A. Bundy
G. Luger
M. Stone
R. Welham

MECHO[1]; YEAR ONE

This is a progress report on the MECHO project originally announced in D.A.I. Working Paper No. 8, Bundy, Luger and Stone, 1975. The project is to write a computer program which can solve mechanics problems stated in English. This is motivated by a desire to understand how it is possible to form a mathematical model of a real world situation. A problem, typical of those solved by our program, is given and the natural language analysis, equation extraction and the solution of the problem discussed.

0 MOTIVATION

This work is motivated by a desire to understand how it is possible to change the representation of a problem in order to make its solution easier. In particular how it is possible to go from the natural language statement of a problem to a mathematical model from which the problem can be solved. Mechanics seems a suitable area to study this because it provides a rich source of problems which are stated in English using a limited domain of discourse and which are hard enough to be interesting without being intractable.

Secondary motives for studying mechanics problems are that they (a) provide an opportunity to study how semantic knowledge (of physics) can be used to guide the search for the solution to a problem in pure mathematics (equation solving) and (b) there may also be educational spinoff from our formalization of the intuitive physical knowledge required in problem solving. This knowledge is not normally stated in text books, but is essential for solving the problem, and is often overlooked by the unsuccessful problem solver.

1 METHODOLOGY

Our initial approach to solving mechanics problems has been to

---

1 As all the best projects seem to need a silly name (HACKER, DENDRAL, PARRY ...) we have called ours MECHO, short for MECHanics Oracle.

divide the task into parts and to tackle each of the problems relatively independently. The reasons for this approach are explained in Bundy, Luger and Stone (1975). Accordingly we have written programs to

(a) Translate the syntactic parse of a problem statement into a surface-level meaning representation,

(b) Extract equations from a deep level meaning representation,

(c) Solve the resulting simultaneous equations.

This preliminary work has enabled us to build up our descriptive theory. That is, we are developing an ontology for each of the meaning representations and are getting a better idea of the kind of inferences required to go from one representation to another. We also have a better understanding of how the inferencing should be controlled (i.e., about the search strategy).

The second section is divided into three parts. These describe the natural language analysis (A), equation extraction using the "Marples Algorithm" (B) and the solving of the equations (C). The third section discusses the merits of PROLOG in this project.

2 A MECHANICS PROBLEM AND ITS SOLUTION

The following mechanics problem is used throughout this paper to illustrate the action of our programs:-

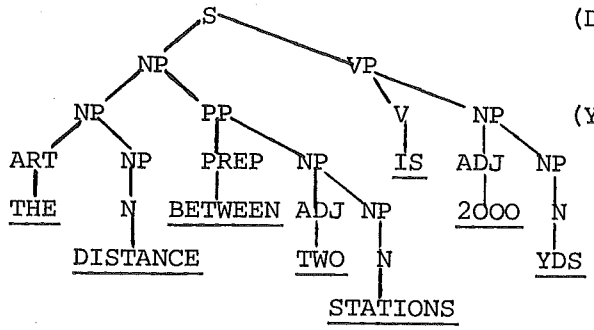
> The distance between two stations is 2000 yards. An electric train starts from rest at one station with a uniform acceleration of A1 ft/sec^2; it comes to rest at the other station with a uniform retardation of A2 ft/sec^2. The speed for the intermediate portion of the journey is constant. Find the constant velocity if the journey is to be completed in three minutes.

A. Natural Language Analysis

The existing natural language program is written in POP-2 and consists mainly of a hierarchical (Bundy & Stone, 1975) database and a set of functions that make use of the database. The database is very important as it contains the information used to guide the parse. At the moment the program only partially performs the task outlined below.

We depend on semantics to dis-ambiguate any parsing problems, and we presently concentrate on developing mechanisms to deal with semantics rather than syntax. The program as it stands accepts as input an embedded list structure that is similar to the output from a simple syntactic parser.

Syntactic Parse



Program Input

(DISTANCE (STATION1
STATION2)
(YDS 2000))

The basic strategy is straightforward. Each clause is examined sequentially. The key word in each phrase (noun for noun phrase, verb for verb phrase) is looked at first. These words, generally representing entities or relationships, will each have at least one major entry in the database. This entry contains information such as which entities can be objects of relationships and which have certain attributes or entities associated with them. For example, acceleration is a quantity, and quantities are known to have a measure, a unit, and a direction. In the case of an entity, the words on the same phrase level are checked against the database to see how they can be used to expand the definition of the entity. For example, in the clause

((TRAIN ELECTRIC) (STARTS(REST) (STATION ONE))
(ACCELERATION UNIFORM FT/SEC^{2}A1)),

ACCELERATION as a quantity is first recognized as an attribute of a particle. UNIFORM is defined as "CONSTANT", an attribute of a quantity; and FT/SEC^2 A1 is recognized as a particular instance of a quantity. When it was recognized that a particle (the TRAIN) had been put into motion, it was noted that there should be a corresponding acceleration. If one assumes that A1 is the appropriate acceleration, the following assertions can be made:

ACCEL (TRAIN,AQ1,PERIOD1)
MEASURE (AQ1,A1)
UNIT (AQ1,FT/SEC^2)
CONSTANT (AQ1)

(PERIOD1 refers to a period of time at the start of which the motion of the train begins)

For a relationship, any previously mentioned entities are examined to see if they qualify as arguments. As the rest of the clause is parsed, new entities are also tested for suitability as arguments.

In the third clause: ((IT) (COMES-TO-REST (STATION OTHER)
(RETARDATION UNIFORM (FT/SEC^2 A2)))),

"IT" is recognized as a pronoun referring to a solid object. "COME-TO-REST" is a verb that indicates the completion of a period of motion of a solid object. This would allow the following assertion:

MOTION (IT,PERIOD2).

But before making this assertion, the data structure is searched for a referent for "IT". The "TRAIN" is the only solid object presently described as being in motion. "IT" and "TRAIN" can be equated, and the following assertion made:

MOTION (TRAIN,PERIOD2)

After the first pass a focal point for the problem is established, in this case the journey of the train. Any isolated pieces of the data structure are examined in a second pass to see if they can be linked to this overall structure. The first clause implied these assertions:

PATH (DØ,STATION1,STATION2)
MEASURE (DØ,2ØØØ)
UNIT (DØ,YDS)

These assertions were not directly related to anything. On the second pass STATION1 and STATION2 are seen as endpoints of PATHØ, the path taken by the train, and

EQUAL(PATHØ,DØ) can be asserted.

The structure is carefully checked for contradictions, and eventually will be passed on to the equation extractor in the form of PROLOG clauses:

```
JOURNEY (TRAIN,EPISODE,PATHØ)
SEGMENT (EPISODE,PERIOD1.PERIOD3.PERIOD2.NIL)
INITIAL (PERIOD1,DEPARTURE)
INITIAL (PERIOD3,CHANGE1)
INITIAL (PERIOD2,CHANGE2)
FINAL (PERIOD2,ARRIVAL)
          etc.
```

B. The Marples Algorithm

All programs for solving applied mathematics problems employ some device for extracting equations from a semantic database. Often this is very simple as in Charniak's program which extracted all possible equations. It is possible to do better than this by avoiding irrelevant equations. The best explanation we have seen of how to do this is Marples' description (Marples 1974) of the behaviour of engineering students. Our algorithm is based on his description.

Our algorithm distinguishes between symbolic quantifiers for which solutions are required (sought unknowns such as the "constant velocity" in the example above) and those which can legitimately appear in such solutions (givens such as accelerations A1 and A2). The first sought unknown is removed from the list of sought unknowns and an equation is formed which contains it. In general this can be done in several ways. We prefer equations which involve only sought unknowns or givens and do not introduce any further intermediate unknowns. Unfortunately we may be forced to introduce some intermediate unknowns in which case these are added to the end of the list of sought unknowns. The sought unknown for which we have just solved is added to the list of givens and a record of the equation we have just formed is remembered. The process is repeated recursively until there are no further sought unknowns to be solved for. The equations are then subjected to an "independence" check, to make sure the same equation does not appear twice and that only two (of a possible five) constant acceleration equations are used.

Equations are actually formed by the PROLOG procedure "MAKEEQN", each clause of which corresponds to a certain physical formula. For example, consider the clause for constant acceleration, v=u+a.t (present velocity equals the initial velocity plus the acceleration multiplied by the time).

```
+MAKEEQN (*V = *U + *A. *T, (CONSTACCEL.1).(*P. *OBJ),*US)
     - ACCEL (*OBJ, *A, *P)    - UNUSED (CONSTACCEL. 1,
       *P. *OBJ, *US)
     - ISVINVAR (*A)      - DIFF (*A, *ZERO)
     - PERIOD (*P)       - DURATION (*P, *T)
     - INITVEL (*OBJ, *U, *P)      - FINVEL (*OBJ, *V, *P).
```

"MAKEEQN" is the name of the procedure; *V = *U . . . . , *US is the calling pattern. The rest of the clause is the body, where - ACCEL (*OBJ, *A, *P) etc. are sub-routine calls. "CONSTACCEL.1" is the name we give to the resulting equation. *US, the list of equations already produced, is the only input to the procedure. These are both used by the independence checking procedure "UNUSED".

The clause can be read as follows:

"We can make an equation V = U+A.T provided: A is the acceleration of some object, OBJ, during some period of time, P; CONSTACCEL.1 in situation P.OBJ is independent of all equations used so far; vector A is constant; A is different from ZERO; P is a period; T is the duration of P; U is the initial velocity of OBJ in period P and V is the final velocity."

These facts are usually checked in the semantic database, although the last two might involve some trivial inferences. The equations extracted from the example considered above are:

```
EQUATIONS - EXTRACTED
V = ZERO + AQ1. TI1 &
T1Ø = TI1 + (TI2 + (TI3 + Ø)) &
V . TI2 = D2  &
ZERO = V + AQ3. TI3 &
DØ = D1 + (D2 + (D3 + Ø)) &
D1 = ZERO . TI1 + 1 / 2 .AQ1 . TI1 : 2 &
D3 = V . TI3 + 1 / 2 .AQ3 . TI3 : 2 &
TRUE
(':' is exponentiation)
```

The units (of velocity, time, etc.) are then standardized and the equations are simplified. The following PROLOG clause, including the list of sought unknowns, is sent to the equation solver:

```
-SOLVE(

Z=A1.T1                                          &
3=T1.6Ø:-1+T2.6Ø:-1+T3.6Ø:-1                     &
Z.T2=X2                                          &
Ø=Z+A2.T3                                        &
6ØØØ=X1+X2+X3                                    &
X1=2:-1.A1.T1:2                                  &
X3=Z.T3+2:-1.A2.T3:2                             &
TRUE,
Z.T1.T2.T3.X2.X1.X3.NIL).
```

The string at the end of the seven equations is the list of "sought unknowns".

C. The Equation Solver

This program is capable of symbolically solving sets of simultaneous algebraic equations for a given list of unknowns. The first step is to select one of the equations and one of the unknowns and to solve the equation for the unknown. The solution is then substituted in the remaining equations and the process repeated until the last equation is solved for the last unknown. At the moment the equation to be solved and the unknown to be solved for are selected by stepping sequentially through the lists of each until a pair is found for which the unknown is solvable by the program. The order in which the equations were extracted (previous section), determines the order within the lists. Later it is hoped to make this process more intelligent by having the program formulate an overall plan or optimum order for the solution of these equations.

The bulk of the program is concerned with solving one equation for one unknown. The strategy consists of successively applying members of a set of rewrite rules to the equation. At present there are 61 such rules in the program, but

since the program is written in PROLOG, an approximation to predicate logic, additional rewrite rules may be added at any time. The rules are not applied at random and the computation is guided by indexing them into sets labelled as useful for a particular strategy. For example, a strategy known as isolation is applied to an equation as soon as there is just one occurrence of the unknown in that equation. The idea behind this strategy is to change the equation to one of the form X=T where X is the unknown and T is a term not containing the unknown. Only the rewrite rules marked "useful to isolation" are invoked by isolation. A typical rewrite rule useful for isolation is "Replace log(U)=V by U=e:V".

Other strategies used by a super-strategy called the basic method are known as collection and attraction. Collection has the job of collecting together occurrences of the unknown and thus reducing the number of occurrences of the unknown in the equation. Thus a typical rewrite rule labelled as useful for collection is "replace 2.sin U.cosU by sin 2U".

Attraction brings occurrences of the unknown "closer together", thus preparing the way for collection. Typically, to attract U and V in the expression U.W+V.W we use the rewrite rule "replace U.W+V.W by (U+V).W".

The basic method implemented in the program tries to apply the strategies of isolation, collection and attraction, recursively. In addition to the basic method, the program has the capability to recognize certain special classes of equations such as linear or quadratic, and can also make a change of unknown. Thus when solving the equation a(sin x):2 + bsinx +c = 0 for x the program first substitutes y for sinx and then recognizes that the resulting equation a.y:2+by+c = 0 is a quadratic in y.

Technical features of the program include a pattern matcher which knows about the commutativity of addition and multiplication,and a package for regarding terms dominated by addition or multiplication function symbols as "bags".

3 EXPERIENCE WITH PROLOG

In most of our mechanics work to date we have used the experimental programming language PROLOG, first developed at Marseille (Roussel, 1975), and currently being improved at Edinburgh (Warren; 1975, 1976). We have written two programs in PROLOG and are in a position to draw some conclusions based on our experiences.

We were very pleased with both the expressive power and speed of PROLOG. It offered all the normal facilities of functional language like LISP or POP-2, with no significant loss of speed. The provision of pattern directed invocation and non-determinism resulted in smaller, more transparent sub-

routines in our programming and a consequent reduction in programming effort. The search mechanism was faster than anything we could have written in a short time. We found PROLOG very easy to learn.

The biggest drawback was the space requirements. At the present time PROLOG is available at the University of Edinburgh in two sizes, 50K and 75K. The 75K PROLOG can only be used in unsocial hours. Earlier versions of our programs exhausted the 50K PROLOG. The latest versions now exhaust 75K. Because the PROLOG default is to be prepared to backtrack to every choice point unless specifically told not to, it uses a lot of space at run time recording these choice points. These choice points are kept even if PROLOG has been told not to backtrack to them. Warren plans to correct this fault (Warren, 1976).

The second major drawback is that the debugging aids are primitive, especially in the early version of PROLOG (SVI) which we are using. Warren has now issued an improved version (SVW) which, together with some further planned improvements (Warren, 1976) seems to meet most of our criticisms. In general, we feel that PROLOG is an exciting new language, fully justifying further development to make it a viable alternative to other A.I. languages.

4 SUMMARY AND CONCLUSIONS

The achievements of the MECHO project so far include three separate programs that respectively:

1) partially implement the task outlined in the natural language section,

2) implement the Marples algorithm for the extraction and independence checking of equations and the conversion of these into a uniform set of units, and

3) implement the "basic method" (Bundy, 1975) for successfully solving sequences of simultaneous equations.

The most important tasks for the future are concerned with providing intelligent links between these programs. The first desirable link would be between a syntactic parser and the existing natural language programs. For instance, contextual information could be used intelligently to determine which of several entries for one word would be the most appropriate in the parse. The final goal would be to have one program that performed the "syntactic" and "semantic" parsing simultaneously, exchanging information between the two.

The gap between the data structure obtainable from the natural language input and the deep level database necessary for extracting equations will eventually be bridged by inferences invoked at this equation extracting stage. This will

inevitably slow down the Marples algorithm. This situation could be improved by employing heuristics designed to reduce the time spent forming irrelevant equations. One possible heuristic might be to divide problems into types corresponding to the chapter headings in applied mathematics textbooks. Associated with each problem type would be an ordered list of equation names. If the problem type could be identified by contextual cues the associated equations would be formed first.

Another line of improvement would be to search for optimality as well as relevancy and irredundancy in the equations extracted. For instance, we could prefer equations which introduced the smallest number of intermediate unknowns.

Finally, there are further questions concerning the ability of PROLOG in the MECHO Project that can only be answered in light of the further development of the storage and debugging features of the language (Warren, 1976).

REFERENCES

Bundy, A., 1975. "Analysing Mathematical Proofs (or reading between the lines)", Proceedings of IJCAI4, Georgia, Cambridge, MIT-AI press.

Bundy, A., Luger, G. and Stone, M., 1975. "A Program to Solve Mechanics Problems Stated in English", Department of Artificial Intelligence Working Paper 8, University of Edinburgh.

Bundy, A. & Stone, M., 1975. "A Note on McDermott's Symbol Mapping Problem", SIGART Newsletter 53, 9-10.

Charniak, E., 1969. "Computer Solution of Calculus Word Problems", pp 303-316, Proceedings of IJCAI1, eds. Walker, D.E. and Norton, L.M. Washington, D.C.

Marples, D.L., 1974. "Argument and Technique in the solution of Problems in Mechanics and Electricity", CUED/C-Educ/TRI, Dept. of Engineering Memo, University of Cambridge, England.

Roussel, P., 1975. "PROLOG: Manuel de Reference et d'Utilisation", Groupe d'IA, Marseille-Luminy, September, 1975.

Warren, D., 1975. "Epilog": Users' Guide to DEC 10 PROLOG (Internal Memo, Department of Artificial Intelligence, University of Edinburgh).

Warren, D., 1976. Proposal to Reduce PROLOG Storage Requirements (Internal Note, Department of Artificial Intelligence, University of Edinburgh).

ACKNOWLEDGEMENTS

We would like to express our gratitude to our colleagues in the

Department of Artificial Intelligence at Edinburgh for their help and encouragement, especially David Warren, for making PROLOG available and advising us on how to use it. This research was funded by SRC Grant B:RG:9449.3 and by an SRC research studentship awarded to Bob Welham.