On Automating Diagrammatic Proofs of Arithmetic Arguments*

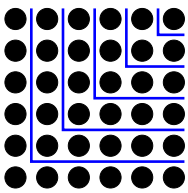
Mateja Jamnik, Alan Bundy and Ian Green
Division of Informatics, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, UK
matejaj@dai.ed.ac.uk, A.Bundy@ed.ac.uk, I.Green@ed.ac.uk

December 15, 1998

Abstract. Theorems in automated theorem proving are usually proved by formal logical proofs. However, there is a subset of problems which humans can prove by the use of geometric operations on diagrams, so called diagrammatic proofs. Insight is often more clearly perceived in these proofs than in the corresponding algebraic proofs; they capture an intuitive notion of truthfulness that humans find easy to see and understand. We are investigating and automating such diagrammatic reasoning about mathematical theorems. Concrete, rather than general diagrams are used to prove particular concrete instances of the universally quantified theorem. The diagrammatic proof is captured by the use of geometric operations on the diagram. These operations are the "inference steps" of the proof. An abstracted schematic proof of the universally quantified theorem is induced from these proof instances. The constructive ω-rule provides the mathematical basis for this step from schematic proofs to theoremhood. In this way we avoid the difficulty of treating a general case in a diagram. One method of confirming that the abstraction of the schematic proof from the proof instances is sound is proving the correctness of schematic proofs in the meta-theory of diagrams. These ideas have been implemented in the system, called DIAMOND, which is presented here.

Key words: Automated Reasoning, Diagrammatic Reasoning, Theorem Proving

1. 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

* Portions of this work were published in the Proceedings of the International Joint Conference on Artificial Intelligence in (Jamnik *et al*, 1997).

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 a challenge and are trying to capture the kind of diagrammatic reasoning that Penrose is talking about so that we will be able to emulate 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. Diagrams have been extensively used in the history of mathematics to *aid informal mathematical reasoning*. The use of diagrams in explanations of theorems and proofs of geometry dates back to Ancient Greece, and the time of Aristotle and Euclid. Thus it is surprising perhaps that more recently, starting with the invention of formal axiomatic logic in the sense of Frege, Russell and Hilbert, diagrams have been denied a *formal* role in theorem proving. It is generally thought by logicians that diagrams have no accepted syntax nor semantic theory in a logical formalism which would make them rigorous enough to be used in formal proofs. Only very recently, in the last two decades, have there been efforts to fill this gap and investigate whether and how diagrams can be used in formal proofs. For instance, the investigation of Pierce's existential graphs in (Sowa 84), the work on GROVER in (Barker-Plummer & Bailin, 1997), the work on Hyperproof in (Barwise & Etchemendy, 1991), the introduction of computational models for interpreting Euler's circles in (Stenning & Oberlander 95), the analysis of the use of Venn diagrams as a formal system in (Shin 95), and the formalisation of a logical theory of Venn diagrams in (Hammer 95).

Our work contributes in some sense to the effort in the research from the formal perspective on the use of diagrams, especially that of automated reasoning systems which use diagrams in the reasoning process. Our aim is to formalise diagrammatic reasoning and to show that diagrams can be used for proofs. In this paper we show how diagrams can be used for proofs in a formal system. We look into how theorems

[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.

of mathematics can be expressed as diagrams for some concrete values, *i.e.*, ground instantiations of a theorem. The initial diagram is manipulated using some geometric operations. The sequence of geometric operations on a diagram represents the inference steps of a diagrammatic proof. Such a concrete proof instance is called an example proof. The set of all available operations defines the proof search space. A general pattern is extracted from these proof instances, and is captured in a recursive program. This recursive program constitutes a general diagrammatic proof for the universally quantified theorem. An existing technique in logic, namely the constructive ω-rule, justifies the step from schematic proofs to theoremhood.

We also aim 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 of a formal symbolic (as opposed to diagrammatic) proof are 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 that there is no mistake in the proof. We want to have such a guarantee in our proof system, and moreover, to gain an 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.

We implemented a diagrammatic proof system called DIAMOND, which automates such diagrammatic reasoning and applies it to problem solving in mathematics. The user interactively constructs example proofs by choosing an initial diagram which represents the theorem, and then applies diagrammatic operations to build a proof. DIAMOND then automatically extracts a general pattern from these instances, and captures it in a recursive program.

First, we list some of the theorems and their diagrammatic proofs. These help us define our problem domain. Second, we present DIAMOND's architecture, some operations required, the abstraction mechanism employed, and indicate how to verify the abstracted proof. Next, we report on some of our results and discuss future work. Then, we discuss some of the related diagrammatic reasoning systems. Finally, we conclude by summarising the main points of this paper.

2. 'Diagrammatic' Theorems

We are interested in mathematical theorems that admit diagrammatic proofs. In order to clarify what we mean by diagrammatic proofs

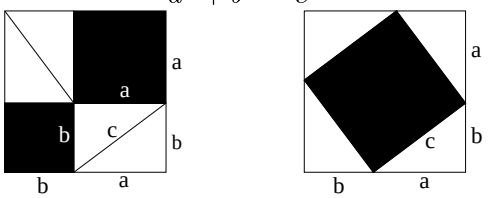
we first list some example theorems. Then, we introduce a taxonomy for categorising these examples in order to be able to characterise the domain of problems under consideration.

2.1. Examples

2.1.1. *Pythagoras' Theorem*

Pythagoras' 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):

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

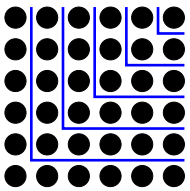


The proof consists of first taking any right angle triangle, completing a bigger square by joining to it identical triangles and squares along its sides, and then rearranging the triangles in a bigger square.

2.1.2. *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:

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

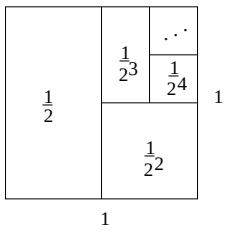


Note the use of parameter n. If we take a square we can cut it into as many ells (which are made up of two adjacent sides of the square) as the magnitude of the side of the square. Note that one ell is made out of two sides, *i.e.*, $2n$, but the shared vertex has been counted twice. Therefore, one ell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

2.1.3. *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 magnitude. Cut it down the middle. Now, cut one half of the previously 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.

2.2. Classification

From the analysis of the examples that we presented above, and many others (see (Jamnik, 1998) and (Nelsen, 1993)), three categories of proofs can be distinguished:

Category 1: Non-inductive theorems. Usually, there is only one representative diagram for all instances of the theorem. There is no need for induction to prove the general case: proofs are not schematic. Simple geometric manipulations of a diagram prove the individual case. Abstraction is required to show that this proof will hold for all a, b. Theorems are of continuous space. Example theorem: *Pythagoras' theorem.*

Category 2: Inductive theorems with a parameter. A diagram is a representative of a particular instance of a theorem. Proofs are schematic: they require induction for the general diagram of magnitude n (a concrete diagram cannot be drawn for this instance). An alternative method can sometimes be used to capture the generality of the proof. Theorems are of discrete space. Example theorem: *sum of odd naturals.*

Category 3: Theorems whose proofs 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. Theorems are of continuous space. Example theorem: *geometric sum.*

2.3. 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, abstractions 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 without some abstraction device, *e.g.*, ellipsis. Our challenge is to find a mechanism for extracting a general proof that does not require using abstractions in diagrams.[2] The generality of the proof will be captured in an alternative way (by using the constructive ω-rule — see §2.4).

Third, to date we consider theorems of natural number arithmetic only. Diamond is designed to prove examples of Category 2, where diagrams represent natural numbers. A more formal definition of diagrams is given in §3.4.1. We may extend diagrammatic theorem proving for examples of Category 1 as well.

One of the possibilities for future work is to consider a need for a more formal problem domain definition.

2.4. Constructive ω-Rule

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

2.4.1. *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.

[2] Note that (Barker-Plummer & Bailin, 1997) formalises the use of abstractions, however in the domain of well founded relations.

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

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). Let the proof of $P(n)$ be captured using a recursive function $\mathsf{proof}(n)$. 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. Abstract (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) from the individual proof instances by (learning type) inductive inference. By meta mathematical induction we mean that we introduce system META such that for all n:

$$\vdash_{\text{META}} \mathsf{proof}(n) : P(n)$$

where ":" stands for "is a proof of". Baker used PA_ω (*i.e.*, Peano arithmetic with ω-rule) for the system META (Baker *et al*, 1992). The meta inductive rule is defined as follows:

$$\frac{\vdash_{\text{META}} \mathsf{proof}(0) : P(0) \quad \mathsf{proof}(r) : P(r) \vdash_{\text{META}} \mathsf{proof}(s(r)) : P(s(r))}{\vdash_{\text{META}} \quad \forall n \;\; \mathsf{proof}(n) : P(n)}$$

This essentially says that by using the rules on $P(s(n))$ we can reduce it to $P(n)$. For more information, see (Baker *et al*, 1992).

2.4.2. *Diagrams and Schematic Proofs*

We claim that we can extend Baker's work on schematic proofs to our diagrammatic proofs so that the generality of the diagrammatic proof is embedded in the schematic proof. Thus, we eliminate the need for abstractions in diagrams, and can extract a general schematic proof 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. This step corresponds to the first step of the schematic proof procedure given in the previous section §2.4.

The second step is to abstract the operations involved in the schematic proof for n. Note that the generality is represented as a recursive program which specifies a sequence of diagrammatic procedures (operations) that are used on a diagram, and not as a general representation of a diagram. 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 abstracted schematic proof.

The last step in the schematic proof procedure is to prove by meta-induction that the abstracted diagrammatic schematic proof is indeed correct. One way of proving the correctness of schematic proofs is to create a theory of diagrams that models the processes in a diagrammatic reasoning system and prove correctness there. A formal definition of a diagrammatic proof of an arithmetic statement, and the correctness of this diagrammatic proof will be discussed in §3.4.

2.4.3. *Schematic Diagrammatic Proof for the Sum of Odd Naturals*
Now we can 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 ells, where an ell consists of 2 adjacent sides of the square.

2. For each ell, continue splitting from an ell pairs of dots at the end of two adjacent sides of the ell until only 1 dot is left (note that for each ell of magnitude n, we will have $n-1$ pairs of dots plus another dot which is a vertex of the two adjacent sides, *i.e.*, $2(n-1)+1$).

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 squares of magnitudes 15 and 16), and find the general pattern of the proof that will hold for each case (*e.g.*, the schematic proof given above).

3. DIAMOND System

The diagrammatic proof system DIAMOND is an embodiment of some of the ideas presented in this paper. DIAMOND stands for **Dia**gra**m**matic Reas**on**ing and **D**eduction.

Clearly, an important issue in the development of DIAMOND is the *internal* representation of diagrams and operations on them. It was George Pólya who was first to advise us on the importance of knowledge representation (see (Pólya, 1945) and (Pólya, 1965)). Simon argued Pólya's point further in (Simon 96) by stating that solving a problem means representing it so that the solution becomes trivial, or at least transparent. In automated reasoning it is difficult to see how to use this advice, since there is normally only one representation scheme for the problem which is available to the system. In DIAMOND we choose a representation which we hope captures the intuitiveness, rigour and simplicity of human reasoning with diagrams. We aim to represent diagrams in a way which enables a theorem prover to prove theorems using diagram.

In DIAMOND we use a mixture of Cartesian and topological representations. DIAMOND uses a primitive notion of a diagram, a dot. All other *elementary* and *derived* diagrams (*e.g.*, rows, columns, ells, frames, squares, triangles, rectangles, *etc.*) are composed in various ways out of dots. The advice of Pólya about alternative representations can readily be used in DIAMOND. Namely, diagrams can be represented in a variety of different ways. For instance, a square is represented as: a sequence of rows; a sequence of columns; a concentric sequence of circumferences, each of which is called a frame; a nested sequence of ells; a sequence of four similar squares; a matrix; a sequence of diagonals.

The choice of the representation that DIAMOND uses is important. Most of the proofs that DIAMOND proves require some kind of recursive decomposition of a diagram. Each alternative representation makes available a different form of recursive decomposition. For more information on the choice of internal representation for diagrams in DIAMOND see (Jamnik, 1998).

The architecture of DIAMOND consists of two parts. The *diagrammatic component* forms and processes the diagram. It is the interface between DIAMOND and the user. The *inference engine* deals with the diagrammatic inference steps. It processes the operations on the diagram. An important submodule is the abstraction mechanism which is used to extract general schematic proofs from example proofs.

The rest of this section presents the operations used to construct proofs, the structure of proofs and the abstraction mechanism used in DIAMOND.

3.1. 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.

Diamond is targeted to prove theorems of discrete arithmetic. Diagrams are a way of representing natural numbers. The interest lies in the effect on the numbers that diagrams represent after an operation has been applied on the diagrams. Thus, the operations join and split diagrams apart in various ways. Some operations are just simple ones (*e.g.*, split a row from a square), and some are more complicated ones (*e.g.* decompose a square into a sequence of rows). Hence, Diamond distinguishes between two types of operations, *atomic* and *composite*:[3]

Atomic operations: are basic one-step operations that can be combined into more complex operations. Examples of such operations are: rotate, translate, cut, split, join, remove, insert a segment,... To date, there are fourteen atomic operations implemented in Diamond.

Composite operations: are more complex, typically recursive operations, composed from simple atomic ones. One can think of them as tactics in automated reasoning. Composite operations are defined in terms of decomposition of different recursive representations of diagrams. Depending on the theorem at hand, the diagram is viewed using a particular representation, which enables one to use a particular recursive composite operation. Ideally, the internal representation of the diagram is pertinent to the composite operation that is being carried out on it. Such a representation would render an operation very easy to apply. It would be just a simple decomposition of the representation of a diagram. Examples of such operations are: recursive decomposition of a square into rows, or columns, or ells, or frames,...

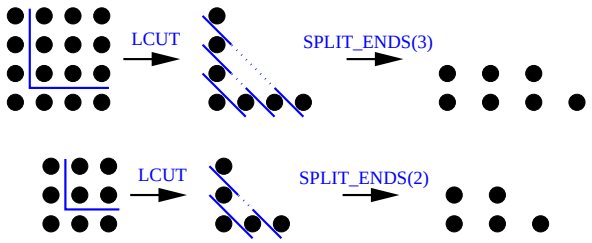
In the example of the theorem for the *sum of odd naturals* the proof consists of the following operations: lcut and split_ends.

[3] A complete list of operations can be found in (Jamnik, 1998), however, a more formal definition of operations as part of the diagrammatic theory can be found in §3.4.3.

3.2. Constructing a Proof

Diamond's example proof consists of a sequence of applications of geometric operations on a diagram. The abstraction is then carried out automatically, if any such abstraction exists for the two example proofs given.[4] 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 of $(n+1)$ 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:

$$\begin{array}{lcl} \mathsf{proof}(n) & = & \mathcal{A}(n), \mathcal{A}(n-1), \ldots \mathcal{A}(1), \mathcal{B} \\ \mathsf{proof}(n+1) & = & \mathcal{A}(n+1), \mathcal{A}(n), \mathcal{A}(n-1), \ldots \mathcal{A}(1), \mathcal{B} \end{array}$$

where for each n, $\mathcal{A}(n)$ is a step case consisting of a sequence of applications of some operations and $\mathcal{B}$ is a base case for $n = 0$. Alternatively, we seek this recursive reformulation:

$$\begin{array}{lcl} \mathsf{proof}(n+1) & = & \mathcal{A}(n+1), \quad \mathsf{proof}(n) \\ \mathsf{proof}(0) & = & \mathcal{B} \end{array}$$

Note that $\mathsf{proof}(0)$ is often an empty list of operations, because often no diagram is defined for $n = 0$, *i.e.*, a diagram which consists of no dots.

[4] 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 abstract 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 abstract. This will be further explained in § 3.3.2.

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.

3.3. Abstraction

Given some example proofs Diamond needs to abstract 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 is a general program which specifies the applications of some operations, where the number of application of each operation is dependent on n or is a constant.

We distinguish between two types of example proofs: *destructor*, *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 *constructor*, *i.e.*, the base case operations are performed first followed by the step case operations. In Diamond we arbitrarily choose to use destructor schematic proofs.

A proof that has the same structure for all n, *i.e.*, one recursive function defines a complete proof for all n, is called a 1-homogeneous proof. Proofs can be c-homogeneous; then there are c cases of the proof. For instance, when there are two cases of a proof, one for odd and one for even natural numbers, then there need to be two recursive functions defining each case of the proof – the proof is 2-homogeneous. 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 abstracted, then the proof is c-homogeneous. If there are c cases, then there are c different abstracted proofs, one for each case. We seek the smallest complete recursive definition of a proof, *i.e.*, c potentially different schematic proofs, if there are c cases. The following theorem and corollary will help us define what we mean by the smallest complete proof:

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 schematic proof is defined to be the smallest complete proof if there is no other f-homogeneous proof obtainable from a c-homogeneous

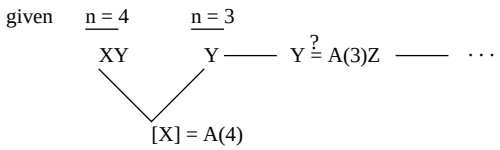
proof for any factor f of c, and all f schematic proofs for f cases are defined.

The general representation of a destructor proof is formalised as follows – let: $n = kc + r$ where $c = \textit{number of cases}$ and $r < c$, and $i \geq 1$. Then the recursive definition of a general proof is:

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

where $\mathcal{A}_r$ is a step case and $\mathcal{B}_r$ is a base case for a class of proofs where $n \equiv r(mod\, c)$. "," denotes concatenation of sequences of operations: "do operations of $\mathcal{A}(n+1)$, then $\mathsf{proof}(n)$". The formalisation of abstracted proof for *constructor* proofs is symmetric to the one given above.

3.3.1. *Abstracting 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 abstract them for all n. The general methodology employed for doing this can be demonstrated as:



The first step of the abstraction 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 abstracted, will be the step case $\mathcal{A}$ of the proof. This is done by commutative and associative matching[5] 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: $an + b$. This is a heuristically adequate choice. 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 abstraction is to extract the next step case from the rest of the example proof for the corresponding new n

[5] Using commutative and associative matching reduces the sensitivity to the order of proof steps (Jamnik, 1998).

(*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 abstractions of the step case. The aim is to eliminate those that are impossible. After checking if step cases for all n down to the base case are structurally consistent one hopes to be left with at least one possible abstraction 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 abstraction of the two given example proofs.

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

$$\begin{aligned} \mathcal{A}(n) &= [(\mathsf{lcut},1),\ (\mathsf{split_diagonal_ends},\mathrm{n}\text{-}1)] \\ \mathcal{B} &= [\,] \end{aligned}$$

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

3.3.2. *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 ell 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 abstract 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 abstraction would determine that the step case consists of two lcuts. However, a complete recursive function for abstraction requires a step case to consist of one lcut only.

DIAMOND checks this by trying to split the step case into a 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 abstraction of the step case $\mathcal{A}(n)$. If the method succeeds, and DIAMOND finds a new abstraction of the step case, call this $\mathcal{A}'(n)$, then it also needs to find a new base case $\mathcal{B}'_{r'}$ 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 $r' < f$.

3.4. CORRECTNESS OF THE SCHEMATIC PROOF

The last stage of extracting a diagrammatic proof is to check that the *guessed* general schematic proof is indeed correct. To prove that the schematic proof is correct we need to show in some meta-theory that $\mathsf{proof}(n)$ uniformly proves $P(n)$ for all n, *i.e.*, it gives a proof tree with $P(n)$ at its root, and axioms at its leaves. This requires reasoning about proofs, *i.e.*, meta-level reasoning. A meta-level proof using general diagrams would be an obvious method for verifying our schematic proof. However, such a meta-level proof would reintroduce the need for abstractions (*e.g.* ellipsis) of diagrams, which we are trying to avoid.

One way of overcoming this problem is to define diagrams and operations in a theory of diagrams where we can express abstract diagrams symbolically rather than diagrammatically. In this theory we can verify schematic proofs by defining the notion of applicability of a posited proof. Given that a particular theorem is expressed as an equality, its schematic proof is correct if applying the operations specified in the schematic proof of the diagrammatic representation of the left hand side of the theorem results in the diagrammatic representation of the right hand side of the theorem. There are two conditions that need to be satisfied. The first condition is that there is an appropriate diagrammatic representation available for the conversion of the theorem into its diagrammatic representation. The second condition is that the operations of the schematic proof are defined on those diagrams.

Before we can state the definition of the correctness property of schematic proofs, we need to formalise the machinery which will enable us to model the processes of a diagrammatic proof. Therefore, we need to formally define diagrams, operations on them, and the applicability of operations of a schematic proof.

3.4.1. *Diagrams*
Diagrams in the theory are defined to be of object type. Some examples of the different kinds of object names in the theory are: row, column, ell, frame, square, rectangle, and triangle.

Diagrams of the theory model natural numbers. DIAMOND's primitive notion of a concrete diagram, a dot, is represented in the theory as the natural number 1. Objects are introduced via a constructor function, diagram, which takes the name of the type of a diagram and the list of parameters of its magnitude. Thus, the type of constructor function diagram is name $\times$ pnat list $\rightarrow$ object. So, for instance, a square of magnitude 4 is expressed in the theory as diagram(square,[4]). All elementary and derived concrete diagrams are expressed using a primitive object dot, hence in the theory they can be expressed using a constructor function, the object name and some parameter representing a natural number for the magnitude of the diagram.

Constant $\emptyset$ denotes a null diagram, or in other words an empty diagram. We define that any diagram that is of 0 magnitude is an empty diagram (note that $a \in b$ denotes that a natural number a is an element of a list b; thus the type of an infix $\in$ is: pnat $\times$ pnat list $\rightarrow$ boolean):

$$0 \in s \rightarrow \mathsf{diagram}(x, s) \equiv \emptyset \tag{1}$$

Note also, that all triangles are equilateral.[6] Here are some examples of diagrams: diagram(row,n), diagram(column,n), diagram(square,n) and diagram(ell,n).

3.4.2. *Operators*
This section gives the operators available in the theory. First, we write diagrammatic equality using $\stackrel{d}{=}$ which denotes that two lists of diagrams are identical. Here is the definition of $\stackrel{d}{=}$:

$$\mathsf{X} \stackrel{d}{=} \mathsf{Y} \longleftrightarrow \forall \mathsf{d}.\ \mathsf{count}(\mathsf{d}, \mathsf{X}) = \mathsf{count}(\mathsf{d}, \mathsf{Y})$$

where the function count can be defined by:

$$\begin{aligned} \mathsf{count}(\mathsf{d}, [\,]) &= 0 \\ \mathsf{count}(\mathsf{d}, \mathsf{d} :: \mathsf{D}) &= 1 + \mathsf{count}(\mathsf{d}, \mathsf{D}) \\ \mathsf{d} \neq \mathsf{e} \rightarrow \mathsf{count}(\mathsf{d}, \mathsf{e} :: \mathsf{D}) &= \mathsf{count}(\mathsf{d}, \mathsf{D}) \end{aligned}$$

[6] It is hard to represent discrete triangles that are of any magnitude, *i.e.*, the sides are of different and any magnitudes. Triangles are represented in a discrete space. Hence, they appear to be right-angle triangles, despite the fact the all the sides of any triangle are of equal discrete magnitude. Were we to extend DIAMOND to prove theorems of real arithmetic (see §4), then there would be a need for continuous space, and therefore a scope for triangles of any magnitude.

Diagrammatic equality $\stackrel{d}{=}$ is a larger relation than an arithmetic equality $=$, because it has all the properties of $=$, *i.e.*, reflexivity, symmetry, transitivity and substitution properties, plus an additional one — the order of elements in a list does not matter. Therefore, two lists of diagrams, X and Y, are diagrammatically equal, $\mathsf{X} \stackrel{d}{=} \mathsf{Y}$, even if the orders in which the diagrams are listed in both lists differ.[7]

We now define some operators that introduce the existence of several diagrams (note that the data type pnat stands for non-negative natural number of Peano arithmetic): @ is append on lists,; :: and nil are list constructors (concatenation of elements onto a list, and an empty list); $\otimes$ is an infix operator which introduces a combination of a number of identical lists of diagrams; $\biguplus$ denotes a collection of diagrams of increasing magnitudes which are all of the same kind – it is analogous to $\sum$ for summation of integers. Here is the recursive definition of $\biguplus_{i=a}^{b}$ for all $a \leq b$:[8]

$$\biguplus_{i=a}^{a} \mathsf{diagram}(\mathsf{name}, f(i)) \stackrel{d}{=} [\mathsf{diagram}(\mathsf{name}, f(a))] \tag{2}$$

$$a \leq b \rightarrow \biguplus_{i=a}^{b+1} \mathsf{diagram}(\mathsf{name}, f(i)) \stackrel{d}{=} \biguplus_{i=a}^{b} \mathsf{diagram}(\mathsf{name}, f(i))@ [\mathsf{diagram}(\mathsf{name}, f(b+1))] \tag{3}$$

Note that f is some function which generates a list of natural numbers for a given number i. This list denotes the parameters of a magnitude of a diagram.

3.4.3. *Operations*

Diagrammatic operations are represented via a function $\mathsf{op} : \mathsf{opname} \times \mathsf{object\ list} \rightarrow \mathsf{object\ list}$. We give here a definition of one operation only, but there are many more operations defined in the theory – see (Jamnik, 1998).

$$\mathsf{op}(\mathsf{lcut}, \mathsf{diagram}(\mathsf{square}, n) :: \mathsf{D}) \stackrel{d}{=} [\mathsf{diagram}(\mathsf{square}, n-1), \mathsf{diagram}(\mathsf{ell}, n)]@\mathsf{D} \tag{4}$$

3.4.4. One_Apply *and* Apply

Here we define what it means to apply an operation on a diagram several times. We use a function apply and function one_apply. Let:

[7] Note that our definition of diagrammatic equality of lists is equivalent to bag equality. The order of the elements in a bag does not matter.

[8] Note that to simplify the notation we write $\biguplus_{i=a}^{b} D(i)$ instead of $\biguplus(a, b, \lambda i.D(i))$.

$$\mathsf{one_apply}(0, \mathsf{opnm}, \mathsf{D}) \stackrel{d}{=} \mathsf{D} \tag{5}$$

$$\mathsf{one_apply}(n+1, \mathsf{opnm}, \mathsf{D}) \stackrel{d}{=} \mathsf{op}(\mathsf{opnm}, \mathsf{one_apply}(n, \mathsf{opnm}, \mathsf{D})) \tag{6}$$

$$\mathsf{apply}([\,], \mathsf{D}) \stackrel{d}{=} \mathsf{D} \tag{7}$$

$$\mathsf{apply}((\mathsf{opnm}, x) :: \mathsf{opss}, \mathsf{D}) \stackrel{d}{=} \mathsf{apply}(\mathsf{opss}, \mathsf{one_apply}(x, \mathsf{opnm}, \mathsf{D}) \tag{8}$$

3.4.5. *Equations*
Here we give a theorem which will be needed. (9) is provable from (7) and (8). Its proof is not given here, but can be found in (Jamnik, 1998).

$$\mathsf{apply}(\mathsf{ops}, \mathsf{D} :: \mathsf{D}_s) \stackrel{d}{=} \mathsf{apply}(\mathsf{ops}, [\mathsf{D}])@\mathsf{D}_s \tag{9}$$

3.4.6. *Conversion relation* dmap
Let dmap denote a relation between a particular class of statements of arithmetic and their equivalent diagrammatic expressions in the theory of diagrams. The equivalence is defined to be over the *size* of the diagram. The size of a diagram is defined to be the number of counters (dots) in the diagram, *i.e.*, the natural number that the diagram represents. dmap takes two arguments, an arithmetic expression and a list of diagrams which could collectively represent this expression. Hence, the type of the relation dmap is pnat $\times$ object list. Here are some general conversions:

$$\mathsf{dmap}(0, [\,]) \tag{10}$$

$$\mathsf{dmap}(n^2, [\mathsf{diagram}(\mathsf{square}, [n])]) \tag{11}$$

$$\mathsf{dmap}(2n-1, [\mathsf{diagram}(\mathsf{ell}, [n])]) \tag{12}$$

$$\mathsf{dmap}(\textstyle\sum_{j=a}^{b} f(j), \biguplus_{j=a}^{b} \mathsf{D}_j) \text{ such that } \forall j, a \leq j \leq b,\ \mathsf{dmap}(f(j), [\mathsf{D}_j]) \tag{13}$$

We have now formalised enough machinery to be able to define the correctness property of a schematic proof.

DEFINITION 1 (Correctness of Schematic Proofs).
proof *is a correct schematic proof of a particular conjecture* $\forall n\ \ L(n) = R(n)$ *if for all* n *there exist two lists of diagrams* D *and* E *such that* $\mathsf{dmap}(L(n), \mathsf{D})$ *and* $\mathsf{dmap}(R(n), \mathsf{E})$*, and*

$$\mathsf{apply}\,(\mathsf{proof}\,(n),\ \mathsf{D}) \stackrel{d}{=} \mathsf{E}$$

It is possible to prove the property in Definition 1 only if $L(n)$, $R(n)$ and proof are known, *i.e.*, for a specific case of a conjecture and a schematic proof. Knowing $L(n)$ and $R(n)$ allows us to infer some conversion relations which specify two lists of diagrams D and E. This satisfies the first part of Definition 1. In the next section we prove the correctness of a schematic proof for a particular conjecture at hand.

3.4.7. *Proof of Correctness of Schematic Proofs for an Example*
Here we prove the property given in Definition 1 for an example of a schematic proof of a theorem about the *sum of odd naturals*. The theorem is stated as $n^2 = \sum_{i=0}^{n}(2i-1)$. The schematic proof of this theorem is given as:[9]

$$\mathsf{proof}(0) \;=\; [\,] \tag{14}$$
$$\mathsf{proof}(n+1) \;=\; [(\mathsf{lcut}, 1)],\ \mathsf{proof}(n) \tag{15}$$

The proof of correctness of a schematic proof for this particular example requires induction on n. The base case for $n = 0$ is trivial, since by (10) no operations are applied to an empty diagram list which results in [ ]. We consider a step case of induction.

Step case:

Hypothesis: for n
Using (11) notice $\mathsf{dmap}(n^2, [\mathsf{diagram}(\mathsf{square}, [n])])$,
hence let $\mathsf{D} = [\mathsf{diagram}(\mathsf{square}, [n])]$.
Using (13) and (12) notice $\mathsf{dmap}(\sum_{i=0}^{n}(2i-1), \biguplus_{i=0}^{n} \mathsf{diagram}(\mathsf{ell}, [i]))$,
hence let $\mathsf{E} = \biguplus_{i=0}^{n} \mathsf{diagram}(\mathsf{ell}, [i])$.

$$\mathsf{apply}(\mathsf{proof}(n), [\mathsf{diagram}(\mathsf{square}, [n])]) \stackrel{d}{=} \bigoplus_{i=0}^{n} \mathsf{diagram}(\mathsf{ell}, [i])$$

Conclusion: for $n+1$
Similarly to the hypothesis, D and E are converted for $n+1$.

$$\mathsf{apply}(\mathsf{proof}(n+1), [\mathsf{diagram}(\mathsf{square}, [n+1])]) \stackrel{d}{=} \bigoplus_{i=0}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i])$$
$$\mathsf{proof}(n+1) = [(\mathsf{lcut}, 1)],\ \mathsf{proof}(n) \;\Downarrow$$
$$\mathsf{apply}(((\mathsf{lcut}, 1),\ \mathsf{proof}(n)), [\mathsf{diagram}(\mathsf{square}, [n+1])]) \stackrel{d}{=} \bigoplus_{i=0}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i])$$
$$(8) \;\Downarrow$$

[9] For the brevity of presentation we take a simpler version of the schematic proof which does not include the operation split_ends.

$$\mathsf{apply}(\mathsf{proof}(n), \mathsf{one_apply}(1, \mathsf{lcut}, [\mathsf{diagram}(\mathsf{square}, [n+1])])) \stackrel{d}{=} \biguplus_{i=0}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i])$$

(6) and (5) $\Downarrow$

$$\mathsf{apply}(\mathsf{proof}(n), \mathsf{op}(\mathsf{lcut}, [\mathsf{diagram}(\mathsf{square}, [n+1])])) \stackrel{d}{=} \biguplus_{i=0}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i])$$

(4) $\Downarrow$

$$\mathsf{apply}(\mathsf{proof}(n), [\mathsf{diagram}(\mathsf{square}, [n]), \mathsf{diagram}(\mathsf{ell}, [n+1])]) \stackrel{d}{=} \biguplus_{i=0}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i])$$

(9) $\Downarrow$

$$\mathsf{apply}(\mathsf{proof}(n), [\mathsf{diagram}(\mathsf{square}, [n])]) @ [\mathsf{diagram}(\mathsf{ell}, [n+1])] \stackrel{d}{=} \biguplus_{i=0}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i])$$

(RHS of hypothesis) $\Downarrow$

$$\biguplus_{i=0}^{n} \mathsf{diagram}(\mathsf{ell}, [i]) @ [\mathsf{diagram}(\mathsf{ell}, [n+1])] \stackrel{d}{=} \biguplus_{i=0}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i])$$

(3) $\Downarrow$

$$\biguplus_{i=0}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i]) \stackrel{d}{=} \biguplus_{i=0}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i])$$

■

3.5. Arithmetic Conjecture and Diagrammatic Proof

Definition 1 makes no claims about the link between a schematic proof and the theoremhood of a conjecture $\forall n\ L(n) = R(n)$. We still need to consider the possibility of a *correct* schematic proof of a *false* conjecture. To establish that the conjecture is true when proved by a schematic proof, an explicit algebraic link between them needs to be defined. We establish this link via the *size of diagrams*. We first define the size of a diagram, and later, in Theorem 2, we state the theorem about the algebraic correctness of a schematic proof for a given conjecture.

Let us denote the size of the diagram D by $|\,\mathsf{D}\,|$. Here is a definition for the size of a diagram:

DEFINITION 2 (Size of Diagrams).
The size of a list of diagrams is equal to the value of the arithmetic expression that it represents: if $\mathsf{dmap}(e, \mathsf{D})$ *then* $|\,\mathsf{D}\,| = e$.

Note that the type of $|\ |$ is: $\mathsf{object\ list} \to \mathsf{pnat}$. Using the property of size defined in Definition 2 on formulae from (10) to (13), we have the

following:

$$|\,[\,]\,| = 0 \tag{16}$$

$$|\,[\mathsf{diagram}(\mathsf{square}, [n])]\,| = n^2 \tag{17}$$

$$|\,[\mathsf{diagram}(\mathsf{ell}, [n])]\,| = 2n - 1 \tag{18}$$

$$\left| \biguplus_{j=a}^{b} \mathsf{D}_j \right| = \sum_{j=a}^{b} |\,[\mathsf{D}_j]\,| \tag{19}$$

Apart from being diagrammatically correct, we want every schematic proof to be *algebraically correct* as well. A schematic proof is algebraically correct if the sizes of the diagrams representing both sides of the proposition after the operations of the schematic proof have been applied are the same. Theorem 2 states the property of algebraic correctness for any schematic proof.

THEOREM 2 (Algebraic Correctness of Schematic Proofs).
For all instances of a schematic proof P *and for all pairs of lists of diagrams* D *and* E*, a schematic proof* P *is algebraically correct if and only if*

$$\mathsf{apply}\,(\mathsf{P}, \mathsf{D}) \stackrel{d}{=} \mathsf{E} \longrightarrow |\,\mathsf{D}\,| = |\,\mathsf{E}\,|$$

The proof of Theorem 2 is straightforward by appealing to the properties of diagram size invariance under applications of multiple operations. The lemmas about these properties and the proof of Theorem 2 are not given here, but can be found in (Jamnik, 1998).

There is one last theorem needed in the formalisation of diagrammatic theory which will allow us *to prove* theorems of arithmetic using *diagrammatic* proofs. We state in Theorem 3 the property about the diagrammatic provability of arithmetic arguments.

THEOREM 3 (Diagrammatic Provability of Arithmetic Conjecture).
A conjecture $\forall n\ L(n) = R(n)$ *is diagrammatically provable if and only if for all* n *there exist two lists of diagrams* D *and* E *such that* $\mathsf{dmap}(L(n), \mathsf{D})$ *and* $\mathsf{dmap}(R(n), \mathsf{E})$*, and*

$$|\,\mathsf{D}\,| = |\,\mathsf{E}\,| \longrightarrow L(n) = R(n)$$

The proof of Theorem 3 is trivial by the definition of size of a list of diagrams given in Definition 2.

3.5.1. *Diagrammatic Provability for an Example*
We consider now an example of an arithmetic conjecture and prove it diagrammatically using a schematic proof that DIAMOND extracts. Let the arithmetic conjecture be

$$\forall n\ n^2 = \sum_{i=0}^{n} 2i - 1$$

and the schematic proof proof that DIAMOND extracted be as defined in (14) and (15). Here are the reasoning steps of the proof:

1. Appealing to Theorem 3 we can discharge the conjecture by:
   - using (11) notice $\mathsf{dmap}(n^2, [\mathsf{diagram}(\mathsf{square}, [n])])$, hence let $\mathsf{D} = [\mathsf{diagram}(\mathsf{square}, [n])]$,
   - using (13) and (12) notice $\mathsf{dmap}(\sum_{i=0}^{n}(2i-1), \biguplus_{i=0}^{n} \mathsf{diagram}(\mathsf{ell}, [i]))$, hence let $\mathsf{E} = \biguplus_{i=0}^{n} \mathsf{diagram}(\mathsf{ell}, [i])$,

   and proving for all n

$$|\,[\mathsf{diagram}(\mathsf{square}, [n])]\,| \;=\; \left| \bigoplus_{i=0}^{n} \mathsf{diagram}(\mathsf{ell}, [i]) \right| \tag{20}$$

2. Appealing to Theorem 2 and $\mathsf{proof}(n)$ that DIAMOND extracted, we can discharge the expression in (20) by proving for all n

$$\mathsf{apply}\,(\mathsf{proof}(n), [\mathsf{diagram}(\mathsf{square}, [n])]) \stackrel{d}{=} \bigoplus_{i=0}^{n} \mathsf{diagram}(\mathsf{ell}, [i]) \tag{21}$$

3. Finally, notice that we already proved (21) in §3.4.7.

■

4. Results and Further Work

DIAMOND is implemented in Standard ML of New Jersey, Version 109.[10] The code is available upon request to the first author.

The entire process of interactive construction of proofs, automatic abstraction from example proofs, and automatic verification of schematic proofs in the theory of diagrams have been implemented in DIAMOND. In the evaluation of DIAMOND we distinguished between a development and a test set of theorems which we proved using DIAMOND. The development set of theorems included three theorems: *the sum of odd naturals*, *i.e.*, $n^2 = \sum_{i=0}^{n} 2i - 1$, *the sum of all naturals*, *i.e.*, $\frac{n(n+1)}{2} = \sum_{i-0}^{n} i$, and *an odd triangular sum*, *i.e.*, $Tri_{2n+1} = Tri_{n+1} + 3Tri_n$, where Tri_i is an i-th triangular number (Nelsen, 1993). The

[10] Standard ML of New Jersey (SML/NJ) is a compiler and programming environment for the Standard ML programming language. SML/NJ is publicly available via the internet on the following site: `http://cm.bell-labs.com/cm/cs/what/smlnj/-index.html`

test set included twenty six theorems. Some proofs of these theorems are reported more elaborately in (Jamnik, 1998). All of these theorems contribute to the significant range and depth of theorems proved using DIAMOND. For more information, the reader is referred to (Jamnik, 1998).

We want to relax the restriction currently imposed on the formulation of example proofs. Our abstraction mechanism can deal with a linear sequence of operations. This sequence is in fact a linearisation of some partially ordered sequence of operation. We want an abstraction mechanism which would be sensitive to partially ordered sequences of operations.

There is also a possibility of allowing non-linear dependency functions in general schematic proofs: *e.g.* exponential or polynomial function.

Some recognition and generalisation of diagrams using abstractions could be an interesting issue to consider. This requires some formalisation of abstractions (*e.g.* ellipsis) in diagrams.

There is a possibility to extend DIAMOND's problem domain from natural number arithmetic to geometry or even further to a different field such as hardware verification. Extending our problem domain to geometry would enable us to prove theorems of Category 1 which are usually geometric theorems of continuous space. These do not require induction, hence there would be no need for DIAMOND's abstraction mechanism. The generality is embedded in the use of continuous space and diagrams of general magnitude. The existing operations and the formalisation of schematic proofs can be used. Additional operations for moving diagrams in various directions would need to be implemented. For more information on the possible extension of DIAMOND to other problem domains, see (Jamnik, 1998).

DIAMOND is an interactive proof checker. A long term goal is to design an automated theorem prover capable of discovering diagrammatic proofs.

5. Related Work

Several diagrammatic systems such as the Geometry Machine (Gelernter, 1963), Diagram Configuration model (Koedinger & Anderson, 1990), GROVER (Barker-Plummer & Bailin, 1997), and Hyperproof (Barwise & Etchemendy, 1991) have been implemented in the past and are of relevance to our system. Additional information about issues in reasoning with diagrams can be found in (Chandrasekaran *et al*, 1995) which is a good reference for demonstrating how extensive and important this field is.

One of the first systems to use a diagram in proving theorems was Gelernter's Geometry Machine (Gelernter, 1963). The diagram in the geometry machine has two roles. Its *negative* role is to reject hypotheses (subgoals) that are not true in the diagram. In this way the search space is pruned. The *positive* role of the diagram is to shorten the inference paths by assuming various facts that are obvious in the diagram as true.

(Koedinger & Anderson, 1990) implemented a geometry problem solver called the Diagram Configuration (DC) model. The key feature of the system is that it organises its data in perceptual chunks, called diagram configurations. These are analogical to key features of diagrams that humans recognise when they inspect a diagram. Therefore, during the process of generating a solution path, DC infers the key steps first, and ignores along the way the less important features of the diagram, *i.e.*, the less important inference steps.

"&"/GROVER, developed by (Barker-Plummer & Bailin, 1997) is an automated reasoning system which uses information from a diagram to guide proof search. Its problem domain are theorems of well founded relations. It consists of the "&" automated theorem prover, based on the sequent calculus for Zermelo set theory, and GROVER, which is the diagram interpreting component of the system. It passes information extracted from the diagram and translated into logical formulae in the language of "&" to the "&" theorem prover. These formulae are then used as additional hypotheses to the main proof of the conjecture. GROVER considers only subgoals that are known to be true in the diagram.

The main common feature of these three systems is their use of diagrams. The diagram is used to model algebraic statements, and the system uses these models for heuristic guidance while searching for an *algebraic* proof. Thus, the basic underlying reasoning process is non-diagrammatic. In contrast, proofs in DIAMOND are explicitly constructed by operations on diagrams, thus the inference steps of the proof are entirely diagrammatic.

Perhaps a more closely related system to DIAMOND is Hyperproof (Barwise & Etchemendy, 1991). Hyperproof reasons about the blocks world. It is an educational tool to teach principles of logic reasoning and proof construction. Hyperproof is an interactive tool for proof checking, as opposed to an automated theorem prover. It is a heterogeneous logic system, because it models inferencing between different kinds of representation. Unlike traditional systems for first-order logic, which use sentential representation, Hyperproof uses sentential and diagrammatic (graphical) representation. It uses a diagram for a concise representation of a complex system aiming to aid human reasoning. The user

can take advantage either of conventional sentential inference rules or diagrammatic inference rules. It differs from DIAMOND in that Hyperproof's diagrammatic inference rules deduce from a diagram to a sentential formula or vice versa. It does not have diagrammatic inference rules between two diagrams, as it is the case in DIAMOND. Moreover, in Hyperproof the sentential inference rules (as well as diagrammatic description of a situation) are essential to construct a proof.

Closer to our work is work done by Siani Baker (Baker *et al*, 1992) described in §2.4 on constructive ω-rule, whereby she exploits the uniform structure of inductive proofs to abstract from example proofs. The main difference between Baker's work and ours is the problem domain. Baker implemented the use of constructive ω-rule for proving arithmetic theorems. Our domain, on the other hand, is diagrammatic theorems. Furthermore, Baker's motivation was to use schematic proofs for theorems that require a cut rule in the inductive proof, otherwise the proof cannot be carried out automatically. Schematic proofs avoid the need for a cut rule. On the other hand, we use the constructive ω-rule in order to justify the automatic provision of general arguments about theorems and their proofs from particular instances.

6. Conclusion

One of the aims in the research reported here is to see whether it is possible to automate the use of diagrams in formal proofs. 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. We have made good progress in exploring this important and difficult area. In particular, we have an explicit handle on abstraction. We showed that diagrams *can* be used for *formal* proofs. We presented, as an example, a diagrammatic reasoning system, DIAMOND, which supports interactive construction of diagrammatic proofs. DIAMOND applies diagrammatic reasoning to problem solving in mathematics. The user proves concrete examples of a theorem, and the system automatically abstracts these instances to give a general schematic proof which we hope holds for all n. In DIAMOND we have the logical machinery (meta-theory, constructive ω-rule) to subsequently justify that the schematic proof does indeed prove the original theorem.

Acknowledgements

The research reported in this paper was supported by an Artificial Intelligence Department Studentship, from 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.), *International Conference on Logic Programming and Automated Reasoning — LPAR 92, St. Petersburg*, Lecture Notes in Artificial Intelligence No. 624, pages 214–225. Springer-Verlag.

Barker-Plummer, D. and Bailin, S. C. (1997). The role of diagrams in mathematical proofs. *Machine Graphics and Vision*, 6(1).

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

Chandrasekaran, B., Glasgow, J. and Narayanan, N. H., (eds.). (1995). *Diagrammatic Reasoning: Cognitive and Computational Perspectives*. AAAI Press/The MIT Press.

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

Hammer, E. M. (1995). *Logic and visual information*. Stanford, California: Center for the Study of Language and Information.

Jamnik, M., Bundy, A. and Green, I. (1997). Automation of diagrammatic reasoning. In Pollack, M.E., (ed.), *Proceedings of the 15th IJCAI*, volume 1, pages 528–533. International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers. Also published in the Proceedings of the 1997 AAAI Fall Symposium. Also available from Edinburgh as DAI Research Paper No. 873.

Jamnik, M. (1998). Automating diagrammatic proofs of arithmetic arguments. Forthcoming PhD thesis. Available from Edinburgh University.

Koedinger, K. R. and Anderson, J. R. (1990). Abstract planning and perceptual chunks. *Cognitive Science*, 14:511–550. Reprinted in "Diagrammatic Reasoning: Cognitive and Computational Perspectives", Glasgow, J., Narayanan, N. H., and Chandrasekaran B. (eds.), AAAI Press/The MIT Press, 1995, pages 577-625.

Larkin, J. H. and Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11:65–99. Reprinted in "Diagrammatic Reasoning: Cognitive and Computational Perspectives", Glasgow, J., Narayanan, N. H., and Chandrasekaran B. (eds.), AAAI Press/The MIT Press, 1995, pages 69-109.

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

Pólya, G. (1945). *How to solve it*. Princeton University Press.

Pólya, G. (1965). *Mathematical discovery*. John Wiley & Sons, Inc, Two volumes.

Shin, S. J. (1995). *The Logical Status of Diagrams*. Cambridge University Press.

H.A. Simon. (1996). *The Sciences of the Artificial*. MIT Press, Cambridge, Mass., third edition.

Sowa, J. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley.

Stenning, K. and Oberlander, J. (1995). A cognitive theory of graphical and linguistic reasoning: Logic and implementation. *Cognitive Science*, 19:97–140.