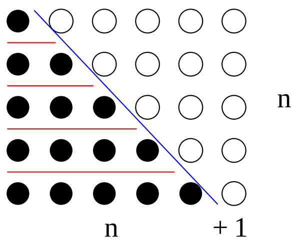
Verification of Diagrammatic Proofs

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

Human mathematicians often "prove" theorems by the use of diagrams and manipulations on them. We call these diagrammatic proofs. In (Jamnik, Bundy, & Green 1997) we presented how "informal" reasoning with diagrams can be automated. Three stages of proof extraction procedure were identified. First, concrete rather than general diagrams are used to prove particular instances of the universally quantified theorem. The diagrammatic proof is captured by the use of geometric operations on the diagram. Second, an abstracted schematic proof of the universally quantified theorem is automatically induced from these proof instances. Third, the final step is to confirm that the abstraction of the schematic proof from the proof instances is sound. Our main focus in this paper is on the third stage, the verification of schematic proofs. We define a theory of diagrams where we prove the correctness of a schematic proof. We give an example of an extraction of a schematic proof for a theorem about the *sum of odd naturals*, and prove its correctness in the theory of diagrams.

Introduction



$$1+2+3+\cdots+n=\frac{n\times(n+1)}{2}$$

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 natural numbers*.

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 emulate it on a computer. It should be pointed out that we are not trying to discover automatically the diagrammatic proofs, but rather to mechanise and explore them in order to understand them better.

In (Jamnik, Bundy, & Green 1997) we presented a way of formalising diagrammatic reasoning for natural number arithmetic and showed how diagrams can be used for proofs in a formal system. Rather than using general diagrams, which need abstractions to represent their generality, we reason with concrete diagrams (*i.e.* of a particular size). Theorems of mathematics can be expressed as diagrams for some concrete values, *i.e.* ground instantiations of a theorem. We presented a three-stage algorithm for extraction of a diagrammatic proof. First, 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. Second, a general pattern is extracted from these proof instances, and is captured in a recursive program. This recursive program, referred to as a schematic proof, constitutes a general diagrammatic proof for the universally quantified theorem. Third, the induced schematic proof has to be verified to show that it indeed proves a proposition for all cases.

The main part of this paper deals with the last step in the extraction of a diagrammatic proof, *i.e.* the verification step. In particular, we define a theory of diagrams

[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.

which models the processes in a diagrammatic proof. In this theory, we can prove by a meta-level inductive proof that a particular schematic proof is correct. There are two main motivations for defining a theory of diagrams in which schematic proofs are verified. The first one is that a diagrammatic schematic proof is an intelligent guess by a machine of the general form of a proof. In this sense it is much the same as what humans do when they induce an abstraction from examples. In an automated reasoning system, this guess needs to be formally checked for correctness. Secondly, we choose to avoid using abstractions (such as ellipsis) in general diagrams which would be needed if such general diagrams were used for checking correctness. We discussed in (Jamnik, Bundy, & Green 1998) why abstractions are problematic to use.

All three stages of the algorithm for formalisation of diagrammatic proofs are implemented in a diagrammatic proof system called DIAMOND (DIAgraMmatic reasONing and Deduction). Rather than automatically discovering diagrammatic proofs, we use DIAMOND to try to understand better informal diagrammatic proofs. 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. The correctness of a schematic proof is verified in the theory of diagrams. The verification proof is carried out in the proof planning system *Clam*.

The work reported in this paper is intended to be self contained. Therefore, the next three sections present the background information on the formalisation of diagrammatic reasoning to enable us to put the main results of this paper in the appropriate context. First, we give an example of a diagrammatic theorem. In the subsequent couple of sections we introduce schematic proofs. Next, the implementation of DIAMOND is demonstrated. The presentation of the main result of this paper follows, *i.e.* we define the theory of diagrams and prove the correctness theorem for an example of a diagrammatic schematic proof. Then we report some of our results and discuss future work. We mention some of the related diagrammatic reasoning systems next. Finally, we conclude by summarising the main points of this paper.

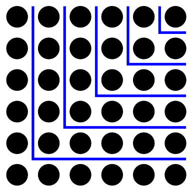
'Diagrammatic' Theorems

We are interested in mathematical theorems that admit diagrammatic proofs. 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. We presented several examples of diagrammatic theorems and their proofs in (Jamnik, Bundy, & Green 1997). We distinguished between three categories of examples, which are by no means exhaustive, and decided to concentrate on the examples of Category 2, an example of which is the theorem about the *sum of odd naturals*. For examples of other categories, see (Jamnik, Bundy, & Green 1997). Other specification of our problem domain include theorems whose instances can be represented as diagrams without the need for abstraction (*e.g.*, the use of ellipsis), and theorems of natural number arithmetic.

Sum of Odd Naturals

This example is taken from (Nelsen 1993). The theorem about the *sum of odd naturals* states the following:

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



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 size 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 size of $(2n - 1)$, where n is the size of the side of the square.

From the analysis of the example above and many others (see (Jamnik, Bundy, & Green 1997) and (Jamnik, Bundy, & Green 1998)), we summarise now the characteristics of examples of Category 2. Theorems of Category 2 are theorems of discrete space. A diagram is a representative of a particular instance of a theorem. Proofs are schematic: they 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 in section "Schematic Proof and Constructive ω-rule") is used to formally justify the extraction of a general proof from the individual proof instances.

Schematic Proof and Constructive ω-rule

Schematic proofs use the constructive ω-rule. The constructive ω-rule allows inference of the sentence $\forall x P(x)$ from an infinite sequence $P(n)$, $n \in \omega$ of sentences by requiring to provide 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 $f_R(n)$ times. By instantiation $\mathsf{proof}(n)$ generates a proof of $P(n)$ for every n.

Diagrams and Schematic Proofs

Baker did some work on the use of schematic proofs of arithmetic theorems (Baker, Ireland, & Smaill 1992). We 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 following is an algorithm for extraction of a *diagrammatic* schematic proof:

1. The diagrammatic schematic proof starts with a few particular concrete cases of a theorem represented by a diagram. The geometric operations on the diagram are performed after that, capturing the inference steps of the diagrammatic proof.
2. Next, we abstract (using a learning type inference) the operations involved in the schematic proof for n. Note that the generality is represented as a recursive program which specifies a sequence of geometric operations that are used on a diagram, and not as a general representation of a diagram.
3. The last step is to prove by meta-induction that the abstracted diagrammatic schematic proof is indeed correct. We carry out the verification in a theory of diagrams that models the processes in a diagrammatic reasoning system and prove correctness there. This will be discussed in section "Correctness of Schematic Proofs".

The schematic diagrammatic proof for the *sum of odd naturals* can be more formally expressed as:

- Cut a square into n ells, where an ell consists of 2 adjacent sides of the square.
- For each ell, continue splitting from an ell $n-1$ pairs of dots at the end of two adjacent sides of the ell until only 1 dot is left, hence $2(n-1)+1$.

DIAMOND System

The diagrammatic proof system DIAMOND is an embodiment of the ideas presented in this paper (see also (Jamnik, Bundy, & Green 1998)).

Clearly, an important issue in the development of DIAMOND is the *internal* representation of diagrams and operations on them. 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. It is the interface between DIAMOND and the user. The *inference engine* deals with the diagrammatic inference steps, processes the operations on the diagram, extracts general schematic proofs from example proofs, and checks the correctness of schematic proofs.

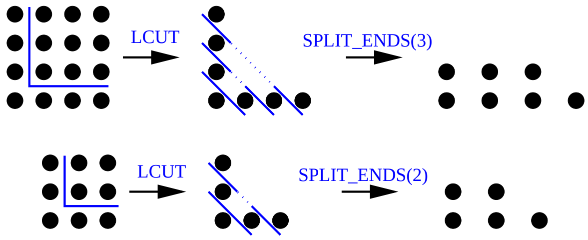
Geometric operations capture the inference steps of the proof. The user is expected to select from a set of all available operations the ones which are applied in an example proof. We distinguish between two types of operations: *atomic* operations (simple, one-step; *e.g.* split, rotate) and *composite* operations (complex, recursive; *e.g.* decompose into ells).

The rest of this section presents how instances of proofs are constructed, the structure of proofs and the abstraction mechanism used in DIAMOND.

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. 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. The abstraction mechanism, which will be described next, extracts the structure common to example proofs.

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. The aim is to reformulate example proofs for n and $n+1$ in the general case into a schematic proof. 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.

The general representation of an abstracted schematic proof is formalised as a recursive program:

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

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 $n=1$. "," denotes concatenation of sequences of operations: "do operations of $\mathcal{A}(n+1)$, then $\mathsf{proof}(n)$".

The number of applications of each operation in the step case $\mathcal{A}$ of the schematic proof is dependent on the parameter n. DIAMOND can automatically detect any linear dependency in the number of applications of an operation. For more information, the reader is referred to (Jamnik, Bundy, & Green 1998).

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

$$\begin{aligned}\mathcal{A}(n) &= [(\mathsf{lcut}, 1), (\mathsf{split\_diagonal\_ends}, n-1)]\\ \mathcal{B} &= [\mathsf{lcut}, 1)]\end{aligned}$$

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

Correctness of Schematic Proofs

The last stage of extracting a diagrammatic proof is to check that the *guessed* general schematic proof is indeed correct. In particular, the verification ensures that the transition from concreteness to generality of a diagrammatic proof is correct. In human reasoning this step is often omitted when humans are convinced that the examples used to induce a general schematic proof uniformly account for all cases of a theorem. This can sometimes result in erroneous proofs. In an automated reasoning system, however, we need to *formally* show the correctness of a schematic proof. 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 meta-level proof reintroduces the need for manipulating abstractions (*e.g.* ellipsis) on 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 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 on 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 mapping of the theorem into its diagrammatic representation. The second condition is that the operations of the schematic proof are defined. A verification proof is a meta-level proof, because it is a proof about a property of an object level schematic proof.

Before we can state the definition for 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 define diagrams, operations on them, and the applicability of operations of a schematic proof, which we do next.

Diagrams

Diagrams in the theory are defined to be of object type. Here are examples of names of diagrams in the theory: row, column, ell, frame, square, $\mathsf{rectangle}$, $\mathsf{triangle}$,...

Diagrams of the theory model natural numbers. DIAMOND's primitive notion of a diagram, a dot, is modelled in the theory as the natural number 1. Objects are introduced via a function $\mathsf{diagram}$ which takes the name of the type of a diagram and the parameters for its size.[2] So for instance, a square of size 4 is expressed in the theory as $\mathsf{diagram}(\mathsf{square}, [4])$. All elementary and derived objects are expressed using a primitive object dot.

Constant $\emptyset$ denotes a null diagram, or in other words an empty diagram. Note also, that all triangles are isosceles. Here are some examples of diagrams: $\mathsf{diagram}(\mathsf{row}, n)$, $\mathsf{diagram}(\mathsf{column}, n)$, $\mathsf{diagram}(\mathsf{ell}, n)$, $\mathsf{diagram}(\mathsf{square}, n)$.

Operators

This section gives the operators available in the theory. First, we denote the diagrammatic equality by $\stackrel{d}{=}$ which denotes that two lists of diagrams are identical.[3] This is to distinguish it from the usual arithmetic equality. These are the operators that introduce the existence of several diagrams: @ for append on lists, :: for list constructor, $\otimes$ for an infix operator which introduces a combination of a number of identical diagrams, $\biguplus_{i=a}^{b} : \mathsf{object} \rightarrow \mathsf{object\ list}$. Here is the recursive definition of $\biguplus_{i=a}^{b}$ for all $a \leq b$:

$$\biguplus_{i=a}^{a} \mathsf{diagram}(\mathsf{name}, f(i)) \stackrel{d}{=} [\mathsf{diagram}(\mathsf{name}, f(a))] \qquad (1)$$

$$\biguplus_{i=a}^{b+1} \mathsf{diagram}(\mathsf{name}, f(i)) \stackrel{d}{=} \biguplus_{i=a}^{b} \mathsf{diagram}(\mathsf{name}, f(i))@ [\mathsf{diagram}(\mathsf{name}, f(b+1))] \qquad (2)$$

Note that $f(i)$ is some list of functions of i that denote the parameters of a size of a diagram.

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

[2] The size of a diagram should not be confused with the arithmetic size of the diagram. The notion of an arithmetic size of a diagram will be explained in section "Mapping relation dmap".

[3] To be more precise, $\stackrel{d}{=}$ denotes an equality of two lists of diagrams modulo additional information about the position of a diagram in the proof tree attached to each diagram, and modulo the order of the list, *i.e.* two lists with the same diagrams but in different order are still the same. Therefore, rather than lists we could use bags which are sometimes called multisets. This is what is used in the implementation of the theory, but for the simplicity of presentation in this paper, we use lists as a datatype for collections of diagrams.

more operations defined in the theory. Any other, non-defined combination of a diagram and an operation is invalid.

$$\mathsf{op}(\mathsf{lcut}, \mathsf{diagram}(\mathsf{square},[n])@\mathsf{D}) \stackrel{d}{=} [\mathsf{diagram}(\mathsf{square}, [n-1]), \mathsf{diagram}(\mathsf{ell}, [n])]@\mathsf{D} \quad (3)$$

Function Definitions

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

$$\mathsf{one\_apply}(0, \mathsf{opnm}, \mathsf{D}) \stackrel{d}{=} \mathsf{D} \quad (4)$$

$$\mathsf{one\_apply}(\mathsf{n}+1, \mathsf{opnm}, \mathsf{D}) \stackrel{d}{=} \mathsf{op}(\mathsf{opnm}, \mathsf{one\_apply}(\mathsf{n}, \mathsf{opnm}, \mathsf{D})) \quad (5)$$

$$\mathsf{apply}([\,], \mathsf{D}) \stackrel{d}{=} \mathsf{D} \quad (6)$$

$$\mathsf{apply}((\mathsf{opnm}, \mathsf{x}) :: \mathsf{opss}, \mathsf{D}) \stackrel{d}{=} \mathsf{apply}(\mathsf{opss}, \mathsf{one\_apply}(\mathsf{x}, \mathsf{opnm}, \mathsf{D}) \quad (7)$$

Note that opss is a list of pairs of operation and the number of times that this operation is applied to a diagram.

Mapping relation dmap Let the dmap relation denote a mapping of a particular class of statements of arithmetic to their equivalent diagrammatic expressions in the theory of diagrams. dmap is used for linking a theorem of arithmetic which is stated symbolically to its diagrammatic representation and diagrammatic proof. The equivalence is defined to be over the *arithmetic size* of the diagram. The arithmetic size of a diagram is defined to be the the number of counters (dots) in the diagram. Note that dmap is a relation rather than a function, because there are several choice in mapping the same arithmetic expression to different diagrams. Here are some general mappings:

$$\mathsf{dmap}(n^2, [\mathsf{diagram}(\mathsf{square}, [n])]) \quad (8)$$

$$\mathsf{dmap}(2n-1, [\mathsf{diagram}(\mathsf{ell}, [n])]) \quad (9)$$

$$\mathsf{dmap}(\sum_{j=a}^{b} f(j), \biguplus_{j=a}^{b} \mathsf{D}_j) \text{ such that } \forall j, a \leq j \leq b,\ \mathsf{dmap}(f(j), [\mathsf{D}_j]) \quad (10)$$

Let us denote the arithmetic size of the diagram D with $|\,\mathsf{D}\,|$. The following holds:

Theorem 1 (Arithmetic size of a diagram) *The arithmetic size of the diagram 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: object list $\to$ pnat. The proof of Theorem 1 is carried out straightforwardly by induction on the structure of the relation dmap. Consequently, we have the following:

$$|\,[\mathsf{diagram}(\mathsf{square}, [n])]\,| = n^2 \quad (11)$$

$$|\,[\mathsf{diagram}(\mathsf{ell}, [n])]\,| = 2n-1 \quad (12)$$

$$\left| \biguplus_{j=a}^{b} \mathsf{D}_j \right| = \sum_{j=a}^{b} |\,[\mathsf{D}_j]\,| \quad (13)$$

Now, we state a lemma about the preservation of the arithmetic size of the sum of all existent diagrams when an operation is applied on a diagram. For all operations that were just introduced, the following holds:

Lemma 1 (Arithmetic Size Invariance Under One Operation) *The arithmetic size of the sum of the resulting diagrams when an operation is applied to some diagrams is the same as the arithmetic size of the diagrams on which the operation was applied. Let* D *be some diagrams such that* $\mathsf{dmap}(e, \mathsf{D})$ *then:*

$$|\,\mathsf{op}(\mathsf{opname}, \mathsf{D})\,| = |\,\mathsf{D}\,|.$$

The proof of this lemma consists of a case analysis of operations and mappings of arithmetic expressions. We shall not give it here. The immediate consequence of Lemma 1 is the preservation of size when a list of operations is applied to some diagram.

Lemma 2 (Arithmetic Size Invariance Under Multiple Operations) *The size of the sum of the resulting diagrams when a list of operations is applied to some diagrams is the same as the size of the diagrams on which the operations were applied. Let* D *be some diagrams such that* $\mathsf{dmap}(e, \mathsf{D})$ *then:*

$$|\,\mathsf{apply}(\mathsf{ops}, \mathsf{D})\,| = |\,\mathsf{D}\,|.$$

The proof of this lemma is by a straightforward induction on the structure of ops. We shall not give it here.

Equations

Here we give some axioms (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 $\to$ boolean):

$$0 \in s \to \mathsf{diagram}(x, s) \stackrel{d}{=} \emptyset \quad (14)$$

$$\emptyset :: \mathsf{D} \stackrel{d}{=} \mathsf{D} \quad (15)$$

Here is theorem (16) which is provable from (6) and (7).

$$\mathsf{apply}(\mathsf{ops}, \mathsf{D} :: \mathsf{D}_s) \stackrel{d}{=} \mathsf{apply}(\mathsf{ops}, [\mathsf{D}])@\mathsf{D}_s \quad (16)$$

Finally, we have the machinery for stating a desirable property about the correctness of a schematic proof formally:

Definition 1 (Correctness of Schematic Proofs) *For all schematic proofs of a particular theorem* $L(n) = R(n)$ *for which the following is given:*

1. *there is a mapping of the arithmetic expressions* $L(n)$ *and* $R(n)$ *into a diagram representation:* $\mathsf{dmap}(L(n), \mathsf{D})$ *and* $\mathsf{dmap}(R(n), \mathsf{E})$,

Base case: n=1

$$
\begin{array}{rcl}
\mathsf{apply}(\mathsf{proof}(1), [\mathsf{diagram}(\mathsf{square}, [1])]) & \stackrel{d}{=} & [\mathsf{diagram}(\mathsf{ell}, [1])] \\
\mathsf{proof}(1) = [(\mathsf{lcut}, 1)] & \Downarrow & \\
\mathsf{apply}([(\mathsf{lcut}, 1)], [\mathsf{diagram}(\mathsf{square}, [1])]) & \stackrel{d}{=} & [\mathsf{diagram}(\mathsf{ell}, [1])] \\
(7) \text{ and } (6) & \Downarrow & \\
\mathsf{one\_apply}(1, \mathsf{lcut}, [\mathsf{diagram}(\mathsf{square}, [1])])) & \stackrel{d}{=} & [\mathsf{diagram}(\mathsf{ell}, [1])] \\
(5) \text{ and } (4) & \Downarrow & \\
\mathsf{op}(\mathsf{lcut}, [\mathsf{diagram}(\mathsf{square}, [1])]) & \stackrel{d}{=} & [\mathsf{diagram}(\mathsf{ell}, [1])] \\
(3) & \Downarrow & \\
[\mathsf{diagram}(\mathsf{square}, [0]), \mathsf{diagram}(\mathsf{ell}, [1])] & \stackrel{d}{=} & [\mathsf{diagram}(\mathsf{ell}, [1])] \\
(14) & \Downarrow & \\
[\emptyset, \mathsf{diagram}(\mathsf{ell}, [1])] & \stackrel{d}{=} & [\mathsf{diagram}(\mathsf{ell}, [1])] \\
(15) & \Downarrow & \\
[\mathsf{diagram}(\mathsf{ell}, [1])] & \stackrel{d}{=} & [\mathsf{diagram}(\mathsf{ell}, [1])]
\end{array}
$$

Figure 1: Base case of the proof of correctness of a schematic proof.

2. all the operations in the schematic proof are defined, then,

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

The property in Definition 1 is impossible to prove for the general case for any theorem, unless very strict conditions are imposed. The problem is in the mapping of the expression stating the theorem into its diagrammatic representation, *i.e.* $L(n)$ and $R(n)$. In the general case it is not known what $L(n)$ and $R(n)$ are, so it is not possible to map them into diagrammatic representations. It is possible, however, to prove the property in Definition 1 for a particular theorem at hand.

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 $n^2 = \sum_{i=1}^{n}(2i-1)$. The schematic proof of this theorem is given as:[4]

$$\mathsf{proof}(1) \ \stackrel{d}{=}\ [(\mathsf{lcut}, 1)] \tag{17}$$

$$\mathsf{proof}(n+1) \ \stackrel{d}{=}\ [(\mathsf{lcut}, 1)],\ \mathsf{proof}(n) \tag{18}$$

Notice that we can use (8), (10) and (9) to map the sentential theorem into its diagrammatic representation: $\mathsf{dmap}(n^2,\ [\mathsf{diagram}\ (\mathsf{square}, [n])])$ and also $\mathsf{dmap}(\sum_{i=1}^{n}(2i-1),\ \biguplus_{i=1}^{n} \mathsf{diagram}(\mathsf{ell}, [i]))$. Thus the first condition of the definition for correctness is satisfied. The operations of a schematic proofs are also defined, so the second condition is satisfied as well. The proof of correctness of a schematic proof for this particular example is a meta-level proof which requires induction on n. Figure 1 shows the base case of the proof of correctness. Figure 2 shows the step case of this inductive proof. ∎

[4] For the brevity of presentation we take a simpler version of the schematic proof which does not include the operation split_ends. This version of the proof assumes that it has been previously proved that any ell is of odd size.

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. DIAMOND passes an induced schematic proof to *Clam*, the inductive proof planner developed in Edinburgh (see (Bundy *et al.* 1991)), where the theory is implemented and the correctness proof is carried out.

Thus far, the interactive construction of proofs, automatic abstraction from example proofs, and automatic verification of some schematic proofs have been implemented in DIAMOND. We will extend the communication between DIAMOND and *Clam* so that other schematic proofs that DIAMOND extracts can be passed to *Clam* and be verified in the theory of diagrams. To date, we can prove about fifteen theorems of significant depth and range (see (Jamnik, Bundy, & Green 1998)). The correctness proof for most of these theorems can be carried out in *Clam*. We are extending the theory of diagrams, and working on more examples.

Some interesting properties of this theory which could be investigated include algebraic correctness of all schematic proofs, incompleteness, and characterisation of the class of theorems that we can prove in this theory.

An alternative method to proving correctness of schematic proofs is to carry out a meta-inductive proof

Step case:

Hypothesis: for n

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

Conclusion: for $n+1$

$$\begin{array}{rcl}
\mathsf{apply}(\mathsf{proof}(n+1), [\mathsf{diagram}(\mathsf{square}, [n+1])]) & \stackrel{d}{=} & \biguplus_{i=1}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i]) \\
\mathsf{proof}(n+1) = [(\mathsf{lcut}, 1)],\ \mathsf{proof}(n) & \Downarrow & \\
\mathsf{apply}(((\mathsf{lcut}, 1), \mathsf{proof}(n)), [\mathsf{diagram}(\mathsf{square}, [n+1])]) & \stackrel{d}{=} & \biguplus_{i=1}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i]) \\
(7) & \Downarrow & \\
\mathsf{apply}(\mathsf{proof}(n), \mathsf{one_apply}(1, \mathsf{lcut}, [\mathsf{diagram}(\mathsf{square}, [n+1])])) & \stackrel{d}{=} & \biguplus_{i=1}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i]) \\
(5) \text{ and } (4) & \Downarrow & \\
\mathsf{apply}(\mathsf{proof}(n), \mathsf{op}(\mathsf{lcut}, [\mathsf{diagram}(\mathsf{square}, [n+1])])) & \stackrel{d}{=} & \biguplus_{i=1}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i]) \\
(3) & \Downarrow & \\
\mathsf{apply}(\mathsf{proof}(n), [\mathsf{diagram}(\mathsf{square}, [n]), \mathsf{diagram}(\mathsf{ell}, [n+1])]) & \stackrel{d}{=} & \biguplus_{i=1}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i]) \\
(16) & \Downarrow & \\
\mathsf{apply}(\mathsf{proof}(n), [\mathsf{diagram}(\mathsf{square}, [n])]) @ [\mathsf{diagram}(\mathsf{ell}, [n+1])] & \stackrel{d}{=} & \biguplus_{i=1}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i]) \\
(\text{RHS of hypothesis}) & \Downarrow & \\
\biguplus_{i=1}^{n} \mathsf{diagram}(\mathsf{ell}, [i]) @ [\mathsf{diagram}(\mathsf{ell}, [n+1])] & \stackrel{d}{=} & \biguplus_{i=1}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i]) \\
(2) & \Downarrow & \\
\biguplus_{i=1}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i]) & \stackrel{d}{=} & \biguplus_{i=1}^{n+1} \mathsf{diagram}(\mathsf{ell}, [i])
\end{array}$$

Figure 2: Step case of the proof of correctness of a schematic proof.

on diagrams. This necessitates reasoning with general diagrams which use abstractions to represent generality. Formalising abstractions (*e.g.* ellipsis) in diagrams to use them in meta-inductive proofs could be an interesting issue to consider.

DIAMOND is an interactive proof checker. A long term goal is to design an automated theorem prover capable of discovering diagrammatic proofs. For each new schematic proof that such a theorem prover would discover, the theory of diagrams will need to be extended automatically to incorporate and be able to check the correctness of the new schematic proof.

Related Work

Several diagrammatic systems such as the Geometry Machine by (Gelernter 1963), Diagram Configuration model by (Koedinger & Anderson 1990), GROVER by (Barker-Plummer & Bailin 1992), and Hyperproof by (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.

Other projects on diagrammatic reasoning which are of interest are by (Furnas 1992), by (Anderson & McCartney 1996), and by (Lindsay 1998).

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

Conclusion

In this paper we presented a theory of diagrams which enables us to check the correctness of diagrammatic schematic proofs. This constitutes the last stage of the procedure for extraction of diagrammatic proofs as presented in our previous work (see (Jamnik, Bundy, & Green 1997)). A schematic proof is correct if it proves all cases (*i.e.* for all n) of the proposition. It consists of a sequence of operations applied some function of n times. In the theory we defined the notion of applicability of a schematic proof and defined the correctness property of schematic proofs. We finally proved the correctness property for an example of a schematic proof of a theorem.

One of the aims of our research is to see whether it is possible to automate the use of diagrams in formal proofs. 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.

The first step is to prove interactively concrete examples of a theorem. Second, the system automatically abstracts these instances to give a general schematic proof which we hope holds for all n. Finally, in DIAMOND we have a logical machinery (a theory of diagrams, constructive ω-rule) to subsequently justify that the schematic proof proves the original theorem.

Acknowledgements

The research reported in this paper was supported by an Artificial Intelligence Department 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 and the computing facilities for the first author.

References

Anderson, M., and McCartney, R. 1996. Diagrammatic reasoning and cases. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1004–1009. AAAI Press.

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, 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 Narayanan, N. H., ed., *Working Notes of the AAAI Spring Symposium on Reasoning with Diagrammatic Representations.* AAAI Press.

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.

Bundy, A.; van Harmelen, F.; Hesketh, J.; and Smaill, A. 1991. Experiments with proof plans for induction. *Journal of Automated Reasoning* 7:303–324. Earlier version available from Edinburgh as DAI Research Paper No 413.

Furnas, G. W. 1992. Reasoning with diagrams only. In Narayanan, N. H., ed., *AAAI Spring Symposium on Reasoning with Diagrammatic Representations: Working Notes*, 118–123. American Association for Artificial Intelligence.

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 reasoning. In Pollack, M., ed., *Proceedings of the 15th IJCAI*, volume 1, 528–533. International Joint Conference on Artificial Intelligence. Also published in the Proceedings of the 1997 AAAI Fall Symposium. Also available from Edinburgh as DAI Research Paper No. 873.

Jamnik, M.; Bundy, A.; and Green, I. 1998. On automating diagrammatic proofs of arithmetic arguments. Research Paper 910, Dept. of Artificial Intelligence, University of Edinburgh. To appear in *Journal of Logic, Language and Information*.

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.

Lindsay, R. 1998. Using diagrams to understand geometry. *Computational Intelligence* 14(2).

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