Available online at www.sciencedirect.com

ScienceDirect

Artificial Intelligence 171 (2007) 197–238

Artificial Intelligence

www.elsevier.com/locate/artint

Conservation principles and action schemes in the synthesis of geometric concepts

Luis A. Pineda

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Ciudad Universitaria, Coyoacán, México, D. F., México, 04510, Mexico

Received 17 February 2006; received in revised form 5 December 2006; accepted 15 December 2006

Available online 22 December 2006

Abstract

In this paper a theory for the synthesis of geometric concepts is presented. The theory is focused on a constructive process that synthesizes a function in the geometric domain representing a geometric concept. Geometric theorems are instances of this kind of concepts. The theory involves four main conceptual components: conservation principles, action schemes, descriptions of geometric abstractions and reinterpretations of diagrams emerging during the generative process. A notion of diagrammatic derivation in which the external representation and its interpretation are synthesized in tandem is also introduced in this paper. The theory is exemplified with a diagrammatic proof of the Theorem of Pythagoras. The theory also illustrates how the arithmetic interpretation of this theorem is produced in tandem with its diagrammatic derivation under an appropriate representational mapping. A second case study in which an arithmetic theorem is synthesized from an underlying geometric concept is also included. An interactive prototype program in which the inference load is shared between the system and the human user is also presented. The paper is concluded with a reflection on the expressive power of diagrams, their effectiveness in representation and inference, and the relation between synthetic and analytic knowledge in the realization of theorems and their proofs.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Diagrammatic reasoning; Diagrammatic theorem-proving; Knowledge representation; Geometric description; Geometric abstraction; Conservation principles; Action schemes; Structured learning; Synthetic concepts

1. Conservation principles and action schemes

Diagrams have been used as representation aids in reasoning and theorem-proving since ancient times; the Pythagoreans liked to illustrate things with pictures, and Euclid's proofs made extensive use of diagrams [21]. In its original formulation, Euclid's method had two main features: on the one hand, every inference step was truth preserving and sustained on secure knowledge, and proofs were valid. On the other, a geometric concept was revealed in the mind of the one who made or followed the proof; that is, the proof also rendered the geometric concept of what the theorem was about. This knowledge can be thought of as the meaning of the representation expressing the theorem, and could be used operationally for problem-solving. Also, although in Euclid's method a proof was presented as a sequence of propositions, it was developed in conjunction with the construction or inspection of a diagram, and very often the propositions only made sense in relation to the diagram.

E-mail address: luis@leibniz.iimas.unam.mx.

0004-3702/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2006.12.003

Euclid's method of proof stood solidly for more than two millennium; however, the formalization of mathematics that followed Hilbert's program at the beginning of the twentieth century changed the conception of the axiomatic method in an essential way: although a theorem was still derived out of the axioms by the application of valid inference steps, the construction of the concept associated with the theorem was no longer an essential part of the process. While in the proof-theoretic sense a proof is rendered as a syntactic object consisting of a set of sentences, the concept expressed by such sentences can be best thought of as semantic object, and it was the construction of this latter object that became a contingent process. Furthermore, concepts may involve intuitions that cannot be easily formalized, and this aspect of the process was better left out, as the main motivation for the modern proof-theoretic method was to secure truth. Also, as axioms and theorems were expressed as linguistic propositions, diagrams were no longer required.

However, despite the tremendous success of the proof-theoretic method, and the logical interest in the validity of diagrammatic proofs and heterogeneous reasoning (e.g. [4,6]), the construction of the concept rendered by diagrammatic inference is still relevant for the philosophy and psychology of reasoning, and for knowledge representation and learning in AI, among many other disciplines. These areas of AI are concerned with the acquisition, construction and use of concepts, and the study of the processes by which geometric and arithmetic knowledge is synthesized presents a very interesting and challenging problem. Also, the role of diagrams in inference is an important concern to all of these disciplines; furthermore, as diagrams are ubiquitous in computer science and technology, understanding the semantic aspect of diagrammatic reasoning and problem-solving has a theoretical and practical value.

In the present paper a theory of the synthesis of some geometric and arithmetic concepts in which diagrams play an essential role is presented; the theory is based on a referential machinery that permits the construction of complex geometric descriptions, and on a novel inferential scheme, and we hope that these mechanisms, in conjunction with their associated theory, will increase our understanding of the reasoning and learning processes involved in diagrammatic proofs, and more generally, in the processes underlying diagrammatic interpretation and inference.

In this paper, geometric and arithmetic concepts are represented directly by functions expressed through the lambda calculus; although this is in fact a representational choice, implemented in the prototype system presented below, the functions can also be thought of as the specification of the agent's knowledge, independently of implementation considerations. In this view, learning consists in synthesizing a new function out of the functions already available and the interaction of the agent with the world. This representational format has also the advantage that the computational agent can 'know' the extension of a concept by simply evaluating the function representing such concept for a given argument.

In the present theory the synthesis of geometric concepts is based on two main knowledge objects which we refer to as conservation principles and action schemes. The theory is focused on a constructive process that acts both on the external representation in which a sequence of diagrams is produced, and at the interpretation level, where the process results in a function in the geometric domain representing a geometric concept. Geometric theorems are instances of this kind of concepts. Conservation principles are concepts of a very abstract character that permit an agent to know whether a property of an object is preserved after a process of change, if the change itself is produced by an action scheme that preserves such property. Action schemes, in turn, are rules of a more concrete character that specify actions that can be performed by an agent in relation to a focus of attention and a geometric context. In the present theory conservation principles are represented through higher-order functions; the form of these functions is illustrated with the principle of conservation of area, as follows:

$$\lambda P \lambda Q \lambda x.area(P(x)) = area(Q(x)) \tag{1}$$

In (1), λ is the functional abstractor, P and Q are higher-order variables, and x is a first order variable. In the statement of the principle, P stands for a geometric description of a set of objects and Q stands for the description of the same set of objects after a process of change has taken place; as will be shown below, P and Q are functional objects that map geometric objects into geometric objects, and $P(x)$ and $Q(x)$ denote a geometric object or configuration before and after a change, if the change itself is produced by an action scheme that preserves the property associated with the conservation principle, in this case, the area. The argument x is a focus object that remains fixed during the change, and the principle as a whole asserts that the area of $P(x)$ is the same as the area of $Q(x)$ if the change is produced in relation to the invariant reference x by an action scheme that is area preserving. An area preserving action scheme is illustrated in Fig. 1.

This action scheme rotates a right triangle by $3/2\pi$ in a clockwise direction, in a context in which there is a second right triangle with the same dimensions exactly, and the effect of the scheme is to align the two triangles by

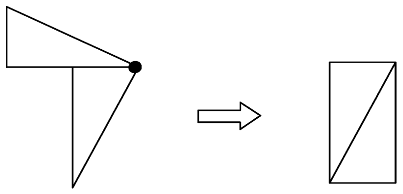
Fig. 1. First action scheme.

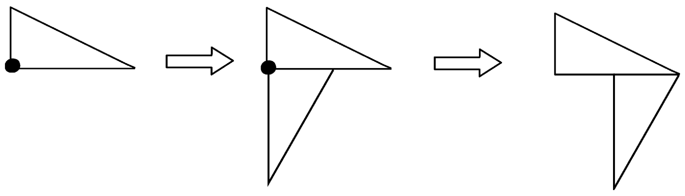
Fig. 2. Second action scheme.

their hypotenuses, as shown in the figure. Action schemes are also defined relative to a focus object that remains fixed during the change; in addition, action schemes may have a "pivot" object which is a function of the focus, and provides an additional parameter for the articulation of the change. In the first scheme, the focus is the fixed right triangle, and the pivot is the vertex aligned with a vertex of the triangle that is rotated by the scheme, as indicated by the bold dot on the corresponding vertex; action schemes are generic and the objects involved can have any position, dimension, and orientation, as long as the relation between the objects on the left hand side of the scheme holds. In this first scheme, the geometric configurations before and after the change are built up with the same "tiles", do not overlap, and have the same area; accordingly, this action scheme is area preserving and its application grants the application of the conservation principle in (1). In the present theory, the application of an action scheme preserving a property and its corresponding conservation principle are coordinated actions. The application of the action scheme produces an action on an external representation, while the application of the conservation principle is a semantic operation, and both of these actions are defined relative to the focus object, which is the same in both planes of expression.

An action scheme that is not area preserving, on the other hand, is illustrated in Fig. 2. This second action scheme introduces a new instance of the focus right triangle (i.e. an exact copy of the focus, with the same dimensions, position and orientation), and involves the rotation of the new triangle in relation to the vertex at the right angle of the focus triangle, and also its translation along the right side of the focus until their vertices are aligned, as shown in Fig. 2.

The action specified by an action scheme is a unit act performed by an 'encapsulated' process; action schemes may be defined in terms of a single operation, as the first one, or may have an internal structure and involve several operations, as the second, but the action itself involves the execution of all its operations in the order specified by the scheme exactly. In this sense, action schemes are thought of as 'holistic' processes whose internal structure is given and cannot be analyzed or used in problem-solving.

An action scheme may preserve none, one, or many properties; however, regardless of their associated conservation principles, all action schemes are defined in relation to a focus object or configuration that remains fixed during the change. Also, conservation principles and action schemes are related systematically, and every conservation principle is associated to all action schemes that preserve the principle's conservation property and vice versa.

In the present theory, a geometric reasoning agent has at its disposal a number of conservation principles and action schemes at any given time, and also the knowledge of the relation between them. This constitutes the primitive knowledge upon which geometric concepts can be synthesized or learned by the agent. The synthetic process aims to establish that a property of an object holds in different states of a construction process; if the property holds for a specific or concrete construction, the result of the process is a concrete relation; however, if the property holds for an abstract class of objects or configurations, then the concept resulting from the construction process is a general relation, or a geometric theorem. In particular, action schemes that preserve a geometric property and permit the application of the corresponding conservation principle play the role of valid inference schemes in the construction of synthetic theorems, in the same way that valid inference schemes of logical derivations underlie a principle of conservation of truth.

In the present theory the synthesis of geometric concepts depends on a process involving the systematic interaction of conservation principles and action schemes. The successful interaction between these two kinds of knowledge renders a diagram or a diagrammatic sequence on the external representation, and a function in the geometric domain representing the concept expressed by the diagram. The purpose of this theory is to model the inference through which such objects are synthesized or discovered.

In the rest of this paper the theory is presented in detail. In Section 2, the geometric language for representing diagrams and their interpretations is illustrated. This language is expressive enough to represent concrete and abstract interpretations of diagrams, descriptions of diagrams and its parts, and also higher-order functions representing conservation principles. The functions resulting from the synthetic process can also be expressed in this language, and evaluated by the language's interpreter. In Section 3 a diagrammatic proof of the Theorem of Pythagoras is presented; this proof is the main case study in the present paper. In this section the descriptions of overt or emergent diagrammatic objects appearing in the diagrammatic proof are also illustrated. In Section 4 the notion of diagrammatic derivation, through which the function representing the theorem is synthesized in tandem with the synthesis of the diagrammatic proof (i.e. the external representation) is presented. The relation between geometric concepts and their corresponding arithmetic interpretations is discussed in Section 5. In Section 5.1 the diagrammatic derivation is augmented with an additional tandem process in which the geometric and arithmetic concepts are synthesized hand in hand. In Section 5.2 a second case study is presented; this is a diagrammatic proof of the theorem of the sum of the odds. It is shown that conservation principles and action schemes are not only effective for the synthesis of geometric concepts, but also for the synthesis of arithmetic functions representing arithmetic theorems. In Section 6 the production of descriptions from external representations through perceptual inferences is discussed. Section 7 is devoted to the description of the prototype program that was developed with the purpose to test the theory; the two case studies presented in the paper have been fully tested with this program.

The present theory is relevant to and has antecedents from diverse disciplines such as AI, cognitive science, programming languages, semantics, psychology and design, and Section 8 is devoted to survey the related work. In this section the AI and cognitive science antecedents of the two case studies included in the paper are also briefly discussed. In Section 9 a brief summary of the theory is presented, and the limitations and further work are discussed. The paper concludes with a discussion, in the light of the present theory, on the expressive power of diagrams, their effectiveness for representation and inference, and the nature of synthetic and analytic knowledge. The paper is complemented by five appendices, as follows: the formalization of the geometric representation language and its interpreter are presented in Appendices A, B and C; the arithmetic interpreter in which the arithmetic expressions of the Theorem of Pythagoras and the theorem of the sum of the odds are expressed and evaluated is included in Appendix D, and the actual formal expressions of the most relevant functions and descriptions used in the paper are included in Appendix E.

2. Geometric description and abstraction

Diagrams on a piece of paper or a computer screen are concrete representational objects (i.e. external marks on a medium) that can have concrete or abstract interpretations; in the current discussion an expression is said to have a concrete or an abstract interpretation depending on the nature of its referent. In an ontological sense, geometric objects belong to the geometric domain and as such, they are abstract objects; from this perspective, numbers and concepts, for instance, are abstract objects, but cars and people in the world are concrete individuals. In a related sense, but from a psychological perspective, experience directed towards the external world, with specific stimuli and actions, is concrete, but experience directed to internal representations or conceptualizations is abstract; also, experience directed to proximate objects in space and time is concrete, but thinking of distant objects in these two dimensions involves abstractions [35]. From this psychological perspective, and according to Piaget, concrete thinking is directed towards individuals, properties and relations of immediate experience, while abstraction is the ability to think about hypothetical entities [36].

From a linguistic perspective, on the other hand, an expression has a concrete interpretation if the expression refers to a particular entity, property or relation, and an abstract interpretation if the reference is a class of these objects. This is the sense used in knowledge representation (e.g. conceptual hierarchies) in which a concept is more abstract or general depending on how many classes of individuals it has within its extension. It is in this latter linguistic and representational sense in which formulae of the propositional logic refer to specific objects and relations (i.e. concrete), while first order logic formulae or the lambda calculus permit expressing abstractions through variables, quantifiers, and the functional abstractor, regardless of how "concrete" or "abstract" the interpretation domain happens to be (e.g. whether the interpretation domain is populated by people or cars, or numbers or geometric objects).

The psychological and semantic views on the relation between the concrete and the abstract are usually thought of as consistent, as the objects at the bottom of the conceptual hierarchy correspond often to the concrete objects out there in the world, and the higher the concept is in this hierarchy the lesser the degree of concreteness of its extension [35]. However, this symmetry is broken down with geometric and arithmetic objects, that are not "out there" in the external world; the number one, for instance, is an abstract object, but an expression that refers to this object has a concrete interpretation, because it refers to this specific object and not to another. In the present discussion, ontological questions about the nature of arithmetic and geometric objects are factored out by adopting the semantic view and it is taken that expressions, whether these are linguistic or diagrammatic, express concrete or abstract concepts, depending on whether their intended interpretation is a specific entity, property or relation, or a class of these objects.

A description that refers to a specific dot in a specific position in the space has a concrete interpretation, which is that particular dot, but an expression representing a function that has a dot and a line as its arguments, its value being true or false depending on whether the dot is on the line, for all dots and all lines in the space, expresses an abstraction. The interpretation of an expression may have concrete and abstract elements; for instance, if a function has a line as its argument, and its value is true or false depending on whether or not a specific dot is on the argument line, then it is an abstraction only in part, as the specific dot is a concrete object. Here, all expressions (i.e. functions) involving reference to no specific or concrete geometric object are referred to as full abstractions. In particular, a function representing a theorem needs to be a full abstraction. In a similar way, although diagrams often have concrete interpretations (i.e. when they represent specific geometric objects) they can have abstract interpretations too; that is, a diagram can stand for a class of geometric objects or relations, under the intended interpretation conditions, and represent a full abstraction. It is in this latter sense that a sequence of diagrams can be the proof of a geometric theorem, as is argued below in this paper. Next, a geometric representation language is presented; this language is expressive enough to represent concrete and abstract geometric objects and configurations, and also conservation principles and action schemes. The language is based on Pineda [39], which in turn is based on Goguen et al. [18], and permits the specification of geometric abstract data-types; several versions of this formalization are available [15,43], and the current version is presented in the appendices of the present paper. The syntax is a term grammar, and the full signature of the geometric representation language is presented in Appendix A; the semantics is specified directly in the interpreter's program, and the actual Prolog code of the interpreter is given in full below in Appendix B.

The language is illustrated with descriptions of concrete geometric objects and configurations, and also with descriptions of classes of geometric objects and relations. The notation reflects the Prolog implementation directly with minor stylistic conventions. The description of a concrete right triangle is shown in (2), where d_i is a constant of sort dot and rt_i a constant of sort right triangle, t stands for a boolean value that denotes whether the term has a well-defined interpretation in the evaluation state, as will be explained below, and $x_i{:}y_i$ stands for a specific or concrete coordinate position:

$$\begin{aligned} &d_1 = dot(t, x_1{:}y_1) \\ &d_2 = dot(t, x_2{:}y_2) \\ &d_3 = dot(t, x_3{:}y_3) \\ &rt_1 = right\_triang(t, d_1, d_2, d_3) \end{aligned} \tag{2}$$

The equalities in (2) are definitions associating constant symbols of sort *dot* with the specific dots, and a constant symbol of sort *right_triangle* with the description of the specific right triangle. In the intended interpretation the constant name and the description denote the same concrete object in the geometric domain. The evaluation of a geometric term produces the extensional definition of a geometric object, the term's denotation, in the evaluation state. A diagram is represented through a list of terms of this kind.

The first boolean argument of a term is true (i.e. t) if all its arguments have a well-defined value in the evaluation state and false (i.e. f) otherwise, for all terms. In the same way that division by zero is not defined and should be prevented in arithmetic operations, geometric arguments must be well-defined in a geometric construction operation. For instance, the interpretation of $intersect(t, l_1, l_2)$, where *intersect* is a dot constructor operator (i.e. constructs the dot at the intersection between the argument lines), produces the term $dot(f, l_1, l_2)$ if the lines in question are parallel.

If a geometric argument of a term is not well-defined (i.e. the value of its first boolean argument is false), the term itself neither is well-defined. For legibility, this first boolean argument of geometric and logical terms is omitted in the expressions below.

The definition of the language specifies a set of geometric sorts including *dot*, *line*, *right_triangle*, *square* and *polygon* among others; the non-graphical sorts *boolean*, *real* and *real_pair* are also defined. A set of constant symbols of every sort and a set of constructor and selector operators for constructing terms of every sort, and for selecting their properties and relations, are also provided. In particular, type and equality predicates are defined for all sorts. Geometric terms are interpreted as functions from the term's arity to the term's sort and denote the corresponding objects in the geometric domain. For geometric interpretation, there is a geometric algorithm associated with each geometric operator which computes the named geometric property or relation, or constructs the geometric object in question. The Prolog code of an instance of a geometric data-type implementing this algorithmic interpretation is given in Appendix C below.

Basic predicates permit testing whether a geometric context holds. For instance, in Fig. 1, the right side configuration of the first action scheme is satisfied by the predicate in (3) which is true if two concrete right triangles are aligned by their hypotenuses exactly, and their areas do not overlap.

$$\mathit{aligned\_HH}(rt_1, rt_2) \tag{3}$$

Similar predicates for testing other complex geometric conditions, and functions for generating geometric objects, are also available. This basic machinery permits us to describe concrete geometric objects and test whether they have particular properties or stand in a given relation.

Next, the representational devices for expressing geometric abstractions are presented. For this purpose symbols for variables of all sorts and the functional abstractor are introduced in the representation language. The form of geometric functions is as follows:

$$\lambda(L, G) \tag{4}$$

where L is a list of variables and G is a geometric expression including, possibly, the variables in L, where these variables are free in G. The body of a function can also be a function permitting the expression of complex geometric conditions. It is now possible to represent, for instance, the class of all pairs of right triangles aligned by their hypotenuses, as follows:

$$\lambda([x, y], \mathit{aligned\_HH}(x, y)) \tag{5}$$

Expression (5) is a full abstraction. In functional application, arguments are first bound, and then the body is evaluated. If rt_1 and rt_2 are the concrete right triangles at the right side of the first action scheme, the application of (5) to these arguments results in the truth value of true.

Functions permit us to express the action part of action schemes too; for instance, the triangle generated in the second action scheme in Fig. 2 is produced by function (6) in terms of a focus or reference right triangle by its rotation through an angle θ about its own right vertex, and its translation by $\delta x{:}\delta y$:

$$\lambda([\mathit{focus}, \theta, \delta x{:}\delta y], \mathit{translate}(\mathit{rotate}(\mathit{focus}, \theta, \mathit{right\_vertex}(\mathit{focus})), \delta x{:}\delta y)) \tag{6}$$

The expressive power presented up to this point allows us to describe objects of the basic geometric sorts in a context-independent fashion, and also to describe classes of these objects, as well as classes of geometric properties and relations; however, in diagrammatic reasoning it is also required to be able to refer to objects and configurations relative to a geometric context, or to geometric objects that emerge from the composition of other shapes. For this purpose the geometric description operator "<=" is introduced as follows:

$$T <= f \tag{7}$$

where T is a term of any geometric sort and f is a boolean geometric function, and the value of the geometric description is the value of the term T itself if the application of f is true. The function f stands for an arbitrary geometric concept, and the intuition is that if a geometric object satisfies such a concept, a geometric description of the form $T <= f$ refers to the denotation of T in the context of f; in case the boolean function is false for the given arguments, the description has no denotation and the interpreter returns the truth value of false as an implementation convention. In the interpretation of descriptions the variables in T are bound to the variables in f. For instance, the description in (8) refers to an arbitrary triangle x that is aligned through its hypotenuse with another arbitrary but different triangle y (as the geometric interpretation of the geometric predicate includes the specification that the triangles do not overlap) as follows:

$$x <= \lambda([x, y], \mathit{aligned\_HH}(x, y)) \tag{8}$$

Descriptions are applied to arguments, as follows:

$$(x <= \lambda([x, y], \mathit{aligned\_HH}(x, y)))([rt_1, rt_2]) = rt_1 \tag{9}$$

The geometric description operator "<=" is a constructor of identity functions that resemble the diverse identity functions of Recursive Functions theory [7]. Intuitively, a geometric context characterized by a function (e.g. as in (5)) can be thought of as an abstract composite object in a multidimensional space with one dimension for each of the objects involved in the definition of the context, and each of these objects can be thought of as the projection of the composite object in the corresponding dimension. For instance, the description in (8) is the first identity (or projection) of the context described by (5). That is, the context formed by two arbitrary triangles aligned through their hypotenuses is thought of as an abstract object in a bi-dimensional space, and (8) is the identity function in the first of these two dimensions; so, the application of (8) to the two concrete right triangles that are aligned through their hypotenuses renders the triangle that is bound to the first argument (i.e. x), as shown in (9). If the variable at the left side of the descriptor were y instead, the constructed function would be the second identity. In (7) f can be a saturated boolean term (i.e. a term with only constant symbols in its argument positions), and the descriptor can be used to refer to concrete geometric objects too. Also, as the left side of the descriptor is an arbitrary term of any geometric sort, the descriptor operator can be used to define emerging objects that are defined in terms of the projections of a given geometric context, as exemplified in (15) below, and in this sense, it can also be used as a constructor, in addition to its basic use as a selector. This expressive device will be essential to describe shapes that emerge in the course of diagrammatic sequences, as will be shown below.

The geometric descriptor also resembles Russell's ι-operator and Hilbert's ε-operator for the definition of descriptions of the form $\iota x.P$ and $\varepsilon x.P$ respectively, where P is an arbitrary predicate that may contain x; the intuitive interpretation of these operators is that a term of form $\varepsilon x.A$ selects any object for which A is true, and has an arbitrary reference otherwise, and the term $\iota x.A$ selects a unique object satisfying the predicate if there is such an object in the interpretation domain, and is undefined otherwise. The motivation of Russell's operator is to capture the meaning of definite descriptions and Hilbert's operator was introduced to replace quantifiers in ordinary predicate logic, but it can also be used to model the meaning of indefinites. The present geometric descriptor is also a device for representing the semantics of descriptions without the explicit use of quantifiers; however, unlike ε-terms and ι-terms that select an object from the interpretation domain, an expression formed with the geometric descriptor is interpreted as a function from individuals (i.e. the arguments of the boolean function in the description) to individuals; the description (i.e. the function itself) resembles an ε-term, and is used to represent indefinites, as will be shown below, but the description applied to its arguments provides a specific reference, loosely resembling the interpretation ι-terms in this respect. Also, as the term on the left side of the geometric descriptor refers to an arbitrary object that is defined from the arguments of a description, as in (15), allowing the description of objects that emerge in the geometric context, the denotation of a geometric description may not satisfy the boolean function on its right side, and the geometric operator differs in this respect from Russell's and Hilbert's operators, which do not have this "constructive" interpretation. Although the motivation of the geometric descriptor is to have a representational device to refer to geometric objects in context, the present descriptor could be used in other conceptual domains, and may have interesting applications in a more general theory of reference and description.

The presentation of the geometric representation language is concluded with the definition of the *apply* operator that allows the expression of functional application explicitly; the arguments of this operator are a function or a description, and its corresponding list of arguments. The use of this operator is illustrated with the expression of the principle of conservation of area in the geometric representation language, as follows:

$$\lambda([P], \lambda([Q], \lambda(X, \mathit{eq}(\mathit{area}(\mathit{apply}(P, X)), \mathit{area}(\mathit{apply}(Q, X)))))) \tag{10}$$

In the synthesis of geometric concepts P and Q will be instantiated with geometric descriptions and X is a list containing the focus; if these descriptions express full abstractions, the resulting equality will be a full abstraction too: a theorem.

The actual form of concept (10) in the geometric representation language is given in Appendix E; this appendix also includes the definition of the main concepts and descriptions used in the two case studies included in this paper, as well as the representation of arbitrary concrete geometric objects to launch the synthetic process and to test the resulting concepts, as will be explained below.

3. Emerging objects and reinterpretation

The synthesis of geometric concepts involves the recognition of shapes that emerge along the synthesis of geometric concepts. Consider the diagrammatic sequence of a proof of the Theorem of Pythagoras in Fig. 3 which is used as the main case study in the present paper. This very well known theorem states that the area of a square on the hypotenuse of a right triangle is equal to the addition of the areas of the squares on its legs (i.e. its right sides). This proof, presented by Bronowski [9], shows very clearly the Pythagorean concept. The diagrammatic sequence is generated from an initial triangle "seed"; from this, three consecutive applications of the second action scheme produce the top-right figure; in this diagram two squares "emerge" out of the triangle's configuration. In the present study, perceptual acts performed by the agent in order to recognize emerging patterns of this kind are referred to as *re*interpretations. The bottom-right diagram is generated from the top-right diagram by the application of the first action scheme twice; this scheme is area preserving, and the application of the principle of conservation of area is allowed. The theorem holds because the square on the hypotenuse emerges from the triangle's configuration in the top-right figure while the squares on the two right sides emerge, aligned side by side, in the bottom-right figure, and these two figures are built with the same tiles. The ability to visualize these emerging objects through a reinterpretation of the diagrams is essential for the realization of the Pythagorean concept, the theorem and the proof.

The perceptual reinterpretation task can be specified as follows: given a diagram produce a geometric description of the relevant emerging object. The generation of the description involves the recognition of the emerging entity, and this latter process can be modeled by an action scheme too; for instance, the inner square can be recognized through the action scheme illustrated in Fig. 4, where the focus is the four-triangle configuration. The context of this scheme matches the top-right diagram, and the square appears as a side effect of the synthetic process. The square on the hypotenuse can be realized by a similar scheme.

In the cycle of perception and action the production of the description may be followed by the reification of the emerging entity (i.e. making the emergent object an actual instance of the diagram). In the current example, the reification of the inner square extends the set of geometric objects that has been produced up to the current state, in

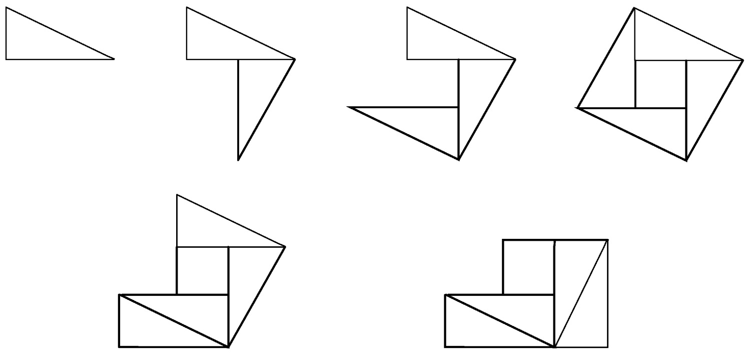
Fig. 3. Theorem of Pythagoras.

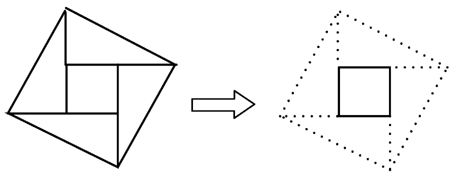
Fig. 4. Third action scheme.

the same way that each application of the second scheme extends the context with a new triangle; for this reason, in the present formulation, the reification of the inner square is included in the definition of the third action scheme. This new object can be described through the basic square constructor operator in terms of the right vertices of all four triangles, as shown in (11).

$$square(right\_vertex(rt_1), right\_vertex(rt_2), right\_vertex(rt_3), right\_vertex(rt_4)) \tag{11}$$

The three action schemes above generate or modify the geometric objects in a particular state of the generative process, and consequently their corresponding diagrammatic external representations; in this respect, the operations performed by these schemes resemble drawing actions; as drawing is a motor behavior, these schemes are thought of as motor action schemes. Action schemes which only produce a description of the recognized entity, on the other hand, are referred to as perceptual schemes. For instance, an action scheme that recognizes the square on the hypotenuse, without reifying this entity in the diagram, is a perceptual scheme.

Motor actions are directed to the domain of external immediate experience and the objects generated or modified by these schemes (i.e. the diagrams on a piece of paper) are also concrete in this respect. For this reason, extensional descriptions in the geometric representation language, and their corresponding external representations, the diagrams, are thought of as the same concrete geometric objects exactly. Perceptual schemes, on the other hand, may produce a concrete or an abstract description as their output, as will be argued extensively in Section 6.

For representing the external square emerging in the top-right diagram in Fig. 3 its intuitive linguistic description *a square on the hypotenuse of a right triangle* is translated into a geometric description; this natural language description has a *de re* (i.e. specific) and *de dicto* (i.e. generic) readings [12] and it is ambiguous between these two senses. In the *de re* sense the description, and also the diagram, has a concrete interpretation, and the intended referents are a specific square and a specific triangle in the geometric domain, and also in the picture; as there are four triangles, this reading under-specifies which one is meant in the description and, in the *de re* interpretation, there is a referential ambiguity in the phrase *a right triangle*; however, in the *de dicto* reading the objects mentioned are any square and triangle standing in the mentioned relation, and there is no referential ambiguity. It is also possible to give a *de re* interpretation to one of indefinites (e.g. *a square*), at the time a *de dicto* reading is given to the other (e.g. *a right triangle*), and in this latter case the expression expresses a partial abstraction. In the present task (i.e. finding a general geometric concept) the required interpretation is the *de dicto* reading for both of the indefinites; the translation of this reading into the geometric representation language is presented next; first the concept of a square on the hypotenuse of a right triangle is represented by the following function:

$$\lambda([x], \lambda([w], is\_right\_triang(x)\ and\ is\_square(w)\ and\ side(hypotenuse(x), w))) \tag{12}$$

In (12) *is_right_triang* and *is_square* are type predicates, *hypotenuse* is a selector for right triangles (i.e. selects the triangle's hypotenuse), and *side* is a boolean predicate that is true if its first argument (i.e. the line), is one side of its second argument (i.e. the square) and false otherwise. The argument of the function with the widest scope is the triangle referred to in the natural language description, and this argument captures the *de dicto* interpretation of the corresponding noun phrase. Let f_1 be the function in (12); then, the square on the hypotenuse is represented through a geometric description as follows:

$$w <= f_1 \tag{13}$$

In (13) the variable w on the left side of the descriptor is bound to the corresponding argument variable of the function at the right side. This description is a full abstraction as it depends on no concrete right triangle or square, and its application to particular instances of these objects will return the square if one of its sides coincides exactly with the hypotenuse of the right triangle, and will be undefined otherwise.

Most people immediately see the emerging squares in the top-right diagram in Fig. 3; however, identifying the squares on the legs of the right triangle in the lower right diagram is not so easy. These squares must be visualized to be reliably identified, as they are not marked in the diagram, and to be able to make this visualization is the essential aspect of the realization of the theorem and its proof. The visualization is shown explicitly in Fig. 5.

The objects resulting from the visualization can be described in the geometric representation language just as can any other explicit object. The concept of two squares on the legs of a right triangle is represented through a geometric function, as follows:

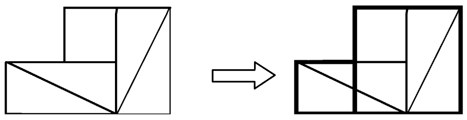

Fig. 5. The visualization.

$$\begin{aligned}&\lambda([x], \lambda([y,z], \mathit{is\_right\_triang}(x) \text{ and } \mathit{is\_square}(y) \text{ and } \mathit{is\_square}(z) \text{ and}\\&\qquad ((\mathit{side}(\mathit{side\_A}(x), y) \text{ and } \mathit{side}(\mathit{side\_B}(x), z)) \text{ or}\\&\qquad (\mathit{side}(\mathit{side\_B}(x), y) \text{ and } \mathit{side}(\mathit{side\_A}(x), z)))\\&))\end{aligned} \tag{14}$$

Let f_2 be the function in (14); then, the visualization is represented through the following geometric description:

$$\mathit{union}(y,z) <= f_2 \tag{15}$$

where *union* is an operator in the signature of the geometric language and its interpretation is a function of type *square* × *square* → *square_pair*. A square pair is well-defined if the squares forming the pair are well-defined and do not overlap. The application of f_2 to a right triangle and two squares in the present diagram is true, and (15) denotes the union of the two squares in the last diagram of the sequence. The actual forms of descriptions (13) and (15) in the geometric representation language, including their corresponding concepts in (12) and (14), are given in Appendix E of the present paper.

The description (15) and its associated function in (14) are defined relative to a focus triangle that has a *de dicto* interpretation. In the visualization in Fig. 5, all four concrete triangles are equivalent, and each instance can be thought of as the representation of a generic triangle, as the original "seed" upon which the whole construction is based is used in this generic sense. For this reason, the triangle argument of the function in (14) is the reference triangle for aligning both of the squares to each of its legs, despite the fact that in the diagram two different concrete triangles are aligned to the two squares. In addition, the whole of the concrete sequence is a generic diagram in which the concrete diagram stands for its class. The position, size and orientation of the triangles and squares can vary, but the relation between these objects holds in any possible concrete realization. Description (15) satisfies any diagram in which there is a right triangle and there are two squares on the sides containing the right angle, as in the diagram used in Euclid's proof, in Fig. 12 below; similarly, description (13) satisfies any diagram in which there is a square on the hypotenuse of a right triangle, and (13) satisfies Fig. 12 too. Generic descriptions capture not only similar diagrams that differ only in their concrete parameters, but also different geometric configurations satisfying the predicates in the description; furthermore, the same diagram can satisfy different descriptions, and the same description can be satisfied by different diagrams. However, different descriptions can only be related through the equality relation stated by a conservation principle if the diagrams satisfying the descriptions are causally related by action schemes preserving the corresponding conservation property.

4. Diagrammatic derivations

In this section the notion of diagrammatic derivation is introduced, and this notion is illustrated with the synthesis and proof process of the Theorem of Pythagoras. This process is modeled as a derivation at the external or "syntactic" level of representation, where a concrete diagram is constructed, and at the semantic level, where geometric knowledge gathered incrementally through the synthetic process is represented. At the external level the actual diagram sequence is generated and can be rendered directly; at the semantic level, on the other hand, the process synthesizes a function representing a geometric proposition. If this function is a full abstraction, the function represents a theorem. In our case study, the output of the synthetic process is a function of four arguments: a right triangle and three squares, and this function is true if its arguments satisfy the Pythagorean relation and false otherwise. The function is expressed in the geometric representation language, and can be applied and evaluated by the geometric interpreter.

The process starts from the seed right triangle in the diagram, and the principle of conservation of area at the semantic level of representation; in the present theory, the knowledge of conservation principles is given in advance to the reasoning agent. Action schemes generate new diagrams or diagrammatic states, and if they have an associated

Table 1
Diagrammatic derivation

DS	Geometric objects	Action scheme	Knowledge accumulated in the synthetic construction process
D_0	$[rt_1]$	*Seed*	f
D_1	$[rt_1, rt_2]$	2	f
D_2	$[rt_1, rt_2, rt_3]$	2	f
D_3	$[rt_1, rt_2, rt_3, rt_4]$	2	f
D_4	$[rt_1, rt_2, rt_3, rt_4, sq_1]$	3	f
D_5	$[rt_1, rt_2, rt_3, rt_4, sq_1]$	*ReInt*	$f(w <= f_1)$
D_6	$[rt_1, rt_2, rt_3, rt_4, sq_1]$	1	$f(w <= f_1)$
D_7	$[rt_1, rt_2, rt_3, rt_4, sq_1]$	1	$f(w <= f_1)$
D_8	$[rt_1, rt_2, rt_3, rt_4, sq_1]$	*ReInt*	$f(w <= f_1)(union(y, z) <= f_2)$

conservation principle its application is allowed. The diagrammatic derivation for our case study is shown in Table 1. The table shows the set of concrete graphical objects at the diagrammatic state D_i, the identifier of the action scheme that produced the current diagram D_i from the previous diagram D_{i-1}; and the geometric knowledge accumulated at each state of the proof process. In the table, f stands for the principle of conservation of area, as formulated in (10). The agent must know this concept in order to be able to realize the theorem and its proof. Reinterpretation acts, tagged as *ReINT*, are perceptual actions that produce a geometric description out of the current diagram; reinterpretations are thought of in terms of perceptual schemes and, as was mentioned, do not reify the recognized entity; consequently, these perceptual acts allow the application of conservation principles if the description refers to all of the relevant objects in the diagram; also, a given diagram can have as many reinterpretations as the number of perceptual schemes satisfying the diagram or its parts that are available to the recognition agent.

Next the detailed form of the semantic objects is reviewed. The introduction of the seed triangle and each application of the second action scheme modify the diagram but these acts do not accumulate knowledge or enrich the interpretation of the diagram. However, these diagrammatic actions are required for providing the diagrammatic scaffolding for construction of the geometric concept, which is properly developed in the second part of the sequence from D_5 to D_8. In this part of the sequence, only area preserving action schemes are allowed. The first reinterpretation produces the description of the square on the hypotenuse, and the principle of conservation of area can be applied to this description, as follows:

$$\lambda([P], \lambda([Q], \lambda(X, eq(area(apply(P, X)), area(apply(Q, X))))))(w <= f_1) \tag{16}$$

$$\lambda([Q], \lambda(X, eq(area(apply(w <= f_1, X)), area(apply(Q, X))))) \tag{17}$$

In general, the process is non-deterministic and several action schemes and conservation principles may be applied at a given point of the derivation process; in the case at hand, (17) could be applied to the descriptions of D_6 and D_7 leading to trivial theorems (e.g. that the area of the top-right and bottom-left configurations is the same); here the correct path is illustrated, and the inference control problem is discussed below, when the prototype program is described. The application of (17) to the description of the two adjacent squares on the right sides of the triangle is shown in (18), with its corresponding reduction in (19):

$$\lambda([Q], \lambda(X, eq(area(apply(w <= f_1, X)), area(apply(Q, X)))))(union(y, z) <= f_2) \tag{18}$$

$$\lambda(X, eq(area(apply(w <= f_1, X)), area(apply(union(y, z) <= f_2, X)))) \tag{19}$$

With this last application the higher-order variables standing for descriptions are reduced and the application of (19) to the focus argument results in a first order function stating the equality between the area of square w on the hypotenuse of triangle x and the area of the union of the squares y and z on the legs of the same triangle x (i.e. the area of an object of sort *square_pair*).

The application of higher-order arguments results in a format that cannot be applied directly by the interpreter, as the embedded descriptions have functions with their corresponding variables. In order to put this function in a proper format (i.e. $\lambda(L, G)$ where G is also a function) the following program transformation rule is used:

$$\lambda(X, eq(area(apply(t_1 <= f_1, X)), area(apply(t_2 <= f_2, X)))) \tag{20}$$
$$\longrightarrow \lambda(X, \lambda(L_1, \lambda(L_2, eq(area(apply(t_1 <= f_1, [X, L_1])), area(apply(t_2 <= f_2, [X, L_2]))))))$$

where L_1 and L_2 are the embedded argument lists of f_1 and f_2 respectively. A transformation rule of this kinds needs to be defined for every conservation principle in the system's data-base. The result of transforming (19) through rule (20) is as follows:

$$\lambda(X, \lambda([w], \lambda([y, z], eq(area(apply(w <= f_1, [X, [w]])), \tag{21}$$
$$area(apply(union(y, z) <= f_2, [X, [y, z]]))))))$$

Function (21) is the final output of the synthetic process and represents the Theorem of Pythagoras in a format that can be applied and evaluated by the geometric interpreter. In this function X stands for a list containing the focus for the transformation (i.e. $[x]$, the argument with the widest scope of f_1 and f_2), which is the same for the descriptions in both sides of the equality. This function can be paraphrased as the following natural language statement: *the area of a square w on the hypotenuse of a triangle x is the same as the area of the union of a square y and a square z on the right sides of the same triangle x.* Function (21) captures the *de dicto* interpretation for all noun phrases in this expression. This proposition is not asserted beforehand and the concept is genuinely synthetic.

This result follows from the application of a general concept of structured equality, the conservation principle, and the ability to establish an equality relation between two generic objects through their generic descriptions. The production of these descriptions depends, in turn, on motor and perceptual schemes that permit the generation of diagrammatic sequences, and the recognition of emergent shapes respectively; finally, the structure of the descriptions depends on the geometric signature and its associated geometric algorithms, the abstractor operator, and the availability of a geometric descriptor operator.

Let f_P be the function in (21), rt_1 be a specific right triangle, and sq_1, sq_2 and sq_3 three specific squares; the application of f_P to these objects in (22) results in (23):

$$f_P([rt_1])([sq_1])([sq_2, sq_3]) \tag{22}$$

$$eq(area(apply(sq_1 <= f_1, [[rt_1], [sq_1]])), area(apply(union(sq_2, sq_3) <= f_2, [[rt_1], [sq_2, sq_3]]))) \tag{23}$$

The evaluation of this expression by the geometric interpreter results in a truth value of true if these objects satisfy the Pythagorean relation and false otherwise.

5. Interpretation mappings and diagrammatic theorem-proving

The Theorem of Pythagoras is a geometric concept and its arithmetic expression $c^2 = a^2 + b^2$ is only true whenever the terms in this equality represent the squares on the hypotenuse and on the legs of a right triangle. The arithmetic expression is interpreted through a representational mapping from the arithmetic into the geometric domain in which square numbers represent geometric squares, the arithmetic addition represents the union between areas, and the arithmetic equality represents the equality between areas; in the absence of this mapping the arithmetic expression is simply false. The production of arithmetic concepts of this kind is also a synthetic process that depends, in turn, on an inverse representational mapping from the geometry into the arithmetic; in this process the synthesis of the arithmetic concept proceeds in tandem with the synthesis of the geometric concept and the diagrammatic derivation.

There are also arithmetic theorems that have a diagrammatic representation and can be derived from a geometric concept; in this case and unlike the Pythagorean theorem, the product of the synthetic process is an arithmetic theorem that is not contingent on the representational mapping. This latter result depends on the nature of the concept of equality employed in the synthesis of the arithmetic concept, and also in the definition between the representational mappings between the geometry and the arithmetic domain. In this section, the derivation of the arithmetic concept and expression of the theorem of Pythagoras is shown first, and then a second case study in which the result of the synthetic process is a genuine arithmetic theorem is presented.

5.1. Arithmetic expression of the Theorem of Pythagoras

For the synthesis of arithmetic concepts the diagrammatic derivation is augmented with an additional tandem process in which arithmetic knowledge is constructed incrementally from the concept of the arithmetic equality. This

Table 2
Derivation of geometric and arithmetic concepts in tandem

DS	Geometric objects	Action scheme	Synthesis of geometric concept	Synthesis of arithmetic concept
D_0	$[rt_1]$	*seed*	f	f_A
D_1	$[rt_1, rt_2]$	2	f	f_A
D_2	$[rt_1, rt_2, rt_3]$	2	f	f_A
D_3	$[rt_1, rt_2, rt_3, rt_4]$	2	f	f_A
D_4	$[rt_1, rt_2, rt_3, rt_4, sq1]$	3	f	f_A
D_5	$[rt_1, rt_2, rt_3, rt_4, sq_1]$	*ReInt*	$f(w <= f_1)$	$f_A(\phi(w <= f_1))$
D_6	$[rt_1, rt_2, rt_3, rt_4, sq_1]$	1	$f(w <= f_1)$	$f_A(\phi(w <= f_1))$
D_7	$[rt_1, rt_2, rt_3, rt_4, sq_1]$	1	$f(w <= f_1)$	$f_A(\phi(w <= f_1))$
D_8	$[rt_1, rt_2, rt_3, rt_4, sq_1]$	*ReInt*	$f(w <= f_1)(union(y, z) <= f_2)$	$f_A(\phi(w <= f_1))(\phi(union(y, z) <= f_2))$

concept is represented as a higher-order function too, and it is applied to the arithmetic interpretation of geometric descriptions during the diagrammatic derivation. For the expression of arithmetic knowledge a second representation language and its interpreter program are defined. This language is based on the standard lambda calculus, and permits the expression of arithmetic expressions and functions; in particular, the concept of the global or unstructured arithmetic equality is represented as the following higher-order function:

$$\lambda([R], \lambda([S], R = S)) \tag{24}$$

In the current interpretation, function (24) represents the principle of conservation of area in the arithmetic domain, and its application runs in tandem with the application of the corresponding geometric conservation principle; the interpreter of the arithmetic representation language is given in full in Appendix D and the actual form of (24) is included in Appendix E below. Let f_A be the function in (24), and ϕ a representation mapping from geometric into arithmetic objects. The tandem derivation of the Pythagorean concept and its interpretation in the arithmetic is illustrated in Table 2.

The definition of ϕ is as follows:

$$\begin{aligned} &\phi(x <= f) = \lambda([u], u^2)) \text{ if type of } x \text{ in } f \text{ is } \textit{square} && (25)\\ &\phi(union) = + \\ &\phi(g(y_1, y_2) <= f) = \phi(g)(\phi(y_1 <= f), \phi(y_2 <= f)) \end{aligned}$$

This is, a square is interpreted as a square number, the union between areas as the arithmetic sum, and the interpretation of a composite geometric object is a function of the interpretation of its constituent parts. No other geometric objects or configurations have an interpretation in the arithmetic domain under this mapping. The definition of ϕ depends on the types of its arguments, and geometric descriptions need to be inspected to check the type of variables on the left side of the description. With this definition, and providing new names for each new variable, the reduction of the final line in Table 2 is as follows:

$$\begin{aligned} &\lambda([R], \lambda([S], R = S))(\lambda([u], u^2))(+(\lambda([v], v^2), \lambda([w], w^2)) && (26)\\ &\lambda([R], \lambda([S], R = S))(\lambda([u], u^2))(\lambda([v, w], v^2 + w^2)) \\ &\lambda([S], (\lambda([u], u^2) = S))(\lambda([v, w], v^2 + w^2)) \\ &\lambda([u], u^2) = \lambda([v, w], v^2 + w^2) \end{aligned}$$

In the prototype system the reduction is performed along with the diagrammatic derivation. The second argument (i.e. the addition of two functions) is transformed by a program transformation rule embedded in the interpreter of the function ϕ, rendering the corresponding function in a format that can be directly evaluated by the arithmetic interpreter, as shown in the second line in the derivation. The last expression is transformed by a second program transformation rule into the arithmetic expression of the theorem; this transformation rule is embedded in turn within the arithmetic interpreter, and its application results in the following function:

$$\lambda([u, v, w], u^2 = v^2 + w^2) \tag{27}$$

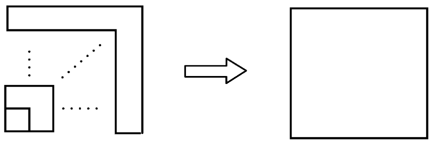

Fig. 6. Diagrammatic proof of the theorem of the sum of the odds.

Function (27) captures the informal statement of the theorem (i.e. $c^2 = a^2 + b^2$) and it holds in case there is a right triangle with hypotenuse c, and legs a and b, and the application of the geometric concept f_P in (21) to this triangle and the geometric squares represented by the square number in (27) results in a truth value of true. The arithmetic interpreter produces the reduction in (26) and permits the application and evaluation of (27).

In the present formulation the synthesis of the geometric and arithmetic proceeds in tandem all the way through, but this is not the only way to think about the process; in particular, the synthesis of the geometric concept can also proceed via the arithmetic interpretation, involving a complex heterogeneous inference; this is illustrated by an alternative proof of the theorem which is due to Bhaskara in the XII Century [21]. This proof is also produced from the top-right diagram in Fig. 3, but instead of the final transformations in which the squares on the two legs emerge side by side, the arithmetic expression is read directly out of the diagram as follows: $c^2 = 4ab/2 + (a - b)^2 = a^2 + b^2$. In this latter case the object produced by the proof is the arithmetic expression, and the geometric concept is placed out of focus somehow, as the squares corresponding to a^2 and b^2 are not realized in the diagram, and this further interpretation of the arithmetic into the geometry is not a part of Bhaskara's proof. However, as the present case study shows, the arithmetic concept is contingent upon the underlying geometric concept, which is the essential knowledge object discovered by the Pythagoreans.

5.2. Diagrammatic arithmetic theorems

The notions of graphical abstraction, conservation principles, action schemes, reinterpretations and interpretation mappings are effective in the synthesis of genuine arithmetic theorems that happen to have a diagrammatic representation; an instance of this kind is the theorem of the sum of the odds which states that $1 + 3 + \cdots + 2n - 1 = n^2$. The truth of this theorem can be realized directly from the diagrammatic proof in Fig. 6. A "corner-frame" of side n in the diagram at the left represents the odd number $2n - 1$, and the sequence of aligned corner-frames represents the sum of these numbers; since the area of the corner-frame sequence is the same as the area of the square resulting from the aggregation of these shapes, the theorem holds.[1]

This theorem has been presented as a representative instance of diagrammatic arithmetic theorems, and it has been studied in AI through the application of inductive theorem-proving methods [22]. In the present terminology, the sequence of corner-frames in the left diagram is reinterpreted through an area preserving action scheme, and the theorem follows from the application of the principle of conservation of area to the descriptions of the geometric configurations on the left and right sides, and from the corresponding mappings into the arithmetic domain.

For the formalization, consider Fig. 7(a) in which an arbitrary square of size $l + 1$ (the length of its side) at an arbitrary position p (its bottom-left corner) is depicted, and Fig. 7(b) where a right corner-frame of size $l + 1$ and width of 1 at position p, is also depicted. The dot p at the bottom-left corner of the square, and at the inner angle of the corner-frame, will be used as pivots for the definition of an additional action scheme, as will be seen below.

For representing these shapes the signature of the geometric language is extended with an abstract geometric object of sort *corner-frame* with its corresponding constructor and selector operators; the arithmetic addition and substraction operations are also included in the representation language.

Functions (28) and (29) express the first and second concepts respectively, as follows:

[1] This representation of odd numbers was known by the Pythagoreans and they also knew the relation between the "corner-frame" and the squares: "... But the ancient commentaries on the passage make the matter clearer still. Philoponus says: 'As a proof... the Pythagorean refer to what happens with the addition of numbers; for when the odd numbers are successively added to a square number they keep it square and equilateral... Odd numbers are accordingly called *gnomons* because, when added to what are already squares, they preserve the square form... Alexander has excellently said in explanation that the phrase 'when gnomons are placed round' means *making a figure* with the odd numbers... for it is practice with the Pythagoreans to *represent things in figure*' " (Heath, 1956, p. 359).

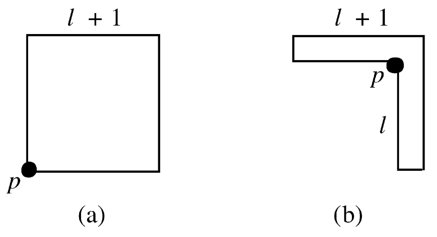

Fig. 7. Shapes used for the theorem of the sum of the odds.

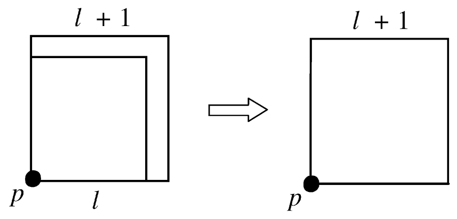

Fig. 8. Fourth action scheme.

$$\lambda([p,l],\lambda([x],\mathit{square}(x)\ \mathit{and}\ \mathit{size}(x)=l+1\ \mathit{and}\ \mathit{position}(x)=p)) \tag{28}$$

$$\lambda([p,l],\lambda([y],\mathit{corner\text{-}frame}(y)\ \mathit{and}\ \mathit{size}(y)=l+1\ \mathit{and}\ \mathit{position}(y)=p)) \tag{29}$$

These functions are defined relative to a dot p at the reference positions of each of these objects and a size parameter l. Let be f_{sq} and f_{c-f} be the functions (28) and (29) respectively. In the same way that abstract descriptions were produced in the case of the Pythagorean theorem, a generic square and a generic corner-frame relative to a position p and size $l+1$ can be described through geometric descriptions, as follows:

$$x <= f_{sq} \tag{30}$$

$$y <= f_{c-f} \tag{31}$$

Next, an additional action scheme for reinterpreting the union of a square and a corner-frame into a new square is defined, as shown in Fig. 8. This scheme is area preserving and its application grants the corresponding conservation principle.

The left-side of this scheme is the union of a square of size l at position p and a corner-frame of side $l+1$ at position $p+l{:}l$; the function expressing this concept is as follows:

$$\begin{aligned}\lambda([p,l],\lambda([w,z],\ &\mathit{square}(w)\ \mathit{and}\ \mathit{size}(w)=l\ \mathit{and}\ \mathit{position}(w)=p\\ &\mathit{and}\ \mathit{corner\text{-}frame}(z)\ \mathit{and}\ \mathit{size}(z)=l+1\\ &\mathit{and\ in\ case}\ l=0\ \mathit{then}\ \mathit{position}(z)=p\\ &\qquad \mathit{else}\ \mathit{position}(z)=p+l{:}l))\end{aligned} \tag{32}$$

On the left side of the scheme, the position of the corner-frame is the position of the top-right vertex of the square (i.e. $p+l{:}l$); however, in the base case (i.e. $l=0$), the top-right vertex of the square of size 0 at position p is also at position p, and the definition of the concept considers this exceptional condition, as shown in the last line of (32). For this, the geometric language includes the control structure *case* which selects its first expression if the condition is true, and the second otherwise. Let f_{sq+cf} be the function in (32); then, the description of the left-side of the scheme is as follows:

$$\mathit{union}(w,z) <= f_{sq+cf} \tag{33}$$

where the interpretation of union is a function of type $\mathit{square}\times \mathit{corner}-\mathit{frame}\rightarrow \mathit{square}$. The description of the right-side of the scheme is (30) directly. The actual forms of descriptions (30) and (33), with the corresponding concepts in (28) and (32), are included in Appendix E in the present paper. With this machinery, the concept of equality of the

Table 3
Diagrammatic derivation

DS	Geometric objects	Action scheme	Knowledge accumulated in the synthetic construction process
D_0	$[sq_{left}, cf_{left}]$	*seeds*	f
D_1	$[sq_{left}, cf_{left}]$	*ReInt*	$f(union(w,z) <= f_{sq+cf})$
D_2	$[sq_{right}]$	4	$f(union(w,z) <= f_{sq+cf})(x <= f_{sq})$

areas between the diagrams at the left and right side of the scheme can be derived from an arbitrary concrete square as shown in Table 3, where f is the principle of conservation of area in (10); in the present case the focus is the pair formed by the position and size of parameters, which remain the same in both sides of the scheme.

Next, the construction of the semantic objects is shown in detail. The application of the principle of conservation of area to the reinterpretation of the square and the corner-frame and the corresponding reduction is shown in (34) and (35) as follows:

$$\lambda([P], \lambda([Q], \lambda(X, eq(area(apply(P, X)), area(apply(Q, X))))))(union(w, z) <= f_{sq+cf}) \tag{34}$$

$$\lambda([Q], \lambda(X, eq(area(apply(union(w, z) <= f_{sq+cf}, X)), area(apply(Q, X))))) \tag{35}$$

The application of the conservation principle to the description of the square resulting from the application of the action scheme, and the corresponding reduction is shown in (36) and (37):

$$\lambda([Q], \lambda(X, eq(area(apply(union(w, z) <= f_{sq+cf}, X)), area(apply(Q, X)))))(x <= f_{sq}) \tag{36}$$

$$\lambda(X, eq(area(apply(union(w, z) <= f_{sq+cf}, X)), area(apply(x <= f_{sq}, X)))) \tag{37}$$

Function (37) represents the concept of the equality of areas before and after the reinterpretation, and this concept is the basic geometric intuition underlying the diagrammatic proof. Finally, the application of the program transformation rule renders (38):

$$\begin{aligned}\lambda(X, \lambda([w, z], \lambda([x], eq(area(apply(union(w, z) <= f_{sq+cf}, [X, [w, z]])),\\ area(apply(x <= f_{sq}, [X, [x]]))))))\end{aligned} \tag{38}$$

The function resulting from the application of (38) to a particular foci list $[p, l]$ represents the concept that the area of the union of a square w of size l at position p and the area of a corner-frame z of size $l + 1$ at position $p + l{:}l$ is the same as the area of a square x of size $l + 1$ at position p.

Let's define a mapping of the geometry into the arithmetic such that a geometric square of size n represents the square number n^2 and a corner-frame of size n for $n \geqslant 1$ represents the odd number $2n - 1$; as before, the union of areas represents the sum operation, and the equality of areas represents the arithmetic equality; this mapping is defined in (39):

$$\begin{aligned}&\phi(x <= f) = \lambda([u], \varphi^2) \text{ if } square(x)\ \&\ size(x) = \varphi \text{ in } f \text{ and } u \text{ is a free variable in } \varphi\\ &\phi(x <= f) = \lambda([u], 2\varphi - 1) \text{ if } corner\text{-}frame(x)\ \&\ size(x) = \varphi \text{ in } f \text{ and } u \text{ is a free variable in } \varphi\\ &\phi(union) = +\\ &\phi(g(y_1, y_2) <= f) = \phi(g)(\phi(y_1 <= f), \phi(y_2 <= f))\end{aligned} \tag{39}$$

The tandem derivation is shown in Table 4, where f_{EQ} is the structured concept of arithmetic equality relative to a focus parameter N as follows:

$$\lambda([P], \lambda([Q], \lambda(N, eq(apply(P, N), apply(Q, N))))) \tag{40}$$

Unlike the concept of equality used for the case of the arithmetic expression of the Pythagorean theorem in which the terms are independent variables, in the present case the interpretations of the geometric descriptions into the arithmetic domain are functions of a parameter N. The form of (40) in the arithmetic representation language is

Table 4
Tandem derivation

DS	Geometric objects	Action scheme	Knowledge accumulated in the synthetic construction process	Synthesis of arithmetic concept
D_0	$[sq_{left}, cf_{left}]$	*seeds*	f	f_{EQ}
D_1	$[sq_{left}, cf_{left}]$	*ReInt*	$f(union(w,z) <= f_{sq+cf})$	$f_{EQ}(\phi(union(w,z) <= f_{sq+cf}))$
D_2	$[sq_{right}]$	4	$f(union(w,z) <= f_{sq+cf})(x <= f_{sq})$	$f_{EQ}(\phi(union(w,z) <= f_{sq+cf}))(\phi(x <= f_{sq}))$

included in Appendix E. Next, the synthesis of the arithmetic concept and the corresponding reduction is shown in detail:

$$\begin{aligned}
&\lambda([P], \lambda([Q], \lambda(N, eq(apply(P,N), apply(Q,N)))))\\
&\qquad\qquad (+(\lambda([u], u^2), \lambda([v], 2(v+1)-1)))(\lambda([w], (w+1)^2))\\
&\lambda([P], \lambda([Q], \lambda(N, eq(apply(P,N), apply(Q,N)))))\\
&\qquad\qquad (\lambda([u,v], u^2+2(v+1)-1))(\lambda([w], (w+1)^2))\\
&\lambda([Q], \lambda(N, eq(apply(\lambda([u,v], u^2+2(v+1)-1), N), apply(Q,N))))(\lambda([w], (w+1)^2))\\
&\lambda(N, eq(apply(\lambda([u,v], u^2+2(v+1)-1), N), apply(\lambda([w], (w+1)^2), N)))
\end{aligned} \tag{41}$$

As before, the actual reduction in the prototype program is performed during the diagrammatic derivation; also, the final function representing the theorem in (42) results from the application of an additional program transformation rule that binds the two arguments of the function in the first *apply* of the last expression in derivation (41) as in this context these two variables are bound to the focus parameter N, as follows:

$$\lambda(N, eq(apply(\lambda([z], z^2+2(z+1)-1), N), apply(\lambda([w], (w+1)^2), N))) \tag{42}$$

In the Prolog implementation, the interpreter of the function ϕ for both the Pythagorean theorem and the theorem of the sum of the odds is the same, and the addition of two functions is transformed into a legal format in this latter case too, as shown in the second line of the derivation (41); the final program transformation binding rule is also embedded within the arithmetic interpreter, and the last expression in derivation (41) can also be applied and evaluated directly by the arithmetic interpreter.

Function (42) corresponds to the informal expression $n^2 + 2(n+1) - 1 = (n+1)^2$ which is the induction step in the theorem of the sum of the odds; also, the two sides of the equality are bound by the focus parameter and this expression holds regardless of the interpretation mappings; as the concept of arithmetic equality employed in the present derivation is a conservation principle that binds the terms produced in the different stages of the derivation, the resulting arithmetic expression is necessarily true.

The equality in (38) provides, in addition, a constructive procedure in which a square of size l can be generated inductively from the union of a square of size $l-1$ and a corner-frame of size l. The basic generation step is defined by the left side of (38) as follows:

$$\gamma = \lambda(X, \lambda([w,z], apply(union(w,z) <= f_{sq+cf}, [X, [w,z]]))) \tag{43}$$

The induction on the parameter l is defined on the basis of (43); in this process the square w in the union term is generated through the application of the function γ itself to the parameter pair $(p, l-1)$; this is, a square of size l at position p is generated through the recursive composition of a square of size $l-1$ and a corner-frame of size l; the basic objects in the recursion are a square of size 0 at position p and a corner-frame of size 1 at position p as well. The definition of this function, following very closely its expression in the geometric representation language, is as follows:

$$\begin{aligned}
\gamma' = \lambda([p,l],\ &\text{case } l = 0 \textit{ then } square(dot(p), 0)\\
&\qquad\qquad \textit{else } apply(union(w,z) <= f_{sq+cf}, [[p, l-1)],\\
&\qquad\qquad\qquad\qquad [apply(\gamma', [p, l-1)]),\\
&\qquad\qquad\qquad\qquad\quad corner\text{-}frame(dot(p+l-1{:}l-1), l]])
\end{aligned} \tag{44}$$

Finally, the area of a square of size l, the right side of (38), is computed by function (45) as follows:

$$\lambda([p, l], area(apply(\gamma', [p, l]))) \tag{45}$$

Function (44) represents the geometric induction on squares and corner-frames in Fig. 6. The essential part of the synthetic process is the generation of the geometric concept in (38) through a diagrammatic derivation, as the equality provides the constructive term of the inductive procedure. The induction on the geometric concept is a further abstraction which is derived from the geometric concept through a symbolic transformation process. In the current implementation, function (44) is defined directly by the human user out of (38); the evaluation of this function by the geometric interpreter produces a square of size l at position p through the corresponding geometric induction. The actual form of γ' in the geometric representation language is included in Appendix E. The automatic synthesis of the function representing the geometric induction out of the basic geometric concept is a complex problem that for the moment is left for further research.

The arithmetic concept itself is produced by an analogous inductive procedure from the equality in (42); in this latter case the square number on the right side is constructed by the recursive application of the term on the left side, and the basic condition in which the parameter $N = 0$. The basic constructive function is:

$$\tau = \lambda(N, apply(\lambda([z], z^2 + 2(z + 1) - 1), N) \tag{46}$$

The induction on the parameter N is obtained by substituting the term z^2 by the function that generates this square number from the function τ' itself applied to the parameter $z - 1$ as follows:

$$\tau' = \lambda(N, apply(\lambda([z], (\mathit{if}\ z = 0\ \mathit{then}\ 0\ \mathit{else}\ apply(\tau', z - 1)) + 2(z + 1) - 1), N)) \tag{47}$$

Functions τ' represents the theorem of the sum of the odds, which in informal mathematics is stated as $n^2 = 1 + 3 + \cdots + 2n - 1$. In this case, the essence of the theorem is the equality in (42), and the expression of the recursion is also a straightforward symbolic transformation process. As in the definition of the geometric induction, function (47) is defined by the human user, and its evaluation by the arithmetic expression produces a square number through the corresponding arithmetic induction. The actual form of τ' in the arithmetic representation language is included in Appendix E.

The purpose of this second example is to illustrate how the construction of geometric abstractions and the application of conservation principles and action schemes permit the synthesis of the underlying geometric concept, which is the basic intuition upon which the arithmetic theorem and proof are realized. The synthesis of this kind of inductive arithmetic theorems is extremely straightforward for human interpreters once the geometric concept has been constructed and the appropriate interpretation mappings are available. It is also plausible that diagrammatic logical reasoning, such as the use of Venn diagrams, can be explained in similar terms, but for the moment such an exploration is left for further research.

Finally, as a conclusion of the presentation of the synthetic process, the present investigation clarifies a taxonomy of diagrammatic theorems proposed by Jamnik [22] and adopted by Foo [13], in which diagrammatic theorems are classified into three kinds: (1) geometric theorems involving a diagram and an interpretation into the arithmetic; (2) diagrammatic theorems involving, in addition, an induction with one parameter, as exemplified by the theorem of the sum of the odds, and (3) diagrammatic theorems whose proofs are inherently inductive. From the perspective of the present theory, functions (21) and (38), representing the Theorem of Pythagoras and the geometric concept underlying the inductive step of theorem of the sum of odds, belong to the same class, as they were produced by the same process and have a similar form; the distinction depends rather on the further ‘symbolic’ process that is required to synthesize the functions representing the geometric and arithmetic inductions involved in the full theorem of the sum of the odds, and this is a further abstraction that is independent of the knowledge domain and also of the representational medium. In addition, the inductive theorem involves a structured concept of equality, which is itself a conservation principle in the arithmetic domain.

The study of theorems of class (3) may also be informed by the present theory. An instance of this latter case is the diagrammatic representation of the infinite sum $1/2 + 1/4 + 1/8 + \cdots + 1/2^n = 1$; to visualize this equality, consider that the right side can be represented through a unit square, which can be partitioned into an aggregation of non-overlapping areas corresponding to each of the fractions on the left side; this can be done by decomposing the unit square into two rectangles a half of its size, and one of this rectangles into two further squares a quarter of the size of the original unit square, as shown in Fig. 9.

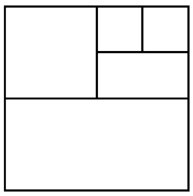

Fig. 9. Inherently inductive diagrammatic proof.

This reinterpretation yields the basic geometric concept underlying the theorem, where the first rectangle corresponds to the first fraction (i.e. $1/2$, one of the squares to the second (i.e. $1/4$), and the other to the rest of the terms in the sum; the infinite expansion is produced by reinterpreting inductively this latter square (i.e. the top-right square in Fig. 9) by the same procedure. In this case, the geometric and arithmetic inductions could be represented through functions analogous to (44) and (47) respectively. The knowledge of these functions is independent of their actual application to a particular parameter, and the only difference between theorems in classes (2) and (3) would depend on the evaluation process that would render only an approximation for theorems in class (3). The basic inductive step for this latter geometric construction can also be synthesized through the application of the principle of conservation of area and an appropriate action scheme, as in the two case studies developed above, and the process that synthesizes the basic geometric concept underlying the theorems in the three classes seems to be essentially the same.

6. Perceptual schemes and the generation of geometric descriptions

In the present investigation it is required to produce or synthesize geometric descriptions for the diagrams appearing in a diagrammatic sequence. This task involves the acts of looking at a diagram, realizing a relevant object, perhaps through a visualization, and generating its appropriate description. Descriptions can be thought of as internal representations used for thought and action, but this does not have to be the case, as the perceptual action can be thought of as the act of looking at the diagram and writing down its description on a piece of paper, and both the diagram and its description are external representations. What is relevant for this interpretation or translation process is that the external diagram and its description mean the same: if the diagram has a concrete interpretation, its description should refer to the same concrete object, but if the diagram expresses an abstraction, its description should express the same abstraction too.

The perceptual interpretation task is a process that maps concrete diagrams into descriptions, and reverses in this sense the direction of the motor schemes that produce concrete diagrams out of descriptions. In the same way that motor action schemes are the causal knowledge structures for drafting actions, the interpretative acts mapping external representations into descriptions can be thought of in terms of action schemes too and, in the present theory, this latter kind of schemes are referred to as perceptual action schemes, or simply perceptual schemes. The input to a perceptual scheme is the extensional representation of the diagram, and its output is a geometric description used for problem-solving, and in this case, for the synthesis of geometric concepts. However, this bottom-up view of the operation of perceptual schemes needs be complemented with a top-down perspective; this is the case because the output description has to be relevant for the task at hand, and conceptual knowledge is also involved in the synthesis of the appropriate description. The top-right diagram in Fig. 5 satisfies the description *a square on the hypotenuse of a right triangle*, but this diagram also satisfies a very large number of descriptions denoting the objects, properties and relations in the diagram and its parts (e.g. the inner square, each of the triangles, the relation between the inner square and one triangle, the relation between the sides of the triangle and the size of the small square, etc.), and the production of any of these descriptions signals a perspective, either concrete or abstract, that the human interpreter places on the diagram and in relation to a particular problem solving task. The need for conceptual knowledge for the synthesis of descriptions can be further appreciated in (30) and (33), whose definitions require deep common-sense knowledge about the conceptual domain and the aims of the task.

The definition of perceptual schemes is complex in many respects. First, a conceptual context (i.e. a body of concepts about the knowledge domain) needs to be considered; also, a given object can have many different descriptions, and criteria for choosing the appropriate one must be available; in addition, the selection of a particular description that is relevant for a particular problem-solving task depends on deep, common-sense knowledge of the interpreter, and there is no evident model of how thought and perception interact in the synthesis of this kind of description.

The distinction between the concrete and the abstract could suggest that the task can be broken down into two main parts: the generation of a concrete description from the diagram, and the application of an abstraction operation upon this object. For instance, a perceptual scheme could produce first a concrete description of the square on the hypotenuse out of the top-right diagram in Fig. 5, as follows:

$$sq_2 <= \mathit{is\_right\_triang}(rt_1)\ \mathit{and}\ \mathit{is\_square}(sq_2)\ \mathit{and}\ \mathit{side}(\mathit{hypotenuse}(rt_1), sq_2) \tag{48}$$

Then, the description in (12) and (13) could be produced by an abstraction operation upon (48). In fact, as diagrams have been traditionally thought of as having concrete referents, models of their interpretation have focused on the generation of concrete representations; an antecedent in this line of work is Reiter and Mackworth's logic of depiction for computer vision [44], in which logical representations of diagrams in the scene domain were produced out of the extensional representation of the overt basic shapes in the image domain through constraint satisfaction. Another antecedent is the *Graflog* program which had an algorithm for the synthesis of propositional representations of simple geometric objects in an interactive graphics setting [25,39,40]. In this program, the synthetic algorithm produced the simplest concrete term involving a reference to all basic objects in the context of the emerging object. More recently Chandrasekaran and his team introduced perceptual routines for the identification of concrete, overt and emerging, points, lines, curves and regions in the context of the *DRS* system [10].

A related line of work is the generation of natural language descriptions to pick out an object among a set of objects in terms of its distinguishing properties [11,52]. Also, the natural language translation between natural language descriptions and diagrams construed as expressions of a graphical language is explored along the general guide-lines of Montague's semiotic program by Pineda and Garza [43]. However, none of these approaches is able to produce descriptions that are relative to a geometric context in which the reference object can satisfy many different descriptions, and the description itself is relative to a conceptual domain and a problem-solving task, like (47) or specific instantiations of (30) and (33), and an open line of research is the generation of geometric descriptions of this kind of complexity.

An additional consideration is that pivoting the synthesis on concrete representations is not essential and the problem can be construed in terms of perceptual schemes that produce the abstraction directly, and there may even be tension about different potential interpretations, as shown by ambiguous pictures like the duck-rabbit made famous by Wittgentein [55], shown in Fig. 10.

Thinking in terms of perceptual schemes, the output of this perceptual interpretation process can be thought of as the description of a generic duck (i.e. $x <= \lambda([x], \mathit{duck}(x))$, as the drawing does not refer to any particular duck; however, the figure is very unstable and can be reinterpreted spontaneously by a kind of "perceptual reflex" as a rabbit (i.e. $x <= \lambda([x], \mathit{rabbit}(x))$). This interpretation process has a bottom-up character as the geometry of the diagram selects one concept among the set of potentially available concepts in the interpretation context, but there is a top-down component too, as the expectation of seeing a duck or a rabbit can influence what is perceived. People have very little control over this perceptual task and the perceptual reinterpretation cycle can proceed as long as we keep looking at the drawing. In this perceptual task the hypothesis that the interpreter forms a concrete description in his mind first, perhaps unconsciously, and produces the final interpretation by an abstraction operation upon such an object, does not seem plausible.

At the present state of this investigation we conjecture that the generation of geometric descriptions through perceptual inferences can be posed in terms of abduction, as follows: given a diagram or a part of it (i.e. its extensional

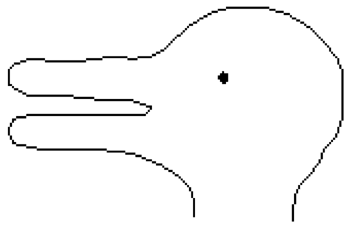
Fig. 10. An ambiguous picture.

representation), find the description of the abstraction expressed by the diagram, among possibly many, that can be produced in terms of a given set of perceptual schemes and a conceptual context, and that satisfies the diagram. In this view the observational sentences are "explained" in terms of the set of available perceptual schemes, which are defined relative to a conceptual domain and a problem-solving task. In a related formulation, a theory of perception as abduction in which the observational sentences produced by the low-level vision process are "explained" by a description that is consistent with a background theory has been proposed by Shanahan [45]; however, it is not clear that such a kind of logical approach for perceptual inference can deal with reinterpretations, as these acts change the set of geometrical entities represented by the external drawing; also, unlike this latter theory that is concerned with the perception of objects in the world, which have a concrete nature, perceptual schemes of the kind suggested here would be concerned with the interpretation of external representations that can have either a concrete or an abstract character.

For all these reasons, the formalization of perceptual schemes, even for this simple domain, is a task of tremendous complexity and the definition of a theory of perceptual schemes and its related perceptual inferences is at this stage beyond the scope of the present investigation. Nevertheless, the notion of perceptual scheme does allow us to specify the task explicitly and suggest an interactive problem-solving architecture in which the inferential load is shared between the system and the human user. In this architecture motor schemes are applied by the system but perceptual inferences are performed by the human user; however, the input and output representations of the hypothetical human perceptual schemes are expressed as descriptions that can be used by the system in problem-solving. This interactive architecture is used for the definition of a prototype program that assists the human user in the synthesis and proof of geometric concepts, as will be shown next.

7. The prototype program

In order to test the theory a prototype program was developed. The core of the system is the interpreter of the geometric representation language; this language permits the definition and interpretation of conservation principles, action schemes, geometric concepts, and descriptions of concrete and abstract objects and configurations. The interpretation of geometric concepts and descriptions, and the application of conservation principles and action schemes is fully performed by the system; the synthetic function representing the Theorem of Pythagoras in (19) and its transformation into (21) are also produced automatically by the system; this function can be applied and evaluated by the geometric interpreter to an arbitrary right-triangle and three arbitrary squares, and its value is true if these objects stand in the Pythagorean relation and false otherwise.

The program includes the definition of an arithmetic language with functional abstraction, and its corresponding interpreter; this language is expressive enough to define and apply functions, both higher-order, like the concept of arithmetic equality in (24), and first-order, like the square function, or the function representing the arithmetic expression of the theorem in (27). The mapping function ϕ from geometric to arithmetic terms in (25) and its corresponding interpreter are given in advance to the system, and the arithmetic derivation is produced in tandem with the geometric one, and the arithmetic expression of the Theorem of Pythagoras in (27) is also synthesized by the system and can be applied and evaluated by the arithmetic interpreter directly. The final output of the tandem process is the function representing the geometric concept and its arithmetic expression under the interpretation mapping.

The diagrammatic derivation process is non-deterministic, and the size of the search space depends on the number of conservation principles and action schemes available, and the ways these can be applied. All drawing action schemes are defined relative to a diagram and a focus object or configuration, and their application produces a new diagram and the patient object that is modified by the scheme, as illustrated in Fig. 11. In the current prototype a state is defined as a list including the current patient, the current diagram, and the current geometric and arithmetic concepts. The system has a main problem-solving loop where a motor action scheme (i.e. a drawing action) or a perceptual reinterpretation action (i.e. the production of a geometric description) is performed in each interactive cycle.

In the case of the Pythagorean theorem the synthetic process starts from an arbitrary triangle seed; although several combinations of objects in a drawing may satisfy the context of an action scheme, only one combination does this in relation to the focus for the second and third schemes, and the application of these schemes is deterministic; in the case of the first action scheme there may be one or two patient objects for a given focus and whenever there are two potential patients, the system presents these options, and the human user selects the patient for the specific application of the scheme.

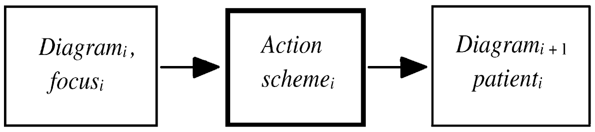

Fig. 11. Shape generation process.

The definition of action schemes includes the specification of the geometric context in which the scheme can be applied in relation to the focus, and the action that modifies the drawing; both the geometric context and the action itself are defined through expressions of the geometric representation language; the definition of action schemes also includes functions that compute other parameters of the scheme if any, and the appropriate control procedures; in addition, the application of an action scheme is allowed only if the object modified by the scheme does not intersect a geometric object in the output drawing. The expressions for the three action schemes used in the proof of the Theorem of Pythagoras are shown next:

Action Scheme 1: (49)

Context: *aligned_AB_NCRV*(*focus*, *patient*)

Pivot: *vertex_aligned_AB_NCRV*(*focus*, *patient*)

Action: *rotate*(*patient*, θ, *Pivot*)

Action Scheme 2: (50)

Context: *is_right_triangle*(*focus*)

Pivot: *right_vertex*(*focus*)

Action: produce *translate*(*rotate*(*focus*, θ, *Pivot*), $\delta x{:}\delta y$)

Action Scheme 3: (51)

Context: *aligned_AB_NCRV*(x, y) and

aligned_AB_NCRV(y, z) and

aligned_AB_NCRV(z, w) and

aligned_AB_NCRV(w, x)

Focus: combination of four right triangles satisfying the context

Action: produce *square*(*right_vertex*(x), *right_vertex*(y),

right_vertex(z), *right_vertex*(w))

In (49) and (50) the angle θ is $3/2\ \pi$ or $-3/2\ \pi$ and $\delta x{:}\delta y$ is the difference between the lengths of the right sides of the focus right triangle in the relevant direction; these parameters are computed by functions defined within the body of the corresponding action scheme, and their value depends on the relative orientations of the right triangles involved.

In the synthetic process only the second action scheme can be applied out of the initial right triangle seed, which is also the focus for the application of the scheme; however, once a right triangle has been added both the first and the second action schemes can be applied and a search space is defined; during the diagrammatic derivation one, two or even the three action schemes could be applied in relation to one or more foci at a given inference step, giving rise to a very large problem-solving space. In order to control the inference process a procedure that suggests a ranked list of action schemes with their corresponding foci is defined. This procedure is based on three heuristic rules, as follows:

(a) If there is a set of four right triangles satisfying the context of the third action scheme and the "hole square" emerging from the four right triangles is not included in the drawing, the application of the third action scheme is suggested; the focus is the four right triangles configuration satisfying the context of the scheme.
(b) If there are two triangles standing in the output context of the second scheme, which is also the input context of the first scheme, and one of the two triangles in the context is the patient of the previous action scheme, the

application of the second scheme is suggested and the current patient becomes the focus for the next application of the scheme.

(c) If there are two right triangles satisfying the context of the first action scheme and neither of these is the current patient, the application of the first action scheme is suggested and the two right triangles in the context become potential foci for the application of the scheme.

The three heuristics are ranked from (a) to (c); the intuition is that whenever the structured configuration of the third action scheme is produced in a search process the emerging square is recognized by a kind of perceptual reflex, and its application is more likely to produce an interesting diagrammatic sequence than the other two. The context of the second scheme is very unrestricted as it only needs a right triangle to be applicable; however, this scheme is more likely to be relevant if the output configuration of the scheme itself is present in the current drawing, and the current patient object becomes the focus for the next application of the same scheme; heuristic (b) implements in this way a natural focus shift strategy that resembles natural language generation and interpretation heuristics where the object/patient of a sentence becomes the subject/agent of the next sentence in a piece of discourse (e.g. *John loves Mary*. *She loves him back*). The last heuristic rule is preferred when the triangle generation process is exhausted, and the search for square-like structures is more relevant for the problem-solving process. For the definition of these heuristics all combinations of the right triangles in the drawing taking four triangles at a time for heuristic (a), and two triangles at a time for heuristics (b) and (c), are considered.

The three heuristics are not mutually exclusive and there may be a number of action schemes with one or more potential foci that can be applied at an arbitrary inference step. In the current prototype system the inference load is shared between the system and the human user, and the user selects an action scheme with its corresponding focus from the ranked list provided by the selection procedure; in addition, a function to undo the last state in order to explore the search space is also defined. For the present case study the first option provided by the system leads directly to the diagrammatic proof, with the exception of the last inference step; although the right action scheme is suggested for the last transformation of the diagram (i.e. the first one), there are three potential foci for the application of this scheme, and also two potential patients for the focus, and search is required to arrive at the final configuration.

In the current prototype, the human user also decides whether the application of a conservation principle is performed or postponed; as was mentioned, the unrestricted use of the principles leads to trivial theorems, and their application should be constrained by other heuristics. In the present case study, the interpretation constraints in (25) above are used as heuristics to limit the search, as the application of conservation principles is more likely to produce interesting theorems if their arguments, and the principles themselves, have an interpretation in a target conceptual domain.

The synthesis of the theorem of the sum of the odds is also produced by the prototype system with exactly the same control procedures and interpreters. The principle of conservation of area employed in both of the proofs is the same, but instead of the "global" equality concept used for the Pythagorean theorem, in which the higher-order variables on the two sides of the equality are independent, the "structured" concept of equality in (40), in which the higher-order variables in the two sides of the equality stand for functional descriptions of a common argument, is used in this latter case. The drawing seed consist on an arbitrary square of size l at position p and a corner-frame of size $l+1$ at position $p+l{:}l$; a fourth action scheme that reinterprets the aligned square and corner-frame as a new square of size $l+1$ at the same position p is defined too. The focus selection procedure suggests the application of this action scheme in relation to the focus pair formed by the position and size parameters, if the geometric context is satisfied by the current diagram or diagrammatic state. The derivation in Table 4 is produced in three steps; in the first the principle of conservation of area is applied to the description of the input diagram, and the arithmetic principle of conservation of truth is applied to the arithmetic interpretation of the geometric description; in the second, the action scheme 4 reinterprets the drawing; as this action scheme is area preserving the current geometric and arithmetic concepts are applied to the description of the resulting diagram and its corresponding arithmetic interpretation respectively. The output of the process is the geometric function (38) and its corresponding interpretation in the last line in derivation (41); the geometric function can be applied to concrete squares and corner-frames, and its value is true if these objects stand in the corresponding relation and false otherwise; the application of the arithmetic function to an integer number is, on the other hand, necessarily true. Functions (44) and (47), representing the geometric induction in Fig. 6 and the theorem of the sums of the odds respectively, can be also expressed and evaluated by the geometric and the arithmetic interpreters; in the current implementation these functions are input directly by the human user and they are applied to an arbitrary parameter, and the interpreters produce the corresponding inductions.

In the prototype system, the human user is responsible for inspecting the diagrams in every diagrammatic state, and for providing descriptions of emerging objects and configurations. This is a very complex perceptual inference involving perception and thought, as was argued above, and this is the main reason for suggesting the current interactive inference strategy. The diagram produced at every generation step needs to be inspected by the human user in order to see whether there is a description that is worth considering as a part of a process of synthesis and proof, and it is planned to extend the prototype system with a graphical interface in which the system provides interactive devices that help the user to visualize potentially interesting emerging objects. However, the synthesis of descriptions of diagrams or their parts, that emerge in diagrammatic derivations, will still remain on the side of the user. The formalization of this kind of inference is a long term goal, which needs to be achieved before the synthesis and proof of geometric concepts can be performed by the system through a fully automatic process.

8. Related work

The present investigation can be placed in the tradition of research on diagrammatic representation and inference in AI. Diagrammatic reasoning was started by the Geometry-Theorem Proving Machine (*GTP*), developed by Gelernter [16] in the late fifties; since then diagrams have been used to guide the search process in a proof tree [2,19,26,34]; in a more structured setting, diagrams have been used to index plans for constructing geometric proofs [33]. More generally, the role of diagrams as "indices" to support a computational process efficiently has been highlighted by Larkin and Simon [27] in their very well-known paper. Since Gelernter's and Larkin and Simon's seminal papers there has been a substantial body of work in this field and some of the most relevant references are collected in [17]. In this context, the present theory resembles model based reasoning approaches along the lines of Johnson-Laird's Mental Models [23] and Lindsay's model of non-deductive inference [30], although instead of the procedural inference schemes postulated by these latter theories, the present approach relies on functional application as its main knowledge construction tool.

The approaches described above rely mainly on symbolic descriptions for the representation of diagrams. However, there is an alternative view in which diagrams are represented in a format that reflects the structure of the world directly, and are thought of as analogical representations [47]. In spite of the strong arguments for the reduction of analogical to propositional or "fregean" representations [20], there is an ample tradition of systems in the analogical tradition. In Funt's *Whisper* system [14], for instance, diagrams were represented through a computational retina, consisting of a circular array of processors, where each processor was associated with a color representing a spatial region, and the operations on the array were controlled by a symbolic high level reasoning component; this system was able to solve some qualitative physics problems by simulation. Another example is Anderson and McCartney's Inter Diagrammatic Reasoning system (*IDR*) [3] where diagrams are defined as a tessellation of a finite planar area completely covered by tiles, each of some gray scale or color value. This system is able to draw different kinds of inferences, like making decisions, solving constraint satisfaction problems, and learning, etc., by applying a predefined set of operations directly on the tiles of a diagram or a set of diagrams. *IDR* has also a symbolic aspect, and computations on diagrams are stated in the lambda calculus.

The question of whether diagrams are represented by descriptions or as images is the subject of the imagery debate [51], and the opposition between the symbolic and analogical views can be thought of as the AI counterpart of this debate. The present theory does support that descriptions are used in diagrammatic reasoning, but it is not inconsistent with the view that images may play a role in representation and inference. For instance, the representation and manipulation of informal diagrams, like sketches, may require an analogical format, but descriptions for thought and action could also be produced by perceptual and motor schemes, acting upon the represented image, and this interaction may help to explain why informal diagrams can be involved in valid reasoning. In the present theory, geometric predicates have an associated geometric algorithm, which performs specific computations in the interpretation of descriptions; in an analogous manner, a number of procedures associated with specific predicates of the descriptive language, and defined in terms of the structure of the analogical representation, could be responsible for the synthesis of descriptions in perceptual inference, and for the transformation of images through motor schemes, in a heterogeneous architecture.

There is also an important body of work related to the definition of geometrical abstract data-types. The present formulation, as was mentioned, is based on Pineda [39], which in turn was based on Goguen et al. [18]. Along the same lines, Wang [53] presented an order-sorted logic for representing pictorial concepts. In a related line of work Kamada and Kawai [24] and Matsuoka et al. [32] developed a framework for visualizing abstract geometrical data-types, and for the bi-directional translation between abstract objects and their visualization. Also, in the object-oriented framework, Alberti et al. [1] presented a program for interactively modeling geometric objects in Euclidean geometry through the definition and manipulation of geometric abstract data-types. Finally, in a somewhat different orientation, tiles and tile's operations in Anderson and McCarney's *IDR* system can also be thought of as abstract data-types.

Another relevant line of work is the development of diagrammatic architectures to support the full diagrammatic inference and problem-solving cycle (i.e., interpretation, reasoning and action). An early antecedent in this line was the *Graflog* system [39,41]. *Graflog* had a knowledge-base partition for representing the geometry of a diagram and a different partition for representing its interpretation in relation to an arbitrary domain, as well as additional symbolic information. The system could inspect and modify the diagram, and also had a model of heterogeneous inference; it also had a multimodal interface through which diagrammatic and symbolic information was input and output during the interactive cycle. In this architecture the actual inference and problem-solving strategies had to be built on top of the basic representational layer, and several applications involving deductive inference [37,39], simulation [38], constraint satisfaction [40,42] and a form of abduction [41] were developed. A more recent effort along similar lines is exemplified by Chandrasekaran's *DRS* system [10]. This system aims to provide a general architecture to support problem-solving with diagrammatic and symbolic information.

The question of whether diagrammatic proofs are valid has been the subject of much interest by logicians; in particular Barwise and Etchemendy developed an information based approach for deductive inference, the so-called situation semantics, in which implicit information can made explicit regardless of the modality, graphical or linguistic, in which overt information is originally expressed [4,6]. In this context, Shin [46] presented a formal analysis of Venn diagrams; she showed that this diagrammatic deductive system is sound, and also provided a completeness proof. In this work Shin introduced an equivalence relation, the so-called counterpart relation, for defining the class of circles with an arbitrary position and size that represent the same set in a given Venn diagram. This logical device is somehow analogous to the geometric descriptor introduced in the present theory as it permits giving a generic interpretation to concrete diagrammatic objects.

On a more applied orientation, Barwise and Etchemendy developed the *Hyperproof* program in which diagrams are used to teach first-order logic [5]. This system supports heterogeneous inference for proving theorems about a blocks world; in this setting the premises are expressed partially through a diagram and partially through a set of logical formulae, but the information required for the solution of the problem is incomplete in either of the modalities. This kind of problems can also be seen in terms of model construction and solved through constraint satisfaction, in the framework of standard model theory, as illustrated in [42].

In relation to the particular examples developed in this paper, it is somehow surprising that despite its apparent simplicity and its importance to mathematics, and also the emphasis given to theorem-proving in AI at that time, there are no reports of attempts to prove the Theorem of Pythagoras by Gelernter's program, or by any other AI program in the tradition started by *GTP*; the theorem has been used to illustrate heterogeneous reasoning, as in Barwise and Etchemendy [4], to exemplify a taxonomy of diagrammatic theorems [13,22], and to study the role of interpretations in diagrammatic theorem proving [53], but non of these studies presented an automated proof of the theorem.

An attempt in this direction, although from a more cognitive perspective, is the *ARCHIMEDES-STUDENT* program developed by Lindsay [31]. This program can construct a diagrammatic sequence from propositional instructions (i.e. commands) provided by the human user; the program has also a set of perceptual routines that allow the identification of emergent geometric objects and relations, and these routines support drafting actions. Perceptual actions are also stated through propositional instructions that command the program to 'observe' geometric entities in the diagram. In this program, theorems and geometric constraints are also input through propositional instructions whose effect is to observe, remember and verify concrete properties of diagrams along the diagrammatic derivation. Diagrams are modified as a consequence of the instructions, and geometric constraints are satisfied incrementally; for this a constraint maintenance technique is used; this feature of the program is closely related to the tradition of geometric constraint satisfaction [8], that has its roots in Ivan Sutherland's Sketchpad program [50], although within a simulation paradigm. This facility also resembles drafting actions that maintain a given set of geometric constraints, developed in the *Graflog* program [40,41]. Through this machinery, Linsay's program has produced and verified several diagrammatic demonstrations of the Theorem of Pythagoras.

There is a sense in which Lindsay's program 'understands' the diagrammatic demonstrations, as it is able to interpret the instructions provided by the user, and to verify that the constraints representing the theorem hold. However, such a sense of understanding differs from the present approach in which a function representing a concept is synthesized, and this function is effective in computing whether the concept holds of a set of geometric or diagrammatic objects, for all instances in the extension of the concept. In this regard and, as far as this author is aware, a diagrammatic theorem-proving system, focused on the synthesis of the Pythagorean geometric concept and its arithmetic expression, both in terms of external representation and its corresponding interpretation, has not been previously presented in the AI literature.

The theorem of the sum of the odds, in turn, has been studied in AI by Jamnik [22] through the application of inductive theorem-proving methods. In Jamnik's system, human users are able to construct proofs by applying diagrammatic operations to a concrete diagram representing an instance of the theorem, and these operations are equated with the inference steps of the proof. The system can then extract a recursive program from the sequence of diagrammatic operations through the so-called constructive w-rule, and this program represents a general schematic proof of the theorem; finally, the system is able to check that the proof is indeed correct, and can verify specific instances of the theorem. This latter approach differs from the present theory in the view that is taken on graphical abstraction: while the generalization in Jamnik's work is obtained from the diagrammatic operations made by the human-user and the application of the constructive w-rule, in the present approach concrete diagrams are interpreted as generic objects through perceptual schemes, and the abstraction is expressed through the geometric descriptor and the functional abstractor operator. Also, while Jamnik's approach focuses on the validity of diagrammatic theorems by logical means, the present theory is centered on the synthesis of geometric and arithmetic theorems out of conservation principles and action schemes. In addition, Jamnik's method is restricted to inductive theorems, while the present theory shows that the underlying geometric concept of geometric theorems, inductive theorems, and inherently inductive theorems, is of the same kind.

The present theory also has psychological, semantic and design motivations. The notions of conservation principles and action schemes introduced in this paper were inspired in part by the corresponding notions of Piaget's mental development theory [36]. Piaget noticed that the ability of a child to realize that a property is preserved in a process of change depends on the child's mental development stage; in particular, children acquire very early the ability to realize that an object is the same before and after a change (i.e. when children develop the concept of an individual object) and later on children develop conservation principles for specific properties that allow them to determine, for instance, that the size, weight, area or volume remain the same when an object holding such properties undergoes a process of change.

From a semantic perspective, the form of conservation principles is similar to the semantic representation of natural language quantifiers; the word *every*, for instance, can be represented as the function $\lambda P \lambda Q \lambda x.P(x) \subseteq Q(x)$ and a quantified sentence, like *every man is mortal*, can be represented through the application of the function representing the quantifier to the corresponding predicates as follows: $\lambda P \lambda Q \lambda x.P(x) \subseteq Q(x)(man)(mortal)$; this functional application renders the first-order function $\lambda x.man(x) \subseteq mortal(x)$. In this analogy, the conservation relation of the quantifier is set inclusion instead of the equality in the conservation principle in (1), and the preserved property is truth, instead of area in the same principle. Following this analogy, natural language quantifiers can be thought of as conservation principles too; this is, in the same way that the application of the principle of conservation of area is allowed only if the diagram is transformed by an area preserving action scheme, the application $\lambda Q \lambda x.man(x) \subseteq Q(x)(mortal)$ is granted only if this operation is truth preserving (i.e. if all men are indeed mortal).

Action schemes, in turn, can be thought of as knowledge structures causally related to actions; in Piaget's mental development theory, for instance, the realization that a property is preserved through a conservation principle needs to be explained in terms of an action scheme that matches the change process. The generative or synthetic aspect of action schemes also has an antecedent in this theory. Piaget also introduced the notion of "schemes of intuition" through which children are able to explore the space by manipulating forms while playing; schemes of intuition are first directed to the external world, and have a global or holistic character governed by perception, although these are later "interiorized" and can be detached from the external experience. In a later stage, which Piaget calls "structured intuition", action schemes are still governed by perception, but the child is now able to analyze the internal relations between the parts of the objects, and focus his or her attention and coordinate his or her actions in relation to the objects manipulated through the scheme, creating in this way a play space where he or she is able to discover the form and function of objects. The size of this design space would depend on the available intuition schemes, and the ways these can be put together. From a computational perspective, the schemes of intuition can be thought of as productions involving a number of actions, so when a scheme is used, all of its actions are performed as a unit, and the action has a global or holistic character; it is in this sense that the action schemes in the present theory are "encapsulated". However, when several schemes are involved in a sequence of actions (i.e. structured intuition), inference is required for selecting the next scheme to be applied, and also the focus object for the new action, and the process is no longer a whole, as a sequence of actions can be analyzed in terms of its constituent parts (i.e. the basic schemes of intuition and the way in which these are combined). In the present theory, the synthesis of geometric shapes depends on an inference of this kind, and the application of an action scheme depends on a geometric context, and also on the selection of a focus of attention.

The systematic application of action schemes for the production of new shapes has also been the subject of a large body of work in design studies; in particular, the production of design patterns made out of compositions of a given set of basic "tiles" or shape-forms by transformation operations (i.e. translations, rotations, symmetrical operations) based on a reference object within the pattern, has a long tradition in design [9]; from the computational perspective, Stiny's shape grammar formalism for the definition of design families or styles [49] was based on this intuition; in particular, a simple but interesting shape grammar of right triangles that generates the overt shapes (i.e. as opposed to the shapes that emerge from the diagrams) that are used in our first case study was developed by Weissman-Knight [54].

In summary, action schemes have several motivations related to the synthesis of new objects; finally, there is an analogy between geometric design, the synthesis of geometric concepts, and diagrammatic theorem-proving, as all of these domains involve the definition of a problem space of form in which the generation of geometric shapes that are the subject of a number of constraints must be explored. The nature and size of the problem space will be determined by the number and type of "seeds" that generate a family of shapes and also by of the available productions that build composite shapes out of basic ones, and by the different ways the productions can be applied in a generation sequence.

9. Discussion

In this paper a theory of the synthesis of some geometric and arithmetic concepts was presented. The theory rests on the definition of conservation principles and action schemes; conservation principles can be thought of as generalized concepts of equality that, in conjunction with action schemes and a referential machinery, which is in turn based on the geometric descriptor and the functional abstractor, permit the synthesis of first order equalities that correspond to ordinary geometric and arithmetic concepts. Action schemes, in addition, allow the interpretation of external representations through perceptual inference, and the generation of diagrammatic actions through motor schemes, and these processes can be thought of as particular forms of schematic thought. The present machinery may also be effective in synthesis of logical concepts, such as the concepts associated with Venn or Euler diagrams, or the concept that is constructed through syllogistic reasoning in mental models, and we left the study of the synthesis of such kinds of concepts for further work.

The concepts produced through diagrammatic derivations can be involved in further logical inferences too; for instance, although most Euclidean proofs involve synthetic processes, they also involve logical relations, like transitivity, which is asserted as an axiom (i.e. the first "common notion" [21]). This knowledge can also be used in conjunction with other kinds of reasoning modes, and it would be interesting to explore the nature and interaction of diverse inferential schemes through the construction of a diagrammatic reasoning machine able to combine the synthesis of concepts with logical deduction, abduction, and also simulation and constraint satisfaction; an interesting question is whether such a machine could be used to model, for instance, the proofs included in Euclid's Elements.

Another consideration is that the inferential machinery developed in this paper has a constructive character, and synthetic concepts have a non-empty extension; however, the addition of logical deduction could also be effective in modeling proofs by contradiction (e.g. Proposition 7 in the Elements [21]). In summary, the inferential machinery presented in this paper is one, although we think important, among a potentially large set of inferential strategies employed in diagrammatic inference. The identification of such a set may have very interesting theoretical and practical consequences.

Independently of the range of problems that can be addressed with the theory, the present representational and inferential machinery permits the investigation of some properties of diagrammatic representations more generally, and we conclude this paper with a reflection, in the light of this theory, on three open questions: what is the expressive power of diagrams, why diagrams are so effective for communication and inference, and what is the relation between synthetic inferences involved in the generation of concepts and valid inferences involved in their proofs.

9.1. Graphical abstraction

A common view on the expressive power of diagrammatic representations and graphics in general is that this kind of representations are "effective" for communication, reasoning and problem-solving because they limit abstraction and facilitate concrete thinking, while propositional languages (formal and natural) permit full abstractions, although at the expense of "effectiveness", as abstract thought is harder for most people. This position is explicitly developed in the theory of graphical specificity [48]; according to this theory there are representations that require that a certain amount of information is specified in a system concretely, in contrast to linguistic systems of representation that allow the expression of arbitrary abstractions. The theory characterizes three kinds of representational systems according to the restriction on abstraction that is enforced, and reciprocally, on the amount of concrete information that is directly interpretable. These are minimal, limited and unlimited abstraction representational systems (MARS, LARS and UARS, respectively). The theory of graphical specificity assumes that graphical representations can be thought of as expressions of a well-formed language, and can be interpreted in relation to a model, in the model-theoretic sense. In MARS, expressions representing states of affairs in the world are satisfied by a single model in the intended interpretation, but LARS and UARS can have several models. MARS can never be ambiguous or contain symbols standing for incomplete information, and hence are "concrete representations". The theory of graphical specificity introduces the notion of representational key; this consists of the knowledge that has to be made explicit to the users of a representation so they can understand it. The difference between MARS, LARS and UARS depends on the structure of the representational key; in the case of MARS, representational terms have a denotation that is fully determined, but in the case of LARS, the interpretation of a symbol or a composite term can have a limited number of possible interpretations and the representation expresses a limited abstraction. The theory distinguishes two kinds of representational keys: terminological and assertional. The statement "x stands for either one or two" establishes the values a representational object can have, independently of the values taken by other representational structures: then it is a terminological key, and is the only kind of key permitted in LARS. Assertional keys, on the other hand, permit the expression of arbitrary relations within elements of representational structures; assertional keys are illustrated in this theory with the following statement: "x is 1 if the value of y is equal to the value of z". Whenever assertional keys are present there is no limit to the abstraction that can be expressed, and hence the system is a UARS.

In contrast with the view of graphical specificity, the present theory shows that diagrams can express full abstractions, and that under the appropriate interpretation conditions, a diagrammatic sequence can express a theorem and its proof. External representations, whether these are graphical or linguistic, are concrete distinctions on the physical medium (e.g. marks on a piece of paper), and whether they denote concrete or abstract objects or concepts depends on the interpretation; an expression has a concrete interpretation if its denotation is a concrete individual, regardless whether this is a concrete object in the space of immediate experience or a specific dot or line in the geometric domain; if the interpretation is a function defined relative to some specific objects, as is the representational key exemplifying LARS above, the diagram expresses a limited abstraction; however, if the interpretation is a function whose value depends on no specific object, the diagram expresses a full abstraction. Assertional keys are unrestricted functions and, according to the theory of graphical specificity, diagrams cannot be interpreted in terms of such a kind of representational keys. However, as has been shown, the interpretation of diagrams and diagrammatic sequences can be stated in terms of geometric descriptions built with the geometric abstractor operator and these descriptions can be unrestricted functions, that is assertional keys. Furthermore, geometric concepts are full abstractions synthesized through the synthetic process, and this process can be seen as the generation of a novel assertional key from other given assertional keys through the application of conservation principles and action schemes. Accordingly, diagrams can be interpreted in terms of assertional keys, and it is under this interpretation that diagrams represent theorems and their proofs.

The present notion of interpretation is not in opposition to the fact that the concrete properties of the representation are essential in perception. Perceptual processes do act on concrete stimuli, and concrete information facilitates processing. A triangle needs to have three sides, and a right triangle a right angle, and these properties are identified by perception very effectively; in the present theory, geometric predicates in the geometric signature refer to specific geometric properties and relations, and the interpretation of these predicates has a concrete character; that is, there is a specific geometric algorithm associated with each predicate in the signature, and the interpretation of these algorithms is embedded within the interpretation of geometric expressions. The representational key, on the other hand, is the knowledge through which symbols, properties and relations of the representation are related to the interpretation domain. Whether a representation is a MARS, LARS of UARS depends on the nature of the representational key, and this notion is orthogonal to the specific geometric interpretation of geometric descriptions.

Children at the stage of concrete thinking may be able to realize that the area of the top-right diagram in Fig. 3 and the area of the lower-right diagram are the same; they may even be able to realize that the area of the hypotenuse in the top-right diagram is the same as the sum of the areas of the two squares on the right sides of the last diagram, for this particular diagrammatic sequence, provided that they are able to recognize the emerging shapes; but children at this age, or adults looking at the diagram from a concrete perspective, are unable to see that the relation holds not only for this particular diagram, but also for any diagrammatic sequence generated from any right triangle, regardless of its position, size and orientation; in order to see the diagrammatic sequence as a theorem and its proof, the interpreter must possess the capacity and attitude to see the external representation as a general class, and for this, the human interpreter must realize an assertional key and look at the diagrammatic sequence with this representational key in mind. Furthermore, most people cannot avoid this last abstract interpretation once they have recognized the relevant emerging objects, and that the relevant part of the proof is area preserving; this is, when they have recognized the relevant relations, they see diagrammatic sequences as valid proofs.

In summary, the interpretation and reinterpretation of diagrams is a perceptual inference; this process takes as its input the extensional representation of the diagram, which has been equated with the external diagram itself, and produces as its output the intended interpretation of the expression in relation to the current task and context, and also in relation the capabilities and attitude of the interpreter, and this interpretation can be concrete or abstract. If a graphical symbol stands for a class of objects then such a symbol functions as a generic object in which the concrete object stands for the class. If there is a restriction on the abstraction that can be expressed by an external representation, this is a limitation of the perceptual device that produces its interpretation, and of the expressive power of the language in which this interpretation is represented and used for thought and action, but not a limitation of the expressive power of the external representation itself. In the absence of an interpretation process it is meaningless to talk about the expressive power of an external representation. If a diagram has a concrete interpretation, this can be expressed in the language of propositional logic, but nothing prevents us from expressing abstract interpretations of graphics and diagrams in a language with variables, the functional abstractor and the geometric descriptor. The basic visual task consists in finding an extensional representation of concrete objects in the visual field, but perception is a process that places a scene in terms of the knowledge, beliefs, desires and intentions of the agent, in a way that is also relevant in the interpretation context, and its output can be an individual concept, as is often the case, but the product of perception can also be a general concept. In particular, if the object of perception is an external representation, either linguistic or diagrammatic, the output of this process is often a general concept.

9.2. Effective interpretation of graphics

The theory of graphical specificity holds that graphical representations can be interpreted effectively by people because the restrictions on abstraction correspond to the concrete information that can be interpreted directly. As MARS can be computed effectively they are also compared with Levesque's vivid knowledge-bases [28] which contain only ground function-free atomic sentences, use the unique names convention and implement the closed-world assumption and the axioms of equality. Vivid knowledge-bases are computationally tractable, as they express information in a complete and unambiguous manner. If the expressiveness of vivid knowledge-bases is increased by allowing disjunction and subsumption, they are roughly equivalent to LARS, as they permit a restricted form of abstraction. The theory of graphical specificity is also closely related to the so-called knowledge representation trade-off [29] according to which there is a compromise between the expressive power of a representation language and the computational cost of making inferences in such a language (i.e. its tractability); subsequently, reasoning with diagrams is easy but only limited abstractions can be expressed, while natural or formal languages permit the expression of full abstractions, but at much higher computational cost. The view is also related to formal language theory: regular expressions have a limited expressive power but accepting a string can be done with limited computational power while grammars are much more expressive, but decision problems have a much higher computational cost in this latter formalism.

However, these views contrast with the intuition that the solution of a complex problem may be very difficult and even impossible with concrete information, but when the appropriate abstractions are in place, the solution can often be read off from the representation directly. More generally, abstraction is required to solve complex problems, and computational resources (memory and computing time) run out very quickly precisely when the solution of a problem that demands abstraction is attempted with a concrete representation. The availability of variables, quantifiers and the functional abstractor in first and higher-order logics permit the expression of richer abstractions than propositional logic, and variables and productions augment the expressive power of grammars in relation to regular expressions. However, the opposition between "the concrete" and "the abstract" is not only about the expressive power of different representation languages: it is also a question about how effectively the expressive power available within a particular representational system is used, and the same language can be used to express things in both a concrete form or through an abstraction; for instance, the regular expression $(a + b + \Lambda)^n$ is interpreted as the language including all strings of a's and b's of length n or less, but this language can also be expressed as the explicit disjunction of all such strings in the same formalism (i.e. $\Lambda + a + b + ab + ba + \cdots + b_1 \cdots b_n$); the former regular expression expresses the information in a compact and direct way (i.e. an abstraction), while the second is a concrete representation in which all instances of the problem are mentioned explicitly (i.e. 3^n terms in the disjunction!), with the subsequent memory and computational cost; what is expensive is to find the appropriate abstraction, but this cost is greatly compensated at the inferential level as reasoning with appropriate abstractions in mind has a very low computational cost.

So, one of the reasons why diagrammatic concepts and proofs can be interpreted very effectively is because the diagrams express the right abstractions directly. In the same way that the abstraction is launched from a finite representation in model based approaches (e.g. Johnson-Laird's mental models), the concrete diagram is the platform for launching the abstraction in the present theory. Finding these abstractions may be very difficult, as was argued in relation to the production of abstract descriptions through perceptual inferences; however, when the abstractions are found, they are used very effectively in synthetic thought processes, as illustrated by the diagrammatic derivations.

Graphical abstractions also facilitate reasoning about change; in diagrammatic derivations the construction of synthetic knowledge is monotonic, simplifying greatly the control of the inferential task, and reducing its computational cost. In particular, despite the fact that the synthetic process requires reasoning about a diagrammatic sequence in which not only the properties but also the ontology changes from diagram to diagram or diagrammatic interpretation state, the frame problem of cognitive science and AI simply does not arise. In a manner consistent with Lindsay's observations [30], predicates and functions of the geometric representation language compute properties and relations of diagrams through their related algorithms in each diagrammatic state, minimizing the effects of the change; also intensional definitions like (2) help to control the change in diagrammatic sequences. However, in addition to these mechanisms, the application of conservation principles accumulates knowledge about the interpretation of different diagrams or diagrammatic states monotonically, and the relevant part of the diagrammatic derivation is read off as a single proposition, and there is no change to account for.

A natural language semantics analogy can help in seeing this latter point. As was mentioned, the sentence *every man is mortal* is represented through the application of the quantifier to the predicates, $\lambda P \lambda Q \lambda x.P(x) \subseteq Q(x)(man)(mortal)$, and renders the function $\lambda x.man(x) \subseteq mortal(x)$. That is, during the incremental interpretation of the sentence *every man is mortal* there is no change to account for, as this is a single proposition; similarly, in the incremental construction of the geometric concept the diagrammatic sequence is interpreted as a single proposition and the theorem is read off out of the diagram, and there is no change to account for. More generally, the conservation principles and geometric descriptions involved in the derivation are very powerful abstractions, and these abstractions account for the change implicitly. This view of change contrasts somehow with traditional approaches to the representation of change in AI (e.g. through frame axioms and the frame problem), in that instead of aiming for establishing all properties in the world or a situation that holds after a process of change, or assuming that all the properties of the world remain the same unless they are altered as a consequence of the change process, conservation principles focus on whether a single property is preserved in a change process, and the change itself is understood in terms of an action scheme that matches the change in some relevant dimension.

An abstraction is an expression of the representation language that is compact enough to be processed as a unit, but its interpretation is a general class of objects or relations; what makes the interpretation of diagrams effective is that once the right abstractions have been found people are able to use these objects for thought and action very easily, and this representational format is a very good compromise in the knowledge representation tradeoff.

9.3. Synthetic and analytic knowledge

The main product of the present work is the synthesis of functions in the geometric domain, and these functions have been equated with geometric concepts; it has also been shown that some arithmetic theorems are constructed from a synthetic geometric concept and an appropriate representational mapping; in the present discussion, an inference with the effect of producing these kinds of concepts, both geometric and arithmetic, is referred to as *synthetic*. As synthetic concepts are not known beforehand, these kinds of truths are *a posteriori*, but nevertheless, as was shown by the application of conservation principles and action schemes, necessarily true. Synthetic inferences can be contrasted with deductive inferences that permit the realization of implicit truths (i.e. theorems) from the set of axioms of a logical theory and a set of valid inference schemes; in this latter case axioms are *a priori* truths, and theorems are proved rather than synthesized; here, an inference to this latter effect is referred to as *analytic*.

Although the main goal of the synthetic process is to find a function that represents a concept, and hence, to enrich the knowledge of the interpreter, the synthetic and the analytical are potentially alternative ways to prove theorems; in the analytic case the theorem is usually known, and its proof is found in a problem space through a valid inference. In the synthetic case, on the other hand, the proof depends on a construction from an underlying source of knowledge. The first is the subject mater of theorem-proving, but the second is a case of structural learning. The study of arithmetic theorems that have a diagrammatic representation has been addressed in AI from the analytic perspective, as was mentioned above, and symbolic theorem-proving methods developed for the study of inductive theorems have been applied to the diagrammatic case. However, the study of geometric and arithmetic concepts from the synthetic perspective is a contribution of the present theory.

The distinction between the synthetic and the analytic can be further discussed with the Theorem of Pythagoras which was shown to be a synthetic concept; in principle, the theorem could be "proved" rather than synthesized, and an inference to such an effect would be analytic. However, a genuinely analytic proof would require a formalization of the geometry through a set of axioms, and the theorem would have to be derived from those basic truths by the systematic application of valid inference schemes, and it would not depend on a given diagram. A diagram can be used as an aid in the proof process as it was the case in Gelernter's *GTP* which, on the basis of a geometric theory (i.e. a set of geometric axioms), was able to prove several simple geometric theorems through the application of valid inference rules, and these proofs were analytical or logical in this sense; however, the diagram had no proper or formal role in the proof and it was used as a heuristic device to guide the search process only, and the proof itself was rendered as a logical derivation from the set of axioms to the theorem.

Whether it is possible to design an AI program able to prove the Theorem of Pythagoras or similar geometric theorems using a purely deductive method, in the tradition started by Gelernter, is an open question; however, as the kind of proofs in Euclidean geometry often involve the generation of a diagrammatic sequence and reinterpretations, this does not seem to be a straightforward exercise; for an additional illustration, consider the proof of the Theorem of Pythagoras in proposition 47 of book 1 of Euclid's Elements [Heath, 1956, p. 349], which is reproduced in Fig. 12. This proof involves the generation of the three squares on the sides of a seed right triangle, and the generation of a vertical line that has the vertex at the right angle of the right triangle as its pivot; the theorem follows from the fact that the rectangle that emerges from the square on the hypotenuse at the left side of the vertical line has the same area as the

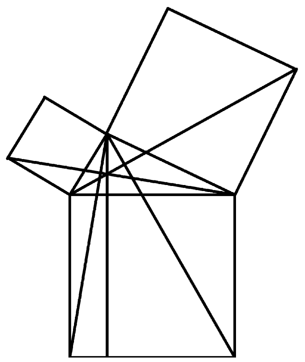
Fig. 12. Euclidean proof of the Theorem of Pythagoras.

square on the leftmost right side of seed right triangle, and similarly for the corresponding rectangle and square on the right side of the diagram. This equality between areas is also realized through a further generation and reinterpretation process; in this new generation step a line from the vertex at the right angle of the right triangle to the bottom-left corner of the square on the hypotenuse, and also a line from the leftmost corner of the square on the leftmost right side of the right triangle to the top-right corner of the square on the hypotenuse, are generated in the diagram. In the new diagrammatic state two new equivalent triangles emerge from the diagram (among other figures), each of these with one side on the leftmost square, and also a side on the square on the hypotenuse. The equivalence between the areas holds because the area of these two triangles is half the area of the rectangle and the square on which their bases lie respectively, and consequently the rectangle and the square have the same areas themselves. This is the case because the base and height of each of these two triangles are the same as the base and height of the corresponding rectangle and square. A similar relation holds for the right side of the diagram.

In the traditional terminology this proof is said to be "axiomatic", despite the fact that the diagram is already given, that the axioms are not stated explicitly, and the propositions mentioned above are the product of the "inference steps"; however, this would not count as an axiomatic proof in the proof-theoretical sense, as for this the axioms and inference rules would have to available in advance, and would have to be applied systematically without taking into account the interpretation of the symbols involved in the derivation, nor the reinterpretations (a notion that may not be even meaningful in formal logical derivations), and it is this formalization that seems problematic. Also, the questions of where the diagram comes from in the first place and how the concrete diagram is promoted into a general class, would still remain unanswered.

The difference between the synthetic and analytic ways of proving a theorem can also be appreciated in the case of the theorem of the sum of the odds. This arithmetic theorem was realized from an underlying geometric synthetic concept and an appropriate interpretation mapping, as was shown above, but it can also be proved by mathematical induction. The geometric concept is a synthetic truth based on the conservation principles and action schemes, but in this latter case, and unlike the arithmetic expression of the Pythagorean theorem, its corresponding arithmetic concept is also a theorem, and it holds independently of the mappings used in the synthetic process. The reason for this result is that the arithmetic concept was derived from a stronger concept of equality in which the terms related by this latter higher-order function are bound by a focus parameter, and the equality concept itself is also a conservation principle in which the conservation relation is equality and the conservation property is truth. Consequently, concepts derived from this arithmetic conservation principle are necessarily true. This synthetic proof is clearly very different from the proof of this theorem by mathematical induction, which has a solid axiomatic foundation in the arithmetic, and does not involve synthetic concepts or reinterpretations.

In summary, the distinction between the synthetic and the analytic demarcates the domain of logical questions, that can be answered through analysis, from questions about learning, that depend on constructive processes. Learning induction has traditionally been concerned with the synthesis of generalizations from a number of concrete instances of given phenomena; however, the realization of a geometric concept depends on the observation that some structural relations hold necessarily, and this knowledge can be gathered from a single instance of an object that exhibits such structural relations. In the present theory, a kind of structural learning has been addressed, as the primary product of the synthetic process is a geometric concept which is not known in advance and cannot be found through analysis. If this process is applied for the first time, the product is a genuine discovery.

Acknowledgements

I would like to thank B. Chandrasekaran and Ivan Bratko for lively discussions and their hospitality; John Lee for useful comments, long term collaboration and careful proof-reading; Phil Johnson-Laird for useful comments and encouragement; also Kees van Deemter, Rafael Morales, Ivan Meza and Yann Frauel for useful comments; also the anonymous reviewers of this and previous versions of the present paper. I also wish to thank Ewan Klein, Henk Zeevat, Dejuan Wang, Aart Bijl and Keith Stenning, at the University of Edinburgh (at the time), and to Gaby Garza, at Instituto de Investigaciones Eléctricas, México, who collaborated or commented on my previous work in diagrammatic reasoning. I also thank Paty Pérez Pavón for her assistance in the production of the Latex files. I also acknowledge the support of CONACyT and PAPIIT, UNAM. Finally, I dedicate this paper to my family who had to endure the evolution of the present theory over the years: to my wife Nydia, who also handed me Piaget's little book at the right time, and my daughters Nydia, Lilián and Alejandra: "Daddy... Pythagoras again? Oh no!!!"

Appendix A. Geometric signature

```
%Basic geometrical constructors
op_g(dot,[bool,real_pair],dot).
op_g(line,[bool,dot,dot],line).
op_g(path,[bool,list_of_dots],path).
op_g(right_triang, [bool,dot,dot,dot],
          right_triang).
op_g(triang,[bool,dot,dot,dot],triang).
op_g(square,[bool,dot,dot,dot,dot],
          square).
op_g(rectangle,[bool,dot,dot,dot,dot],
          rectangle).
op_g(parallelogram,[bool,dot,dot,dot,dot],
          parallelogram).
op_g(polygon,[bool,list_of_dots],polygon).
op_g(square_pair,[bool,square,square],
          square_pair).
op_g(corner_frame,[bool,dot,real],corner_frame).

%Logical constants
op_g(and,[bool,bool,bool],bool).
op_g(or,[bool,bool,bool],bool).
op_g(if,[bool,bool,bool],bool).
op_g(iff,[bool,bool,bool],bool).
op_g(not,[bool,bool],bool).

%Arithmetic
op_g(add,[bool,real,real],real).
op_g(dif,[bool,real,real],real).
op_g(mult,[bool,real,real],real).
op_g(add_real_pair,[bool,real_pair,real_pair],
          real_pair).

%type predicates
%for all geometrical sorts
op_g(is_sort,[bool,sort],bool).

%Equality predicates
%for all geometrical sorts including
%real (e.g. angles, distances) and
%real_pair (e.g. positions)
op_g(eq,[bool,sort,sort],bool).

%Geometric transformations
%for all geometrical sorts

%Translation
op_g(translate,[bool,sort,real_pair],sort).

%Rotation in relation to the origin
op_g(rotate,[bool,dot,real],dot).

% Rotation in relation to a pivot dot
op_g(rotate,[bool,sort,real,dot],sort).

%Focal symmetry
op_g(symmetrical,[bool,sort,dot],sort).

%Axial symmetry
op_g(symmetrical,[bool,sort,line],sort).

%Geometrical selectors

%Dot
op_g(position,[bool,dot],real_pair).

%Line
op_g(origen,[bool,line],dot).
op_g(end,[bool,line],dot).
op_g(length,[bool,line],real).
op_g(angle,[bool,line],real).

%Path
op_g(origen,[bool,path],dot).
op_g(end,[bool,path],dot).
op_g(num_segments,[bool,path],int).
op_g(segment,[bool,path,int],line).
op_g(length,[bool,path],real).

%Right-triangle
op_g(side_A,[bool,right_triang],line).
op_g(side_B,[bool,right_triang],line).
op_g(hypotenuse,[bool,right_triang],line).
op_g(angle_AH,[bool,right_triang],real).
op_g(angle_BH,[bool,right_triang],real).
op_g(right_vertex,[bool,right_triang],dot).
op_g(vertex_A,[bool,right_triang],dot).
op_g(vertex_B,[bool,right_triang],dot).
op_g(base,[bool,right_triang],real).
op_g(height,[bool,right_triang],real).
op_g(area,[bool,right_triang],real).

%Triangle
op_g(side_A,[bool,triang],line).
op_g(side_B,[bool,triang],line).
op_g(side_C,[bool,triang],line).
op_g(vertex_A,[bool,triang],dot).
op_g(vertex_B,[bool,triang],dot).
op_g(vertex_C,[bool,triang],dot).
op_g(angle_A,[bool,triang],line).
op_g(angle_B,[bool,triang],line).
op_g(angle_C,[bool,triang],line).
op_g(base,[bool,triang],real).
op_g(height,[bool,triang],real).
op_gf(area,[bool,triang],real).

%Square:
op_g(side_A,[bool,square],line).
op_g(side_B,[bool,square],line).
op_g(side_C,[bool,square],line).
op_g(side_D,[bool,square],line).
op_g(size,[bool,square],real).
op_g(area,[bool,square],real).
op_g(position,[bool,square],real_pair).
op_g(bottom_left_vertex,[bool,square],dot).
op_g(top_right_vertex,[bool,square],dot).

%Square constructors
%At the origin and aligned to the axis
op_g(square,[bool,real],square).

%At its Bottom-Left corner (horizontal)
op_g(square,[bool,dot,real],square).

%Square_pair
op_g(union,[bool,square,square],square_pair).
op_g(area,[bool,square_pair],real).

%Rectangle
op_g(side_A,[bool,rectangle],line).
op_g(side_B,[bool,rectangle],line).
op_g(side_C,[bool,rectangle],line).
op_g(side_D,[bool,rectangle],line).
op_g(base,[bool,rectangle],real).
op_g(height,[bool,rectangle],real).
op_g(area,[bool,rectangle],real).

%Parallelogram
op_g(side_A,[bool,parallelogram],line).
op_g(side_B,[bool,parallelogram],line).
op_g(side_C,[bool,parallelogram],line).
op_g(side_D,[bool,parallelogram],line).
op_g(base,[bool,parallelogram],real).
op_g(height,[bool,parallelogram],real).
op_g(area,[bool,parallelogram],real).

%Polygon
op_g(area,[bool,polygon],real).
op_g(segment,[bool,polygon,int],line).

%Corner-frame
op_g(pivot,[bool,corner_frame],dot).
op_g(position,[bool,corner_frame],real_pair).
op_g(size,[bool,corner_frame],real).
op_g(area,[bool,corner_frame],real).
op_g(union,[bool,square,corner_frame],square).

%Geometrical Predicates

%For dots
op_g(up,[bool,dot,dot],bool).
op_g(down,[bool,dot,dot],bool).
op_g(right,[bool,dot,dot],bool).
op_g(left,[bool,dot,dot],bool).

%For dot and figures
op_g(on,[bool,dot,line],bool).
op_g(on_project,[bool,dot,line],bool).
op_g(on_path,[bool,dot,path],bool).
op_g(origen,[bool,dot,line],bool).
op_g(origen_path,[bool,dot,path],bool).
op_g(end,[bool,dot,line],bool).
op_g(end_path,[bool,dot,path],bool).
op_g(in,[bool,dot,polygon],bool).

%For lines
op_g(horizontal,[bool,line],bool).
op_g(vertical,[bool,line],bool).
op_g(parallel,[bool,line,line],bool).
op_g(perpendicular,[bool,line,line],bool).
op_g(colineal,[bool,line,line],bool).
op_g(on,[bool,line,line],bool).
op_g(intersect,[bool,line,line],bool).

%For lines and figures
op_g(side,[bool,line,right_triang],bool).
op_g(side,[bool,line,triang],bool).
op_g(side,[bool,line,square],bool).
op_g(side,[bool,line,rectangle],bool).

%For right triangles
%Aligned by the hypotenuses
op_g(aligned_HH,[bool,right_triang,
        right_triang],bool).

%Aligned by Hyp. and Right-side
op_g(aligned_H_LR,[bool,right_triang,
        right_triang],bool).

%Aligned by right sides and right-vertex
op_g(aligned_AB_CRV,[bool,right_triang,
        right_triang],bool).

%Aligned by right-sides and a vertex
%(which is not the right vertex)
op_g(aligned_AB_NCRV,[bool,right_triang,
        right_triang],bool).

%Pythagorean triplet: A, B, HYP:
op_g(is_right_triang,[bool,real,real,real],
        bool).

%For polygons
op_g(in,[bool,polygon,polygon],bool).
op_g(intersect,[bool,polygon,polygon],bool).

%Geometrical functions

%For dots
op_g(distance,[bool,dot,dot],real).
op_g(distance,[bool,dot,line],real).
```

```
%IF pos(dot1) = pos(dot2)= pos
%Then pos(dot3) = pos
op_g(cons_dot,[bool,dot,dot],dot).

%For lines
op_g(angle,[bool,line,line],real).
op_g(line,[bool,dot,real],line).
op_g(middle_dot,[bool,line],dot).

%Intersection types
op_g(cross_at,[bool,line,line],dot).
op_g(t_join_at,[bool,line,line],dot).
op_g(e_join_at,[bool,line,line],dot).

%any actual or projective intersection
op_g(intersect_at,[bool,line,line],dot).

%For right-triangs
op_g(square,[bool,right_triang,
        right_triang],square).
op_g(rectangle,[bool,right_triang,
        right_triang],rectangle).
op_g(vertex_aligned_AB_NCRV,[bool,
        right_triang,right_triang],dot).

%RT constructor given triplet A, B, HYP
op_g(right_triang,[bool,real,real,real],
        right_triang).
```

Appendix B. Prolog's code of the geometrical interpreter

```
%Geometric operator
:- op(800,xfx,'<=').

%Interpretation of an expression in relation to
%a list of definitions of form 'g_def(ID, G_object)'
eval_g_exp(Exp, Base, Val) :-
        create_env(Base),
        eval_gterm(Exp, Val),
         abolish(g_def, 2).

eval_g_func(Func, Args, Base, Val) :-
        create_env(Base),
        eval_gterm(Func, Args, Val),
        abolish(g_def, 2).

%Main evaluation cycle

%The expression is a variable
eval_gterm(X, X) :- var(X).

%The expression is a boolean value
eval_gterm(X, X) :- X = true;
X = false.

%The expression is a real number
eval_gterm(X, X) :- number(X).

%The expression is an order pair
eval_gterm(X:Y, Val_X:Val_Y) :-
        eval_gterm(X, Val_X),
        eval_gterm(Y, Val_Y),
        number(Val_X),
        number(Val_Y).

%The expression is a constant in g_def
eval_gterm(ID, Val):-
        atom(ID),
        g_value(ID,Val).

%The expression is an control structure
eval_gterm(case(Cond, Exp1, Exp2), Val) :-
        eval_gterm(Cond, true),
        eval_gterm(Exp1, Val).
eval_gterm(case(Cond, Exp1, Exp2), Val) :-
        eval_gterm(Cond, false),
        eval_gterm(Exp2, Val).

%The expression is a function;
%Args is a list of arguments or a list of
%lists of arguments

%Base case: The function has no arguments
eval_gterm(lambda(Vars, Exp), lambda(Vars, Exp)).

%The function has an ID an is defined in g_def
eval_gterm(ID, Args, Val) :-
        atom(ID),
        eval_gterm(ID, Func),

        %Check that ID is a func. or a desc
        functional_object(Func),

        %Get new variable names
        assert(func(Func)),
        retract(func(NFunc)),

        %eval function
        eval_gterm(apply(NFunc, Args), Val).

functional_object(lambda(Vars, Exp)).
functional_object(T <= F) :-
        functional_object(F).

%Consume the last list of arguments, and eval Exp.
eval_gterm(lambda(Vars, Exp), [Args|[]], Val) :-
        bindings(Vars, Args, Exp),
        eval_gterm(Exp, Val).

%Consume argument and eval embedded function
eval_gterm(lambda(Vars, Func),[HArgs|TArgs], Val) :-
        bindings(Vars, HArgs, Func),
        eval_gterm(Func, TArgs, Val).
```

```
%Basic functional application: Args is a simple list
eval_gterm(lambda(Vars, Exp), Args, Val) :-
          bindings(Vars, Args, Exp),
          eval_gterm(Exp, Val).

%Bind arguments
bindings([],[],_).
bindings([HVar|TVars], [HArg|TArgs], Exp) :-
          HVar = HArg,
          bindings(TVars, TArgs, Exp).

%The expression is a geometrical description: base case
eval_gterm(T <= F, T <= F) :-
          eval_gterm(F, F).

%Eval description: return the value of T if F is true for Args
%otherwise the description is false
eval_gterm(T <= F, Args, Val_T) :-
          eval_gterm(F, Args, true),
          eval_gterm(T, Val_T).
eval_gterm(_<=_,_,false).

%Functional application: Exp is a func. or a description
eval_gterm(apply(Exp, Arg), Val) :-
          eval_gterm(Exp, Arg, Val).

%The expression is a term
eval_gterm(Exp, Val):-
          Exp =.. [OP|Args],
          reduce_gterm(OP, Args, Val).

%All arguments of a term must be well-defined
reduce_gterm(OP,[true|Geometric_Args], Value) :-
          op_g(OP,_,_),

          %Eval arguments
          reduce_g_args(Geometric_Args, Val_Args),

          append([true], Val_Args, Args),
          Basic_Exp =.. [OP|Args],

          %Evalua expression basica
          g_value(Basic_Exp, Value).

%Else, the term has no well-defined value and return:
%            op(false, Args)
%Case 1: One or more arguments have no well-defined
% value
%Case 2: The value of all arguments is true, but the term
% itself does not has a well defined value
reduce_gterm(OP, [Bool|Args], Val) :-
          op_g(OP,_,_),
          Val =.. [OP|[false|Args]],!.

%Reduce list of geometrical arguments
reduce_g_args([],[]).
reduce_g_args([H|T],Values) :-
          eval_gterm(H, Val),
          legal_value(Val),
          reduce_g_args(T, T_Vals),
          append([Val], T_Vals, Values),!.

%The argument is a lsit of terms (path & polygon)
reduce_g_args([Terms|_], [Values]) :-
          reduce_g_args(Terms, Values),!.

legal_value(X) :- X = true; X = false.
legal_value(X) :- number(X).
legal_value(X:Y) :- number(X),number(Y).
legal_value(VAL) :- VAL =.. [OP|[true|_]].
legal_value(VAL) :- VAL =.. [OP|[false|_]],fail.
```

Appendix C. Example of geometrical abstract data-type

```
%Clause g_value computes the geometric algorithm
%associated to each geometrical predicate and
%its form is: g_value(EXPRESSION, VALUE)

%Basic constructor for triangle abstract data-types
g_value(ID,TRIANG) :-
          atom(ID),
          %ID in geometrical def. data-base
          g_def(ID,DEF),
          DEF =.. [OP|_],
          op_g(OP,_,triang),
          eval_gterm(DEF,TRIANG),
          TRIANG = triang(true,D1,D2,D3).

g_value(triang(true,D1,D2,D3),
          triang(T,D1,D2,D3)) :-
                    triang(D1,D2,D3,T).

triang(D1,D2,D3,true) :-
          different_dots([D1,D2,D3]),
          not_aligned([D1,D2,D3]).

triang(D1,D2,D3,false) :-
          not(triang(D1,D2,D3,true)).

%Type predicate
g_value(is_triang(true,TRIANG),true) :-
          TRIANG =.. [F|[true|_]],
          %Right triangs are triangs
          (F = triang; F = right_triang).
g_value(is_triang(true,_),false).
```

```
%Equality predicate
g_value(eq(true,TRIANG1,TRIANG2),T) :-
        TRIANG1 =.. [F1|[true|DOTS1]],
        TRIANG2 =.. [F2|[true|DOTS2]],
        (F1 = triang ; F1 = right_triang),
        (F2 = triang ; F2 = right_triang),

         %The vertices are the directly
         %the same or after 1 or 2 shifts
         eq_figures(TRIANG1,TRIANG2,2,T).

%Main selectors for triang

%Sides
g_value(side_A(true,triang(true,D1,D2,D3)),
         line(true,D1,D2)).
g_value(side_B(true,triang(true,D1,D2,D3)),
         line(true,D2,D3)).
g_value(side_C(true,triang(true,D1,D2,D3)),
         line(true,D3,D1)).

%Vertices opposite to the corresponding side
g_value(vertex_A(true, triang(true,D1,D2,D3)),D3).
g_value(vertex_B(true, triang(true,D1,D2,D3)),D1).
g_value(vertex_C(true, triang(true,D1,D2,D3)),D2).

%Angle at vertex_A
g_value(angle_A(true,
         triang(true,D1,D2,D3)),IANG) :-
         eval_gterm(angle(true,
                      line(true,D3,D1),
                      line(true,D3,D2)),
                 ANG),
         %return interior angle
         int_ang(ANG,IANG).

%Angle at vertex_B
g_value(angle_B(true,
         triang(true,D1,D2,D3)),IANG):-
         eval_gterm(angle(true,
                      line(true,D1,D2),
                      line(true,D1,D3)),
                 ANG),
         %return interior angle
         int_ang(ANG,IANG).

%Angle at vertex_C
g_value(angle_C(true,
         triang(true,D1,D2,D3)),IANG) :-
         eval_gterm(angle(true,
                      line(true,D2,D1),
                      line(true,D2,D3)),
                 ANG),
         %return interior angle
         int_ang(ANG,IANG).
```

```
%Base
g_value(base(true,TRIANG),BASE) :-
        eval_gterm(is_triang(true,
                        TRIANG), true),
        eval_gterm(length(true,
            side_A(true,TRIANG)),BASE).

%Height
g_value(height(true,TRIANG),HEIGHT) :-
        eval_gterm(is_triang(true,
                        TRIANG),true),
        eval_gterm(vertex_B(true,TRIANG),VB),
        eval_gterm(vertex_A(true,TRIANG),VA),
        eval_gterm(side_A(true,TRIANG),BASE),
        eval_gterm(angle(true,BASE),ANG),
        pi(PI),
        PIM is PI/2,
        ANGH is ANG + PIM,
        eval_gterm(line(true,VB,ANG),HAXIS),
        eval_gterm(line(true,VA,ANGH),VAXIS),
        eval_gterm(length(true,
                  line(true,VA,
                  cross_at(true,HAXIS,
                           VAXIS))),
                  HEIGHT).

%Area
g_value(area(true,TRIANG),AREA) :-
        eval_gterm(is_triang(true,TRIANG), true),
        eval_gterm(base(true,TRIANG),BASE),
        eval_gterm(height(true,TRIANG),HEIGHT),
                  A1 is BASE * HEIGHT,
                  AREA is A1/2.

%axial symmetry for right_triang & triang
g_value(symmetrical(true,TRIANG,AXIS), SYM_TRIANG):-
        TRIANG =.. [F|[true,V1,V2,V3]],
        (F = right_triang ; F = triangle),
        eval_gterm(is_line(true,AXIS),true),
        eval_gterm(symmetrical(true,V1,AXIS),SV1),
        eval_gterm(symmetrical(true,V2,AXIS),SV2),
        eval_gterm(symmetrical(true,V3,AXIS), SV3),
        SYM_TRIANG =.. [F|[true,SV1,SV2,SV3]].
```

Appendix D. Prolog's code of the arithmetic interpreter

```
%Eval arithmetic expressions and functions

%Interpretation of a function in relation to
%a list of definitions 'Base' of form 'a_def(ID, Exp)'
eval_a_func(Exp, Arg, Base, Val) :-
        create_env(Base),
        a_eval(Exp, Arg, Val),
        abolish(a_def, 2).
```

```
%The expression is a number
a_eval(X, X) :- number(X).

%The expression is a truth value
a_eval(B, B) :- B = true; B = false.

%The expression is a constant defined in a_def
a_eval(ID, Val):-
        atom(ID),
        a_def(ID, Val).

%The expression is a control structure
a_eval(if(Cond, Exp1, Exp2), Val) :-
        a_eval(Cond, true),
        a_eval(Exp1, Val).

a_eval(if(Cond, Exp1, Exp2), Val) :-
        a_eval(Cond, false),
        a_eval(Exp2, Val).

%Eval basic expression
a_eval(Exp, Val) :-
        Exp =.. [OP|Arg_List],
        OP \== lambda,
        OP \== apply,
        reduce_args(Arg_List, RArgs),
        basic_value(OP, RArgs, Val).

reduce_args([], []).
reduce_args([H|T], RArgs) :-
        a_eval(H, Val_H),
        reduce_args(T, Vals_T),
        append([Val_H], Vals_T, RArgs).

%Transformation rule for functions embedded
within terms
basic_value(OP, [F1, F2], Func) :-
        F1 =.. [lambda|[L1, Exp1]],
        F2 =.. [lambda|[L2, Exp2]],
        Body =.. [OP|[Exp1, Exp2]],
        append(L1, L2, Arg_List),
        Func =.. [lambda|[Arg_List, Body]].

%Signature & semantics
basic_value(add, [X, Y], Val) :- Val is X + Y.
basic_value(dif, [X, Y], Val) :- Val is X - Y.
basic_value(mult,[X, Y], Val) :- Val is X * Y.
basic_value(sq, [X], Val) :- Val is X * X.
basic_value(eq, [X, Y], true) :- E is X – Y, error(E, 0).
basic_value(eq, [X, Y], false).

%Functions
%Base case: the value of a function is the function itself
a_eval(lambda(Vars, Exp), lambda(Vars, Exp)).

%Function ID defined as 'a_def(ID, lambda(Args, Exp))'
a_eval(ID, Args, Val) :-
        atom(ID),
        a_eval(ID, Func),
        Func =.. [lambda|_],

        %Get new variable names
        assert(func(Func)),
        retract(func(NFunc)),

        %eval function
        a_eval(apply(NFunc, Args), Val).

%Consume last arguments in list and eval
a_eval(lambda(Vars, Exp), [Args|[]], Val) :-
        bindings(Vars, Args, Exp),
        a_eval(Exp, Val).

%Consume argument list and return function
a_eval(lambda(Vars, Func), [HArgs|TArgs], Val) :-
        bindings(Vars, HArgs, Func),
        a_eval(Func, TArgs, Val).

%Basic function evaluation: Args is a simple list
a_eval(lambda(Vars, Exp), Args, Val) :-
        bindings(Vars, Args, Exp),
        a_eval(Exp, Val).

%Functional application
a_eval(apply(Func, Arg), Val) :-
        restructure_args(Func, Arg, New_Func),
        reduce_args(Arg, Val_Arg),
        a_eval(New_Func, Val_Arg, Val).

%Arguments restructuring for second transformation rule
restructure_args(Func_ID, _, Func_ID).
restructure_args(lambda(Args, Exp), Pars, lambda(Args, Exp)) :-
        length_list(Args, N),
        length_list(Pars, N).

restructure_args(lambda(Args, Exp), Pars,
              lambda([Z], New_Exp)) :-
        length_list(Pars, 1),

        %bind all variables in Args and replace them by Z
        substitute_vars(Args, Z, Exp, New_Exp).

%replace all variables in Args by Var
substitute_vars([], _, Exp, Exp).
substitute_vars([H|T], Var, Exp, New_Exp) :-
        change_var_exp(H, Var, Exp, Next_Exp),
        substitute_vars(T, Var, Next_Exp, New_Exp).

change_var_exp(Var, NVar, Exp, NExp) :-
        var(Exp),
        Var == Exp,
        NExp = NVar.
```

```
change_var_exp(Var, NVar, Exp, Exp) :-
        var(Exp).

change_var_exp(Var, NVar, Exp, NExp) :-
        nonvar(Exp),
        Exp =.. [Func|Args],
        change_var_args(Var, NVar, Args, NArgs),
        NExp =.. [Func|NArgs].

change_var_args(Var, NVar, [], []).
change_var_args(Var, NVar, [H|T], NArgs) :-
        change_var_exp(Var, NVar, H, NExp),
        change_var_args(Var, NVar, T, RExps),
        append([NExp], RExps, NArgs).
```

Appendix E. Definition of geometric and arithmetic objects

Notation:
A relation of form “g_def(g_id, g_object)” associates a geometrical object to an arbitrary g_id
A relation of form “a_def(a_id, a_object)” associates an arithmetic object to an arbitrary a_id

```
%Geometric concepts
%Principle of Conservation of area: Geometric concept in (10) in the text of the paper
g_def(pca, lambda([P], lambda([Q], lambda(X, eq(true, area(true, apply(P, X)),area(true, apply(Q, X))))))).

%Geometric descriptions
%description of a square on the hypotenuse of a right triangle: description (13) and concept (12)
g_def(sq_on_hyp, Y <= lambda([X], lambda([Y], and(true,is_right_triang(true,X),
                                   and(true,is_square(true,Y),side(true,hypotenuse(true,X),Y)))))).

%Description of the union of the squares on the right sides of a right triangle: description (15) and concept (14)
g_def(union_sqA_sqB, union(true,Y,Z) <= lambda([X], lambda([Y,Z],
                                   and(true,is_right_triang(true,X),
                                   and(true,is_square(true,Y),
                                   and(true,is_square(true,Z),
                                   or(true, and(true,side(true,side_A(true,X),Y), side(true,side_B(true,X),Z)),
                                            and(true,side(true,side_B(true,X),Y), side(true,side_A(true,X),Z))))))))).

%Union of a square at position P of size l & corner-frame of size l + 1 at position p + l:l. Description (33) and concept (32)
g_def(union_sq_cf, union(true, W, Z) <= lambda([P, L], lambda([W, Z],
                                   and(true, is_square(true, W),
                                   and(true, eq(true, size(true, W), L),
                                   and(true, eq(true, position(true, W), P),
                                   and(true, is_corner_frame(true, Z),
                                   and(true, eq(true, size(true, Z),add(true, L, 1)),
                                   and(true, case(eq(true, L, 0),
                                                  eq(true, position(true, Z), P),
                                                  eq(true, position(true, Z), add_real_pair(true, P, L:L))),
                                   true))))))))).

%Square of size L + 1 at position P: Description (30) and concept (28)
g_def(sq_l1, X <= lambda([P, L], lambda([X], and(true, is_square(true, X),
                                   and(true, eq(true, size(true, X), add(true, L, 1)), eq(true, position(true, X), P)))))).

%Arithmetic concepts
%Concept of arithmetic equality (Global): Concept (24)
a_def(global_arith_eq, lambda([M], lambda([N], eq(M, N)))).

%Concept of arithmetic equality (Structured): Concept (40)
a_def(structured_arith_eq, lambda([P], lambda([Q], lambda(N, eq(apply(P, N),apply(Q, N)))))).
```

```
%Functions representing the geometric and arithmetic inductions of the theorem of the sum of the odds
%Function representing the induction over sq's of size l - 1 & cf of size l to sq of size l: Concept (44)
g_def(g, lambda([P, L],
                    case(eq(true, L, 0),
                        square(true, dot(true, P), 0),
                        apply(union_sq_cf, [[P, dif(true, L, 1)],
                                [apply(g, [P, dif(true, L, 1)]),
                                corner_frame(true,
                                        dot(true,add_real_pair(true, P, dif(true, L, 1):dif(true, L, 1))),
                                        L)
                                ]])))).

%Function representing the theorem of the sum of the odds: Concept (47)
a_def(t, lambda(N, apply(lambda([Z], add(if(eq(Z, 0), 0, apply(t, [dif(Z,1)])), dif(mult(2, add(Z,1)),1))), N))).

%Examples of concrete geometrical objects to launch the synthetic process
%Arbitrary seed for the synthesis of the Theorem of Pythagoras
g_def(tr_seed, right_ triang(true,dot(true,0.4:0.4),dot(true,0.4:0.7),dot(true,0.6:0.4))).

%Arbitrary seeds for the theorem of the Sum of the Odds
g_def(sq_seed, square(true, dot(true, 2:2), 2)).
g_def(cf_seed, corner_frame(true, dot(true, 4:4), 3)).

%Examples of concrete geometrical objects to test the resulting concepts

%reference dots, right triangles and squares to test the Theorem of Pythagoras
g_def(d1, dot(true, 0.5:0.5)).
g_def(d2, dot(true, 0.5:0.9)).
g_def(d3, dot(true, 0.9:0.5)).
g_def(d4, dot(true, 1.3:0.9)).
g_def(d5, dot(true, 0.9:1.3)).
g_def(rt1, right_triang(true,d1,d2,d3)).
g_def(sq1, square(true,d1,d2,dot(true,0.1:0.9),dot(true,0.1:0.5))).
g_def(sq2, square(true,d1,d3,dot(true,0.9:0.1),dot(true,0.5:0.1))).
g_def(sq3, square(true,d2,d3,d4,d5)).

%Concrete objects to test the function representing the geometric concept underlying the theorem of the sum of the odds
g_def(sq4, square(true, dot(true, 3:4), 5)).
g_def(cf1, corner_frame(true, dot(true, 8:9), 6)).
g_def(sq5, square(true, dot(true, 3:4), 6)).
```

References

[1] M.A. Alberti, E. Bastioli, D. Marini, Towards object-oriented modeling of Euclidean geometry, The Visual Computer 11 (1995) 378–389.

[2] J.R. Anderson, C.F. Boyle, G. Yost, The geometry tutor, in: Proceedings of the International Joint Conference on Artificial Intelligence, Los Angeles, California, 1985, pp. 1–7.

[3] M. Anderson, R. McCartney, Diagram processing: Computing with diagrams, Artificial Intelligence 145 (2003) 181–226.

[4] J. Barwise, J. Etchemendy, Visual information and valid reasoning, CSLI report, Stanford, 1990.

[5] J. Barwise, J. Etchemendy, Hyperproof, CSLI Lecture Notes, vol. 42, CSLI Publications, Stanford, CA, 1994.

[6] J. Barwise, J. Etchemendy, Heterogeneous logic, in: J. Glasgow, N.H. Narayanan, B. Chandrasekaran (Eds.), Diagrammatic Reasoning, AAAI Press/The MIT Press, Menlo Park, CA, 1995, pp. 211–234.

[7] G.S. Boolos, R.C. Jeffrey, Computability and Logic, third ed., Cambridge University Press, Cambridge, 1990.

[8] A. Borning, The programming language aspects of THINGLAB, a constraint-oriented simulation laboratory, ACM Transactions on Programming Languages and Systems 3 (1981) 353–387.

[9] J. Bronowski, The Ascent of Man, BBC Corporation, London, 1973.

[10] B. Chandrasekaran, U. Kurup, B. Banerjee, J. Josephson, R. Winkler, An architecture for problem solving with diagrams, in: Proceedings of the Third International Conference, Diagrams 2004, in: Lecture Notes in Artificial Intelligence, vol. 2980, Springer-Verlag, Berlin, 2004, pp. 151–165.

[11] R. Dale, E. Reiter, Computational interpretations of Gricean maxims in the generation of referring expressions, Cognitive Science 18 (1995) 233–263.
[12] D.R. Dowty, R.E. Wall, S. Peters, Introduction to Montague Semantics, D. Reidel Publishing Company, 1985.
[13] N. Foo, M. Pagnuco, A. Nayak, Diagrammatic proofs, in: Proceedings of the 16th International Joint Conference on Artificial Intelligence, IJCAI-99, Morgan Kaufmann, Stockholm, 1999, pp. 378–383.
[14] B.V. Funt, Problem-solving with diagrammatic representations, Artificial Intelligence 13 (1980) 210–230.
[15] G. Garza, L.A. Pineda, Synthesis of solid models of polyhedra from their orthogonal views using logical representations, Expert Systems with Applications 14 (1) (1998) 91–108.
[16] H. Gelernter, Realization of a geometry-theorem proving machine, in: A. Feigenbaum, J. Feldman (Eds.), Computers and Thought, AAAI Press/The MIT Press, London, 1995, pp. 134–152 (first published in 1963 by McGraw-Hill).
[17] J. Glasgow, N.H. Narayanan, B. Chandrasekaran (Eds.), Diagrammatic Reasoning, AAAI Press/The MIT Press, Menlo Park, CA, 1995.
[18] J.A. Goguen, J.W. Thatcher, E.G. Wagner, An initial algebra approach to the specification, correctness and implementation of abstract data types, in: R.T. Yeh (Ed.), Current Trends in Programming Methodology, Prentice-Hall, 1978, pp. 80–149.
[19] J.G. Greeno, Forms of understanding in mathematical problem solving, in: S.G. Paris, G.M. Olson, H.W. Stevenson (Eds.), Learning and Motivation in the Classroom, Lawrence Erlbaum, Hillsdale, NJ, 1983, pp. 83–111.
[20] J.P. Hayes, Some problems and non-problems in representation theory, in: R. Brachman, H. Levesque (Eds.), Readings in Knowledge Representation, Morgan and Kaufmann, Los Altos, CA, 1985, pp. 3–22.
[21] T.L. Heath, The Thirteen Books of Euclid's Elements, translation, introduction and commentary by T.L. Heath, second ed., Dover Publications, Inc., New York, 1956.
[22] M. Jamnik, Automating diagrammatic proofs of arithmetic arguments Ph.D. dissertation, Department of AI, University of Edinburgh, 1999.
[23] P.N. Johnson-Laird, Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness, Cambridge University Press, Cambridge, 1983.
[24] T. Kamada, S. Kawai, A general framework for visualizing abstract objects and relations, ACM Transactions on Graphics 10 (1991) 1–39.
[25] E. Klein, L.A. Pineda, Semantics and graphical information, in: D. Diaper, et al. (Eds.), Proceedings of Human Computer Interaction INTERACT'90, Elsevier Science Publishers B.V./North-Holland, Amsterdam, 1990, pp. 485–491.
[26] K.R. Koedinger, J.R. Anderson, Abstract planning and perceptual chunks: Elements of expertise in geometry, Cognitive Science 14 (1990) 511–550.
[27] J.H. Larkin, H. Simon, Why a diagram is (sometimes) worth ten thousand words, Cognitive Science 11 (1987) 65–99.
[28] H.J. Levesque, Logic and the complexity of reasoning, Journal of Philosophical Logic 17 (1988) 355–389.
[29] H.J. Levesque, R. Brachman, A fundamental tradeoff in knowledge representation and reasoning, in: R. Brachman, H. Levesque (Eds.), Readings in Knowledge Representation, Morgan and Kaufmann, Los Altos, CA, 1985, pp. 41–70.
[30] R.K. Lindsay, Images and inference, Cognition 29 (1988) 229–250.
[31] R.K. Lindsay, Using diagrams to understand geometry, Computational Intelligence 14 (1998) 238–272.
[32] S. Matsuoka, S. Takahashi, T. Kamada, A. Yonezawa, A general framework for bidirectional translation between abstract and pictorial data, ACM Transactions on Information Systems 10 (1992) 408–437.
[33] T.F. McDougal, K.J. Hammond, Using diagrammatic features to index plans for geometry theorem-proving, in: J. Glasgow, N.H. Narayanan, B. Chandrasekaran (Eds.), Diagrammatic Reasoning, AAAI Press/The MIT Press, Menlo Park, CA, 1995, pp. 691–709.
[34] A.J. Nevins, Plane geometry theorem proving using forward chaining, Artificial Intelligence 6 (1975) 1–23.
[35] A. Paivio, Imagery and Verbal Processes, Lawrence Erlbaum Associates Inc., Publishers, Hillsdale, NJ, 1979.
[36] J. Piaget, Six Études de Psychologie, Barral Editores, S.A., Barcelona, 1970 (edition in Spanish).
[37] L.A. Pineda, E. Klein, J. Lee, Graflog: Understanding drawings through natural language, Computer Graphics Forum 7 (1988) 97–103.
[38] L.A. Pineda, N. Chater, Graflog: Programming with interactive graphics and Prolog, in: N. Magnenat-Thalmann, D. Thalmann (Eds.), New Trends in Computer Graphics, Proceedings of CG International'88, Springer-Verlag, Berlin, 1988, pp. 469–478.
[39] L.A. Pineda, Graflog: a theory of semantics for graphics with applications to human-computer interaction and CAD systems, Ph.D. dissertation, Centre for Cognitive Science, University of Edinburgh, 1989.
[40] L.A. Pineda, Reference, synthesis and constraint satisfaction, Computer Graphics Forum 11 (1992) 334–344.
[41] L.A. Pineda, On computational models of drafting and design, Design Studies 14 (1993) 124–156.
[42] L.A. Pineda, A model for multimodal representation and inference, in: R. Paton, I. Neilson (Eds.), Visual Representations and Interpretations, Springer-Verlag, London, 1999, pp. 375–386.
[43] L.A. Pineda, G. Garza, A model for multimodal reference resolution, Computational Linguistics 26 (2000) 136–192.
[44] R. Reiter, A.K. Mackworth, Logical framework for depiction and image interpretation, Artificial Intelligence 41 (1989) 125–155.
[45] M. Shanahan, Perception as abduction: Turning sensor data into meaningful representations, Cognitive Science 29 (2005) 103–134.
[46] S. Shin, The Logical Status of Diagrams, Cambridge University Press, Cambridge, 1995.
[47] A. Sloman, Afterthoughts on analogical representations, in: R. Brachman, H. Levesque (Eds.), Readings in Knowledge Representation, Morgan and Kaufmann, Los Altos, CA, 1985, pp. 431–440.
[48] K. Stenning, J. Oberlander, A cognitive theory of graphical and linguistic reasoning: Logic and implementation, Cognitive Science 19 (1995) 97–140.
[49] G. Stiny, Pictorical and Formal Aspects of Shape and Shape Grammars, Birkhauser Verlag, Basel, 1975.
[50] I. Sutherland, Sketchpad: A man-machine graphical communication system, in: AFIPS SJCC Proceedings, 1963, pp. 329–346.
[51] M. Tye, The Imagery Debate, A Bradford Book, MIT Press, 1991.
[52] K. Van Deemter, Generating referring expressions: Boolean extensions of the incremental algorithm, Computational Linguistics 28 (2002) 37–53.

[53] D. Wang, Studies on the formal semantics of pictures, Ph.D. dissertation, ILLC Dissertation Series 1995-4, Institute for Logic, Language and Communication, University of Amsterdam, 1995.
[54] T. Weissman-Knight, Languages of designs: from known to new, Environment and Planning B 8 (1981) 213–238.
[55] L. Wittgenstein, Philosophical Investigations, Basil Blackwell, Oxford, 1953.