Part V

Mathematical Invention

Chapter 13

Concept Formation

- This chapter describes how a computer can form definitions and conjectures.
- Section 13.1 gives examples of conceptbeing formed.
- Section 13.2 analyses these examples to identify various concept formation processes.
- Section 13.3 formalizes these processes and the knowledge they operate on. Concept formation is described as inference.
- Section 13.4 describes how to control this inference using heuristic search.
- Section 13.5 describes the performance of a program, AM, based on these ideas.

Up to now this book has concerned itself with only one aspect of mathematical reasoning: the proving of theorems. But this is not the only kind of mathematical reasoning indulged in by human mathematicians. In the next two chapters we consider two of these other aspects, namely

- the making of interesting definitions and conjectures
- and the formation of a mathematical model from an informal problem specification.

This chapter is a of a program, AM, written by Doug Lenat [41], for making interesting definitions and conjectures, a process we will call *concept formation*. That reasoning is involved in these activities is illustrated by the following two examples of the sort of thing that AM can do. The first example shows how AM discovered for itself the concept of prime number and the second example shows how AM went on to conjecture the Prime Unique Factorization Theorem.

13.1 How Definitions and Conjectures Can Be Made

- Given times, a function from s [1] of numbers to numbers:

[1] See section 4.3.3 for a definition of bags.

- Find some examples of times, e.g. $times([2,1,2]) = 4$;
- Create inv_times, a function from numbers to sets of bags of numbers, by forming the inverse of times; (AM is using the trick discussed in section 4.3.1 for ensuring that inv_times is a function by forming a set of values.)
- Find some examples of inv_times, e.g. $inv\_times(4) = \{[2,1,2],[1,4],[4],[2,2]\}$; (In order that this set should not be infinite, AM only allowed one occurrence of the unit element, 1, in each bag.)
- Create divisors, a function from numbers to sets of numbers, by composing inv_times and generalized union, i.e. by applying inv_times and then taking the union of all the bags in the resulting set;
- Find some examples of divisors, e.g. $divisors(4) = \{1,2,4\}$;
- Create primes, the set of those numbers whose divisors are doubletons, e.g. $divisors(3) = \{1,3\}$.

Notice how the reasoning in this example consisted of a succession of creating new concepts from old and finding examples of the new concepts. As we will see, the finding of examples, while not strictly necessary to the progression of definitions, does help AM to decide what definitions are interesting to make.

In order to make interesting conjectures, a third sort of operation is required: the checking of the examples in an attempt to discover regularities and patterns. This is illustrated by the next example.

- Given inv_times, a function from numbers to sets of bags of numbers:
- Find some examples of inv_times, e.g. $inv\_times(4) = \{[2,1,2],[1,4],[4],[2,2]\}$;
- Check examples of inv_times and notice that inv_times always contains a bag of primes, e.g. $[2,2] \in inv\_times(4)$;
- Create $prime\_times$, a function from numbers to sets of bags of primes, by restricting the range of inv_times to bags of primes;
- Find some examples of $prime\_times$, e.g. $prime\_times(4) = \{[2,2]\}$;
- Check examples of $prime\_times$ and notice that all examples of $prime\_times$ are singletons;
- Conjecture the Unique Factorization Theorem, namely that $prime\_times$ will always be a singleton.

13.2 Operations of Concept Formation

The operations that were involved in the above two examples of concept formation were: creating new concepts (e.g. functions, sets, etc), finding examples of concepts, checking examples of concepts and making conjectures. We will consider each of these in more detail.

13.2.1 Creating New Concepts

New concepts were created from old ones above, in a variety of ways.

- inv_times was created from times by a process of *inversion*, that is

  $$inv\_times(X) = \{Y : times(Y) = X\}.$$

  As noted in section 4.3.1, this particular way of forming inverses guarantees that they are functions: $inv\_times(X) = Y \leftrightarrow times(Y) = X$ would not be a proper definition unless there was only one such Y for each X.

- divisors was created from inv_times by *composition* with generalized union (gen_union), that is

  $$divisors(X) = gen\_union(inv\_times(X)).$$

- $prime\_times$ was created from inv_times by *restriction*, that is

  $$prime\_times(X) = \{Y : Y \in inv\_times(X) \wedge bag\_of\_primes(Y)\}$$

- In addition, AM can also form new concepts by *coalescing* old ones, i.e. by identifying their parameters. For instance, it forms the function, square, from binary multiplication by identifying its parameters, i.e.

  $$square(X) = X \cdot X$$

- And AM can form new concepts by *structural modification* of definitions of old ones, e.g. it drops a conjunct from the definition of equal to form a new definition - of the concept equal-length.

13.2.2 Finding Examples of Concepts

Having defined a concept AM tries to find examples of it. This enables AM to decide how important the concept is and to suggest new concepts and conjectures that might be made. It uses a variety of ways of finding examples. Some of these are listed below.

- *Instantiating the definition.* For instance, given that AM has an example of times, it might insert it in the definition of inv_times to generate an example of that. This is how the examples above were generated.

- *Look at the concept's generalizations.* For instance, given that AM has several examples of numbers, and that it knows that numbers are a generalization of primes, then each of the example numbers is a candidate for an example prime.

- *Run calculation procedures.* If AM has a function whose range is the concept in question and if there is a calculation procedure associated with that function and if examples of the domain of the function are known, then the procedure can be run to generate examples of the concept. For instance, suppose AM were trying to find examples of sets. It knows that divisors is a function from numbers to sets of numbers and it knows some numbers. If it also has a procedure for calculating divisors then this can be run on the numbers and will produce some example sets.

13.2.3 Checking Examples of Concepts

AM checks these examples periodically for a variety of reasons. We have met one of these, namely to look for regularities and patterns among them.

- This may suggest the creation of a new concept, as the noticing that inv_times always contained a bag of primes led to the definition of $prime\_times$,
- or it may suggest the making of a conjecture, as the noticing that $prime\_times$ was always a singleton led to the conjecture of the Unique Factorization Theorem.

Another role of checking is to vet the examples. Some of the methods of obtaining examples, like the use of examples of generalizations, are liable to error. Thus some 'examples' may need to be rejected or modified.

Lastly, if there are just not enough examples, the checking operation may suggest finding some more. How it does this we discuss below.

13.2.4 Making Conjectures

It is one thing to say 'look for regularities' among the examples, but what does this mean in practice: what sort of regularities should be checked? AM looks for four kinds of regularities and these are listed below.

- One concept is an example of another, e.g. AM notices that all the examples of $prime\_times$ are singletons and hence conjectures that this is always the case.
- One concept is (almost) a specialization of another, e.g. AM notices that all the examples of primes were also examples of odd_primes, except the boundary prime 2. It conjectures that all primes except 2 are odd.
- One concept is (almost) the same as another, e.g. AM notices that an operation it had just defined was the same as times. It merges the two concepts and boosts its interest in the resulting concept.
- One concept is related to a second by a third, e.g. AM notices that bag_of_primes is related to inv_times by set membership, ie inv_times always contains a bag of primes. It restricted inv_times to the range, bag of primes, to define the new function $prime\_times$. Since new relations are defined from time to time, the notion of 'is related' is continually being broadened.

13.3 Formalizing the Knowledge

In order to get a computer to perform these operations on concepts it is necessary to formalise the knowledge involved. In the discussion above we have seen the need for all sorts of knowledge about the concepts of mathematics.

Firstly, since AM's purpose is to make definitions and conjectures it needs to be able to record these. This can be done using predicates, '$definition$' and '$conjecture$', between a particular definition or conjecture and the name of the concept, e.g.

name	*numbers*	*times*
definition	$\{X : bag\_of\_ts(X)\}$	$parallel\_join2(bag, bag, proj2)$
conjecture	$intersect(X, Y) \in numbers$	$associative$
example	0, 1, 2, 3	$times([2, 2]) = 4$
isa	object	operation
generalization	sets	$parallel\_join2$
specialization	primes	square
procedure	–	'a computer program'
type	numbers	$bags\_of\_numbers \mapsto numbers$
in_domain_of	inv_times	–
in_range_of	times	–

Table 13.1: Record of Information about Concepts

$$definition(primes, \{X : divisors(X) \in doubletons\})$$
$$conjecture(primes, prime\_times(X) \in singletons)$$

You might think that '$definition$' should be a function, but AM sometimes records several alternative definitions for a concept, so a predicate is appropriate.

As we have seen, AM also finds examples of concepts, so it needs to be able to record these. We can do this using a predicate, '*example*', between the example and the name of the concept, e.g.

$$example(numbers, 2)$$
$$example(sets, numbers)$$
$$example(times, times([2, 2]) = 4)$$

To help AM find examples of concepts, some additional information must be recorded. For instance, if generalizations of a concept are to suggest candidate examples, then the generalizations of the concept must be stored. If examples are to found by running calculation procedures of functions whose range is the concept, then that function's calculation procedure, domain and range must all be recorded, e.g.

$$generalization(primes, numbers)$$
$$generalization(odd\_primes, primes)$$
$$procedure(times, \text{'}a\ computer\ program')$$
$$type(times, bags\_of\_numbers \mapsto numbers)$$

For reasons of efficiency and presentation, AM recorded all the above information in the form of a table (or see appendix C). It also stored the information redundantly, e.g. the table for numbers records *both* its generalizations and its specializations, the table for times and numbers both record the fact that numbers are the range of times. An example table is given in figure 13.1.

Note that specialization is the inverse of generalization, isa is the inverse of example and in_domain_of and in_range_of are both inverses of . The relation between specialization and isa is analogous to that between set inclusion and set membership, e.g. all primes are numbers and both primes and numbers are examples of objects. AM employs a representation language to make this redundant storage efficient. Knowledge is entered by typing in data structures, such as (13.1), and the representation language automatically adds or changes the redundant inverse .

Figure 13.1: AM's Initial Concepts

13.3.1 Initial Concepts

In the same way that an automatic theorem prover needs a set of axioms as a basis before it can derive any theorems; AM needs a set of initial concepts before it can derive new ones and make conjectures. AM's initial concepts were chosen without any deep analysis of what would be parsimonious or psychologically plausible, but they reflect the sort of knowledge that a preschool child might be expected to have. They are represented diagrammatically in figure 13.1.

In this diagram specialization relationships are represented by single links and example relationships by treble links.

AM has a represention, not only of sets, s and bags, but also of *osets*, which are ordered like lists but, like sets, have no multiple elements. These four kinds of structure form a little quartet.

The operations of inversion, restriction, composition, coalsceing and structural modification are all represented explicitly as concepts. Thus, AM can form the inversion of times (*inv_times*) by composing inversion and times; or the restriction of *inv_times* to primes (*prime_times*) by composing restriction and *inv_times*.

13.3.2 Formalizing Operations

Just as the knowledge about the initial concepts can be represented as axioms, the operations of finding examples and making definitions and conjectures can be represented as rules of inference.

For instance, the fact that all known examples of one concept are examples of another can be represented by the formula,

$$\forall Ex\ example(C1, Ex) \rightarrow example(C2, Ex),$$

where $C1$ is the first concept and $C2$ the second. In this case AM can conjecture that $C1$ is a specialization of $C2$, i.e.

$$conjecture(C1, C1 \subseteq C2).$$

We can represent this operation as a rule of inference, i.e.

$$\frac{\forall Ex\ example(C1, Ex) \rightarrow example(C2, Ex)}{conjecture(C1, C1 \subseteq C2)} \tag{i}$$

AM makes new definitions by inverting, restricting, composing, coalsceing, structurally modifying, etc old definitions. Since all these operations are represented explicitly as initial concepts AM need only use composition. Thus the making of new definitions can be represented by the rule,

$$\frac{\begin{array}{l} definition(C1, D1(X)) \\ definition(C2, D2(Y)) \end{array}}{\exists C3\ definition(C3, D2(D1(X)))}$$

If AM knows of a function, F, from $D \mapsto R$ and it knows an example of concept D and it has a calculation procedure associated with F then it can find an example of R by running the procedure on the example of D. Let $run(Proc, Ex)$ represent the result of running program, $Proc$, on input, Ex, then this operation can be represented by the rule of inference,

$$\frac{\begin{array}{l} type(F, D \mapsto R) \\ procedure(F, Proc) \\ example(D, Ex) \end{array}}{example(R, run(Proc, Ex))}$$

Exercise 43 *AM knows that any example of a concept is an example of a generalization of the concept. Represent this operation as a rule of inference.*

13.4 Concept Formation as Heuristic Search

Thus concept formation can be viewed as a forward search process, by applying the above rules of inference to AM's initial concepts. Like many of the search processes we have considered in this book, there are many possible applications

Figure 13.2: AM's Development Viewed as Growing a Tree of Tasks

which can be made at every stage: if the search process is not to become bogged down in a combinatorial explosion there must be some guidance.

Lenat's solution to this problem was to use meta-level inference [2] (cf chapter 12). AM reasons about the representation of concepts that it has derived so far: 'Which slots in a concept's table have been filled in?', 'Which slots are empty?', etc. On the basis of this reasoning AM suggests new tasks: 'Fill in this slot on this concept.', 'Check the contents of this slot.'. As these tasks are achieved new tasks are suggested and the representation of mathematical concepts is gradually expanded.

To conduct this meta-level inference AM uses a systems of meta-level rules. [3] Examples of such rules are:

> 29. To fill in examples of X, where X is a kind of Y,
> Inspect the examples of Y; some of them may be examples of X as well. The further removed Y is from X, the less cost-effective this rule is.
>
> 48. If the totality of examples of concept C
> is too large to be interesting,
> then consider these three possible reactions:
>
> (i) specialize C;
> (ii) forget C completely;
> (iii) replace C by one conjecture.

We follow Lenat by writing these meta-rules in English. The meta-rules fall roughly into two classes: those, like the first example above, which enable a particular task to be achieved; and those, like the second example, which suggest new tasks. The meta-level inference process can also be visualised as a forwards search of a tree, in which the nodes correspond to the 'check' or 'fill in' tasks and the arcs are labelled by meta-rules.

In order to guide the growth of this task tree (see figure 13.2), AM used the technique of . Each task was assigned a numeric score. As each new task was

[2] Although he did not call it this and, indeed, prefers not to distinguish meta-rules from rules.

[3] Called 'heuristic rules' or 'heuristics' by Lenat.

suggested its numeric score was calculated and this determined its position on an . When AM had finished its current task and was ready to start a new one, it would take off the front of the agenda, the task with the best score.

To calculate the numeric score of a task, AM combines numeric values from various aspects of the task, namely

- A value, R, is inherited from the meta-rule which originally put the task on the agenda, or, R_i, from each distinct meta-rule, if several meta-rules suggested the same task.
- A symbolic 'reason' is also inherited from the meta-rule. AM uses these 'reasons' to ensure that the meta-rule which is fired several times only contributes its R value to a task once. (The 'reasons' are also printed to the user to explain AM's current activity.)
- A value, C, is associated with the concept whose table is being worked on and
- a value, S, with the sort of slot which is being filled in or checked.
- A different value, A, is given depending upon whether the current task is to 'fill in' or to 'check'.

These values are combined to give an overall score for the task using the formula:

$$|\sqrt{\sum_i R_i^2}| \cdot [0.2 \cdot A + 0.3 \cdot S + 0.5 \cdot C]$$

Lenat found that the performance of AM was unaffected by slight variations to this arithmetic formula, suggesting that little hangs on its precision.

The R values of this formula are stored with each meta-rule. The values A and S are global parameters associated with 'fill in', 'check' and the slot names. The C value must be associated with each concept and this is most conveniently done by storing it in that concept's table as the value of a slot called the concept's *worth*.

When a new concept is created it must be assigned a worth value for use in future calculations of a task's score. The worth of a new concept is calculated from the worths of the concepts which define it. Thus, when *inv_times* is created, its worth is calculated from the worths of inversion and times. The formula used to make this calculation is local to the particular concepts involved. Information about how to do the calculation is stored on yet another slot of each concept's table, the *interest* slot. Some of AM's meta-rules can also change the worth of a concept and also the score of a task.

13.5 The Performance of AM

AM started with the 76 concepts in figure 13.1. By running its 242 meta-rules to suggest and achieve about 256 'fill in' and 'check' tasks, it gradually filled out the representation of these concepts and defined and filled out new concepts. After about an hour of cpu time AM had about 300 concepts. Of the new concepts defined about 25 were really interesting and about 100 more were acceptable. The remainder were worthless. The amount of information stored about the original concepts approximately tripled.

Among the interesting new concepts developed by AM were: numbers, prime numbers and maximally divisible numbers. This latter kind of number is the dual of the prime numbers: each maximally divisible number having more divisors than any smaller number. Lenat originally dismissed maximally divisible numbers as uninteresting, but later took them more seriously and managed to prove some theorems about them. Finally, Lenat discovered that his theorems had previously been discovered by the famous Indian mathematician, Ramanujan.

Among the interesting conjectures made by AM were: de Morgan's Law, the Prime Unique Factorization Theorem and Goldbach's Conjecture. The definition of primes and the proposing of the Unique Factorization Theorem were given as examples in section 13.1. In [41] the trace of AM which is claimed to be the proposing of Goldbach's Conjecture is in fact a weaker conjecture, namely that every even number is the sum of some number of primes, rather than just two primes. This weaker conjecture is trivial to prove, e.g. $6 = 2 + 2 + 2$, etc. AM did propose Goldbach's Conjecture on a previous run and Lenat was misled by the similarity of the printout.

Lenat performed various experiments with AM: changing the way that the tasks were selected and varying the initial concepts. AM was remarkably robust under changes of the method of task selection. Slight changes produced almost no difference in AM's behaviour; more radical changes slowed down the development of interesting new concepts by a factor of three, but did not prevent that development.

AM's meta-rules were shown to be robust by exchanging the set-theoretic initial concepts for geometric ones, i.e. the concepts of point, line, angle, triangle, etc. AM was still able to make reasonably interesting new definitions and conjectures, e.g. the congruence and similarity of triangles. Although some of the geometric results had a strong arithmetic flavour.

In all the experiments, AM's concept formation slowed down after about an hour of cpu time. The problems were twofold.

- As the number of concepts grew and the amount of information stored about each one increased, the computer's memory was gradually exhausted.

- The quality of the new information being discovered began to degrade.

Lenat attributes this degradation of quality to the fact that to discover interesting information about a concept, meta-rules are required especially geared to that concept. The new concepts discovered by AM are more and more specialized, but it cannot also discover new meta-rules geared to these special concepts. It must, instead, rely on very general purpose rules and this becomes an increasing liability.

13.6 Summary

It is possible to represent the formation of new mathematical concepts and the making of conjectures as a process of inference. This inference process is extremely explosive, but can be guided by meta-level inference, which in turn can be guided by heuristic search. The proof that this is a practical proposition is embodied in a computer program, AM.

Starting from a base of very general concepts, AM was able to discover many interesting arithmetical concepts and make several interesting conjectures. AM is guided by a simple numeric model of 'interestingness', in which the 'worth' of a concept is inherited from the concepts from which it was defined. AM did not use any notion of proof to confirm its conjectures or to determine interestingness, but this would be an interesting extension of the AM ideas.

Further Reading Suggestions

[39] is a short introduction to AM. [41] is longer and more detailed account. [40] is a more general discussion of the issues.

Chapter 14

Forming Mathematical Models

- This chapter describes how mathematical problems, stated in English, can be solved by translating them into equations.
- Section 14.1 describes and criticises the simple technique of keyword replacement used in the STUDENT program.
- Section 14.2 describes the intermediate representation used, in the MECHO program, as a staging post between the English and algebraic representations of a problem.
- Section 14.3 describes the role of inference in bridging the gap between the information in the intermediate representation and that required to form the equations.
- Section 14.4 describes the formalization of physical laws.
- Section 14.5 describes the process, used by MECHO, to form equations from the intermediate representation.
- Section 14.6 summarises the use of meta-level inference in MECHO.

In this chapter we consider another 'non-theorem proving' aspect of mathematical reasoning: the formation of mathematical models from an informal specification. In particular, we will consider how the description of a problem in English can be translated into a system of mathematical formulae, e.g. equations.

14.1 Keyword Replacement

One class of problems, sometimes called *Algebra Word Problems*, (Not to be confused with the word problems of group theory, ring theory, etc. defined in section 9.6.3.) admit of a particularly simple solution. Various key words in the English statement of the problems can be translated directly into mathematical symbols. Phrases which described objects, called *noun phrases*, can then be translated into variables, so that each clause gives rise to an equation.

The translation rules are as follows:

- the keyword 'is' is replaced by '$=$';
- noun phrases in which a number is followed by a unit, e.g. '5 inches', are replaced by the multiplication of the number and the unit, e.g. '5.inches';
- some special phrases, e.g. 'X percent less than', have special translations, e.g. '$(100 - X)/100$';
- questions and commands, e.g. find X, what is X, have the special effect of marking X as a variable to be solved for;
- any remaining noun phrases are replaced by variable names: identical or similar noun phrases being replaced with the same variable.

Consider, for example, the problem,

> The price of a radio is 69.70 pounds. If this price is 15 percent less than the marked price, find the marked price.

We will translate this problem statement clause by clause.

The first clause is,

> The price of a radio is 69.70 pounds.

'is' translates to '$=$' and '69.70 pounds' to '$69.70 \cdot pounds$'. This leaves the noun phrase 'The price of a radio' which we will replace by a new variable, *price*. Thus the whole clause translates to,

$$price = 69.70 \cdot pounds$$

The second clause is,

> this price is 15 percent less than the marked price

Again 'is' translates to '$=$' and the special phrase '15 percent less than' to '$.85 \cdot$'. This leaves the noun phrases 'this price' and 'the marked price'. A procedure is needed which can recognise that the first of these refers to 'the price of a radio' while the second is a new variable, 'marked-price'. The problem this procedure has to solve is called the *noun phrase reference problem*. We will be meeting it again. Assuming that this procedure is provided, the translation of the second clause is,

$$price = .85 \cdot marked\_price$$

The final clause is,

> Find the marked price.

This is a command, and so has the effect of marking '*marked_price*', as a variable to be solved for.

The equations can now be solved for '*marked_price*'. Since they are linear simultaneous equations a procedure to solve them can be easily provided. This will yield the answer,

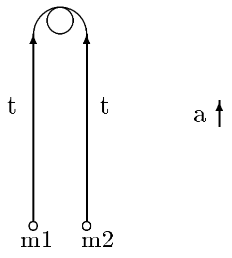

Figure 14.1: A Pulley System

$$marked\_price = 82 \cdot pounds$$

To finish off the process this equation should be translated back into English by applying the rules above in reverse. Thus '=' should be replaced by 'is'; '$82 \cdot pounds'$ by '82 pounds' and '$marked\_price$' by 'The marked price'. This will give the clause,

> The marked price is 82 pounds.

Exercise 44 *Translate into an equation the clause: 'The distance between London and Edinburgh is 400 miles'.*

A program based on these techniques, called STUDENT, has been built by Danny Bobrow [6]. It can solve a wide range of simple algebra word problems, like that above. It does, however, have its limitations, for instance,

- it can only deal with a limited subset of English: verbs like 'is', 'find', etc. connecting a small range of noun phrases;
- its procedure for solving the noun phrase reference problem is successful only in simple cases, where the similar phrases shared some common string of words;
- it can only solve sets of linear simultaneous equations.

These problems could have been solved by a process of gradual improvements and extensions, however, STUDENT had a more fundamental limitation, and this is illustrated by the following example.

> Two particles of mass $m1$ and $m2$ lbs are connected by a light string passing over a smooth pulley. Find the acceleration of the particle of weight $m2$ and the tension of the string supporting it.

Figure 14.1 is a diagram of the situation being described here.

There are noun phrases in this problem specification which cannot be translated into equations, or any other kind of algebraic formula, e.g.

> Two particles .... connected by a light string passing over a smooth pulley.

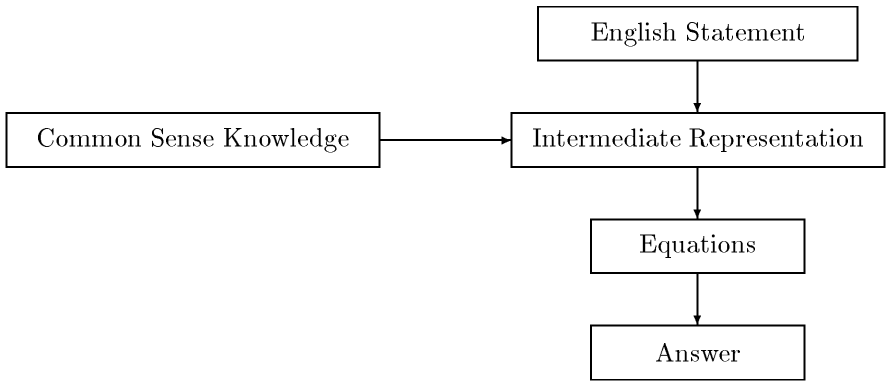

Figure 14.2: Strategy for Problem Solving

But this is not an optional part of the specification which can be ignored: it sets the scene for the algebraic information. STUDENT has no way of representing such information.

Furthermore, the problem cannot be solved without a piece of 'scene setting' information, which is not provided in the problem statement, but is expected to be assumed by the problem solver, namely that the string is hanging vertically on either side of the pulley. The problem solver is expected to deduce this from his knowledge of gravity and freely hanging objects or to assume it from his familiarity with pulley systems in Mechanics problems.

Thus in order to solve problems like the pulley one above an automatic problem solver must be able to represent 'scene setting' information provided in the English statement and common-sense information about gravity, etc. A representation of the problem is needed which is intermediate between the English statement and the sets of equations. The English statement must be translated first into this intermediate representation, where the common-sense knowledge can act on it and fill out the problem statement; and then this intermediate representation must be translated into sets of equations. This strategy is diagrammed in figure 14.2.

A Mechanics problem solving program, called MECHO, has been built by the author and others [12]. It can solve the pulley problem above and a variety of problems from other areas of Mechanics. This chapter is a rational reconstruction of the MECHO program.

We will deal mainly with the formalization of the intermediate representation and of the common-sense knowledge, and with the procedure for extracting equations from the intermediate representation. The problem of translating from English into an intermediate representation is called *Natural Language Understanding*. The study of this problem is a separate area of Artificial Intelligence. It has come a long way from the simple keyword replacement techniques described earlier and requires a book all of its own. The interested reader is refered to [64, 46].

14.2 Formalising the Intermediate Representation

The scene described in the pulley problem above consists of a configuration of four objects. The intermediate representation must represent this configuration and associate various physical quantities with them. There is an element of arbitrariness about any representation of such configurations, that is there are many choices of predicates, functions and constants to be made and most of the alternatives are equally acceptable. Where a particular choice is important we will point this out below.

The four objects – the two particles, the string and the pulley – can be represented by the constants, $part1$, $part2$, str and $pull$, respectively. We will want to record what sort of object each one of these constants represents, and we can do this with a predicate, isa, e.g.

$isa(part1, particle)$
$isa(part2, particle)$
$isa(str, string)$ and
$isa(pull, pulley)$

Every object has exactly one , so we could have made isa a function. In fact none of the relationships in the intermediate representation will be represented by functions. There are several reasons for this design choice, one of which was first described in section 4.3.2: that using no functions causes the arguments to be essentially boolean. We will meet further reasons in section 14.6, where we will also see how the missing functional information can be reintroduced into the problem solving process.

Now we must represent the relationships between these objects. These can be represented at various levels. At the highest level the four objects form a very common configuration in Mechanics problems – a pulley system. Thus the relationship might be represented by a single predicate, $pulley\_sys$, e.g.

$pulley\_sys(pull, str, part1, part2)$.

At a lower level the configuration can be represented as a series of contact relations between the point objects and parts of the string. In problems of this kind the particles and the pulley can be idealised as 0 dimensional objects and the string as a 1 dimensional object. We need to specify three points on the string: the two ends and the point of contact with the pulley. This can be done with the predicates end and $contact\_pt$, i.e.

$end(str, end1, left)$
$end(str, end2, right)$
$contact\_pt(str, cpt)$

Contact relations can then be asserted using the predicate $contact$, i.e.

$contact(end1, part1)$
$contact(end2, part2)$
$contact(cpt, pull)$

To complete this low level picture the shape of the string needs to be described. It is divided into two parts. The part supporting $part1$ we will call $bit1$

Figure 14.3: The Inclination of the String Parts

and the part supporting $part2$ we will call $bit2$. The relationship between these bits and the whole string can be captured with the predicate, $partition$,

$$partition(str, \langle bit1, bit2 \rangle)$$

where the second parameter is a disjoint list of subparts of the first, and the order of this list indicates the left-right order of the subparts.

Finally, $bit1$ and $bit2$ must be asserted to be straight with vertical inclination. This can be done with the predicates *incline* and *concavity*. Both of these were developed to express the shape of curved 1 dimensional objects: hence *incline* gives the angle between the tangent at a point and the x-axis; and *concavity* would normally say on which side a curve was concave, but here takes the special value of $stline$.

$$incline(bit1, end1, 90)$$
$$incline(bit2, cpt, 270)$$
$$concavity(bit1, stline)$$
$$concavity(bit2, stline)$$

Here the inclination of $bit1$ is directed from $end1$ to cpt and the inclination of $bit2$ is directed from cpt to $end2$ (see figure 14.3). This asymmetry is introduced to simplify the task of representing the asymmetry of the direction of travel of the two particles, i.e. if one goes up then the other goes down.

In practice both high and low level relationships are required. Some of the lower level relations can be recovered by translating the English statement of the problem, e.g. the *contact* and *isa* relations. These relationships can be used as cues for the higher level, $pulley\_sys$, relationship and the remaining low level relationships, e.g. the string shape relations can be inferred from this. The high level, $pulley\_sys$, proposition is an example of a *schema*. This word was chosen following Bartlett [2] who first used it to describe the behaviour he observed when people recognised an object: a few properties of the object are perceived and these are used to cue an object schema, whose remaining properties are then verified by further perception.

In the case of Mechanics problems the schemata correspond to standard configurations (pulley systems) or situations (motion in a straight line). Schema cueing allows the unstated assumptions of the problem (called 'house rules' in [44]) to be brought in. In addition to the shape of the string these will also include assumptions about physical quantities, e.g. that the string is inextensible unless otherwise stated. The schema cueing mechanism has to allow all these default assumptions to be overruled by contradictory information in the problem statement. For instance, if a non-zero coefficient of extensibility for the string were given then the default assumption that the string is inextensible would be overruled.

This completes the description of the configuration, so we go on to consider how physical quantities, masses, accelerations, etc., can be associated with the various objects. For instance, a mass of $m1$ lbs must be associated with $part1$ and $m2$ lbs with $part2$. A particle has only one mass, but this may be expressed in various units, lbs, stones, grams etc. Translation between units may be necessary during the solution of the equations, so the mass of a particle is expressed in two bites: one asserting a unique mass and the other expressing this in terms of a number of units, i.e.

$$\begin{array}{l} mass(part1, mass1) \\ measure(m1, lbs, mass1) \\ mass(part2, mass2) \\ measure(m2, lbs, mass2) \end{array} \tag{i}$$

A similar technique can be used to represent the tension of the string.

$$\begin{array}{l} tension(bit2, tsn) \\ measure(t, lbs.ft/sec^2, tsn) \end{array} \tag{ii}$$

and the acceleration of the particles

$$\begin{array}{l} accel(part2, acc, 90) \\ measure(a, ft/sec^2, acc) \end{array} \tag{iii}$$

but since the coefficients of friction and extensibility are dimensionless quantities no measure proposition is required.

$$\begin{array}{l} friction(pull, 0) \\ extensibility(str, 0) \end{array}$$

Exercise 45 *Represent the information that the mass of a stone is 5 oz and its acceleration is 32 ft/sec^2 downwards.*

Finally, the status of the various physical quantities has to be represented so that MECHO knows what it is supposed to solve for in terms of what. This is done using two meta predicates, *sought* and *given*, which specify which quantities are to be solved for and which may appear in the answer, respectively. Thus the pulley problem above yields the propositions,

$$\begin{array}{l} sought(acc) \\ sought(tsn) \\ given(mass1) \\ given(mass2) \end{array}$$

14.3 Bridging the Gaps

The propositions above constitute the intermediate representation of the problem. This is extracted from the English problem statement by natural language understanding programs. But in order to solve the problem we have seen that additional information is required: we need to bridge the gap between the information provided in the problem statement and that needed to solve the problem.

We have already seen one 'bridging' technique – the cueing of schemata. But this technique does not solve all the problems. For instance, to solve the problem it is necessary to resolve forces about both particles. To do this MECHO needs to know the acceleration and mass of both particles and the tensions in the bits of string which they are connected to. This information is recorded above for $part2$ (equations (iii), (i) and (ii)), but vital parts of it are missing for $part1$, because they were not mentioned in the English statement of the problem.

However, the missing information about $part1$ can easily be recovered by inferring it from the information about $part2$. All that is required are inference rules of the form:

$$\begin{array}{l} tension(Bit2, Tsn) \wedge \\ partition(Str, \langle Bit1, Bit2\rangle) \wedge \\ extensibility(Str, 0) \wedge \\ pulley\_sys(Pull, Str, P1, P2) \wedge \\ friction(Pull, 0) \\ \qquad\qquad \rightarrow tension(Bit1, Tsn) \end{array}$$

which says that the tension in one bit of the string of a pulley system is the same as the tension in the other bit, provided that the string is inextensible and the pulley is frictionless.

$$\begin{array}{l} accel(Part2, Acc, Dir2) \wedge \\ pulley\_sys(Pull, Str, Part1, Part2) \wedge \\ extensibility(Str, 0) \wedge \\ end(Str, End1, left) \wedge \\ incline(Str, End1, Dir1) \\ \qquad\qquad \rightarrow accel(Part1, Acc, Dir1) \end{array}$$

which says that the acceleration of one particle in a pulley system has the same magnitude as the other, provided that the string is inextensible.

The schema cueing mechanism can also be implemented as an inference process in two stages, namely:

- the cueing of the schema;
- and the use of this to infer default information.

The cueing stage consists of the application of axioms like:

$$\begin{array}{l} isa(Part1, particle) \wedge isa(Part2, particle) \wedge \\ isa(Pull, pulley) \wedge isa(Str, string) \wedge \\ end(Str, End1, left) \wedge end(Str, End2, right) \wedge \\ contact\_pt(Str, Cpt) \wedge \\ contact(End1, Part1) \wedge contact(End2, Part2) \wedge \\ contact(Cpt, Pull) \\ \qquad\qquad \rightarrow pulley\_sys(Pull, Str, Part1, Part2) \end{array}$$

Thus when sufficient evidence has accumulated about the existence of a pulley system the high level proposition asserting the existence of one can be deduced.

There may be alternative sets of sufficient evidence which would confirm the existence of a pulley system. This is reflected by having additional axioms which differ from the one above by having different conditions on the left hand side of the implication arrow.

When axioms of the above sort have been used to assert the existence of a schema it can be used to infer default information. This is done with axioms like:

$$pulley\_sys(Pull, Str, Part1, Part2) \rightarrow extensibility(Str, 0)$$

and

$$pulley\_sys(Pull, Str, Part1, Part2) \rightarrow \exists Bit1\ \exists Bit2\ \ partition(Str, \langle Bit1, Bit2 \rangle)$$

The fact that this is *default* information means that these axioms must be treated in a special way by the inference system: they must not be used if contradictory information is already known. The MECHO inference system has such axioms specially marked. Before it uses them it tries to see if a contradictory conclusion is already known. In the first example above it sets up the subgoal

$$extensibility(Str, X)$$

if this succeeds then the default axiom is never used.

Exercise 46 *Draw the Lush Resolution search tree for the following clauses, using (vii) as top clause.*

(i) $accel(Part2, Acc, Dir2) \wedge$
$pulley\_sys(Pull, Str, Part1, Part2) \wedge$
$extensibility(Str, 0) \wedge$
$end(Str, End1, left) \wedge incline(Str, End1, Dir1) \wedge$
$\rightarrow accel(Part1, Acc, Dir1)$

(ii) $pulley\_sys(Pull, Str, Part1, Part2)$
$\rightarrow extensibility(Str, 0)$

(iii) $\rightarrow pulley - sys(peg, rope, man, bucket)$

(iv) $\rightarrow accel(bucket, quickly, up)$

(v) $\rightarrow end(rope, knot, left)$

(vi) $\rightarrow incline(rope, knot, down)$

(vii) $accel(man, quickly, down) \rightarrow$

14.4 Extracting Equations from the Intermediate Representation

We have seen how all the information in the problem statement can be represented as propositions and how gaps in this information can be bridged by inference rules and schema cueing, but how can a mathematical model in the form of equations be formed from this intermediate represention?

Physical formulae, like the 'resolution of forces' formula, $F = MA$, must clearly be brought to bear. But what is the relation between the F, M and A of the formula and the propositions of the intermediate representation? M is the mass of whatever object we are resolving about, e.g. if MECHO decides to resolve about $Object$ then

$$mass(Object, M)$$

The situation for A is more complicated: A depends both on the acceleration of $Object$ and on the direction, Dir, in which MECHO decides to resolve, i.e.

$$A = Acc \cdot \cos(Dir - Dir1)$$
$$\text{where } accel(Object, Acc, Dir1)$$

And the situation for F is most complicated of all since F is the sum of all forces acting on $Object$.
MECHO represents this situation using a meta-predicate, $is\_formula$,

$$\begin{aligned} is\_formula(\ & resolve, \\ & situation(Object, Dir), \\ & F = M \cdot A, \\ & mass(Object, M) \wedge \\ & accel\_compt(Object, A, Dir) \wedge \\ & sum\_forces(Object, F, Dir)) \end{aligned}$$

The first parameter is the name of the physical formula. The second parameter is the situation in which the formula is being used, in this case the object being resolved about and the direction of resolution. The third parameter is the formula itself and the fourth parameter relates the second to the third by a series of propositions. When it has decided what formula to use and in what situation, MECHO can fill in the blanks in the formula – the F, M and A – by setting up the propositions in the fourth parameter place as subgoals, i.e. by trying to satisfy $mass(Object, M)$, where the variable $Object$ is already matched to some particular constant, say $part1$. There is already a proposition in the intermediate representation which will match $mass(part1, M)$, but this is not so for $accel\_compt(part1, A, 270)$. There is a gap between the information provided in the problem statement and the information needed to solve the problem.

As already discussed, inference is used to bridge this gap. The first axiom MECHO will need is

$$\begin{aligned} & accel(Object, Acc, Dir1) \wedge \\ & A = Acc \cdot \cos(Dir - Dir1) \\ & \qquad \rightarrow accel\_compt(Object, A, Dir) \end{aligned}$$

and then the axiom relating the accelerations of the two particles in a pulley system discussed earlier.

To satisfy $sum\_forces(part1, F, 270)$ a series of inferences are made, in which the various kinds of forces which might be involved, e.g gravitational, reactions, friction, etc. are considered in turn. The assumption is made that only those forces are acting which we can deduce are acting – that we have complete information about the situation.

14.5 Choosing Equations

We have seen how MECHO can form an equation from the intermediate representation, provided it knows what formula to use and what situation to use it in, but how can it make this choice?

The guiding principle is to choose equations which will help it to solve for the *sought* quantities in terms of the *given* ones. Suppose, for instance, it decides to form an equation which solves for the acceleration, acc. This quantity provides a lot of clues as to what formula and situation might be used. For instance, not all the formulae relate accelerations to other quantities. Those that do not can be ignored. acc is the acceleration of a particular object, $part2$, and this particle is moving in direction 90. This suggests the situation, $situation(part2, 90)$. The equation chosen should be independent of any which have already been chosen to solve for other quantities. If any choices remain we would prefer an equation which does not introduce any new unknowns .

MECHO uses all this information when choosing an equation. Its procedure can be summarized as follows.

- Analyse the quantity currently being solved for. Find out what sort of quantity it is and what situation it is defined in, e.g. the acceleration in direction 90 of $part2$.
- On this basis select a particular formula and situation.
- Fill in the blanks in the formula using inference if necessary.
- If any of these inference steps would require the introduction of some new quantity, e.g. the velocity of $part1$, then remember this node as a continuation node, but go back and try another formula and/or situation.
- If an equation cannot be formed without introducing new quantities then introduce some by resurrecting a continuation node. Mark the new quantities as sought unknowns and solve for them later.
- Check that the new equation is independent from any that were formed earlier.
- Repeat the process on any remaining sought quantities.
- Use the measure propositions to translate all the quantities into numbers in compatible units.

The result of this procedure is a list of equations, e.g.

$$m1 \cdot g - t = m1 \cdot a$$
$$t - m2 \cdot g = m2 \cdot a$$

together with a record of which equations solve for which unknowns. MECHO feeds these to PRESS (see chapter 12) for solution.

14.6 Meta-Level Knowledge

In the last three sections we have seen how MECHO cues schemata, makes inferences and chooses equations. All these processes involve choices: choices which could cause combinatorial explosions unless they are controlled. To control them MECHO uses the technique of meta-level inference described in chapter 12. Using this technique necessitates the representation of meta-level knowledge about the expressions of the intermediate representation, physical formulae, etc.

We have already met one such piece of meta-level knowledge. The is-formula predicate is used to represent knowledge about the way that the variables of physical formulae are related to objects in the intermediate representation. Further examples are the predicate *kind* which relates a physical quantity to its type and its original defining proposition, e.g.

$$kind(acc, acceleration, accel(part2, acc, 90))$$

This information is used to analyse the quantity being solved for. The quantity type is used, in conjunction with the meta-predicate, *relates*, to draw up a short list of physical formulae which involve quantities of this type. That is,

$$relates(acceleration, \langle resolve, const-accel1, const-accel2, ...\rangle)$$

says that the formulae in the list, $\langle resolve, ...\rangle$ all involve accelerations. The defining proposition is further analysed by a complex process of meta-level inference to help suggest what situation to focus on when forming equations, e.g. the proposition $accel(part2, acc, 90)$ will be broken apart to get $part2$ and 90. In an application of resolve, these might suggest resolving about $part2$ in direction 90 degrees, but the analysis will also range wider and suggest resolving about objects in contact with $part2$ etc.

To control the making of inferences, various kinds of meta-knowledge are used. One of the most powerful techniques uses the functionality of some of the predicates, i.e. the fact that some parameters of a predicate are functionally dependant on others. For instance, in $mass(part1, m1)$, $m1$ is functionally dependant on $part1$, and hence $m1$ has the function properties of and , i.e. every particle has exactly one mass (see section 4.3.1). MECHO uses this knowledge in three ways.

- *Uniqueness test*: before attempting to prove that $mass(part1, m2)$, say, MECHO looks to see that it does not already know $mass(part1, m1)$, where $m1 \neq m2$. If it does then the original goal is abandoned as unsatisfiable.

- *Back-up Prevention*: if the goal $mass(part1, M)$ is satisfied by $\{m1/M\}$ then further choice points are deleted, since $part1$ can have no other mass.

- *Controlled Creation*: if the goal $mass(part1, M)$ should fail then MECHO can choose to force satisfaction by creating a Skolem constant, say m, defined to be the mass of $part1$. The decision as to whether to do this depends on the state of the problem solving process, e.g. whether an attempt has already been made to form an equation not containing intermediate unknowns.

This incorporation of the uniqueness and existence properties of functions into the inference control mechanism, more than compensates for the loss of functional information implied by MECHO's use of predicates to represent functional relationships like the mass and acceleration of a particle. The extra power comes from MECHO's ability to use the functional knowledge to control what inferences are made and what terms are introduced into the Herbrand Universe.

14.7 Summary

We have seen that it is possible to model aspects of mathematical reasoning other than the proving of theorems, in particular, the formation of a mathematical model from an informal specification of a problem.

When this informal specification is in a natural language like English, a natural language understanding program is required. This translates from English to some more formal representation of the problem. In the case of very simple algebra word problems this formal representation can be sets of equations. But when the problem specification includes 'scene setting' information an intermediate representation is needed.

To bridge the gap between this intermediate representation and the knowledge required to form equations it is necessary to cue schemata and to make inferences. These inferential processes involve choices, so some control mechanism is required. The technique of meta-level inference, described in chapter 12, can be adapted to this task.

Further Reading Suggestions

[6] is a good introduction to STUDENT. [12] is a good introductory account of MECHO. [11] is a more detailed account.