View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Edinburgh Research Explorer

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Doing arithmetic with diagrams

Citation for published version:
Bundy, A 1973, 'Doing arithmetic with diagrams'. in Proceedings of IJCAI-3.

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Preprint (usually an early version)

Published In:
Proceedings of IJCAI-3

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 20. Feb. 2015

DOING ARITHMETIC WITH DIAGRAMS

by

Alan Bundy
Department of Computational Logic,
University of Edinburgh,
Edinburgh, Scotland.

ABSTRACT •

A theorem prover for part of arithmetic in described which proves theorems by representing them in the form of a diagram or network. The nodes of this network represent 'ideal integers', i.e. objects which have all the properties of integers, without being any particular intoger. The links in the network represent relationships between 'ideal integers'. The procedures which draw these diagrams make elementary deductions based on their built-in knowledge of the functions and predicates of arithmetic. This theorem prover is intended as a model of some kinds of human problem-solving behaviour.

DESCRIPTIVE TERMS.

Theorem Proving, Heuristic Method, Representation, Psychological Modelling, Semantic Network, Arithmetic, Ideal Integer, Logic, Semantic: Tableaux.

0. MOTIVATION.

The objective of this work is to investigate human problem-solving behaviour by trying to simulate it on a machine. The main source of information about human behaviour is still self-observation and so the author has deliberately chosen a domain in which he has some experience, arithmetic (see 1). Arithmetic hus the additional advantage that as one of the oldest branches of mathematics it is rich in proof techniques and easily stated but difficult theorems (e.g. Format's Last Theorem). Also an efficient arithmetic theorem prover is likely to find applications in Program Correctness Proof3.

1. INTRODUCTION.

This is a report *of* work in progress. It describes what is, to the best of the author's knowledge, a new kind of automatic theorem prover, called SUMS (a System which Understands Mathematical Symbols). SUMS does not explicitly use axioms or rules of inference to prove theorems. Instead it represents the candidate theorem as a network (or diagram) in which the nodes are the property lists of arithmetic terms and the links describe relationships (e.g. = <, |) between them. Statements are asserted by adding new links and proved by accessing the diagram. Knowledge about arithmetic is built into the procedures that draw the diagram so that when links are added to it, elementary deductions are made (and more links added) automatically.

The domain of SUMS is the Elementary Theory of Natural Numbers (i.e. the arithmetic of the non-negative integers). At present it can only handle the classes of terms and formulae defined below;

(a) An arbitrary skolem constant or natural number is a term.

(b) If A and B are terms then SUC(A) ; Pre(A); A + B; A - B and A x. B are terms.

(c) If A and B are terms then $A = B$; $A \neq B$; $A \leq B$; $A > B$; $A \geq B$; $A < B$; $A|B$ and $A \nmid B$ are atomic formulae. (Note that for each predicate symbol, S, there is a symbol $\not S$ such that $\neg S(x_1,\dots,x_n) \leftrightarrow \not S(x_1,\dots,x_n)$.)

(d) An atomic formula is a formula.

(e) If P and Q are formulae then $\neg P$; $P \& Q$; $P \vee Q$; $P \rightarrow Q$ and $P \leftrightarrow Q$ are formulae,

Where Suc and Pre are the successor and predecessor functions.

i.e. $Suc(A) = A + 1$ and $Pre(A) = A \dot{-} 1$

and $\dot{-}$, the modified difference function, is just ordinary subtraction modified so that its result is always a natural number

$$\text{i.e.}\quad A \dot{-} B = \begin{cases} A - B & \text{if } B < A \\ 0 & \text{if } A \leq B \end{cases}$$

2. EXAMPLES.

Before describing the representation of the diagram and the procedures which draw it, it is best to illustrate how the theorem prover works with some examples.

(i) The Associative Law of Addition.

$$a + (b + c) = (a + b) + c \qquad (1)$$

SUMS always starts by constructing nodes for 0 and 1 and drawing the appropriate links between them (e.g. $0 \neq 1$, $0 \leq 1$, $1|0$, etc.) New nodes are always appropriately related to 0 and 1. However, since these links play no part in the proof which follows they have been omitted for clarity.

The machine then creates nodes for each of the subterms involved in (1) and draws in the appropriate links.

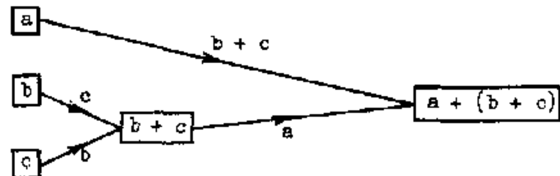

Fig. 1.

The links in the above diagram are distance and direction links, e.g. 'a' is a distance 'b + c' less than 'a + (b + c)'. When a new link is added to a node it is compared with the old links to see if any conclusion can be drawn.

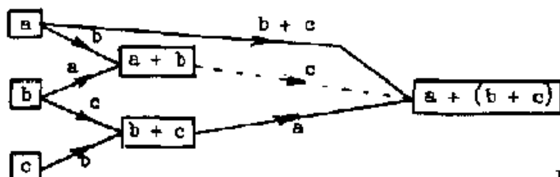

Fig. 2.

For instance in Fig. 2 the link from 'a' to 'a + b' is compared with the link from 'a' to 'a + (b + c)', which causes the distance label 'b' to be compared to the distance label 'b + c'. The node 'b' is found to be a distance 'c' less than the node 'b + c' in the diagram; so SUMS deduces that 'a + b' must be a distance 'c' less than 'a + (b + c)', and draws in the appropriate link (dotted).

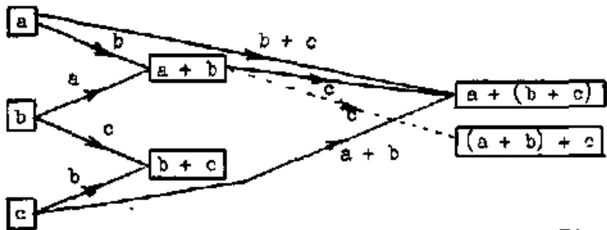

Fig. 3.

Finally the node for '(a + b) + c' is created but as the first link is being drawn (dotted in Fig. 3) it is compared with the link connecting 'a + b' to 'a + (b + c)' and found to be of equal length.

SUMS deduces that $a + (b + c) = (a + b) + c$ and achieves this by *merging the two nodes and not adding the new link* (Fig. 4).

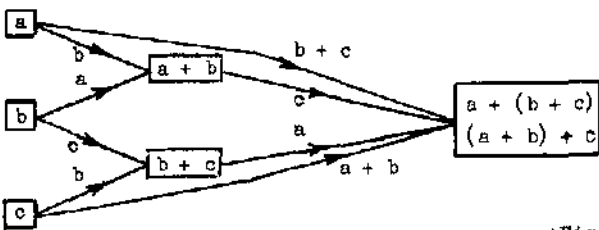

Fig. 4.

SUMS now proves the theorem by accessing the diagram and discovering that these two terms have the same node.

(ii) Commutativity of Maximum.

$$a + (b \dot{-} a) = b + (a \dot{-} b) \qquad (2)$$

If we define $\max(x,y)$ as $x + (y \dot{-} x)$, (2) becomes the commutative law of maximum.

As before SUMS creates nodes for each of the subterms involved in (2), except that this time it considers two cases. SUMS always draws terms of the form $A \dot{-} B$ by first interrogating the diagram to see if either of the cases $A \leq B$ or $B \leq A$ hold. Otherwise it duplicates the diagram, asserts $A \leq B$ in one and $B < A$ in the other and then tries to prove the theorem in both cases.

So in this example the final diagrams are:

Case $a \leq b$

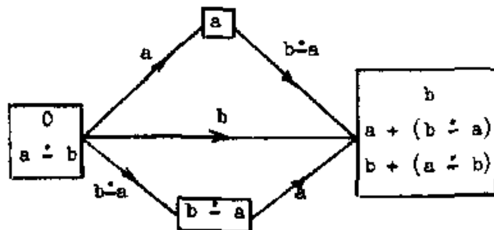

Fig. 5.

Case $b < a$

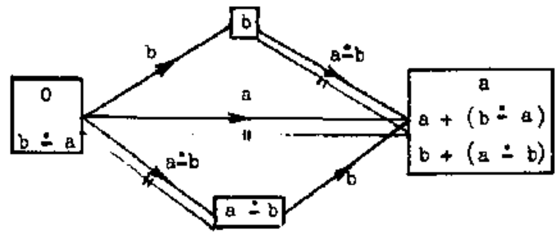

Fig. 6.

A link of the form

Fig. 7.

means that A = B. These links are not needed in this example but are included for completeness. They occur because asserting B < A involves asserting both B < A and B / A.

3. THE REPRESENTATION OF THE DIAGRAM.

The diagrams are represented using lists and POF-2 records. We find records convenient, but they could, if necessary, be replaced by Lists or arrays.

Each diagram is a record with 6 components (or slots) in which information is recorded. These slots are called and contain:

(a) Title: a formula in list notation which says which case the diagram represents, e.g. [LESS A B].

(b) Contra: a 1 or 0 according as the diagram has or has not been found contradictor;'.

(c) Network: a list of the entries (defined below) in the diagram.

(d) Nought and Unity: 2 redundant slots pointing to the, frequently used, entries for 0 and 1 respectively.

(e) File: a list of terms which were not represented in the diagram when wanted, but which SUMS may decide to represent later.

Every term is represented, in the diagram, by a unique record with 3 slots, called an entry. These slots are called and contain:

(a) Label: The name of the term being represented in list notation, e.g. [ADD A [DIFF B A]]

(b) Proplist: A record of type 'node' which contains all the links to the other entries. When two terms are made equal all the information from the first term's 'proplist' (property list) is put into the second term's 'proplist', which then replaces the first wherever it appears. Thus equal terms share 'proplists' which justifies the adoption of the name 'nodes' for the 'proplist' rather than the entries. This method of dealing with equality is more convenient than using equality links, because: it makes checking for equality more efficient (the 'proplists' must be EQ) ; the equality axioms are automatically incorporated, and all the information about a single node is kept in one place.

(c) Replace: A slot used in the copying of diagrams.

STRUCTURE OF DIAGRAM

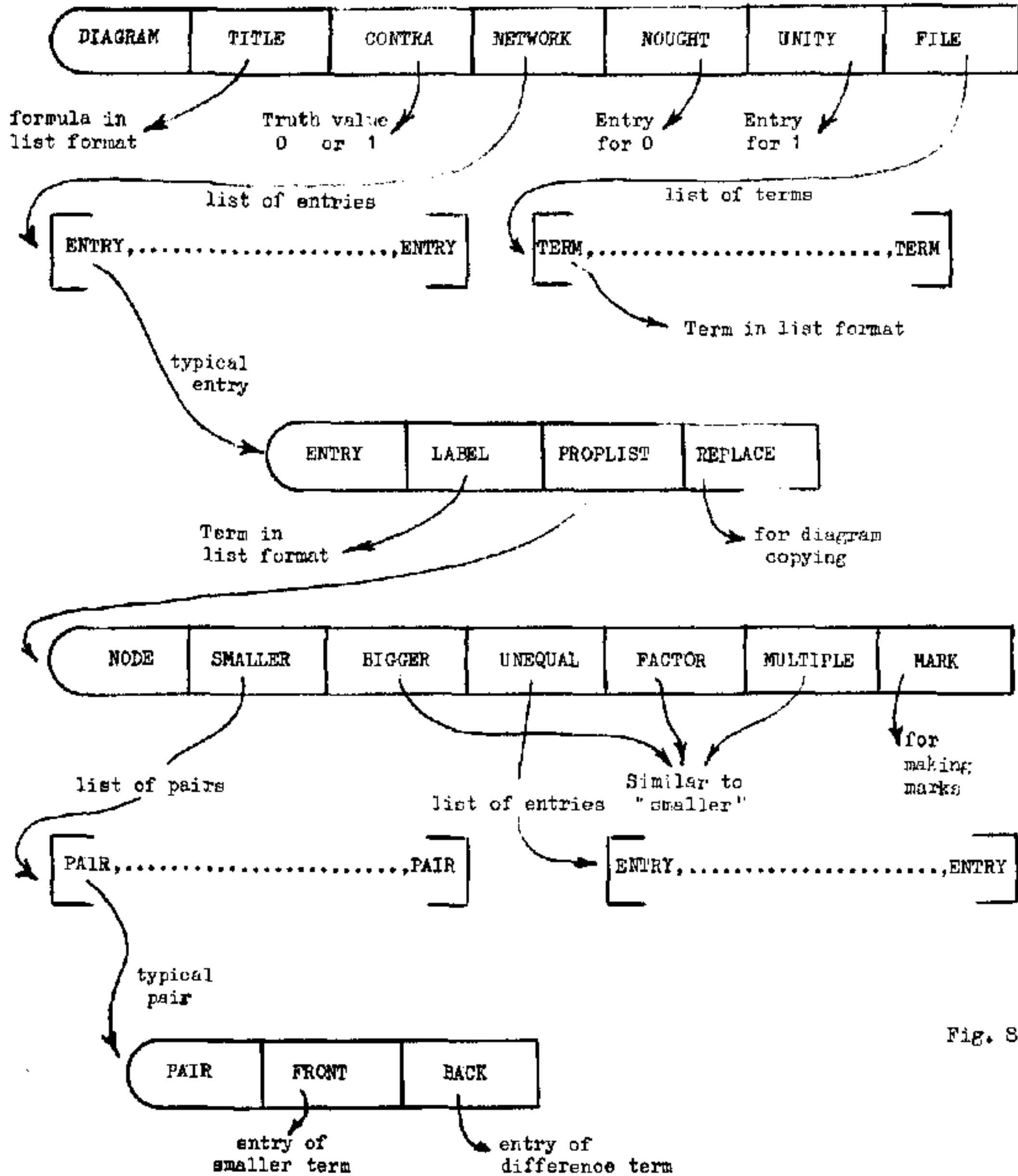

Fig. 8.

FIRST ELEMENT,.......,LAST ELEMENT

notation for lists

DATAWORD | FIRST COMPONENT | .... | LAST COMPONENT

notation for records.

KEY

Each node is a record with 6 slots called and containing:

(a) Smaller: A list of pairs of entries. The front of each pair represents *a* term smaller than the present one; the back represents the distance between them.

(b) Bigger: Similar to (a) except that the front of each pair is the entry of a term bigger than the present one.

{c) Unequal: A list of entries representing terms known to be unequal to the present one.

(d) Factor: Similar to (a) except thnt the front of each pair is the entry of a term which exactly divides the present one, and the back is the entry representing the quotient produced.

(e) Multiple: Similar to (d) except that the front of each pair is the entry of a term which the present term exactly divides.

(f) Mark; A slot used for markers when measurements are being made in the diagram.

4. THE PROGRAM.

In order to distinguish functions in SUMS from functions in Arithmetic, we will call the former procedures.

There arc four classes of top level procedures arranged roughly in a hierarchy. They are:

(a) Logical Procedures: which analyse the original formula and *decide which atomic* formulae *to* assert and which to prove.

(b) Drawing Procedures: which analyse the terms in the formulae, draw them in the diagram and assert relationships between them.

(c) Asserting Procedures: which make relationships between terms hold in the diagram by adding links.

(d) Interrogating Procedures: which discover whether relationships between terms hold by accessing the diagram.

(i) The Logical Procedures.

After the initial diagram has been created all negations are eliminated from the candidate tliooreni by passing them down to, and absorbing them .into, the atomic formulae. i.e. each negated atomic formula -S(x1,...xn) is replaced by #(x1.... xn). The candidate, P, is then passed to the procedure 'Prove' whose description follows.

To prove P:

(a) If P is a conjunction of atomic formulae, its terms are drawn in the *diagram, which* is then interrogated to see if P is true.

(b) If P is of the form Q & R then a copy of the diagram is made. Q is proved in the first diagram and R in the second.

(c) If P is of the form Q v R (or Q - E) then we assert -Q (Q) in the diagram and prove R.

(d) If P is of the form QR then a copy of the diagram is made. In the first diagram, Q is asserted and R is proved; in the second R is asserted and ft proved.

To assert a formula in the diagram the following procedure is called.

To assert P:

(a) If P is already true in the diagram then the procedure is exited.

(b) If P is already false the diagram is closed, by making it contradictory, and the procedure exited.

(c) If F is atomic, its subterms are drawn, and it i-. made true in the diagram by calling an appropriate asserting procedure.

(d) If P is of the form Q & R then both Q and R are asserted.

(e) If P is of the form Q v R, (Q - R) then:

If Q is false (Q is true), R is asserted;
If R *is* false, Q is asserted [-Q is asserted);

Otherwise a copy of the diagram is made and Q is asserted (- Q is averted) in the first diagram and R in the second,

(f) If P is of the form Q <--> R then:

If Q is true, R is asserted;
If R is true, Q is asserted;
]f Q is false, - R is asserted;
If R is false, - Q is asserted;

Otherwise a copy of the diagram is made and Q and R are asserted *in the first* diagram *and -Q and* -R *am* asserted in the second.

The referee has pointed out the similarity of these procedures to Beth's Semantic Tab]eaux (see *').* The main difference;; are that:

(a) SUMS cannot yet handle arbitrary quantification. Semantic Tableaux provides some valuable clues as to how to correct this defect.

(b) in SUMS the left (valid) and right (invalid) columns of Beth's tableaux have been combined in u single diagram, making a neater and more powerful procedure.

(c) At present SUMS does not assert the negation of the theorem to be proved, a practice which would certainly lead to an increase in power.

(d) Before dividing into *2* cases SUMS checks the present diagram to see if either of the new diagrams would be contradictory (see the checks in (e) and (f) above, and in the drawing procedure for - below). This limits the number of cases to be considered, but is not perfect, and unnecessary cases are sometimes considered.

(e) Most importantly, the assertion of an atomic formula, in SUMS, is not just the passive addition of the formula to a list, but an act which may have wide repercussions within the diagram.

(ii) The Drawing Proceduren.

Between them the drawing and asserting procedures are mainly responsible for drawing the diagram. Their general philosophy is to limit the number of nodes in the diagram to those representing terms mentioned in the candidate theorem, but to draw as many links

CONTROL FLOW CHART

(NOT TO BE TAKEN TOO SERIOUSLY)

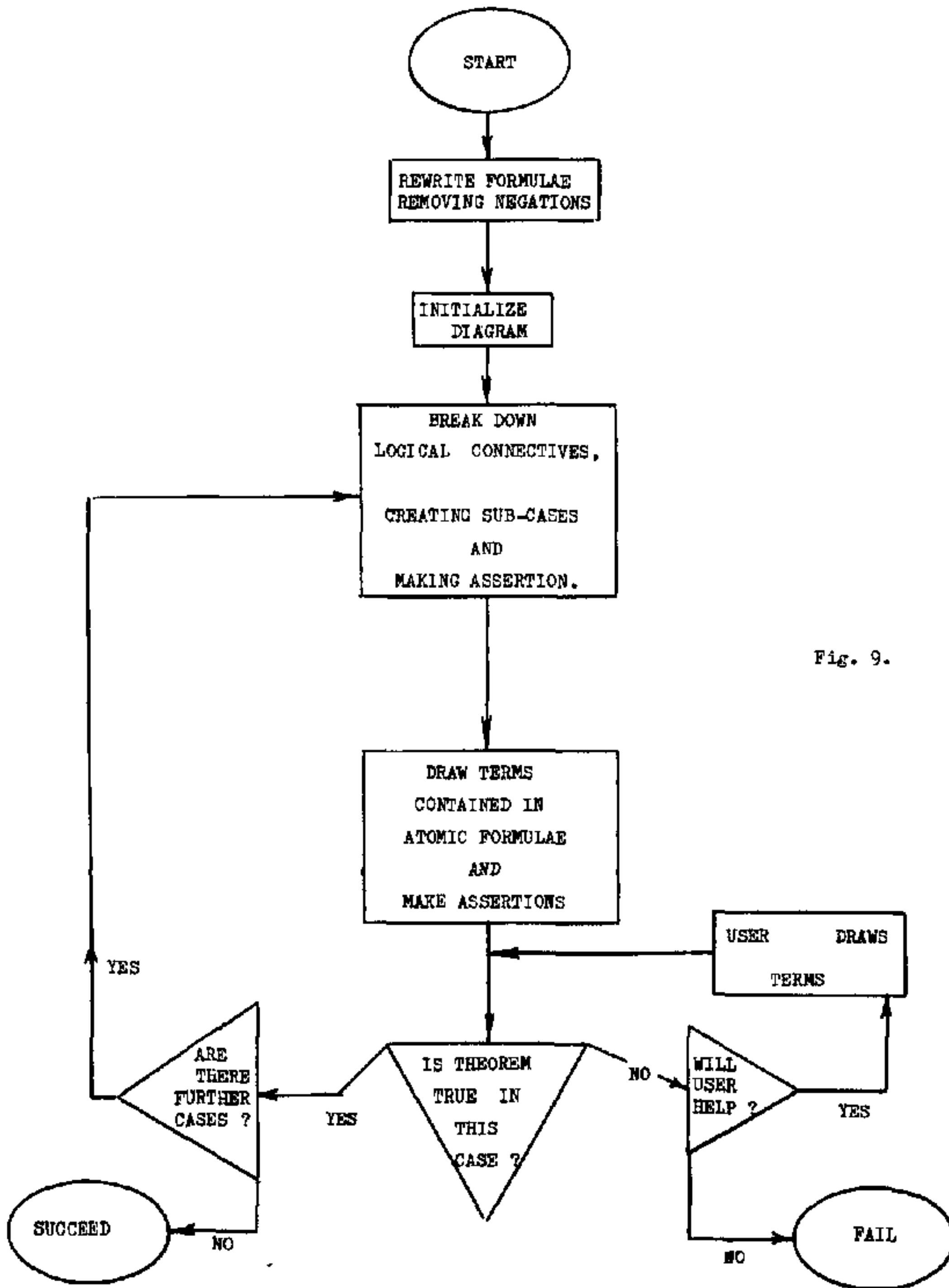

Fig. 9.

between these nodes as possible. Thus the diagram is prevented from exploding, since only a finite number of links is possible between a finite number of nodes, but quite sophisticated relationships are deduced between the nodes that are represented. This kind of heuristic is quite common in mathematics for instance as advice to students proving theorems in Euclidean geometry, to deduce as much as possible about a diagram without constructing any further points (ef. 3), The only exceptions to this rule are;

(a) the creation of nodes for 0 and 1, often needed because they are the identity elements for addition and multiplication.

(b) the limited creation of nodes needed to label links between existing nodes, e.g. if $a \leq b$ is one of the premises of the candidate theorem and $b - a$ is not already represented, then we represent it.

(c) the creation of new nodes as part of a proof strategy (sec Section 6).

(d) the suppression of those clauses of links which do not appear to be necessary to prove the theorem, e.g. multiplicative links in a theorem only involving addition. This has not yet been implemented.

(a) and (b) only produce a strictly limited number of extra nodes.

We now turn to a description of the procedures themselves. A formula is drawn in the diagram by creating entries for each of its subterms which do not already have them, from the bottom up, and then passing these entries *to* procedures which extract information from the term's function symbols. There is one of these drawing procedures for each arithmetic function symbol, and the procedure receives every term beginning with this symbol. It assumes that the term's arguments have already been represented. If necessary it can use the interrogation procedures to discover the state of the diagram before using its knowledge of arithmetic to assert relationships with the asserting procedures.

As an example we describe the procedure for terms of the form A - B:

```
If A ≤ B then make A ∸ B equal to 0.
else if B ≤ A then make A ∸ B less than or
equal to A by an amount B.
else
  Copy the present diagram.
  If P is the title of the old diagram make
  P & A ≤ B the title of the old one and
  P & B < A the title of the new one.
In the old diagram
  create an entry for B ∸ A if one does not
  already exist.
  make A less than B by amount B ∸ A
  make A ∸ B equal to 0.
In the new diagram
  make A ∸ B unequal to 0
  make A ∸ B less than A by amount B.
end.
```

These drawing procedures are sufficiently close to the original function definitions to make one feel optimistic about having the machine 'learn' a new function by constructing its drawing procedure from its definition. The creation of an entry for a term involves constructing a new entry, inserting the term in its 'label' slot and a new node in its 'proplist' slot and adding this entry to the diagram's 'network' slot. The property list is then filled with as much information as possible. For instance, all terms are made bigger than 0 and divisible by 1. In addition if the term is a natural number we compute and record, subject to the normal limitations, its relationships to all previously created natural numbers. e.g. 2 is made bigger than *1* by an amount 1 and unequal to 0 and 1 etc. This is still non-explosive if the number of nodes is strictly controlled, and ensures that all nodes relevant in a particular situation can be conveniently recovered.

(iii) <u>The Assorting Procedures</u>.

There is an asserting procedure for each predicate symbol. These procedures update the property lists of the entries in the diagram thus creating new links in the network. Before and after these links are created various 'antecedent' procedures (cf. 4) are called, which examine the local state of the diagram to see whether any further facts can be deduced. In particular a contradiction may be detected in some diagram and then registered by assigning 1 to the diagram's 'contra' slot.

These antecedent procedures contain a pood deal of the arithmetic knowledge which is built into the theorem prover. They are constantly evolving as new information is included or neater ways. of achieving the same effect are discovered, und so it. would be misleading to over-emphasise their present state by describing one in detail. However, wo can give some examples of their action.

Suppose that an asserting procedure is making A divide C with quotient B (i.e. $A \times B = C$). Let A, B and C be represented in the diagram by the entries Ae, Be and Ce respectively, and in general if *X* is *a term,* let Xe be the entry representing it. Before adding any new links between Ae, Be and Cc, SUMS compares,for instance, Ce and Be with each of the pairs, (Ee.I)e) , in the 'multiple' slot of Ae. We illustrate this situation in the following diagram, in which the double arrows indicate that the links are multiplicative.

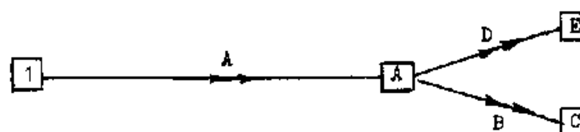

Fig. 10.

If say B is equal to D then because:

$$(A \times B = C \ \& \ A \times D = E \ \& \ B = D) \rightarrow C = E \qquad (3)$$

C is made equal to E and the procedure is exited.

Adding the new links involves consing the pairs

(Ae.Be) and (Be.Ae) onto the list contained in the 'factor' slot of Ce;
(Ce.Ae) onto the list contained in the 'multiple' slot of Be and
(Ce.Be) onto the list contained in the 'multiple' slot of Ae.

The redundancy of this method *of* link stor*age ia* justified by the convenience of having all the information about a particular node stored in that node.

After the links have been added, for instance, Ce and Be are again compared with each of the pairs (Ee.De) in the 'multiple' slot of Ae. If, say, A is known to be unequal to 0 and C exactly divides E with quotient P.. then B is made to divide D exactly with quotient F. Because:

$$(A \times B = C \;\&\; A \times D = E \;\&\; A \neq 0 \;\&\; C \times F = E) \rightarrow B \times F = D \quad (4)$$

Again if, say, B ia smaller than D by an amount P and there ia a term, G, equal to A x F represented in the diagram then D is made less than E by an amount C. Because:

$$(A \times B = C \;\&\; A \times D = E \;\&\; B + F = D \;\&\; A \times F = G) \rightarrow C + G = E \quad (5)$$

Formulae like (3), (4) and (5) are the mainstay of the antecedent procedures. Although apparently arbitrary, they often have a simple geometric interpretation and turn out to be surprisingly powerful in antecedent mode. In general the asserting and antecedent procedures fall into the following pattern:

(a) Check whether the relationship already holds. If so, exit without adding any more links.

(b) Check whether the opposite relationship already holds. If so, declare a contradiction and exit.

(c) Otherwise add all new links.

(d) Now do a lot of very cheap (in time and space) checks to discover the current local state of the diagram. Deduce and assert as many new facts as possible without creating any new nodes in the diagram.

(iv) The Interrogating Procedures.

There is an interrogating procedure for each predicate symbol. They are used to settle all questions about the relationships between terms and they do thi3 by accessing the diagram. Because they are used so frequently the present procedures are designed, for efficiency, to do very little searching. For instance, the procedure which asks whether A is equal to B returns true if and only if A and B have the same node stored in their 'proplists'. The procedure which asks whether A is less than B starts at A and climbs up through the 'bigger' slots, marking its passage, until either it comes to B and returns true or it exhausts all possibilities and returns false. It would be possible to design procedures which tried a lot harder than this, but their use would have to be selective.

There is a subclass of the interrogating procedures, called the measuring procedures, and there are measuring procedures corresponding to most of the function symbols. For instance, the measuring procedure for + takes as arguments two entries, Ae and Be, (say) representing the terms, A and B, and returns true if and only if it finds an entry for a term equal to A + B, in which case it also returns this entry. It does this by looking in the 'bigger' slot of Ae for a pair whose back is equal to Be and returns the corresponding front if successful. This procedure may well succeed even if A + B is not represented. e.g. B + A may be represented or in a diagram where a $\leq$ b, A is a and B is b - a we may return the entry for b. If the procedure does not succeed the term A + B is stored in the diagram's 'file' slot where it may be used at a later date as grounds for constructing a node for A + B.

5. RESULTS.

74 of the 86 theorems which have been attempted so far were drawn from , which contains about 700 formal theorems. The remaining 12 were invented to test SUMS ability to deal with particular numbers (see numbers 14 15 and 20 below). SUMS can now prove 64 of them with times ranging from 0.125 sees, of C.P.U. time (number 13) to 45.94 secs. of C.P.U. time (number 12). There follows a selection of 15 of these 64 successfully proved theorems together with 5 of the 22 failures.

(i) Successes.

	Theorem	Time Spent (in secs. of C.P.U. Time)
1.	$a + (b + c) = (a + b) + c$	3.5
2.	$a + (b \dot{-} a) = b + (a \dot{-} b)$	4.625
3.	$a \dot{-} (b + c) = (a \dot{-} b) \dot{-} c$	13.0
4.	$(a + b) \dot{-} c = (a \dot{-} c) + (b \dot{-} (c \dot{-} a))$	23.38
5.	$0 < Suc(a)$	1.063
6.	$(b < a) \vee (a \leq b)$	0.9375
7.	$a \neq 0 \rightarrow (Pre(a) < a \;\&\; a = Suc(Pre(a)))$	5.375
8.	$a < b \leftrightarrow Suc(a) < Suc(b)$	11.81
9.	$(a + b < c) \leftrightarrow (a < c \dot{-} b)$	11.63
10.	$a \times (b \times c) = (a \times b) \times c$	3.813
11.	$a \times (b + c) = a \times b + a \times c$	4.188
12.	$1 \dot{-} (a + b) = (1 \dot{-} a) \times (1 \dot{-} b)$	45.94
13.	$a \mid a$	0.125
14.	$2 + 2 = 4$	3.5
15.	$a + a = 2 \times a$	3.188

Failures.

	Theorem	Time Spent (before automatic termination)
16.	$a \leq b \;\&\; c \leq d \rightarrow a + c \leq b + d$	8.938
17.	$2 \leq a \;\&\; 2 \leq b \rightarrow a + b \leq a \times b$	19.69
18.	$a \mid b \;\&\; a \mid c \rightarrow a \mid (b + c)$	5.813
19.	$a \mid b \;\&\; a \mid Suc(b) \rightarrow a = 1$	9.25
20.	$(a + 2) \times (a + 3) = a \times a + 5 \times a + 6$	23.38

Most of these theorems are quite difficult to prove from the Peano axioms of arithmetic. For instance, the normal proof of number 1 involves two induction steps. Although normalization algorithms might be used to prove some of them, it is extremely difficult to see how such a method would deal with number 4, number 7 or number 12.

6. GOALS.

In order to prove the remaining 636 theorems in , SUMS requires abilities outside its present scope. Plans for adding some of these abilities are well advanced and only await implementation; others are further in the future. A description of these plans follows.

(i) Further Arithmetic.

In order to tackle more sophisticated arithmetic formulae, SUMS needs to be able to deal with the quotient; remainder; exponential; nth odd prime; greatest common divisor; is-a-prime and other functions and predicates. All these abilities require a drawing procedure for each new function, an asserting and an interrogating procedure for each new predicate and its negation, and new slots in the property lists. These will be added soon.

(ii) Quantification.

At present SUMS is not capable of dealing with any formula that would contain existential quantifiers if it were written in Prenex normal form. Beth's Semantic Tableaux[2] and Robinson's Resolution[5] both provide valuable clues as to how to correct this deficiency. There follows a proposed solution.

Firstly, replace any call of the procedure 'To prove P', described in Section 4, by a call of 'To assert -P' and

then the theorem is proved when all the diagrams become contradictory. Non-contradictory diagrams will be used to suggest counter-examples or will be candidates for more strenuous proof strategies (see next sub-section).

Secondly, update the 'to assert P' procedure by adding:

(g) If P is of the form V x Q(x) then an entry of type 'variable' is created for x. x is added to a local variable list 'vars' and Q.(x) is asserted.

(h) If P is of the form 3x Q(x) then an entry of type 'skolem function[1] is created for x. x is made dependent on all the variables :in the current value of 'vars' and Q(x) is asserted.

Thirdly, during the course of the proof we may 'substitute[1] for a variable x, any term A, which is not dependent on x. This involves making a copy of any entry containing x, replacing each occurrence of x in each copy by the term A and making equal any terms with the same label. The ability to substitute needs to be handled with the same circumspection as the other proof strategies mentioned in the next sub-section.

(iii) <u>Proof Strategies</u>.

With the additions mentioned above, SUMS should be able to prove a wide class of straightforward theorems. However, it performs no search and always terminates wit!) or without a solution after it has represented the candidate theorem. To prove more sophisticated theorems SUMS needs to have and to know how to use, a store of proof strategies (PLANNER consequent theorems[4]).

For instance, it may decide to try mathematical induction, to construct some new terms, to make a substitution, to divide into cases, to use some previously proved theorem or to set up some intermediate sub-goals. All these abilities are relatively straightforward to apply once a decision about how to use them has been made, but deciding on what to use and how is difficult. Our intention, therefore, is to implement this stage in two parts. The first task will be to make SUMS interactive. Procedures will be written to correspond to the mathematical use of such instructions as: Try induction on...; Consider the term...; Substitute... for...; Consider the cases...; Use...; First prove...: etc. Then SUMS will be led through the proofs of some moderate]y difficult theorems in Number Theory. The second task is to use this experience to evolve a language for classifying candidate theorems and selecting and applying suitable proof strategies.

Some progress has now been made on the first task. If SUMS fails to prove a theorem it asks for help in constructing new terms and offers the contents of the diagram's 'file' slot as evidence. The user may then tell SUMS to construct some terms before continuing with the proof. For instance, after representing a < b & c < d -» a + c < b + d (number 16 in Section 5) in the diagram, and failing to prove it, SUMS asks for help and suggests the constructions: a + d; c + b; (d - c) + b and (b - a) + d. If the user orders a + d or c + b to be constructed a proof is immediate. If instead he chooses (d - c) + b or (b - a) + d, help is requested again. On the initial trial run through the 22 previous failures (see Section 5), 16 were proved using this method - 7 on the basis of the proffered evidence alone. Of the remaining 6 theorems 3 were abandoned and 3 produced stack overflow because the length of deductions got too deep.

7. <u>CONCLUSION</u>.

This theorem prover uses an analogical representation of arithmetic and lets this representation do all the work. In the traditional representations of mathematical theories, e.g. a set of axioms in a resolution theorem prover, it is quite easy to assert a contradictory set of formulae without this situation being readily detected. In the present representation assertions once made are readily accessible and any new assertion causes all sorts of conclusions to be drawn and asserted so that contradictions are usually Spotted quickly.

In [3] Gelernter achieved a similar effect by usi.ng two representations: a syntactic one for proving theorems with rules and axioms, and a diagram to guide this proof. His 'syntax computer' discovered groups of sub-goals which would imply the present goal, and his 'diagram computer' vetted those sub-goals, rejecting those that were false in the diagram. The diagram was also used to <u>prove</u> certain very basic sub-goals {see 3 *p.* 42 No. 7*)* but this is a risky process, because although precautions were taken to prevent spurious coincidences entering into the diagram, there were still statements, true in the diagram, hut not provable from the hypothesis of the candidate theorem. The analogue of Celernter's diagrams in arithmetic would be to substitute particular numbers for the variables in some subgoal, and then compute the truth or falsity of the resulting formula.

SUMS arose from an attempt to represent in the machine the concept, fairly common among mathematicians (see for instance [2]) , of an 'ideal (or typical) integer', i.e. an object with all the properties of an integer (e.g. being equal to, less than, not equal to or a divisor of some other interer) but which is not any particular integer (e.g. 3, 13 or 53). In a 'diagram' composed of 'ideal integers' no spurious coincidences arise, so that anything true in the diagram is provable from the hypothesis of the theorem and anything false is not provable. So this diagram can be legitimately used for rejecting <u>and proving</u> sub-goals. Of course something may be neither true nor false in the diagram.

Does SUMS prove theorems or does it check their validity? It certainly does not produce proofs in a formal logical system, but neither does it exhaustively test the candidate theorem in some model. Nor, of course, does the practising mathematician confine himself to cither of these techniques. Rather he is prepared to use a variety of methods to achieve his ends (sec [6]). To convince himself, and others, of the soundness of his final proof, he produces a protocol. Formal logical systems were introduced to analyse and justify this procedure, and not to replace it as a method of discovery. SUMS is designed to simulate the behaviour of mathematicians. During the course of a proof it 'proves' many facts (i.e. convinces itself of their truth) and records these as true; it also produces a protocol which is intended to convince others of their truth (i.e. a proof).

It is the author's hope that the method of theorem proving outlined in this paper will prove applicable not just to arithmetic, but to all mathematical theories, especially classical systems with a single standard model, like analysis, geometry and set theory. In fact similar systems for geometry and set theory are now being built by Aaron Sloman and Mike Liardet respectively.

The author also hopes that this representation may make

some contribution to the study of Psychological Modelling. In particular, maybe it sheds some light on the curious blackboard diaerams which mathematicians use to help them 'understand' problems.

ACKNOWLEDGEMENTS.

My debt to the M.I.T. Progress Report[7] will be obvious. Not quite as obvious, but equally important, are the conversations with my colleagues - Aaron Sloman, Bo'b Boyer, J Moore, Mike Liardet and numerous others.

REFERENCES.

Bundy, A. 'The Metatheory of the Elementary Equation Calculus'. Ph.D. Thesis. University of Leicester, England. 1971.

Beth, E.S. 'Semantic Entailment and Formal Derivability'. Mededelingen der Ron. Med. Akad. v. Wet. New Series, Vol. 18 No. 13, Amsterdam, 1955.

Gelernter, H. 'A Geometry Theorem Proving Machine', Computers and Thought pp. 134-52, McGraw Hill, 1963.

Hewitt, C. 'PLANNER: A Language for Manipulating Models and Proving Theorems in a Robot'. Proceedings of 1st IJCAI. Washington D.C., 1969.

Robinson, J.A. 'A Machine Oriented Logic based on the Resolution Principle'. J.Assoc.Comput. Mach. 12. pp. 23-41, 1965.

Sloman, A. 'Interactions between Philosophy and Artificial Intelligencer The Role of Intuition and Non-Logical Reasoning in Intelligence'. Proceedings of 2nd IJCAI, pp. 270-6, The British Computer Society, 1971, also Artificial Intelligence 2, pp. 209-25* North Holland Publishing Co., 1971.

Minsky, M. and Papert, S. 'Project M.A.C. Process Report', pp. 129-244, M.I.T., 1971.