Will it Reach the Top? Prediction in the Mechanics World *

Alan Bundy

University of Edinburgh, Edinburgh EH8 9NW, Scotland

Recommended by R. Boyer

ABSTRACT

We describe an extension of a mechanics problem solving program to the set of "roller coaster" problems, i.e. problems about the motion of a particle on a complex path. The reasoning strategy adopted by the program is described and compared to earlier work in this domain. Conclusions are drawn about the representation of motion and prediction. Questions are raised about Frames and Multiple Representations.

1. Introduction

In this paper we describe how the MECHO mechanics problem solver [2] was adapted to deal with the "roller coaster" problems described by de Kleer [6]. Why did we decide to tackle the roller coaster problems? The reasons are listed below.

(i) Our MECHO program is intended as a general problem solver for the mechanics world. de Kleer's NEWTON, on the other hand, was specifically designed for the roller coaster problems and was based on the ideas of "Frames" [11]. We wanted to see whether the study of the roller coaster problems would show up important limitations of MECHO.

Thus our main motivation was to test the existing MECHO mechanism and to see in what way it needed to be extended. We consider such a research methodology at least as valid as the more common one of using a problem domain to explore some new mechanism (e.g. Frames).

(ii) The roller coaster domain is important as it isolates in pure form the problems of motion, which are involved in all dynamics problems.

(iii) These problems are also involved in the prediction of subsequent events given some initial configuration. An issue we had not dealt with before.

* This research was supported by S.R.C. grant BRG 94493.

Artificial Intelligence **10** (1978 129–146),

(iv) We were dissatisfied with the reasoning processes which NEWTON went through to solve the problem. Some steps seemed irrelevant. We resolved to do better. This is not to underestimate the pioneering work of de Kleer, without which the present work would not have been possible.

2. The Problems

Below we describe the three roller coaster problems tackled by de Kleer's NEWTON.

2.1. The sliding block

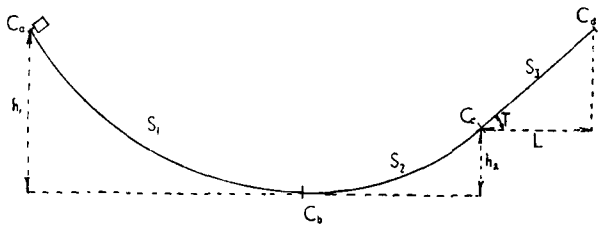


FIG. 2.1.

At time M_a the block starts from rest at point C_a. Will it reach point C_d?

2.2. The loop the loop

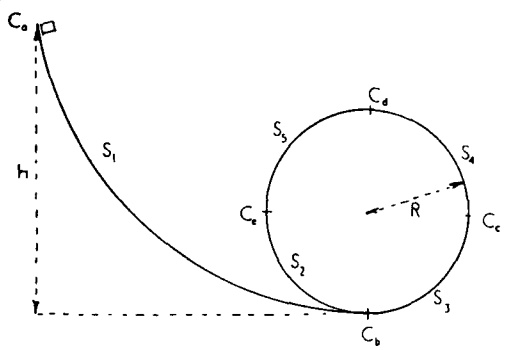


FIG. 2.2.

At time M_a the block starts from rest at point C_a. How small can h be made so that the block still successfully executes the loop the loop?

2.3. The great dome

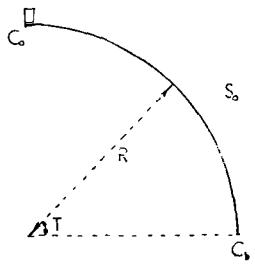


FIG. 2.3.

The block is given a nudge from rest. At what point does it fly off the dome?

All the paths are assumed to be smooth (i.e. friction is ignored).

2.4. Describing the problems

The above problems do not lend themselves to prose description without the aid of a diagram. Since we have excluded visual processing from the current project and they cannot be handled by the natural language processing alone, we decided to bypass this stage and describe the problems to MECHO in the form of symbolic descriptions. Fortunately de Kleer had already developed a symbolic notation for describing these problems and we adopted this in content, though not in form. The essential idea of de Kleer's notation is a description of the paths in terms of their first and second derivatives, i.e. their slope and concavity. These descriptions play a vital part in the prediction process.

The main task which would be required of a visual processing component of MECHO, if we were to build one, would be the partitioning of complex paths into subpaths with invariant slopes and concavities, e.g. given the slope

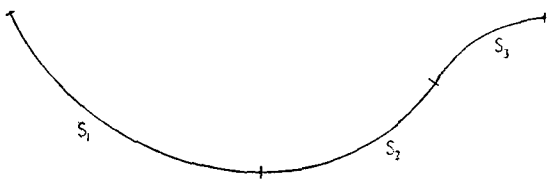


FIG. 2.4.

we would require the "vision" component to partition it into the subpaths: S_1; S_2; S_3, where

Path	*Slope*	*Concavity*
S_1	down	concave
S_2	up	concave
S_3	up	convex

We would be interested in the comments of those working in visual perception as to the feasibility of such processing.

3. Hypothesize and Test

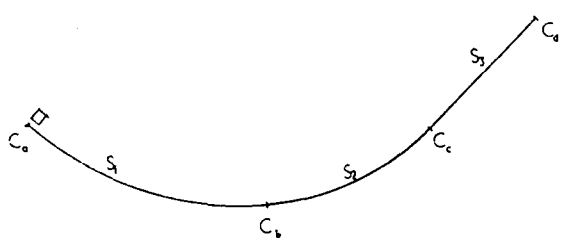
The reasoning strategy we developed for the three problems above is one of hypothesize and test. That is, we answer the question set by the problem in two stages. First the question is subjected to a qualitative test, i.e. is motion feasible given the rough shape of the path? etc. If this proves positive the answer "yes" becomes a hypothesis which is then subjected to a quantitative test, i.e. a detailed analysis involving extracting and solving equations and inequalities. This process can be carried out recursively, if necessary, by breaking the path into subpaths and testing each of them in turn.

It is not necessary, however, to carry out the quantitative testing locally. It can all be saved up to the end and done together.

This division into qualitative and quantitative knowledge was strongly suggested by de Kleer and we have adopted it in our program. The distinction is discussed more fully in Sections 4, 5 and 9. A small modification to the program would stop it from making the distinction, so that qualitative and quantitative testing would be done together. However, the two stage processing seemed to produce a more natural protocol and we have maintained de Kleer's division.

4. A Worked Example

A better idea of the reasoning process can be gained from considering an example. So in this section we consider how our strategy of hypothesize and test deals with the sliding block example.



path S_0

FIG. 4.1.

In our symbolic notation the question to be answered is expressed as

At (Block, C_d, _mom)?

which means "is the block at point C_d at some moment _mom", where _mom is a variable to be bound to a particular moment during the course of the problem

solving. On the grounds that "you can get to a place by travelling there" this question generates the sub-question

Motion (Block, S_0, C_a, left, P_0)?

which means "does the block travel on (above the) path S_0, starting at point C_a, during period P_0" (the fourth argument, left, will be explained later).

This question will be answered affirmatively provided a number of tests prove positive. These tests cannot be answered immediately because not enough is known about the shape of S_0: In particular its slope and concavity are not invariant. The question is, therefore, broken into three subproblems corresponding to the three subpaths, i.e.

Motion (Block, S_1, C_a, left, P_1)?

Motion (Block, S_2, C_b, left, P_2)?

Motion (Block, S_3, C_c, left, P_3)?

where $[P_1, P_2, P_3]$ is a partition of the time period P_0.

These three all pass the qualitative tests, storing up quantitative tests for later. The qualitative tests are expressed as four questions:

(a) Will the block get started?

(b) Will it run out of steam and stop?

(c) Will it go too fast and fly off?

(d) Will it fall off?

Some of these tests are passed without difficulty. For instance, the block will not fall off since it travels above all three paths. Nor will it take off, since none of the paths is convex. There is a possibility of its running out of steam on the second and third paths and this possibility causes a proviso to be stored against the velocities on these two paths. This proviso will be picked up and checked by the quantitative tester in the second stage. The proviso is that the velocities remain real valued. If a block does not reach the upper position of a path its velocity on that section will be imaginary.

When a motion description (e.g. Motion (Block, S_1, C_a, left, P_1)) has passed the qualitative tests and the quantitative provisos have been stored for future testing, it is asserted into the database as a hypothesis. It is necessary to do this because it is needed for the next round of qualitative tests. MECHO cannot decide that the block can "get started" on the next path unless it knows that the block is at the starting point at the right moment. This can be deduced from the fact that the block is at the finishing point of the previous path at the same moment and the two paths are consecutive.

When all the qualitative testing has been done, the provisos are gathered together for quantitative testing. Typically these provisos will involve unknown quantities, which must first be expressed in terms of quantities given in the statement of the problem. Therefore: a list is made of all unknowns in the provisos; equations are

extracted which express these unknowns in terms of the givens; the equations are solved; the solutions are substituted for the unknowns and the provisos are evaluated. By giving different numerical values to the given quantities the evaluation of the provisos can be made to return a positive or negative outcome. Alternatively, if the givens have symbolic values the evaluation process can return an expression as answer.

The algorithm for deciding what equations to extract was based on Marples 1974 study of Cambridge Engineering students. It works by carefully examining the unknown quantity whose value is sought to see what kind of quantity it is (e.g. acceleration, mass etc.) and what situation it occurs in (e.g. what objects, time and direction it is associated with). This information is then used to draw up an ordered short list of equations to be tried. If possible an equation is found which introduces no new intermediate unknowns, but if this is impossible intermediate unknowns are created and these are added to the list of unknowns whose value is sought. For further details see [2]. Notice that no information about the problem type is used, so that the algorithm is general purpose.

5. Finding Minima

The reasoning process for the other two examples is very similar except for the final stage. Instead of the proviso being evaluated to return a simple yes/no answer they are investigated to find minimum values under which they return true.

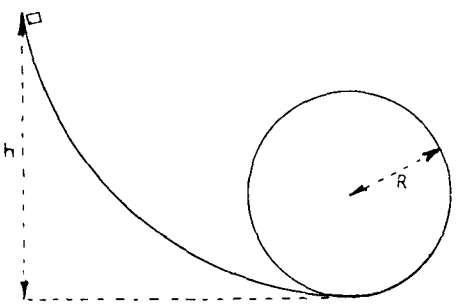


FIG. 5.1.

For instance, in the loop the loop example the original provisos concern conditions on the velocity of the block under which the block will not fall off or stop. After equation extraction, solution and substituting these become inequalities between h, the initial height of the block, and R, the radius of the circle. MECHO then has to find the minimum value h, under which the inequality is still true. This is done by "isolating" h to get an inequality of the form:

$$h \leqslant \tfrac{5}{2} R.$$

That is, the inequalities are manipulated until h is exposed, by itself on the left hand side and the right hand side contains no occurrence of h. One of the main methods of doing this is by repeatedly replacing the outermost left hand side function symbol by its inverse on the right hand side, e.g.

$$x \cdot y \leqslant z \,\&\, x > 0 \rightarrow y \leqslant z/x$$

(see [1] for a description of isolation in the case of equation solving). The problem of finding a minimum value for h to satisfy an inequality, can then be exchanged for the problem of finding the value for h which satisfies an equality, namely:

$$h = \tfrac{5}{2} R$$

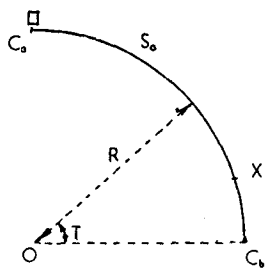


FIG. 5.2.

The great dome problem is handled similarly to the loop the loop problem. That is MECHO tries to prove that motion takes place from C_a to C_b, collects together the provisos, then finds the least T which makes them true. One interesting feature of this process is that we make use of a hypothesis which is actually false, since the block can never reach C_b, without taking off first. For the squeamish an alternative reasoning process is available where we consider motion from C_a to the point of take off (say X above). Similar conditions would be generated on T and the minimum value which makes these conditions true would also be the angle $\widehat{XOC_b}$.

We can easily visualize other processes which could be applied to the provisos, e.g. finding maxima, comparing with an answer expression given in the problem etc. One of the reasons for leaving quantitative testing to the end is that it makes such proviso processing easier to execute. In MECHO the proviso processing procedure is passed down as an argument to QA the main question answering procedure. The other argument is the top level question to be answered. Thus the top level goals of each of the problems is:

sliding block: QA(At (Block, C_d, _mom), eval)

loop the loop: QA(Motion (Block, S_0, C_a, left, _per), Min(h, _ans))

great dome: QA(Motion (Block, S_0, C_a, left, _per), Min(T, _ans))

6. Brief Description of de Kleer's NEWTON

As mentioned above NEWTON also works in two stages: qualitative followed by quantitative reasoning. In our opinion de Kleer's main contribution was in the design of the qualitative reasoner, called the *envisioner.*

The envisioners job was to work out all possible scenarios for the Block, given its initial position and the shape of the path. Thus in the sliding block example the Block might reach the top (point C_a) or it might stop during the course of traversing S_2 or S_3, slide back and oscillate about the lowest point (C_b). The scenarios are built up in the form of a tree, e.g.

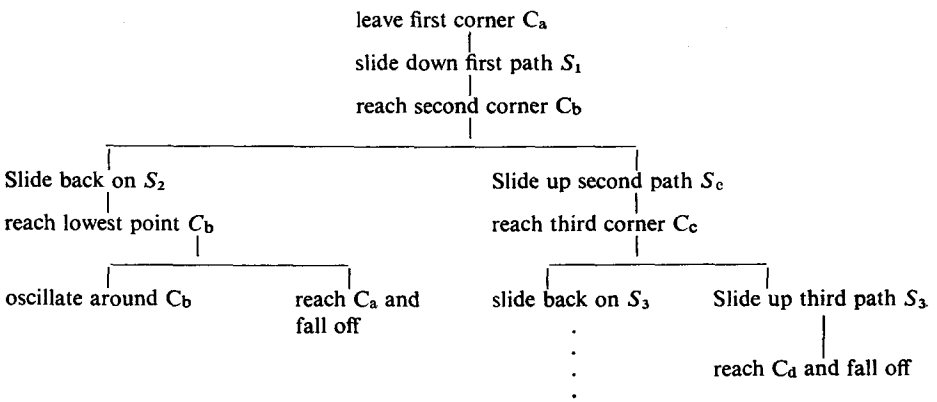


FIG. 6.1. Envisionment tree.

Envisionment is done by a set of 11 production rules. These look at the current state of the Block and local features of the path it is currently on and predict what might happen next. A typical rule is:

$$\text{velocity–u} \;\&\; \text{incline} \;\&\; \text{above} \;\&\; \text{on} \rightarrow \text{slide–u}$$

This means: if the Block is travelling up and is on and above an inclined path then it may continue to slide up.

The envisionment tree is later examined and quantitative tests applied to see which branch of the tree will actually be taken. For instance, the vertical height of the various paths will determine whether the Block reaches the top or oscillates about the lowest point. This is done by associating a different quantitative test with each qualitative ambiguity, e.g. a quantitative test of the velocity is associated with the ambiguity

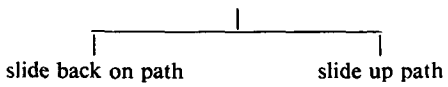


FIG. 6.2. Qualitative ambiguity.

7. A Comparison of NEWTON and MECHO

In applying MECHO to the roller coaster problems we naturally tried for an improved performance, where possible. The main improvement was that MECHO did not generate all the envisionment tree, but only those branches required to answer the question asked. Thus in the sliding block example only the right-most branch of the envisionment tree in Fig. 6.1 was considered. This was done, as described in Section 4, by finding a path between the initial and desired point, and asking only about motion between these points. This more goal-directed envisionment was always an improvement except when the question was a general one like "what happens next" which implies doing a forwards analysis. We did not come across, nor allow for, such questions.

This goal-directed reasoning entailed doing the envisionment process by applying something like de Kleer's production rule system, but in reverse, e.g. trying to prove that the block could slide up the path, by showing that the conditions were right. We first tried simply listing all situations under which sliding was possible, but this proved too cumbersome. Instead we discovered a way of simplifying the problem. Each question about whether motion could take place in a given situation was divided into four sub-questions, namely:

(a) Will the block get started?
(b) Will it run out of steam and stop?
(c) Will it go too fast and fly off?
(d) Will it fall off?

According to the situation these sub-questions may be answered simply on the basis of qualitative information. Alternatively they may require quantitative testing. This is delayed until later by storing the quantitative test (some algebraic expression) as a proviso. These provisos are added up, so that a motion question may generate up to four provisos. As an illustration the procedure for answering sub-question (d) above is explored in detail in Section 10.

This way of organising the qualitative tests seems to be more primitive than de Kleer's. For instance, he has a single quantitative test for the qualitative ambiguity fly-slide-slide illustrated by the following tree and diagram

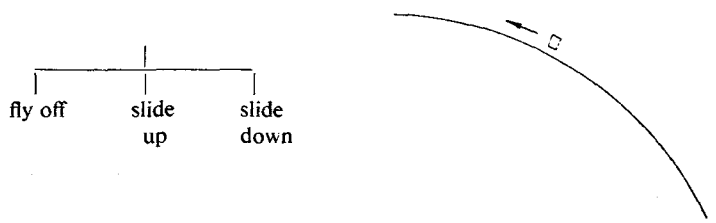


FIG. 7.1.

In MECHO the test for this situation is provided in two parts, by provisos from sub-questions (b) and (c). MECHO's four sub-questions provides a more intimate relationship between the quantitative and qualitative knowledge. A trivial amendment to the program would cause the quantitative testing to be done at the same time as the qualitative testing instead of being delayed to the end.

It is also easy to see how MECHO could be extended to deal with extensions to the roller-coaster domain, e.g. for rings threaded on wires or rough paths. Such extensions are discussed in Section 10. We attribute this flexibility to MECHO's primitive representation, where quantitative tests are automatically built as the situation demands, rather than pre-stored. To update NEWTON to deal with, say, threaded rings, it would be necessary for the programmer to do himself the analysis that MECHO does automatically. Thus the relationship between the two representations is analogous to that between the Huffman/Clowes line labelling [5] and Mackworth's analysis of it [9].

NEWTON's quantitative knowledge was stored as a collection of Frames [11] each of which contained related physical formulae. These Frames were intended to guide the equation extract on process in a sensible way. MECHO on the other hand used the general purpose Marples algorithm for equation extraction which works by backwards reasoning from the unknowns whose values are sought. Despite these apparently different organisations, we have been unable to detect any difference in the manner or efficiency with which equations are actually extracted by the two programs (except that NEWTON seems to have a difficulty with simultaneous equations which was not experienced by MECHO).

8. Representing Paths and Motion

In this and the next two sections we describe in more detail the descriptive terms (i.e. notation) and the procedures used by MECHO to describe and reason about the roller coaster world. We do this to try to make clear exactly how the program works—even down to the actual code in a few carefully chosen cases. To describe the program solely at the less detailed level we have used so far lays us open to the danger of ascribing more agency to the program than is, in fact, justified.

We also hope to promote more discussion of the description terms used in reasoning programs. In recent years the emphasis has been on control structures rather than descriptive power. The choice of terms, however, can make just as big a contribution to successful reasoning. The tradeoffs between different kinds of representation need discussion.

We start with the description of the motion of a particle on a path. This involves making four binary or tenary choices, i.e.

(i) Does the path slope downhill, uphill or is it horizontal?
(ii) Is the path convex, concave or straight?
(iii) Does the particle travel from left to right or right to left?
(iv) Is the particle above or below the path?

In fact, these choices are essentially binary with some degenerate cases. To simplify programming we have chosen the same parity pair left/right for indicating all these choices. Four predicates have argument places for containing either left, right or one of the degenerate cases, they are: End, Slope, Concavity and Motion. For instance, the following situation

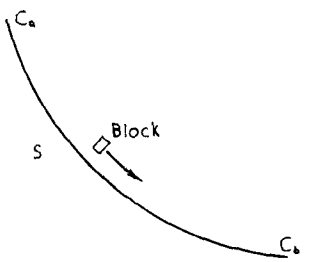


FIG. 8.1.

is described by

End (S, C_a, left) meaning C_a is the left end of S.

Slope (S, left) meaning the left end of S is highest.

Concavity (S, left) meaning, looking from the left end, the left side of S is concave.

Motion (Block, S, C_a, left, P) meaning the Block starts from C_a and travels on the left side of S, looking from C_a.

All these parities can be varied independently, e.g.

End (S, C_a, left)

Slope (S, left)

Concavity (S, right)

Motion (Block, S, C_a, right, P)

would describe the situation

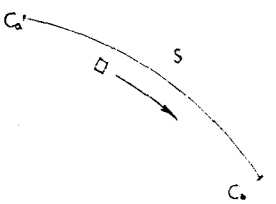


Fig. 8.2.

The choice of predicates may seem a bit arbitrary, but in fact it has been carefully developed to simplify the task of transforming motion descriptions between consecutive paths. Consider the following two situations

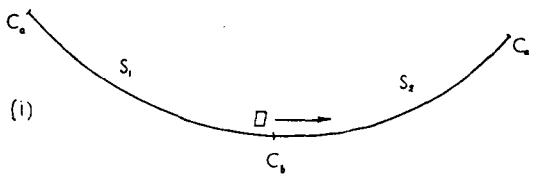


FIG. 8.3.

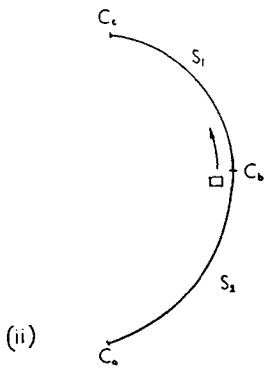


FIG. 8.4.

Using our notation

Motion (Block, S_1, C_a, left, P_1)

transfers to

Motion (Block, S_2, C_b, left, P_2)

in both cases. The fact that the block is now travelling from:

(a) left to right above the path in case (i)

and from

(b) right to left below the path in case (ii)

is handled by the different path descriptions. The task of transferring motion descriptions would have been much more tricky to handle neatly if we had tried to use descriptions (a) and (b) explicitly, to describe the motion.

9. Representing Time

In this section we describe the representation of time in MECHO. All descriptive terms dependent on time, have a time argument place (usually the last place), e.g.

At (Block, C_a, M_a)

or Motion (Block, S_1, C_a, left, P_1).

These time arguments can either be moments or periods. That is a description can be asserted to happen for an instant only, or to hold for some interval of time. Two special moments are associated with each period, namely its initial and final moments. The duration of a period is distinct from the period itself. Thus two different periods can have the same duration. A duration is a quantity, which can be measured in seconds, minutes, hours etc. The notation so far is:

Isa (M_a, Moment).

Isa (P_1, Period).

Initial (P_1, M_a).

Final (P_1, M_b).

Duration (P_1, T_1).

Breakdown (T_1, 2, secs).

To describe a situation in which a period can be divided up into several non-overlapping subperiods, the Partition predicate is used. This arises in the sliding block example where P_0 the period in which the block slides over the whole path is divided into P_1, P_2 and P_3, the periods in which the block slides over the three subpaths. This is represented as

Partition (P_0, [P_1, P_2, P_3]).

The second argument is always a list of sub-periods arranged in time order.

The same predicate, Partition, is used to describe the division of the whole path into its subpaths

Partition (S_0, [S_1, S_2, S_3]).

Here the subpaths are arranged in left/right order. That is the left end of S_0 is one of the ends of S_1.

In the description of the three roller coaster problems (see Section 2), the structure of the paths is given, but the structure of time is not. Only the initial moment is given. The rest of the time structure is erected during the course of solving the problem. For instance, in the sliding block problem the original question is

At (Block, C_d, -mom)?

The inference rule which translates this into a question about motion terminating at C_d (see Section 4), also invents a suitable period, P_0, during which the motion takes place. Since the Block is known to be at C_a at moment M_a, the initial moment of P_0 is asserted to be M_a. A suitable final moment M_d is also invented and the path to be traversed is S_0 since this connects C_a and C_d. (We have chosen the names M_d and P_0 for expositional purposes, the program "gensyms" up some fresh symbols different from this. Our gensym is a bit more readable than most, since you can specify a suitable prefix for the number, e.g. Period 1, vel 1, vel 2, etc.).

Not enough is known about S_0 for the qualitative tests to succeed, so MECHO

considers motion on the three subpaths S_1, S_2, S_3. For this it needs three more periods, so suitable ones are invented, say P_1, P_2 and P_3 and the relationship

Partition (P_0, [P_1, P_2, P_3]) is asserted.

MECHO does all this by mimicking the structure of the paths.

10. The Qualitative Tests

We are now in a position to describe the qualitative motion tests in more detail. These are expressed as a series of PROLOG *clauses* (see [12]). Each clause has the form

$A \leftarrow B, C, D.$

where A, B, C, and D are *literals*. A literal is a symbolic description like

Motion (Block, S_1, C_a, left, P_1) or Slope (S, right)

which may or may not contain variables (variables are indicated by the prefix _), e.g.

Motion (_part, _path, _start, left, P_1) or Slope (_path, right).

The meaning of

$A \leftarrow B, C, D.$

is that to prove A it is sufficient to prove B, C and D in that order.

The literals B, C, D are sometimes proved by a direct database look up, but usually involve further clauses of form $B \leftarrow E, F$, say, in a depth first search. In fact we have considerably modified the default PROLOG depth first search in MECHO: inserting traps to detect hopeless goals and reject them; and tests to decide that the search should be shortened in some cases and lengthened in others. For details of these modifications and the reasoning behind them see [3] or [4].

The main motion clause is

Motion (_part, _path, _start, _side, _per) ←
    Getstarted (_part, _path, _start, _side, _per),
    Nostopping (_part, _path, _start, _side, _per),
    Notakeoff (_part, _path, _start, _side, _per),
    Nofalloff (_part, _path, _start, _side, _per),
    Record (Motion (_part, _path, _start, _side, _per)).

The first four literals after the backwards arrow are responsible for making the four qualitative tests. The last literal asserts the new hypothesis into the database, i.e. an attempt to "prove" Record (some-relation) has the effect of putting some-relation in the database.

The four qualitative tests are all handled in a similar way so we will describe only the last of them, Nofalloff. This test is divided into four cases: two cases which can be settled positively on qualitative information alone; one case which can be settled negatively on qualitative information alone and one case which requires a quantitative analysis. Each case is handled by a separate clause.

The first and simplest case is when the particle is above the path, then the particle cannot fall off the path

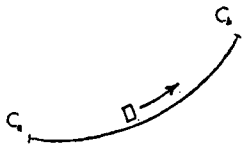


FIG. 10.1.

The clause for this case is:

Nofalloff (_part, _path, _start, _side, _per) ←
Above (_path, _start, _side), ! .

The ! is a control literal which prevents the backtracking mechanism recalculating this positive answer to Nofalloff in the event that later processing fails, i.e. an attempt to "prove" !, causes the search tree to be pruned of branches representing alternative ways of proving Nofalloff. Above (_path, _start, _side) tests that the particle is above the path using End and the value of _side, i.e.

Above (_path, _start, _side) ←
End (_path, _start, _side).

The second case concerns vertical slopes. If a particle is at the top of a vertical path then it is considered not to fall off, but to fall down to the end.

FIG. 10.2.

The representation of vertical paths is a bit messy in our notation. The top of the path is arbitrarily assigned as the left end. The verticality is noticed because the concavity is the degenerate "straight" case and the inclination from the top is 270°. The clause for this case is thus:

Nofalloff (_part, _path, _start, _side, _per) ←
Slope (_path, _start),
Concavity (_path, _),
Incline (_path, _start, 270), !.

The third case concerns convex slopes, where the velocity is sufficient for the particle to stick to the underside by centrifugal force.

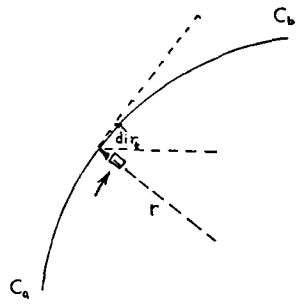


FIG. 10.3.

This case involves quantitative reasoning and so the calculation is delayed until the second stage by asserting a proviso. To calculate this it is necessary to know the velocity of the particle and the radius of curvature of the path. The clause for this case is:

Nofalloff (_part, _path, _start, _side, _per) ←
  Below (_path, _start, _side),
  Concavity (_path, right),
  CC (Vel (_part, _v, _dir, _per)),
  CC (Radius of curvature (_path, _r)),
  Postulate (Proviso ((_v ↑ 2) * Sin (_dir) ⩾ _r * g)), !

The first two literals make sure we are in the correct case. The second two literals recover the numeric quantities needed by the last literal which asserts the proviso into the database. The CC predicate ensures that the request for a numeric quantity succeeds even if this means creating some intermediate unknowns. This predicate is explained fully in [4]. The Postulate predicate is the same as Record except that the assertions are removed on backup.

The last case deals with the situation where the particle is below a non-convex path. In this case it is bound to fall off so the test fails.

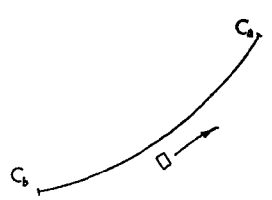


FIG. 10.4.

The clause for this case is

Nofalloff (_part, _path, _start, _side, _per) ←
  Below (_path, _start, _side)
  Concavity (_path, _conc), Diff (_conc, right),
  Record (Fallsoff (_part, _path, _start, _side, _per)),
  !, Fail

The literal Fail always fails thus ensuring that the call of Nofalloff will fail also. The ! guarantees that this failure cannot be reversed by the backtracking mechanism. The first two literals make sure we are in the correct case. The third literal "Record (. . ." puts a record in the database that the particle falls off the path during this period. This information is not currently used, since a suitable opportunity has not yet arisen (see Section 6).

We could easily add another case to Nofalloff designed to deal with rings on wires. Suppose the literal

Threaded (ring, wire, period) means ring is threaded on wire during period

then the extra clause would be

Nofalloff (_part, _path, _start, _side, _per) ←
  Threaded (_part, _path, _per), !.

This should be added first, so that it is tested before the other four cases.

11. Conclusion

In this paper we have described an extension of the mechanics problem solver MECHO to a new range of problems. We have compared the reasoning strategy adopted by MECHO with that used by de Kleer's NEWTON program on the same problems.

MECHO proved capable of solving these new problems provided it was extended to perform the "envisionment" process described by de Kleer. This extension was made and in the process the envisionment process improved by making it more goal-directed. The question of whether motion could take place was answered by breaking it into four sub-questions. This representation seemed more primitive than de Kleer's and thus more easily extended to new problem domains. The machinery developed here to handle the motion of a particle is now being used for a variety of problems in other areas of dynamics.

As de Kleer has said [7] it is important for a problem solver not only to solve hard problems, but to give simple solutions to easy problems. We disagree with de Kleer that this effect can only be achieved with a multiple representation. MECHO will apply the same technique to any problem. In the case of an easy problem this technique will rapidly degenerate and yield an easy solution. For instance, the question may be answered on the basis of qualitative information alone or after a simple calculation. Hard problems will need more work—for instance if friction were involved the algebra required to check the provisos would require detailed consideration of the shape of the curve etc. However, the qualitative knowledge

would still be used and the same basic proof plan would still be followed. Multiple representations are not desirable for their own sake, but only if forced by the nature of the problem, which in this case they are not.

To deal with the equation extraction process it was not necessary to use de Kleer's "Frame like" chunking of the physical formulae, our existing general purpose algorithm proved perfectly adequate. Indeed it was not obvious that the two mechanisms differed substantially in performance. This brings into question the nature of Frame type mechanisms in AI programs. Is there such a thing? Are there different types of Frames mechanisms? Do some of them correspond closely to previously known non-Frame mechanisms? Our suspicion here is that some uses of Frames and the old STRIPS plan formation techniques are much closer than is generally acknowledged (also see [8] and [13].

ACKNOWLEDGEMENTS

My thanks to my colleagues: George Luger, Martha Stone and Bob Welham, for many discussions about the MECHO project, Dave Warren for help with PROLOG and Peggy for her tireless typing.

REFERENCES

1. Bundy, A., Analysing mathematical proofs (or reading between the lines), in: Winston, P., Ed., Proceedings of IJCAI 4, Georgia, U.S.S.R. or DAI Research Report No. 2, Edinburgh (1975) 22–28.
2. Bundy, A., Luger, G., Stone, M. and Welham, R., MECHO: Year one, in: Brady, M., Ed., Proceedings of the 2nd AISB Conference, Edinburgh, or DAI Research Report No. 22, Edinburgh (1976) 94–103.
3. Bundy, A., Can domain specific knowledge be generalized? in: Reddy, R., Ed., Proceedings IJCAI-77, Cambridge, MA (1977) 496.
4. Bundy, A., Exploiting the properties of functions to control search. DAI Research Report No. 45, Edinburgh (1977).
5. Clowes, M. B., On the interpretation of line drawings as simple three dimensional scenes. Internal Memo, Sussex (1970).
6. de Kleer, J., Qualitative and quantitative knowledge in classical mechanics, AI Lab. Technical Report No. AI-TR-352, MIT, Cambridge (1975).
7. de Kleer, J., Multiple representations of knowledge in a mechanics problem solver, in: Reddy, R., Ed., Proceedings of IJCAI-77, Cambridge, MA (1977) 299–304.
8. Fikes, R. E., Automatic planning from a frames point of view, in: Schank, R. and Nash-Webber, B., Eds., TINLAP, Cambridge, MA (1975) 104–107.
9. Mackworth, A. K., Interpreting pictures of polyhedral scenes, Internal Memo, Sussex (1972).
10. Marples, D. L., Argument and technique in the solution of problems in mechanics and electricity, CUED/C–Educ/TRI, Department of Engineering Memo, University of Cambridge, England (1974).
11. Minsky, M., Frame–Systems: A framework for representation of knowledge, AI Lab. Memo No. AIM-306, MIT, Cambridge (1973).
12. Roussel, P., PROLOG: Manuel de reference et d'utilisation, Groupe dIA, Marseille-Luminy (1975).
13. Shank, R. C. and Abelson, R. P., Goals, plans, scripts and understanding: An inquiry into human knowledge structures, Erlbaum Press, N.J. (1977).

Received September 1977