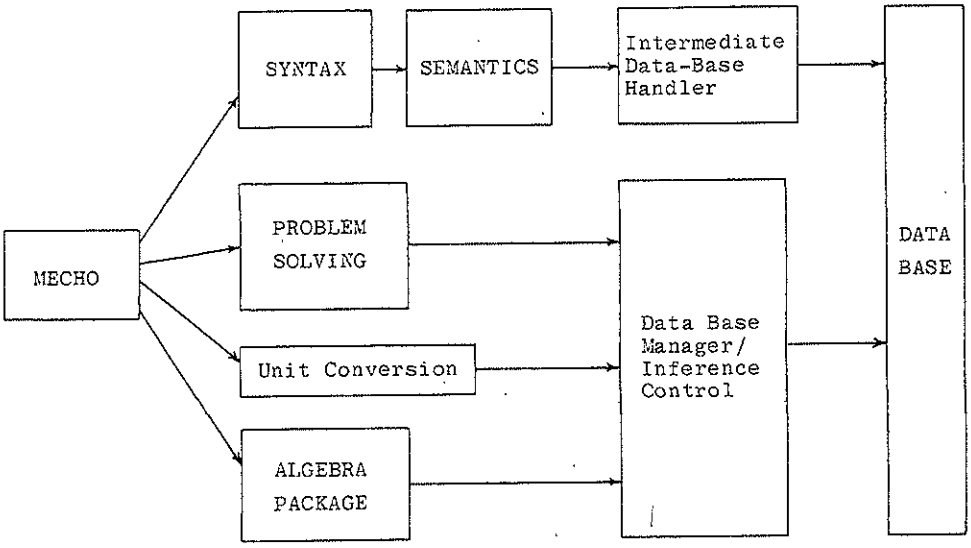


Figure 1. Program block structure

Input to the program is in the form of English text. An example taken from the area of pulley problems would be:

> 'Two particles of mass b and c are connected by a light string passing over a smooth pulley. Find the acceleration of the particle of mass b.'
> (Taken from [10].) (1)

The purpose of the natural language module is to produce a set of predicate calculus assertions which will enable the problem solver to solve the problem. This objective of producing a symbolic representation of the 'meaning' of the problem statement has been used by us as a vehicle for exploring syntax-semantics interaction. The syntactic parser calls semantic routines as soon as possible in order to interpret fragments of text and quickly reject inappropriate syntactic choices.

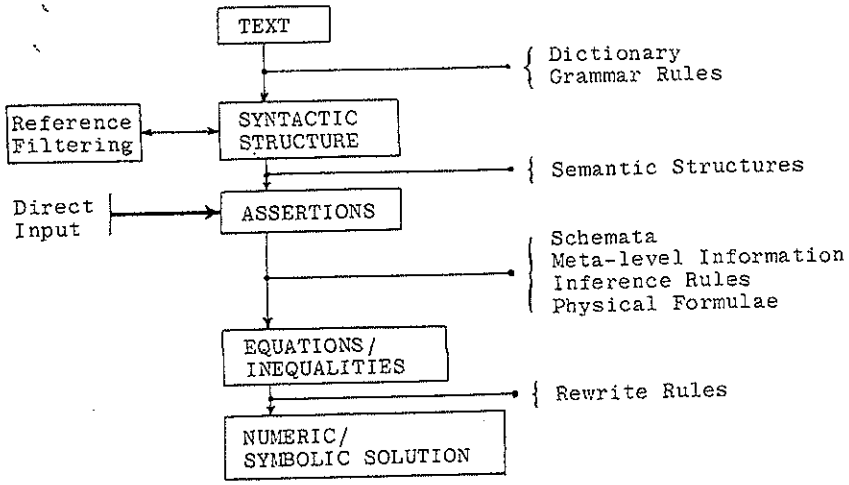


Figure 2. Representation structures

The work of the syntax routines divides into clause syntax and phrase syntax.

The purpose of syntactic analysis at the clause level is to establish clause boundaries and, within each clause, to prepare the ground for the semantic analysis of the main verb. Clause analysis thus involves identifying the start of new phrases, assigning syntactic roles to the phrases and performing phrase analysis to interpret the internal structure of the phrases. The internal phrase analysis typically returns simply a referent (and some typing information) to the higher levels. This means that preliminary reference evaluation is carried out locally, with the information conveyed by a phrase being captured in assertions produced as 'side effects' by semantic routines called during the analysis. These semantic routines are responsible for interpreting what it means for a given object to have a certain property, and indeed for checking whether or not it can have the property. Domain-specific information concerning typing, idealisation and object-property possibilities is used to answer these questions. Failure of the semantics indicates that the parse is invalid. The following (simple) example shows the kind of meta-level structures used in this process.*

meaning (light, Object, mass (Object, zero)).

type_constraint (light, physobj).

The meaning predicate states that the meaning of applying the property light to an object, is that the object has a mass of zero. type_constraint asserts that being a physical object is a necessary condition to having the property light.

* The notation used here, and in following examples, follows the Prolog convention that names starting with an upper-case letter are logical variables which are purely local to the structure (Prolog clause). Atoms, which are in lower case, and compound terms all stand for themselves. Rules are of the form 'P ← Q & R', meaning 'if Q and R then P'. Most examples have undergone slight cosmetic alteration.

It is interesting to note that the declarative assertions which give the meaning of a phrase can be specified independently of how they will be used. Meta-level information concerning the state of the analysis is used to determine whether they are used to add new information or to test necessary constraints on previous information. (This is basically the 'given'/'new' distinction discussed in [8]).

One of the aspects of natural language understanding that has interested us especially is the way in which criteria of semantic well-formedness can be used to resolve cases of ambiguity in reference evaluation. Our program incorporates a full deductive mechanism, as opposed to semantic markers, to capture the global semantic constraints that arise during the interpretation. Reference evaluation proceeds continuously during the combined syntactic and semantic analysis with semantic information being used to filter sets of possible candidates for referents. The method used to achieve this in a general way is basically that of Waltz filtering [16]. As can be seen, the referent returned by the phrase syntax is likely to be incompletely specified and for this reason all interaction between the semantics and the data-base is handled by an intermediate data-base handler which implements the inference system and reference filtering system over these referents.

The syntactic structure built by the clause syntax specifies a main verb, and positions such as 'logical subject' and 'logical object' are filled by referents. (We do not construct a complete parse tree as such). From this structure a set of assertions is generated by invoking semantic routines. The semantic analysis of the verb maps the main verb onto a base verb and establishes a mapping between the syntactic roles of the clause and the deep roles associated with the base verb. As a result the referents are fitted into conceptual slots in a way similar to conventional 'caseframe' analysis. The base verb then specifies the assertions (in terms of the referents) which follow from this mapping. Base verbs differ from case frames in that while they attempt to generalise collections of related verbs, they are not defined in terms of universal primitive roles or slots.

Given a satisfactory parse of a sentence, which produces a set of consistent assertions and disambiguates the referents, we are then able to produce a set of assertions (by instantiating out the referents) about the objects in the problem. These are supplied to the problem solving module. As an example, the assertions produced for the above problem statement (1) would be:

```
isa(period, period 1)
isa(particle,p1)
isa(particle,p2)
isa(string,s1)
isa(pulley,pull)
end(s1,end1,right)
end(s1,end2,left)
midpt(s1,midpt1)
fixed_contact(end1,p1,period1)
fixed_contact(end2,p2,period1)
fixed_contact(midpt1,pull,period1)
```

```
mass(p1,mass1,period1)
mass(p1,mass2,period1)
mass(s1,zero,period1)
coeff(pull,zero)
accel(p1,a1,270,period1)
accel(p2,a2,90,period1)
measure(mass1,b)
measure(mass2,c)
sought(a1)
given(mass1)
given(mass2)                                    (2)
```

In addition the following schema is cued:

```
cue pullsys_stan(sys1,pull,s1,p1,p2,period1)
```

The cueing of schemata is necessary to provide extra information, defaults etc. which are not given explicitly but are 'house rules' in this domain (e.g. that the pulley in a pulley system has negligible weight). We cue schemata, fairly simplistically, by recognising key words and certain object configurations. For example the following structure asserts that a pulley-system schema can be cued if objects can be found which satisfy the ideal-type constraints and have certain relationships between each other.

```
sysinfo( pullsys,
      [Pull,Str,P1,P2],
      [pulley,string,solid,solid],
      [ supports(Pull,Str),
        attached(Str,P1),
        attached(Str,P2) .
      ]).
```

The effect of this cue would be to invoke a schema such as:

```
schema( pullsys,
      [Pull,Str,P1,P2], Time,
      [ constaccel(P1,Time),
        constaccel(P2,Time),
        cue stringsys(Str,[Lpart,Rpart]),
        (tension(Lpart,T1,Time)
              ← coeff(Pull,zero) &
                tension(Rpart,T,Time) )
      ],
      [ coeff(Pull,zero),
        mass(Pull,zero,Time) ]).
```

This schema asserts that in a standard pulley problem the objects undergo constant acceleration, the tension in both parts of the string are equal if there is no

friction (only one rule shown), and that the friction and mass of the pulley default to zero if not otherwise given. (This example has been somewhat simplified).

The predicate calculus notation can be used to input problems directly to the problem solver – and in fact research on the problem solver has resulted in it being able to tackle a wider range of problems than the natural language module can currently handle. The representational principles behind these assertions view the objects of Newtonian Mechanics in terms of simple zero and one dimensional objects (points and lines) which are typed and have properties and relations defined over them. For example particles, pulleys, spatial points, moments of time are all types of POINT while rods, strings, paths (trajectories), and periods of time are types of LINE. Physical quantities, such as length, velocity, force etc., form the other main branch of our type hierarchy (see [4]).

The work of the Problem Solver divides into types of task. There is the overall strategic task of deciding what to do, how to solve the problem by producing equations which solve for unknown quantities (including intermediate unknowns introduced during the solution). On the other hand there is the tactical task of combining the input assertions with general facts and inference rules in order to prove required goals. We shall discuss each of these in turn.

Our overall strategy is a general goal directed algorithm for equation extraction developed from a study by David Marples of student engineers [12]. For instance, suppose a1, the acceleration of particle p1, is the (only) sought unknown. (Here we continue the example started above (1) (2) ). Resolution of forces about p1 will be chosen to solve for a1 and this produces the equation:

$$-\text{mass1.g} + \text{tension1} = \text{mass1.a1} \qquad (3)$$

All possible force contributions on p1 are examined and since p1 is attached to the end of the string this results in the string being considered. tension1 was formerly unknown but the function properties of the predicate 'tension' enable it to be created (see later) to allow the equation to be formed. We have to introduce tension1 as an unknown because it is not possible to solve for a1 without doing so. The next step is to solve for tension1 which is a force and involves the string s1. Again resolution of forces is selected – p1, pull, p2 being objects on the string that are possible candidates for resolving about. p1 has been previously used and only p2 can be used without introducing unknowns. The result is the equation:

$$\text{mass2.g} - \text{tension1} = \text{mass2.a1} \qquad (4)$$

These two equations can be solved to produce a solution for a1.

The input assertions provide meta-level information about whether certain quantities are sought or given. The Marples Algorithm works by traversing the list of sought unknowns in a fixed order: the (quantity) type of each unknown being used to provide a shortlist of formulae that could solve for it, and the definition of the quantity (ie the assertion which introduced it) being used to find the physical objects, times and angles involved. Notice that there is a distinction made between 'formulae', which are composed of variables over quantities (eg

'F = M * A'), and 'equations' which are instantiations of formulae (eg (3) and (4) above). In the Marples algorithm we are reasoning about the properties of formulae in order to successfully produce appropriate equations.

Before the application of a formula to produce an equation, there is a stage of qualitative analysis where general facts about the problem are used to decide applicability. Our interest here is in exploring the selective use of meta-level reasoning to guide the equation extraction process. As well as deciding applicability we have to prepare a situation within which to apply a formula. This may involve, for example, collecting together all the objects connected to a particle if we wish to resolve forces about it. General independence criteria (eg 'You can't resolve forces about the same object in linearly dependent directions') are also used to eliminate redundant equations.

The following are (simplified) examples of the meta-level structures used during the above examples:

```
kind(a1,accel,
        relaccel(p1,earth,a1,270,period1)).

relates(accel,
        [resolve,constaccel–N,relaccel]).

prepare(resolve,relaccel(P,earth,A,Dir,Time),
                situation(P,Set,Dir,Time))
      ←isa(particle,P) &
        findall(X, sameplace(X,P,Time), Set).

isform(resolve,situation(Obj,Set,Dir,Time),
                F = M * A )
      ←mass(Obj, M ,Time) &
        accel(Obj,A,Dir,Time) &
        sumforces(Obj,Set,Dir,Time, F).
```

The kind predicate asserts that a1 is a quantity of type accel defined in the given relaccel assertion. relates states that all the formulae whose names are given in the list, contain variables of type accel and therefore can be used to solve acceleration. prepare gives the criteria for constructing the situation within which to resolve forces, and the isform predicate defines the equation by defining the meaning of its component variables.

The equation extraction algorithm is two pass in that it first tries to produce a solution which does not introduce new unknowns before allowing the introduction of extra (intermediate) unknowns which are added to the unknowns list and have to be eventually solved for. Notice that the quantities manipulated are purely symbolic; they can be introduced by the creation mechanism (see later) without their values being known at this stage (e.g. when the first equation (3) was formed in the above example, the quantity tension1 was introduced without the program knowing, or trying to find, its actual value). As can be seen, it is the Marples algorithm which will eventually produce an equation which solves for tension1.

The data-base stores all the facts supplied by the English statement, but to bridge the gap between the explicit information derived from the problem statement and that needed to solve the problem the program requires a general knowledge of mechanics which is formalised in a set of inference rules. An example of such (object-level) inference rules would be:

```
relaccel(P1,P2,zero,Dir,Period)
        constrelvel(P1,P2,Period).

constrelvel(P1,P2,Period)
        fixed_contact(P1,P2,Period).
```

The first rule says that the relative acceleration between two points of reference is zero if there is a constant relative velocity between them (over a certain period), and the second rule says that two points of reference have a constant relative velocity if they are in contact (again, over a certain period). The inference rules are a set of Horn clauses which have been hand ordered and contain certain typing information to guide selection. The search strategy is depth first, with pruning of semantically meaningless goals, and while this could be improved upon, current performance has not yet necessitated such a step.

An important part of our work has been the investigation of search control mechanisms which will enable effective use of this wealth of implicit knowledge. All requests to retrieve assertions from the data-base, either directly or via inference, are handled by the inference control module. This module uses information from the request along with meta-information and inference rules in an attempt to satisfy the goal. The first step involves normalisation of the goal to remove syntactic sugar or to express it in terms of a smaller set of underlying predicates. This is performed with a one pass rewrite rule set. The resulting goal is then classified according to the instantiation state of its component arguments and the possibility of using function properties and certain other mathematical properties of the predicate (such as reflexivity, symmetry and transitivity). This information is used to select appropriate proving strategies. (A basic strategy of 'unit preference' will always first check the data-base directly).

Our two most important strategies are the use of function properties to prune search and the use of equivalence class type mechanisms to direct it. The representation treats what would normally be considered functions as predicates with extra control information. Being a function means that certain arguments are uniquely determined by certain other arguments. For example, in the predicate 'tension (String,T,Time)' the actual tension T is determined once the String and the Time have been given.

These function properties can be used to prevent useless inference if another (different) value of a function argument is already known (uniqueness property); to create new entities to satisfy a goal if all attempts at inference have failed (existence property); and to automatically eliminate backtracking by disregarding choices made during inference (uniqueness again). Examples of the meta-level structures used by the program in performing the above are:

```
rewrite( accel(P,A,Dir,Time),
         relaccel(P,earth,A,Dir,Time),
         strategy(dbinf) ).

meta( relaccel, 5,
      [P1,P2,A,Dir,Time],
      [pt_of_ref,pt_of_ref,accel,angle,time],
      function([P1,P2,Time] => [A,Dir] ) )
```

The rewrite rule tells us that any accel predicate can be rewritten to a relaccel predicate with the earth as the other point of reference, and that the standard inference strategy is then applicable. The meta predicate specifies the argument types and the function properties of the predicate relaccel.

The second major strategy, which provides an alternative to using the inference rules, is a general similarity class mechanism based on equivalence class ideas. Predicates which are (pseudo-) equivalence relations and would normally produce self-resolving inference rules are defined in terms of this mechanism. A tree is used to represent similarity class membership and the goal (such as 'being in the same place') is proved by establishing equivalence of roots. This can be seen as an alternative (and less explosive) axiomatisation of these predicates. Our extension over traditional uses of this method has been to allow labelled arcs and calculation during the tree traversal. Thus predicates like 'vector separation' and 'relative velocity' which have pseudo-equivalence properties can also use this strategy. Here is an example of a structure used in these cases:

```
rewrite( sameplace(P,Q,Time),
         [ sameclass(P,Q,touch(Time)),
           merge(P,Q,touch(Time))      ],
         strategy(simclass) .    ).
```

This states that the predicate sameplace should use the general sameclass mechanism on the particular tree touch(Time). Also specified is an updating mechanism for adding new sameplace assertions; in this case it would involve merging two separate trees.

These general strategies can be applied to a wide range of predicates and often capture important facts about the domain (eg the fact that an object cannot be in two places at once is a fact about the function properties of 'at(Object,Place, Time)'). The explicit control of new object creation coupled with the goal directed backward reasoning method of the Marples algorithm results in a create/consider-by-need type of behaviour. Restrictions, such as 'don't create' or 'don't infer', can be added to the request for a goal to be proved and this enables the Marples algorithm to be selective over its use of the Inference Control in accordance with its needs at the time.

For some mechanics problems a process of prediction is required to answer questions like 'Will the particle reach the top of the slope if it starts with velocity V ?'. Each question about the motion of a particle on a complex slope unpacks into a series of questions about the behaviour on simple parts of the slope. Some

of these can be answered immediately on the basis of the qualitative shape of the slope, but others involve the solution of inequalities containing unknown quantities. These unknowns are declared as sought and the equation extraction algorithm is called to solve for them. The prediction system is special purpose and built around problems similar to those tackled by De Kleer, i.e. motion problems.

Since the equations produced by the equation extraction algorithm are in terms of symbolic quantities, there is a stage of Unit Conversion where the actual values are substituted and a final unit system is selected – conversion factors being added where appropriate. (Some problems involve a combination of all sorts of different units – feet, yards, miles . . . . . ). The two equations produced above ((3) & (4)) are very simple in that no particular units are involved. The only step will be the substitution of b and c for mass1 and mass2 respectively giving:

$$-b.g + tension1 = b.a1 \quad (5)$$

$$c.g - tension1 = c.a1 \quad (6)$$

The set of simultaneous equations and/or inequalities produced by the problem solving module is passed to the algebra module (PRESS) which will solve them to produce a final answer to the problem. Let us look at how PRESS produces a solution for a1 given (5) and (6). The two equations are solved by isolating tension1 in the second equation (which was intended to solve for tension1), and then using the result as a substitution into the first equation. This final result can then be simplified with a1 being isolated on the left hand side to give the final answer:

$$a1 = g.(c-b) / (c+b) \quad (7)$$

The extension of equation solving techniques to inequalities (there are interesting connections) has enabled us to solve the inequalities produced by the prediction problems, but in addition we have found that the information required to justify the use of certain rewrite rules is often of the form ' only if $X > 0$ ' etc. Solving and proving inequalities is therefore of direct use within the system.

However, PRESS was not developed purely as a service program for MECHO. It was intended as a vehicle to explore ideas about controlling search in mathematical reasoning using meta-level descriptions and strategies [3]. Rather than using exhaustive application over a large set of rewrite rules, it uses the meta-level strategies of isolation, collection and attraction to carefully control application of several different sets of rewrite rules. This selectivity has many advantages: principled methods for guiding search cut down useless work, identical rules may be used in different ways (eg left to right or right to left) in different circumstances without causing problems, and theoretical requirements such as proof of termination of the rewrite rules are made much easier.

When PRESS is used as an equation and inequality solver (ie as a module of MECHO), it classifies the equations (inequalities) to be solved so as to generate guidance information. An exciting area of research that we would like to expand on is that of designing inclusion and ordering criteria to classify algebraic iden-

tities which are produced by a theorem power. This would enable the system to automatically learn new rules. The use of meta-level reasoning to place new rules into a framework where they will be selectively and correctly applied overcomes many of the obvious 'explosion' and 'looping' problems that would occur with haphazard additions to a large rewrite rule set.

## 3. DISCUSSION

Throughout the above discussion of the MECHO program we have constantly emphasised the importance of 'meta-information' in controlling search. This has been applied in the rejection of semantically meaningless parses, the control of inference, the extraction of equations and the guiding of algebraic manipulation.

The theme that has emerged from our work is the benefit to be gained from axiomatizing the meta-level of the domain under investigation and performing inference at this level, producing object level proofs as a side effect. This is the methodology investigated by Pat Hayes in the GOLUX project [9], except that we have developed our meta-level representation for a particular domain rather than adopting general purpose representations based on resolution theorem proving systems.

In [2] we showed how GPS could be viewed in this way. At the object-level the search space can be viewed as an operator/state OR tree in which the states are nodes and the operators are arcs between them. At the meta-level the search space can be viewed as a method/goal AND/OR tree in which the goals are nodes and the methods are arcs between them. A simple depth-first search at the meta-level then induces a highly complex, middle-out search at the object-level.

In order to make clear the distinction we are drawing between meta-level and object-level representations in MECHO, we shall list examples of the descriptions used at each level. When defining a notation it is usual to define the constants, variables, function symbols and predicate symbols of the language; and then to show how terms and formulae can be formed by composing them together with the logical connectives. We shall follow this type of outline in an informal fashion. (To avoid confusion with earlier terminology we shall use the words 'assertion' and 'rule' to replace 'formula').

At the object-level we have the following kinds of primitive:

constants p1, end1, mass2, a1, right, 90, 270, lbs, feet, etc.

variables P1, Str, Period, Accel, F, M, etc.

function symbols +, *, cos, etc.

predicate symbols accel, relaccel, mass, fixed_contact, etc.

These are formed into terms such as 'M * A' and assertions such as 'F = M * A', 'accel(p1,a1,270,period1)' etc. Finally, logical connectives are used to form these assertions into inference rules like:

```
relaccel(P1,P2,zero,Dir,Period)
    ←constrelvel(P1,P2,Period).
```

The only function symbols at the object-level are for straight forward arithmetic and trigonometric functions. This is because we have recorded function properties by making meta-level assertions about object-level predicates.

At the meta-level all these object-level descriptions are meta-constants along with additional meta-constants for schema names, formula names, object types, strategy types etc. As examples of meta-level primitives we have:

constants (Any object level description). physobj, pullsys, resolve, particle, line, length, dbinf etc.

variables Type, Eqn, Goal, Strategy, Expr1, etc. (see below)

function symbols Constructors for lists, sets, bags, etc.

predicate symbols meaning, sysinfo, schema, kind, relates, isform, rewrite, meta etc.

Again these can be formed into meta-terms and meta-assertions and we gave several examples during the program description. What we shall now examine are the meta-rules which are formed from these assertions. (These use the same logical connectives as the object-level rules). We shall take simplified examples from each of the four main areas of our work.

The first example is a rule used by the Natural Language module, which specifies the conditions for a property to be correctly applied to a particular entity.

```
attribute(Property,Entity,State)
  ← type_constraint(Property,Type) &
    isa(Type,Entity) &
    meaning(Property,Entity,Assertion) &
    consistent(Assertion,State).
```

This rule states that a particular Property can be attributed to an Entity in a given State (of the parse), if the Entity satisfies the type_constraint of the Property, and if the meaning of the attribution is consistent with all the other assertions in the current State. (If, for example, the Assertion was 'mass(s1,zero)' then this would involve checking that no other mass was known for s1. It is here that we see one of the key connections with our work on Problem Solving, since this is precisely a matter of 'function properties'!).

The following is an example taken from the Marples Algorithm, and it defines the requirements for an equation to solve for a particular quantity.

```
solves_for(Q,Eqn)
  ←kind(Q,Type,Defn) &
    relates(Type,F_list) &
    select(Formula,F_list) &
    prepare(Formula,Defn,Situation) &
    isform(Formula,Situation,Eqn).
```

This rule states that Eqn solves for Q if Q has type Type and Formula is a formula

that relates Type quantities to other quantities, and if Situation is the situation within which to apply the Formula given the Defn of Q, and if Eqn is the instantiation of the formula in the given Situation. It can be seen that this rule is a direct axiomatisation of our earlier description of how the Marples algorithm extracts equations. (The select goal would specify the qualitative guidance and apply the independence criteria (given extra arguments) ).

In a similar way we give the following example of rules which describe how the Inference Control uses function properties.

```
is_satisfied(Goal)
   ← rewrite(Goal,Newgoal,Strategy) &
      decompose(Newgoal,Pred,Args) &
      meta(Pred,N,Args,Types,Func_info) &
      method(Strategy,Func_info,Newgoal).

method( strategy(dbinf),
          function(Fargs => Vals), Newgoal)
   ← all_bound(Fargs) &
      use_function_properties(Newgoal).
```

The first rule states that Goal is satisfied if it rewrites to Newgoal whose predicate symbol has certain Func_ info, and if a method is used based on the Strategy and this Func_info. (Certain arguments to meta have been ignored). The second rule states that the normal inference method will prove Newgoal given its function properties if all the function arguments, Fargs, are bound, and if object-level inferencing is performed using function property pruning.

As a final example we take a rule concerned with algebraic equation solving. This rule is interesting in that while PRESS does not currently use it, it could be derived from rules PRESS does have. Automating this procedure would be an interesting area for study.

```
solve(U,Expr1,Ans)
   ← occ(U,Expr1,2) &
      collect(U,Expr1,Expr2) &
      isolate(U,Expr2,Ans).
```

This rule states that Ans is an equation which solves for U given Expr1 if Expr1 contains two occurrences of U, if Expr2 is an equation derived from Expr1 in which these two occurrences have been collected together, and if Ans is an equation derived from Expr2 in which U has been isolated on the left-hand side.

All the above rules can be seen as classifying object-level descriptions and using this information in deciding what to do. However the effects are very different in the different areas. In the natural language processing meta-level rules monitor object-level assertions, rejecting semantically unacceptable consequences of a parse. In equation extraction the effect is to select equations using a means/ends analysis technique. In Inference Control the result is use of the most effective axiomatisation for the goal in hand, and in Algebraic manipulation multiple re-

write rules are selectively brought to bear on expressions. Thus relatively simple meta-level search strategies can induce a wide variety of complex object-level behaviours.

These meta-inference techniques were strongly suggested by our use of the programming language Prolog. The fact that Prolog procedures are also predicate calculus clauses and the fact that predicate calculus has a clear semantics, encourages the user to attach meanings to his procedures and these meanings are usually meta-theoretic. However, Prolog as a programming language only offers a single level of 'syntactic' structures (atoms, compound terms etc.), and a lack of care can lead to a blurring of theoretical distinctions. During the development of MECHO, a lack of emphasis (realisation?) of these distinctions resulted in a mixing of object and meta levels (for example, the use of Prolog variables to represent variables at both levels, the mixing of object and meta level assertions in rules such as isform). We plan to remove these aberrations.

Weyrauch's work on the FOL system (See [17]), is of importance in relation to this need for an adequate theoretical formalism. The distinction between the object-level and the meta-level is fundamental within his system, and his use of 'reflection principles' is designed to capture the relation between these levels. We feel that his work is of direct value to workers in the field of expert systems, such as ourselves.

The principle of utilising 'knowledge about knowledge' is becoming increasingly important in practical AI programs. Davis and Buchanan [6] classified four different kinds of meta-level knowledge used by their TEIRESIAS system. They represent knowledge about objects and the data-structure used to describe them in schemata and describe the argument type characteristics of their functions in templates. Their program can classify and build models of the inference rules it uses and meta-rules are used to guide the choice of inference rules to be used and the order of using them. In MECHO, the Natural Language and Inference Control modules both use information like that stored in TEIRESIAS templates. MECHO meta-level inference rules are similar in spirit to TEIRESIAS meta-rules except that the MECHO rules are more general purpose and they generate a variety of different search strategies in different contexts.

## 4. CONCLUSION

In this paper we discussed MECHO, a program for solving mechanics problems. We have shown how the technique of using and controlling knowledge about the domain by inference at the meta-level, can be applied to a range of different areas. Many workers in the field (e.g. [9], [6], [17]), have argued that controlling search by using meta-level inference is superior to built-in, smart search strategies because the search information is more modular and transparant. The argument is for systems to make explicit the full knowledge involved in their behaviour, which in turn aids the modification of their data and strategies, thus improving their robustness and generality. This leads the way to systems which could automatically modify their strategies and explain their control decisions.

We conclude that meta-level inference can be used to build sophisticated and flexible strategies, which provide powerful techniques for controlling the use of knowledge, while retaining the clarity and modularity of a declarative knowledge representation.

## 5. REFERENCES


1. Bobrow, D. *Natural Language input for a computer solving system*. PhD thesis, MIT, 1964.
2. Bundy, A. 'Computational models for problem solving' *Learning and Problem Solving (part 3)*, The Open Univ. Press, 1978. Units 26-27 of the Open University Cognitive Psychology Course D303.
3. Bundy, A. & Welham, R. *Using Meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation.* Forthcoming working paper, Dept. of Artificial Intelligence, Edinburgh, 1979.
4. Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R. and Palmer, M. *Mecho: A program to solve Mechanics problems*. Working Paper No. 50, Dept. of Artificial Intelligence, Edinburgh, 1979.
5. Charniak, E. *Computer solution of calculus word problems*, pages 303-316. IJCAI, 1969.
6. Davis, R. & Buchanan, B.G. *Meta-level knowledge: overview and applications,* pages 920-927. IJCAI, 1977.
7. De Kleer, J. *Qualitative and quantitative knowledge in classical mechanics.* Technical Report AI-TR-352, MIT AI Lab, 1975.
8. Haviland, S.E. & Clark, H.H., What's new? Acquiring new information as a process in comprehension. *Journal of Verbal Learning and Verbal Behaviour* 13:512-521, 1974.
9. Hayes, P. *Computation and deduction*. Czech. Academy of Sciences, 1973.
10. Humphrey, D. *Intermediate Mechanics, Dynamics*. Longman, Green & Co., London, 1957.
11. Larkin, J. *Problem solving in Physics*. Technical Report, Group in Science and Mathematics Education, Berkeley, California, 1977.
12. Marples, D. *Argument and technique in the solution of problems in Mechanics and Electricity*. Technical Report CUED/C-Educ/TRI, Dept. of Engineering, Cambridge, England, 1974.
13. Novak, G. *Computer understanding of Physics problems stated in Natural Language*. Technical Report TR NL30, Dept. Computer Science, Univ. of Texas, Austin, 1976.
14. Pereira, L.M., Pereira, F.C.N. and Warren, D.H.D. *User's guide to DEC-system-10 PROLOG*. Technical Report, DAI, 1978.
15. Stallman, R.M. & Sussman, G.J. *Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis*. Technical Report No. 380, MIT AI Lab, 1976.
16. Waltz, D. *Generating semantic descriptions from drawings of scenes with shadows.* Technical Report MAC AI-TR-271, MIT AI Lab, 1972.
17. Weyhrauch, R.W. *Prolegomena to a theory of mechanized formal reasoning.* RWW Informal Note 8, Stanford University, 1979.
18. Winograd, T. *Understanding Natural Language*, Edinburgh University Press, 1972.