The role of inference in the solving of mechanics problems

Master PLEASE RETURN THIS TO ALAN BUNDY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

UNIVERSITY OF EDINBURGH

DAI Working Paper : 16

The Role of Inference in the Solving of Mechanics Problems

by Alan Bundy

Introduction

The questions addressed in this paper have arisen in a project on the automatic solution of mechanics problems, but are of much wider significance. The aim of the mechanics project is to write a computer program to solve problems stated in English:

e.g.

"One ship is sailing due east at 12 mph and another ship is sailing due north at 16 mph. Find the velocity of the second ship relative to the first." (Humphrey, 1957, p16).

More detail of the project can be found in Bundy, Luger, Stone, 1975 and Bundy, Luger, Stone & Welham, 1976. In these papers different problems are discussed such as problems about "motion in a straight line" and about "pulleys".

The motivation behind the project is to explore the relationship among many different representations of a problem. These include the representation derived from the natural language statement of the problem, a representation sufficient for drawing a diagram, and a representation such as a set of equations suitable for solving the problem. These representations may be conveniently envisaged as forming a sequence with the English statement at one end and the answer at the other. In Bundy et al 1975 we suggest a sequence of 7 such representations.

Most of the processing necessary to go from one representation to the next will be done by a set of inference rules. The value of the "inference paradigm" is that it enables us to escape from a strictly sequential development of these representations and make the development sensitive to the particular situation. It is not necessary for a particular representation to be complete before it can contribute to the development of later stages. Inferences can be made both ways so that a later stage can fill in details in an earlier one. Development can be goal driven so that the precise development of a stage depends on the actual question being answered. As we shall see inferences may be tentative and may later be undone.

In this paper we will be looking at some inference rules used in the mechanics project and contrasting them along several dimensions, such as: plausible versus deductive; how they are controlled and the use of Frames like notions (Minsky 1974).

Some Examples of Inference Rules

Here are some of the inference rules which we either use or plan to use in the mechanics program, with a brief explanation of their role. In the program these inference rules appear in predicate calculus notation (actually PROLOG notation, see Warren 1975), but we have translated them into English.

In mechanics problems there are often hidden assumptions. For instance in the relative velocity example above it is implicit that the precise dimensions and shape of the ships are not important. Further the problem is essentially 2 dimensional: Both ships are envisaged as resting on a horizontal plane (the sea). We plan to handle this by translating each real world object into an ideal object configuration, according to the problem type. In the case of relative velocity problems, ships would be translated into particles on a horizontal plane. In problems using Archimede's Principle, on the other hand, ships might become containers floating in a liquid. We therefore get the inference rule.

Problem type "relative velocity" → ship becomes particle on horizontal plane.(a)

Similarly in "motion in a straight line" problems the directions of vector quantities (accelerations, velocity, etc.)) are not always explicitly stated. One is left to infer from the problem type that unless otherwise stated all vectors have the same direction.

Problem type "motion in a straight line" → all vectors have same direction.(b)

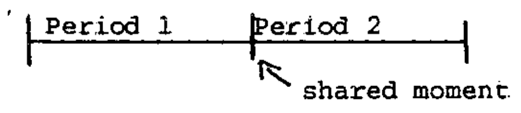
If our program is to use these problem types to make inferences, they must be made known to the program. At the moment the program is told explicitly,just as students are told by the chapter heading of the text they are using to handle miscellaneous examples or exam questions,the program should be able (as a good student is) to infer this for itself.from cues given within the text of the problem. One such cue for relative velocity problems is particularly easy: a request for the velocity of one object relative to another.

Velocity sought of x relative to y → problem type is "relative velocity".(c)

Unfortunately we have to allow that these cues may mislead and that later in the solution processes the inference may have to be withdrawn.

Some of the inference rules are of the more conventional kind relating quantities within a particular representation. For instance, if one period of time immediately follows another, the initial moment of the second must be the same as the final moment of the first.

Period 2 follows Period 1 → the final moment of Period 1 is the initial moment of Period 2. (d)



To solve problems it is often necessary to consider physical quantities (velocities, forces, etc.) not mentioned in the original statement.

Velocity of obj at time t sought, obj is particle, t is time → create V, declare V as velocity of obj at time t. (e)

It is clearly important not to create these whenever possible, but only when the problem solving process demands it. For a discussion of how this is done see Bundy et al 1976, p4.

Logical Status of Inferences

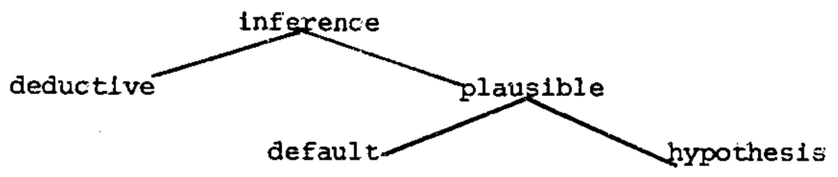
A distinction is often drawn (see for example Wilks 1974) between deductive inference rules and what are variously called inductive or plausible inference rules. What distinguishes the latter is that they are not universally valid but only probable. It should be clear that we use such inference rules in our project and that (b) and (c) in the last section are good examples of them. Whereas (d) is a good example of a deductive rule. We will use the notation of plausible versus deductive inference, (since "induction" is already an overworked word).

If plausible inference making is to mean anything more than rather suspect deductive inference making, we must allow for the possibility of the inference being wrong. There are at least two ways of doing this.

Firstly, and most simply, the inference rule can be treated as a default and used only when all other methods have failed. We have used inference rule (b) above, in this way. So that if the program wants to find the direction of a vector, and the problem type is "Motion in a straight line" then if all other methods fail it will assume the direction to be some standard default.

Secondly the inference rule can be treated as a hypothesis, which can be tested and may be found wanting and rejected. We intend to use inference rule (c) above in this way. That is having hypothesised a particular problem type based on cues in the problem, contrary evidence will be considered (e.g. cues for rival problem types) and the original hypothesis may be rejected.

The classification of the logical status of inferences may be represented in the following tree.



The use of (b) as a default inference pre-supposes that it is used "backwards" i.e. the inference is only used to answer questions about the direction of some vector. It also pre-supposes that we have some control over the order in which inferences are used. When the program wants to know the direction of some vector it first checks to see if the information is directly recorded as a unit assertion in the database. If this fails it will try to infer the direction using deductive inference rules. Only if both these fail will it assign some default value using rule (b) .

On the other hand the use of (c) as a hypothesis pre-supposes it is used "forwards" i.e. to decide the problem type given the presence of certain cues. This usage raises several unresolved questions:

(i) Should the program temporarily halt the development of the representation stages in order to test this hypothesis, or should it continue with the development but be on the lookout for a contradiction or some other indication that the hypothesis is false?

(ii) In the former case i.e. performing an immediate and explicit test (as recommended in the best books on scientific method!), what tests should be used and when should it decide that the hypothesis is confirmed?

(iii) If the program continues assuming the hypothesis to be true, but later finds a contradiction, how is blame to be assigned to one of the several hypotheses which may currently be extant?

(iv) Worse still, once blame is assigned, how is the resulting mess to be cleared up? Not only the hypothesis must be deleted, but also any conclusion it was a party to, however indirectly.

Of course these are general questions and there is unlikely to be a single answer to them. Much work is required to survey solutions to these problems in existing AI programs and to study ways in which humans (including scientists) solve them. Here are the imperfect beginnings of such a survey.

Hypothesis making occurs in Hardy's work on the automatic writing of LISP programs from sample input/output pairs (Hardy 1974). He uses hypotheses to suggest ways of dividing the output between that produced by the current procedure and that produced by the recursive call. What distinguishes this work from ours, is that Hardy's hypotheses are about <u>ways to solve the problem</u>. So that there is no distinction in his program between testing the hypothesis and continuing the development (question (i)). The hypothesis fails, if it fails to lead to a solution (question (ii)). Only one hypothesis is extant at any one time, so there is no credit assignment problem (question (iii)). On setting up a hypothesis a CONNIVER (McDermott and Sussman 1972) like context is created and this is popped on hypothesis failure thus deleting all assertions made since the hypothesis was set up, regardless of whether the hypothesis was instrumental in inferring them (question (iv)). It is worth pointing out that Hardy diminishes the chance of an unsuccessful hypothesis being set up by insisting that it be suggested by some minimum number of cues.

Unfortunately Hardy's solutions are dependent on properties of his particular task, especially the fact that his hypotheses are about ways of guiding the search for a solution to the problem. We are unable to adopt Hardy's solutions since our hypotheses have a different flavour, namely they activate implicit information about the problem. A kind of hypotheses making, perhaps more suitable for us is proposed by Hayes and Rosner in their work on understanding cocktail party conversations. (Hayes, P., Rosner, M., 1976). Their program will proceed by a process of forward inferencing. All assertions will be tentative and assigned some (numeric) probability. The program will not explicitly test any hypothesis, but will be constantly on the lookout for a contradiction (our question (i)). Blame for a contradiction will be assigned to the least likely hypothesis (our question (iii)). Each assertion will point to the assertions which established it, and these pointers will be used to selectively clean up after a contradiction is found (our question (iv)).

In our mechanics project only a few assertions are made by hypothesis rules so we could afford to modify the Hayes/Rosner scheme by having explicit "hypothesis failing" inferences i.e. the presence of certain cues would upset cosy assumptions about the problem type. In the relative velocity example this might be a reference to a 3 dimensional configuration or request for information about the volume of water displaced by the ship. This would enable us to drop the numeric probabilities and the continuous testing for inconsistency. This proposal is only intended as tentative.

Controlling Inference

It is not enough to say what inferences will be made. If a combinatorial explosion is to be avoided and if the representations are to be developed in a way sensitive to the particular question being asked and information given, then it is essential that the way in which these inferences are controlled be carefully specified. In our project much of these specifications remain to be worked out. However, even now, there are some issues that can be discussed.

We will be discussing some of these issues in terms of the forwards/-backwards (or bottom up/top down) distinction. The usual definition of a forwards use of some inference $A \rightarrow B$ is that given A we may deduce B. A backwards use of the same inference is that if we have B as a goal we may exchange it for the goal of establishing A. It should be clear that this is not a logical distinction since a forwards use of $A \rightarrow B$ could be thought of as a backwards use of $\neg B \rightarrow \neg A$ and vice versa. Never-the-less the distinction between forwards and backwards can be maintained if we reject the heuristic equivalence of $A \rightarrow B$ and $\neg B \rightarrow \neg A$.

As Kowalski has pointed out (Kowalski 1975) the distinction between forwards and backwards is too coarse. There are many hybrid schemes. In our program some of these will be forward inferences which assert other inferences into the database e.g. given A and $A \,\&\, B \rightarrow C$ we may assert $B \rightarrow C$ as a forwards or backwards inference.

Because we are using both default inference rules and hypotheses some of the decisions about the use of forward or backward inference are predetermined. Most of the remaining inferences are done backwards guided by two main goals: the drawing of a diagram and the extraction of equations. This ensures that the development of the representation is pertinent to the particular question posed. For instance we do not fill in all physical quantities that could be filled in, nor derive all the equations that could be derived regardless of their usefulness. The general disadvantages of backwards reasoning are that a given fact may be re-derived several times and that we can spend time searching a fruitless branch of the search tree.

The first disadvantage (i.e. re-derivation) can be partially overcome by asserting established facts into the database. The sort of place this will be done will be in the asserting of the initial meaning representation; the ideal object configurations (inference rule (a)) and the problem types (inference rule (c)). The second disadvantage (i.e. fruitless search) can be mitigated by careful choice of inference rules, ordering of the rules and the insertion of tests to reject identifiably fruitless searches. Of course, this is a hard problem and no easy answers can be expected.

A related issue is that of how hard we want to try, to establish some goal. Consider the goal of finding the velocity of some object at some time. This may be represented as a backwards search for the goal vel(obj, v, t). How much search the program does depends on why it wants to know. For instance, it may know that v is a physical quantity but not what kind (mass, velocity, acceleration, etc.). It may try to find out by looking up mass(obj, v), vel(obj, v, t), accel(obj, v, t) in the database. If the call of mass(obj, v) invokes a huge search using inference rules this could be fatal, since the (easy) call of vel(obj, v, t) might never be made. On the other hand the program may be trying to fill in the details of an equation, like $v = u + at$ in which the value of v is unknown. Now it is prepared to try arbitrarily hard using the database, deductive or default inferences and even, as a last resort, creating a new intermediate unknown (see inference rule (e)).

A case intermediate between these two is when the program wants to know the velocity in order to test some property of it. (e.g. whether it is constant over some period). Then it is prepared to use inference, but not to create a new intermediate unknown, since there would be no way of testing the velocity for the required property. Thus the program needs at least three ways of establishing a goal: database only; database with inference and database with inference and creation. These calls are not currently provided by our language, PROLOG, but they can be mocked up by a judicious choice of description (e.g. by having three kinds of velocity predicate).

The Employment of Frames like ideas

Minsky's Frames paper (Minsky 1974) has been very influential in AI. Below we discuss how it has influenced our work. Our view is that Minsky's paper contains a collection of ideas from which one is at liberty to pick and choose. We use his ideas at the level of conceptualizing our program rather than at the implementational level. That is we have not tried to implement a "Frames" programming language. Inference rules which we think of as related in a "Frame" may be distributed throughout the code. We believe this gives us the benefits of the Frames approach without a loss of flexibility. For instance, it could be useful to be in two Frames at once (in the case of problem type Frames).

The main idea people take from the Frames paper is of prestored models of stereotypical situations, which are evoked by cues and are then used to help interpret the input to the problem. We use this idea in two places in our program: the problem types and the ideal object configurations. These are evoked by cues (see inference rules (c) and (a) respectively), and then supply extra information to the program (see inference rules (a) and (b)). The ideal object configurations are a conventional kind of static frame with something of the hierarchical structure advocated by Minsky (Minsky 1974, pl). The problem types seem to be a fairly unconventional type of frame. They do not describe a physical situation or a sequence of events and the inference rules they consist of are scattered throughout the program.

Another Frames idea we use is that of default values associated with a frame. We have mentioned the default direction associated with problems of type "motion in a straight line" (see inference rule (b)), we have similar defaults with other problem types and we expect defaults associated with ideal object configurations e.g. the string in a pulley system will be assumed inextensible and the pulley smooth and weightless.

Conclusion

In this paper we have described some of the inference rules employed in a program which solves mechanics problems. One advantage of using such rules in an AI program is that it allows a flexible flow of control, sensitive to the particular situation. We have compared these inference rules along four dimensions: content; logical status; search control and relationship to Frames. We hope the discussion will be of wider interest and will spark off a debate resulting in the extension and clarification of some of the ideas expressed.

References

Bundy, A., Luger, G. and Stone, M. 1975. "A Program to Solve Mechanics Problems Stated in English", DAI Working Paper 8.

Bundy, A., Luger, G., Stone, M. and Welham, R. 1976. "MECHO; Year One", Proceedings of 2nd AISB Conference, ed. Brady, M., Edinburgh.

Hardy, S. 1974. "Automatic Induction of LISP functions", Proceedings of 1st AISB Conference, p.51.

Hayes, P.J., Rosner, M. 1976. "ULLY: A Program for Handling Conversations", Proceedings of the 2nd AISB Conference, p.137. ed. Brady, M.

Kowalski, R. 1974. "Logic for Problem Solving", DCL Memo No. 75, Edinburgh.

McDermott, D.F. and Sussman, G.J. 1972. "The CONNIVER Reference Manuel", AI Memo No. 259, MIT Project MAC.

Minsky, M. 1974. "A Framework for Representing Knowledge", MIT AI MEMO No. 306.

Schank, R. 1975. "Using Knowledge to Understand", Proceedings of TINLAP, eds. Schank, R. and Nash-Webber, B.L., Cambridge.

Warren, D. 1975. "SVW.UG - A Users Guide to PROLOG Supervisor 'SVW'". Privately circulated Memo, DAI, Edinburgh.

Wilks, Y. 1974. "A Computer System for making Inferences about Natural Language", Prooeedings of 1st AISB Conference, p. 268.

November 1976