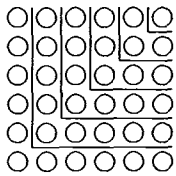
Automation of Diagrammatic Reasoning*

Mateja Jamnik, Alan Bundy, Ian Green
Department of Artificial Intelligence, 80 South Bridge
Edinburgh, EH1 1HN, UK
matejaj@dai.ed.ac.uk, A.Bundy@ed.ac.uk, I.Green@ed.ac.uk

Abstract

Theorems in automated theorem proving are usually proved by logical formal proofs. However, there is a subset of problems which humans can prove in a different way by the use of geometric operations on diagrams, so called diagrammatic proofs. Insight is more clearly perceived in these than in the corresponding algebraic proofs: they capture an intuitive notion of truthfulness that humans find easy to see and understand. We are identifying and automating this diagrammatic reasoning on mathematical theorems. The user gives the system, called DIAMOND, a theorem and then interactively proves it by the use of geometric manipulations on the diagram. These operations are the "inference steps" of the proof. DIAMOND then automatically derives from these example proofs a generalised proof. The constructive ω-rule is used as a mathematical basis to capture the generality of inductive diagrammatic proofs. In this way, we explore the relation between diagrammatic and algebraic proofs.

Introduction



$$1 + 3 + 5 + \cdots + (2n - 1) = n^2$$

It requires only basic secondary school knowledge of mathematics to realise that the diagram above is a proof of a theorem about the sum of odd naturals.

It is an interesting property of diagrams that allows us to "see" and understand so much just by looking at a simple diagram. Not only do we know what theorem the diagram represents, but we also understand the proof of the theorem represented by the diagram and believe it is correct.

Is it possible to simulate and formalise this sort of diagrammatic reasoning on machines? Or is it a kind of intuitive reasoning particular to humans that mere machines are incapable of? Roger Penrose claims that it is not possible to automate such diagrammatic proofs.[1] We are taking his position as an inspiration and are trying to capture the kind of diagrammatic reasoning that Penrose is talking about so that we will be able to simulate it on a computer.

The importance of diagrams in many domains of reasoning has been extensively discussed by Larkin and Simon (Larkin & Simon 1987), who claim that "a diagram is (sometimes) worth *ten* thousand words". The advantage of a diagram is that it concisely stores information, explicitly represents the relations among the elements of the diagram, and it supports a lot of perceptual inferences that are very easy for humans.

It is exactly these characteristics of diagrams that we wish to exploit in our project. In this paper we present a system (which is currently being developed) called DIAMOND (DIAgraMmatic reasONing and Deduction), which reasons with diagrams. With this system, the user inputs a theorem of mathematics to be proved, instructs the system what diagram to start the search for the proof from, and decides what geometric operations to perform during the proof search. Our aim is to investigate the relation between formal algebraic proofs and more "informal" diagrammatic proofs. Usually, theorems are *formally* proved with the use of inference steps which often do not convey an intuitive notion of truthfulness to humans. The inference steps are just statements that follow the rules of some logic. The reason we trust that they are correct is that the logic has been previously proved to be sound. Following and applying the rules of such a logic guarantees us that there is no mistake in the proof. We might not have such a guarantee in DIAMOND, but will gain

*The work reported in this paper has been presented at IJCAI-97 in Nagoya, Japan. The original version of this paper is to be published by Morgan Kaufmann Publishers in the Proceedings of IJCAI-97.

[1] Roger Penrose presented his position in the lecture at International Centre for Mathematical Sciences in Edinburgh, in celebration of the 50th anniversary of UNESCO on 8 November, 1995.

a more informal insight into the proof. Ultimately, the entire process of diagrammatically proving theorems will illuminate the issues of formality, rigour, truthfulness and power of diagrammatic proofs.

First, we list some of the theorems that we aim to prove. Second, we present DIAMOND's architecture, some operations required, the generalisation mechanism employed, and indicate how to verify the generalised proof. Next, we report on some of our results and discusses future work. Then, we discuss some of the related diagrammatic reasoning systems. Finally, we conclude by summarising the main points of this paper.

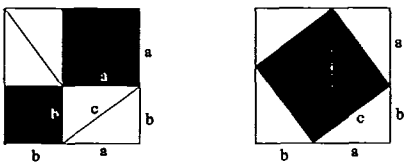
'Diagrammatic' Theorems

We are interested in mathematical theorems that admit diagrammatic proofs. In order to clarify what we mean by diagrammatic proofs we first list some example theorems. Then, we introduce a taxonomy for categorising these examples in order to be able to characterise the domain of problems under consideration.

Examples

Pythagora's Theorem Pythagora's Theorem states that the square of the hypotenuse of a right angle triangle equals the sum of the squares of its other two sides. Here is one of the many different diagrammatic proofs of this theorem, taken from (Nelsen 1993, page 3):

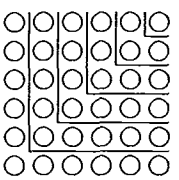
$$a^2 + b^2 = c^2$$



The proof consists of first taking any right angle triangle, completing a bigger square by joining to it identical triangles and squares along its sides, and then rearranging the triangles in a bigger square. For a more elaborate explanation, see (Jamnik, Bundy, & Green 1997).

Sum of Odd Naturals This example is also taken from (Nelsen 1993, page 71). The theorem about the sum of odd naturals states the following:

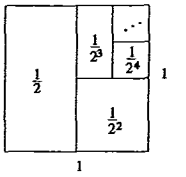
$$1 + 3 + \cdots + (2n - 1) = n^2$$



Note the use of parameter n. If we take a square we can cut it into as many L's (which are made up of two adjacent sides of the square) as the size of the side of the square. Note that one L is made out of two sides, i.e., $2n$, but the shared vertex has been counted twice. Therefore, one L has a size of $(2n - 1)$, where n is the size of the square.

Geometric Sum This example is also taken from (Nelsen 1993, page 118). A theorem about a geometric sum of $\frac{1}{2^n}$ states the following:

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 1$$



Note the use of ellipsis in the diagram. Take a square of unit size. Cut it down the middle. Now, cut one half of the previous cut square into halves again. This will create two identical squares making up a half of the original square. Take one of these two squares and continue doing this procedure indefinitely.

Classification

From the analysis of the examples that we presented above, and many others, three categories of proofs can be distinguished:

Category 1: Proofs that are not schematic: there is no need for induction to prove the general case. Simple geometric manipulations of a diagram prove the individual case. At the end, generalisation is required to show that this proof will hold for all a, b. Example theorem: *Pythagora's Theorem.*

Category 2: Proofs that are schematic: they require no inductive step to prove the theorem for each concrete diagram (i.e., problem), but require induction for the general diagram of size n (a concrete diagram cannot be drawn for this instance). The constructive ω-rule (explained in more detail later on in this paper) is used to generate a generalised proof from the individual proof instances. Example theorem: *Sum of Odd Naturals.*

Category 3: Proofs that are inherently inductive: for each individual concrete case of the diagram they need an inductive step to prove the theorem. Every particular instance of a theorem, when represented as a diagram requires the use of abstractions to represent infinity. Thus, the constructive ω-rule (defined later) is not applicable here. Example theorem: *Geometric Sum.*

Problem Domain

We choose mathematics as our domain for theorems since it allows us to make formal statements about the reasoning, proof search, induction, generalisations, and such issues. Having introduced the examples and their

categorisation, which is by no means exhaustive, we are now able to further restrict our domain of mathematical theorems.

First, we narrow down the domain to a subset of theorems that can be represented as diagrams without the need for abstraction (e.g., the use of ellipsis, as in the above example theorem for *geometric sum*). Conducting proofs and using abstractions in diagrams is problematic, since it is very difficult to keep track of these abstractions while manipulating the diagram during the proof procedure.

Second, we consider diagrammatic proofs that require induction to prove the general case (i.e., Category 2 above). Namely, diagrams can be drawn only for concrete situations and objects. We cannot draw, for example, an $n \times n$ square. Our challenge is to find a generalisation mechanism that does not require using abstractions in diagrams. The generality of the proof will be captured in an alternative way (by using the constructive ω-rule; see next Section).

It is clear that we will need a stronger problem domain definition which remains a subject of our research. One possibility is to consider theorems of arithmetic or number theory only. To date, DIAMOND is targeted to prove examples of Category 2, but we may implement diagrammatic theorem proving of examples for Category 1 as well.

Constructive ω-Rule

As mentioned above we could use the constructive ω-rule to prove theorems of Category 2. Siani Baker in (Baker, Ireland, & Smaill 1992) did some work on the constructive ω-rule and schematic proofs for arithmetic theorems. Here, we explain the idea behind schematic proofs and how it can be applied to diagrammatic proofs.

Schematic Proof Schematic proofs use the constructive ω-rule which is an alternative to induction. The constructive ω-rule allows inference of the sentence $\forall x P(x)$ from an infinite sequence $P(n)$ $n \in \omega$ of sentences.

$$\frac{P(0), P(1), P(2), \ldots}{\forall n.P(n)}$$

where "if each $P(n)$ can be proved *in a uniform way* (from parameter n), then conclude $\forall n P(n)$." The criterion for uniformity of the procedure of proof using the constructive ω-rule is taken to be the provision of a general schematic proof, namely the proof of $P(n)$ in terms of n, where some rules R are applied some function of n (i.e., $f_R(n)$) times (a rule can also be applied a constant number of times). Now, $\mathsf{proof}(n)$ is schematic in n, since we applied some rule R n times. The following procedure summarises the essence of using the constructive ω-rule in schematic proofs:

1. Prove a few special cases (e.g., $P(2)$, $P(16)$, ...).
2. Generalise (guess) $\mathsf{proof}(n)$ (e.g., from $\mathsf{proof}(2)$, $\mathsf{proof}(16)$, ...).
3. Prove that $\mathsf{proof}(n)$ proves $P(n)$ by meta-induction on n.

The general pattern is extracted (guessed, to be exact) from the individual proof instances by (learning type) inductive inference. By meta (mathematical) induction we mean that we introduce system PA_ω (i.e., Peano Arithmetic with ω-rule) such that:

$$\mathsf{proof}(n) : P(n) \vdash_{PA_\omega} \mathsf{proof}(s(n)) : P(s(n))$$

where ":" stands for "is a proof of", and $s(n)$ is a notation for successor of n. This essentially says that by using the rules on $P(s(n))$ we can reduce it to $P(n)$. For more information, see (Baker, Ireland, & Smaill 1992).

Diagrams and Schematic Proofs We claim that we can extend Baker's work on schematic proofs in our diagrammatic proofs in that the generality of the diagrammatic proof is embedded in the schematic proof. Thus, we eliminate the need for abstractions in diagrams, and can generalise from manipulations on concrete diagrams.

The diagrammatic schematic proof starts with a few particular concrete cases of the theorem represented by the diagram. The diagrammatic procedures (i.e., operations) on the diagram are performed next, capturing the inference steps of the diagrammatic proof. In DIAMOND, this step (also referred to as the proof checking step) is done *interactively* with the user, and corresponds to the first step of the schematic proof procedure given in the previous section.

The second step is to generalise the operations involved in the schematic proof for n. Note that the generality is represented as a sequence of diagrammatic procedures (operations) and not as a general representation of a diagram. In DIAMOND, this step is done *automatically*. More precisely, the basic idea is to consider proofs for $n+1$ which can be reduced to proofs for n (or conversely, such proofs for n which can be extended to proofs for $n+1$ by adding to them some additional sequence of operations). The difference between the proof for $(n+1)$ and the proof for n, i.e., the additional sequence of operations in the proof for $(n+1)$ with respect to the proof for n is referred to as the step case of the generalised proof.

The last step in the schematic proof procedure is to prove by meta-induction that the generalised diagrammatic schematic proof is indeed correct. It remains a subject of this research project to determine whether this will be considered at all or not. An alternative at this point could be to translate the diagrammatic proof to an algebraic proof. We are currently exploring this latter possibility (see "Correctness of Schematic Proof" below).

Schematic Diagrammatic Proof for the Sum of Odd Naturals

Now we can attempt to structure the diagrammatic proofs in a more formal way. Here we list the proof

for the theorem about the sum of odd naturals as a sequence of steps that need to be performed on the diagram:

1. Cut a square into n L's, where an L consists of 2 adjacent sides of the square.
2. Cut each L into two segments.
3. For each L, join these two segments one on top of the other length-wise (note that one of the two segments is always one unit longer than the other, thus an L always consists of an odd number of units, i.e., $2n-1$).

Identifying the operations (i.e., geometric manipulations) that were required to prove the theorem will help us define a large repertoire of such operations which will be used in the diagrammatic proofs. The generality of the proof is captured by the use of the constructive ω-rule, by which we take a few special cases of the diagram (say a square of size 15 and 16), and find the general pattern of the proof that will hold for each case (e.g., the schematic proof given above).

DIAMOND System

Clearly, an important issue in the development of DIAMOND is the *internal* representation of diagrams and operations on them. The hope is that these capture the intuitiveness, rigour and simplicity of human reasoning with diagrams. We aim to emulate human visual perception to enable a theorem prover to prove theorems using diagrammatic inference steps. There are several representations available to achieve this. In DIAMOND we use a mixture of Cartesian and topological representations.

The architecture of DIAMOND consists of two parts. The *diagrammatic component* forms and processes the diagram. The *inference engine* deals with the diagrammatic inference steps. It processes the operations on the diagram. An important submodule is the generalisation tool (i.e., implementation of the constructive ω-rule).

The rest of this section presents the operations used to construct proofs, the structure of proofs and the generalisation mechanism used in DIAMOND. For more information see (Jamnik, Bundy, & Green 1997).

Geometric Operations

Geometric operations (also referred to as manipulations or procedures) capture the inference steps of the proof. Thus, a sufficiently large number of such operations which are then available to the user in the search for the proof, needs to be identified and formalised. Since we are not generating, i.e., discovering diagrammatic proofs, but rather we are trying to understand them, we can expect from the user to input these operations. To date, a small number of such operations has been implemented and is available to the user.

We distinguish between two types of operations:

Atomic operations: they are basic one-step operations that will be combined into more complex operations. Examples of such operations are: rotate, translate, cut, join, project from 3D to 2D, remove, insert a segment.

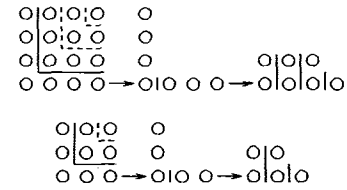
Composite operations: they are more complex, typically recursive operations, composed from simple atomic ones. Perhaps, we can think of them as tactics or tacticals in automated reasoning. In the future we need to investigate several different recursive structures of diagrams. Depending on the theorem that we are proving, we use a different recursive composite operation. Ideally, the internal representation of the diagram should be pertinent to the composite operation that we are performing on it.

In the example of the theorem for the *sum of odd naturals* the proof consists of the following operations: *lcut, split_top_row, rotate90, join, rem_left_column, remove_dot.*

Constructing a Proof

DIAMOND's example proof consists of a sequence of applications of geometric operations on a diagram. The generalisation is then carried out automatically, if any such generalisation exists for the two example proofs given.[2] DIAMOND expects the example proofs to be formulated in a particular way where the order of operations in the user's formulation of the example proofs is crucial. Both example proofs are expected to be given with the same order of operations, but with some extra operations in the case of the proof$(n+1)$ for some particular n.

Consider the example for the *sum of odd naturals.* The step cases for proofs for $n=4$ and $n=3$ look as follows:



The aim is to recognise automatically the structure of the proof from a linear sequence of applications of operations, so that the example proofs for n and $n+1$ can be reformulated in the general case into the following:

$$\begin{aligned} \text{proof}(n) &= \mathcal{A}(n)\mathcal{A}(n-1)\ldots\mathcal{A}(2)\mathcal{B}(1) \\ \text{proof}(n+1) &= \mathcal{A}(n+1)\mathcal{A}(n)\mathcal{A}(n-1)\ldots\mathcal{A}(2)\mathcal{B}(1) \end{aligned}$$

[2] If the proof contains a case split for say, even and odd integers, and the two example proofs given are for two different cases, then DIAMOND cannot generalise from them. However, DIAMOND recognises that the example proofs were given for different cases, and requests the user to supply another example proof for each case, in order for it to be able to generalise. This will be further explained in "f-Homogeneous Proof".

where for each n, $\mathcal{A}(n)$ is a step case consisting of a sequence of applications of some operations and $\mathcal{B}(1)$ is a base case for $n = 1$. Alternatively, we seek this recursive reformulation:

$$\begin{array}{rcl} \mathsf{proof}(n+1) & = & \mathcal{A}(n+1)\ \ \mathsf{proof}(n) \\ \mathsf{proof}(1) & = & \mathcal{B}(1) \end{array}$$

A further issue that we are investigating currently is to relax the requirement for a particular ordering of operations in formulating example proofs. Sets with partial ordering could be used as an alternative.

Generalisation

Given some example proofs DIAMOND needs to generalise from them, so that the final diagrammatic proof is not only for the cases of specific n's, but holds for all n. Such a schematic proof consists of a general sequence of applications of some operations, where the number of application of each operation is dependent on n or is a constant.

DIAMOND distinguishes between two types of example proofs: *destructive*, i.e., the example proofs which are formulated so that the base case operations are performed last (in a sense the initial diagram is "destructed" by the application of operations down to a trivial diagram, forming the proof along in this way); and *constructive*, i.e., the base case operations are performed first followed by the step case operations.

The proofs that have the same structure for all n are called 1-homogeneous proofs. Proofs can be c-homogeneous; then there are c cases of the proof. We say that if all concrete instances of the proof (for instances of numbers that "equal modulo c") have the same structure and can be generalised, then the proof is c-homogeneous. If there are c cases, then there are c different generalised proofs, one for each case. The following theorem can be proved:

Theorem 1: If a proof is c-homogeneous, then it is also (kc)-homogeneous for every natural number $k > 0$.

The immediate consequence of **Theorem 1** is:

Corollary 1: If a proof is *not* c-homogeneous, then it is also *not* f-homogeneous for every factor f of c.

In a c-homogeneous proof we will denote by $\mathcal{B}_r$ a base case for a branch of numbers which give remainder r when divided by c. $\mathcal{B}_r$ is actually a proof for the smallest natural number that gives remainder r when divided by c. A special case is a class of numbers divisible by c, where a base case is going to be $\mathcal{B}_c$, which is a proof for $n = c$.

The general representation of the destructive proof is formalised as follows – let:

- $n = kc + r$
- where c = *number of cases* and $r < c$
- and $i \geq 1$.

Then the recursive definition of a general proof is:

for $r \neq 0$:

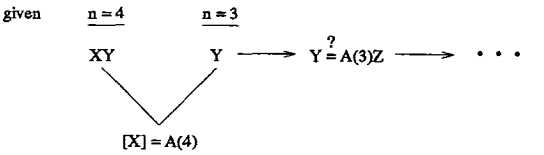
$$\begin{array}{rcl} \mathsf{proof}(ic+r) & = & \mathcal{A}_r(ic+r)\ \mathsf{proof}((i-1)c+r) \\ \mathsf{proof}(r) & = & \mathcal{B}_r(r) \end{array}$$

for $r = 0$:

$$\begin{array}{rcl} \mathsf{proof}(ic) & = & \mathcal{A}_c(ic)\ \mathsf{proof}((i-1)c) \\ \mathsf{proof}(c) & = & \mathcal{B}_c(c) \end{array}$$

where $\mathcal{A}_r$ is a step case and $\mathcal{B}_r$ is a base case for a class of proofs where $n \equiv r(mod\ c)$. The formalisation of generalised proof for *constructive* proofs is symmetric to the one given above.

Generalising For All Linear Functions As mentioned above, we aim to recognise the particular recursive structure of the given example proofs. More precisely, we want to extract the step case $\mathcal{A}$ and the base case $\mathcal{B}$ of the proof and then generalise them for all n. The general methodology employed for doing this can be demonstrated as:



The first step of the generalisation algorithm is to extract the difference between the two given example proofs for n_1 and n_2 $(n_1 > n_2)$, where $c = n_1 - n_2$, in the hope that this, when generalised, will be the step case $\mathcal{A}$ of the proof. This is done by associative matching which detects and returns the difference between the two example proofs. Now we have a concrete step case of the proof. This difference consists of a few operations op_k each applied x_{k,n_1} times for some natural k.

To make a step case general, we need to find the dependency function between every x_{k,n_1} and n_1. This demands identifying a function of n_1, which would give a specific x_{k,n_1}, i.e., $f_k(n_1) = x_{k,n_1}$ for some k and n_1. DIAMOND assumes that the dependency is linear: $an+b$. Thus, let us write for each op_k a linear equation $an_1 + b = x_{k,n_1}$, where n_1 and x_{k,n_1} are known.

The subsequent stage of the generalisation is to extract the next step case from the rest of the example proof for the corresponding new n (i.e., n_2). If successful, continue extracting step cases for the corresponding n's from the rest of the proof until only the base case is left.

Since we are dealing with inductive proofs, it is expected that every step case of a proof will have the same structure, i.e., will consist of the same sequence of application of operations, but a different number of times. Thus, we could in the same way as above for every operation op_k write a linear equation $an_2 + b = x_{k,n_2}$. However, the number x_{k,n_2} of applications of a particular operation op_k in the next step case is not

known. A possible value of x_{k,n_2} is acquired by counting the number x' of times every operation op_k of the initial step case occurs in the rest of the proof. The actual value of the number of occurrences of each operation could be any number from 0 to x'. Thus, we do branching for all such values and thus we have:

$$\begin{aligned} an_1 + b &= x_{k,n_1} \\ an_2 + b &= x_{k,n_2} \end{aligned}$$

where n_1, n_2, x_{k,n_1} and x_{k,n_2} are known, so the equations can be solved for a and b, and x_{k,n_2} takes values from 0 to x'. This results in several possible potential generalisations of the step case. The aim is to eliminate those that are impossible. After checking if step cases for all n down to base case are structurally consistent one hopes to be left with at least one possible generalisation of the example proofs. The step case is rejected when the sequence of operations in the subsequent step cases is impossible, i.e., the functions were wrong. This normally occurs when the dependency function gives a negative number of applications of a particular operation, when the calculated sequence is not identical to the rest of the example proof, or when there is no integer solution to our equations. Usually, there will be only one possible generalisation of the two given example proofs.

The example proof for the *sum of odd naturals* is generalised into the following step case and base case:

$$\begin{aligned} \mathcal{A}(n) &= \{\textit{lcut(1), split\_top\_row(1), rotate90(1), join(1), rem\_left\_column(n-1), remove\_dot(1)}\} \\ \mathcal{B}(1) &= \{\textit{remove\_dot(1)}\} \end{aligned}$$

where the function in parentheses indicates the number of times that the operations are applied for each particular n.

f-Homogeneous Proof Assume two example proofs for the *sum of odd naturals* (the example proof would consist of making n lcuts, and then showing that each L consists of an odd number of dots). If the user supplies two example proofs for values of n and $n+1$, for some concrete n, then there is no problem, so DIAMOND will generalise normally and determine that the proof is 1-homogeneous. However, should the user supply proofs for n and $n+2$ for some concrete n, the first stage of generalisation would determine that the step case consists of two lcuts. However, a complete recursive function for generalisation requires a step case to consist of one lcut only.

DIAMOND checks this by trying to split the step case into further f structurally the same sequences of operations, for all factors f of c in order to obtain an f-homogeneous proof. If the method fails, then there is no such f-homogeneous further generalisation of the step case $\mathcal{A}(n)$. If the method succeeds, and DIAMOND finds a new generalisation of the step case, call this $\mathcal{A}'(n)$, then it also needs to find a new base case $\mathcal{B}'(r')$ if $r' \neq 0$, or $\mathcal{B}'(f)$ if $r' = 0$, where the previous r for c was such that $n = kc + r$ and $r < c$, and the new r' is now such that $n = kf + r'$ and $r' < f$.

Correctness of Schematic Proof

The last stage of extracting a diagrammatic proof is to check that the *guessed* general schematic proof is indeed correct. Currently, we are investigating the possibility of translating a diagrammatic proof into an algebraic proof, and then carrying out meta-induction on this translation[3] to prove the correctness of a schematic proof.

In particular, we need to show in the meta-theory that proof(n) proves $P(n)$ for all n, i.e. it gives a prooftree with $P(n)$ at its root, and axioms at its leaves.

In order to translate a diagrammatic proof into its corresponding algebraic proof it is necessary to translate the geometric operations on the diagram into algebraic rewrite rules. For example, an *lcut* takes a square of size n, and returns a square of size $(n-1)$ and an ell of size n:

$$\text{square}(n) \xrightarrow{lcut} \text{square}(n-1) \oplus \text{ell}(n)$$

where $\oplus$ is an operator on multisets of diagrams that is right associative, such that

$$\text{diagram}_1 \oplus \text{diagram}_2 \oplus \cdots \oplus \text{diagram}_n$$

denotes the "existence" of several (n) diagrams that are available for manipulation at any one time. Algebraically, $\oplus$ is isomorphic to the usual + operator.

Next, we translate diagrams into their algebraic equivalent representation, so square(n) translates into n^2, ell(n) translates into $(2(n-1)+1)$, etc. Therefore, a diagrammatic operation *lcut* corresponds in algebra to the following rewrite rule:

$$n^2 \Rightarrow (n-1)^2 + (2(n-1)+1)$$

Finally, we apply these rewrite rules as indicated by the schematic proof to the statement of the theorem. If we arrive at equality, then the schematic proof is indeed correct.

Results and Further Work

DIAMOND is implemented in Standard ML of New Jersey, Version 109. The code is available upon request to the first author.

So far, the interactive construction of proofs and automatic generalisation from example proofs have been implemented in DIAMOND. We can prove a few theorems: sum of odd naturals, sum of all naturals, sum of Fibonacci squares. We are working on more examples.

[3] It is not possible to carry out meta-induction directly on the diagrams without the use of abstractions (e.g. ellipsis), which has already been discussed above to be problematic.

We want to relax the requirement for a particular formulation of example proofs. Partially ordered sets could be used. However, this would require recognition and generalisation of diagrams (as opposed to a sequence of operations).

There is also a possibility of allowing non-linear dependency functions.

Another issue that will be addressed is whether to prove automatically that the derived generalisation of a diagrammatic proof is indeed correct.

Finally, some recognition and generalisation of diagrams using abstractions could be an interesting issue to consider. The difficulty is to keep track of these abstractions while manipulating the diagram.

Related Work

Several diagrammatic systems such as the Geometry Machine (Gelernter 1963), Diagram Configuration model (Koedinger & Anderson 1990), GROVER (Barker-Plummer & Bailin 1992), and Hyperproof (Barwise & Etchemendy 1991) have been implemented in the past and are of relevance to our system.

However, they all use diagrams to model algebraic statements, and use these models for heuristic guidance while searching for an *algebraic* proof. In contrast, proofs in our system are explicitly constructed by operations on diagrams.

Closer to our work, but not in the domain of diagrammatic reasoning, is work done by Siani Baker (Baker, Ireland, & Smaill 1992) described in Section "Constructive ω-Rule", whereby we exploit the uniform structure of inductive proofs to generalise from example proofs.

Conclusion

We presented a diagrammatic reasoning system, DIAMOND, which supports interactive construction of diagrammatic proofs. The system automatically generalises from examples to give a general proof for all n. The hope is that automating the 'informal' diagrammatic reasoning of humans will shed light on the issues of formality, informality, rigour and 'intuitive' understanding of the correctness of diagrammatic proofs.

Acknowledgements

We should like to thank Predrag Janičić for inspiring discussions about some of the work presented here. The research reported in this paper was supported by an Artificial Intelligence Dept. studentship, the University of Edinburgh, and a Slovenian Scientific Foundation supplementary studentship for the first author, and by EPSRC grant GR/L/11724 for the other two authors.

References

Baker, S.; Ireland, A.; and Smaill, A. 1992. On the use of the constructive omega rule within automated deduction. In Voronkov, A., ed., *LPAR 92, St. Petersburg*, Lecture Notes in AI No. 624, 214–225. Springer-Verlag.

Barker-Plummer, D., and Bailin, S. C. 1992. Proofs and pictures: Proving the diamond lemma with the GROVER theorem proving system. In *Working Notes of the AAAI Symposium on Reasoning with Diagrammatic Representations.*

Barwise, J., and Etchemendy, J. 1991. Visual information and valid reasoning. In Zimmerman, W., and Cunningham, S., eds., *Visualization in Teaching and Learning Mathematics.* Mathematical Association of America. 9–24.

Gelernter, H. 1963. Realization of a geometry theorem-proving machine. In Feigenbaum, E., and Feldman, J., eds., *Computers and Thought.* McGraw Hill. 134–52.

Jamnik, M.; Bundy, A.; and Green, I. 1997. Automation of diagrammatic proofs in mathematics. In Kokinov, B., ed., *Perspectives on Cognitive Science*, volume 3, 168–175. New Bulgarian University.

Koedinger, K., and Anderson, J. 1990. Abstract planning and perceptual chunks. *Cognitive Science* 14:511–550.

Larkin, J., and Simon, H. 1987. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science* 11:65–99.

Nelsen, R. B. 1993. *Proofs Without Words: Exercises in Visual Thinking.* The Mathematical Association of America.