

ABSTRACTION, VISUALISATION AND GRAPHICAL PROOF

Luis Pineda¹, John Lee² and Gabriela Garza

Abstract

In this paper we investigate the process of learning and verifying graphical theorems through abstraction and visualisation. First, the notion of effectiveness of a representation is discussed from both a computational and a cognitive perspective; the nature of the relation between external representations and abstraction in these two views, and its implications for diagrammatic reasoning in AI, is also explored. Then we present a discussion of pragmatic aspects of reasoning with a system and reasoning from a system, and the relations of these views to recognising and learning graphical proofs. To understand more clearly the nature of diagrammatic proofs, a case study is presented from two different symbolic perspectives. In the first, the goal is that a system learns a proof from a sequence of graphical patterns by inductive learning; in the second, the emphasis is on the syntax, semantics and proof procedure of the inductive mathematical proof of the same problem. As both of these approaches lies within a logicist view of diagrammatic reasoning, the question is addressed of whether a diagrammatic argument can be visualised, and to what extent this visualisation constitutes a proof. To this end, a third approach to verifying and learning a diagrammatic proof of the case study through a “visualisation” with a “retina” is presented. The discussion results in a diagrammatic reasoning system with a declarative syntax and a compositional semantics but implemented with a distributed computing architecture. The paper is concluded with a discussion on the relation between abstraction, visualisation, interpretation change and learning, applied to understand a purely diagrammatic proof of the Theorem of Pythagoras.

1. INTRODUCTION

Diagrammatic proofs are for many people usually easier to learn and understand than the corresponding proofs expressed in mathematical or logical notation. In diagrammatic proofs, proof procedures involve a limited number of operations which transform diagrams representing the premises of a theorem into a diagram representing its conclusion; proofs of geometric theorems, like the proof of the Theorem of Pythagoras in Figure 1.1, are probably the most typical examples of this kind. In an informal analogy between geometrical and logical proofs the different diagrams of the graphical proof would correspond to the premises of a logical argument and the final diagram, where the truth of the theorem can be appreciated, would correspond to the conclusion.

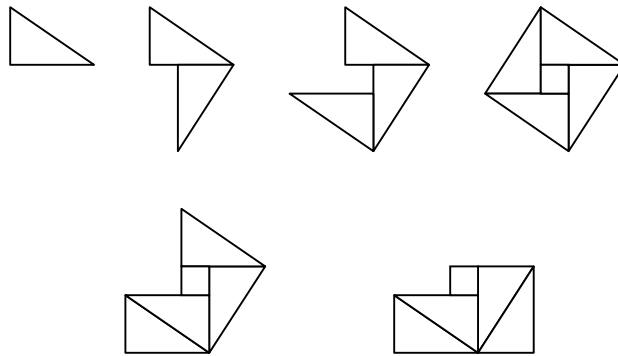


Figure 1.1. Proof of the Theorem of Pythagoras

There are also examples in which both theorem and proof are supposed to be read off from a concrete diagram, without external signs of the reasoning process involved, like the proof of the Pythagorean theorem shown in Figure 1.2. However, the absence of graphical transformations of the diagrams does not mean that the proof is grasped by a single, holistic, inference. In this latter case, the actual proof is a geometrical

¹ Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas (IIMAS), UNAM, Mex., luis@leibniz.iimas.unam.mx

² Human Communication Research Centre (HCRC), University of Edinburgh, UK, john@cogsci.ed.ac.uk

argument in which the appreciation of the truth of the premises and conclusion can be directly verified on the diagram. Furthermore, the nature of the proof can be best elucidated if the construction procedure for the diagram is developed alongside the geometrical argument.

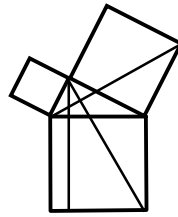


Figure 1.2. Theorem of Pythagoras (Euclid's proof)

Diagrammatic proofs have also been used as logical reasoning systems; for instance, Euler circles and Venn diagrams have been used to reason about syllogisms. In these kinds of systems it can be appreciated more easily that there is a set of valid operations that can be applied to produce the diagram representing the conclusion out of the diagrams representing the premises of a logical argument. Consider the Euler circle representation of the syllogism *All A are B, All B are C, All A are C* which is shown in Figure 1.3. As can be seen, the diagrams representing the premises are aggregated on and aligned by the middle term *B* of the syllogism.

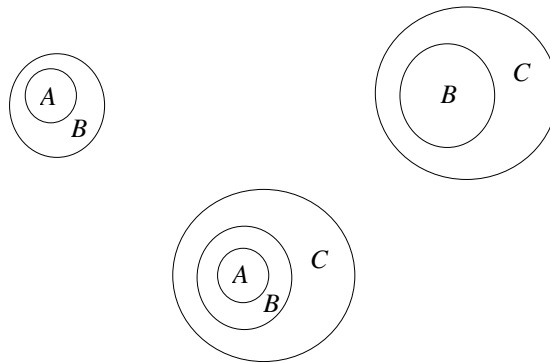


Figure 1.3. Syllogism representation with Euler's circles

Diagrams have also been used to illustrate proofs of arithmetic theorems that are normally proved through mathematical induction, such as the theorem of the sum of odd numbers, $1+3+5+\dots+(2n-1)=n^2$, illustrated in Figure 1.4. A sample of theorems of this kind, including the present one, are given by Nelsen [Nelsen, 1993].

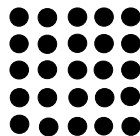


Figure 1.4. Theorem of the sum of the odds.

The figure represents the theorem because the leftmost L's of all the squares whose upper-right corner is at the upper-right corner of the grid can be interpreted as odd numbers, such that an L with a side of size n can be interpreted as the odd number $2n-1$. Any sequence of n consecutive L's (from right to left, starting from the upper-right) can be interpreted as the sum of the corresponding odd numbers and, as the area (number of dots) covered by this sequence is the same as the area of a square of size n^2 , the diagram represents the theorem.

As this sample of proofs suggests, diagrams can be used effectively to present and assess the validity of arguments that would be much harder to understand through logical or natural language representations.

The study of these proofs is important, as they provide a paradigmatic case of study for the use of graphical representations in more open forms of graphical reasoning and problem-solving tasks. The study of diagrammatic reasoning is not only a theoretical concern. As has been pointed out by Herbert Simon in the foreword of a recent collection on diagrammatic reasoning [Glasgow, 1995], it also has important applications in computational technology both for enhancing the effectiveness of visual displays, and for providing a scientific base for the construction of representations that can be stored and manipulated by computers. This is relevant to several fields of research, such as human-computer interaction, multimodal communication, visual programming, artificial intelligence (AI), and, in general, to any discipline which deals with the effective presentation and use of graphical information.

In Section 2 of this paper we discuss the notion of effectiveness of a representation both from a computational and a cognitive perspective, and ask whether these perspectives meet in AI. To look at the question of computational effectiveness we think of the tape of a standard Turing Machine as an external medium, like a piece of paper, and of the set of states and transitions, the algorithm, as an internal abstraction. We emphasise that the notation of a representation is embedded in this abstraction. We also introduce a generalised scanning device that is able to inspect symbols and abstract away from the restrictions imposed on external symbols by the architecture of standard Turing Machines. Then, we turn to consider the notion of effectiveness from a cognitive perspective. For this, we adopt the theory of specificity of graphics [Stenning and Oberlander, 1995] and review the effect of introducing limited abstraction in graphical representations. Finally, we review efforts in the field of AI to design and build intelligent systems that take advantage of the effectiveness of diagrammatic representations to increase the power of theorem-proving and problem-solving systems.

In Section 3, we review a number of pragmatic issues relevant for diagrammatic reasoning. We follow Gurr [Gurr, Lee and Stenning] on the distinction between reasoning *within* and *from* a system. The former is the traditional AI point of view in which the definition of a search space and the formal operations to explore it are the concern of problem-solving systems. However, computational processes are always embedded in agents which interact not only with one symbolic system but with many, and also with the world, in complicated ways. Adopting the view of reasoning from a system permits us to see emerging shapes in diagrams that are essential to the appreciation of diagrammatic proofs and learning.

In Section 4 we turn to investigate diagrammatic reasoning and proof through a case study. The aim is to clarify as much as possible computational models of diagrammatic reasoning, and to see the extent to which the effort captures human diagrammatic reasoning. The discussion is developed around several ways of modelling the diagrammatic proof of the theorem of the odds in Figure 1.4. First, we present an approach developed by Jamnik [Jamnik et al., 1999] in which the arithmetic proof of the theorem, which is usually proved by mathematical induction, is modelled as a learning induction inference produced on the basis of a sample of diagrams which are considered concrete instances of the general relation expressed by the theorem. However, although learning and proving theorems are closely related in problem-solving tasks, they are normally thought of as different phenomena. The common intuition is that a theorem is first learned and then proved. To study this issue we present a system of multimodal representation formalised along the lines of Montague's semiotic programme [Dowty, 1981]. In this system the syntax and semantics of a graphical language in which the diagrammatic proof is expressed are made explicit, and we show how the theorem can be proved graphically by a diagrammatic representation of mathematical induction. These two approaches highlight two different aspects of diagrammatic reasoning. The first focuses on the problem of how to express a theorem graphically, while the induction and the proof itself are thought of as symbolic operations. The second, on the other hand, shows how the proof by mathematical induction can be thought of as a linguistic argument that has a visual counterpart, but assumes that the expression representing the theorem is given. Additionally, as both strategies postulate intermediate symbolic structures to learn or reason about the diagram, the diagram becomes a subordinate external object only used for interfacing with the symbolic process, or as an aid to computing and storing information effectively: the traditional logicist view of AI.

In Section 5 we investigate further whether diagrammatic proofs can be modelled computationally without relying essentially on a symbolic process. We review Funt's work on the Whisper system [Funt, 1980] and discuss whether the diagrammatic proof can be visualised through a computational *retina*. We also discuss whether the diagrammatic inductive argument has to be visualised to verify the theorem, or whether, according to the theory of graphical specificity, the visualisation of the proof on a single diagram can be considered a valid argument.

In section 6 we discuss whether diagrams can be interpreted as universal assertions or proofs. We argue this is the case because diagrammatic or graphical proofs are interpreted by people as limited abstraction representational systems. We argue that the abstraction that makes it possible to interpret a diagram as a graphical proof is a consequence of the abstraction conveyed by the basic notation of the representational system and some properties of the representation medium. The discussion in relation to the pragmatics of theorem-proving is also brought to bear, and it is argued that the process of visualisation is much more interesting in the context of learning a theorem graphically rather than in the context of verifying graphically something that is known already. We then advance a discussion on the relation between abstraction, visualisation and notational change, and how this relation impacts on learning. We conclude the paper with a reflection on how the syntax and semantics of graphics, the interpretation of diagrams as minimal abstraction representational systems, the pragmatic issues related to reasoning within and from a system, and the processes involved in notational change and visualisation are all brought to bear upon learning and verifying the proof of the Theorem of Pythagoras illustrated in Figure 1.1.

2. Notation, Representation and Graphical Interpretation.

2.1 Turing machines and effective computations

In this section we look at the relation of a diagrammatic representation and its interpretation process, and discuss some of the reasons that make diagrams such effective aids in communication and reasoning. One way to think of a diagram is as an expression composed of symbols of a well-defined alphabet which is written down on the tape, probably bidimensional, of a Turing Machine. The tape can be thought of as the *medium* of the representation (in a sense similar to that of [Stenning et al., 1995]). The Turing machine itself, the set of states and transitions that defines the algorithm, can be thought of as an abstract process which interprets the diagram. This intuition corresponds to the actions one takes when performing manipulations on external representations; for instance, for adding two numbers in decimal notation using the traditional algorithm that is taught in elementary school, one has to write the numeral symbols on a piece of paper according to what is seen (in an external representation) and manipulate the symbols following an algorithm, which has no overt description, but which is known internally.

Expressions written on the tape of a Turing Machine are interpreted in relation to a given notation. The string “111”, for instance, can be interpreted as the number three, seven or one hundred and eleven depending on whether the notation is monadic, binary or decimal. However, unlike the alphabet, the set of states and actions, and the transition table, the notation is not formally stated in the specification of a Turing Machine. If the interpreter is a person, the knowledge of what is the notation is kept in her mind, and the interpretation is relative to the standard input and output configuration conditions that have to be explicitly stated [Boolos and Jeffrey, 1990], but if the interpretation process is a standard Turing Machine, the notation is nowhere but implicit in the way the algorithm is constructed.

Algorithms compute functions, but the same function can be computed by different algorithms, and the choice of one algorithm instead of another can have an important impact on the amount of memory and time required for the computation. Indeed, it is possible to compare two algorithms computing the same function in terms of the computation steps and memory cells employed by each of them, and if one performs better than the other, we can say that it computes the function more effectively. Algorithms are also designed taking into account the shape of the medium in which the symbols on the tape are written down. A Turing Machine for computing the sum using a linear tape with monadic notation, for instance, can be easily designed; however, the same computation can be performed by a Turing Machine using decimal notation and an array as its representational medium. This machine would have to navigate not only in a horizontal direction (from left to right and vice versa) but also to move up and down the array according to the current state and the symbol that is scanned. The former machine can be defined with a very small set of states, but the computational steps and tape cells required to perform the computation grow linearly in the size of the arguments. The latter machine, on the other hand, can be designed with a little more than 110 states but, interestingly enough, the computational resources required to perform the computation grow only logarithmically in the size of the arguments. It is not an accident that human beings use this second choice of notation and medium for arithmetic computations. The relation between the abstract process, the scanning device and the grid of a bidimensional Turing Machine is illustrated in Figure 2.1.

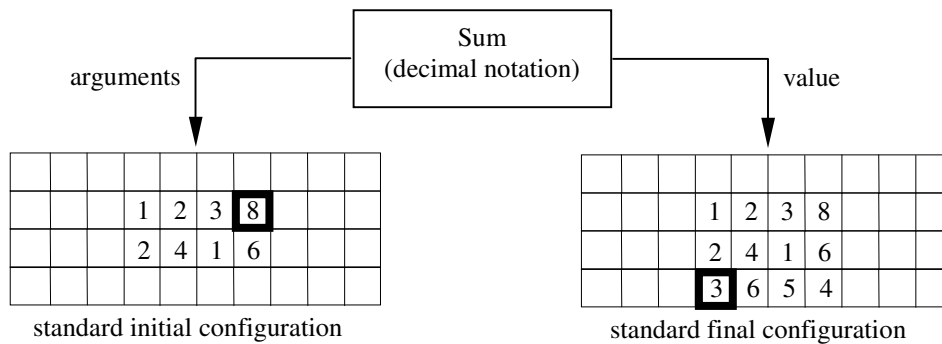


Figure 2.1 Bidimensional Turing Machine for a sum calculation

Another important aspect of the interpretation process performed by a Turing Machine is that many computational steps are used to navigate across the medium to scan and write symbols in the proper place. Increasing the dimensionality of the medium reduces the number of computational steps, as the navigation of the scanning device over the cells will have more access routes. In order to see this, one can easily define an algorithm to compute a sum in decimal notation with a linear tape and compare the number of steps required to move the symbols around the tape with the navigation steps required to compute the sum on a grid. People do follow a scanning protocol when performing arithmetic computations on a piece of paper, and can be aware of it.

Other shapes of the Turing Machine's "tape" can be conceived. Data structures like arrays, trees, dags, or graphs in general can also be thought of as external representations, in the sense that the objects stored in the cells of these kinds of structures (i.e. array cells or nodes) are basic symbols or composite structures of a well-defined language, that are inspected by a scanning device which is able to move along the medium during the interpretation process according to its shape. Navigation on a graph is similar to navigation on a linear tape, as the scanning device follows the topology of the medium and moves from cell to cell through the graph links, one step at a time. Navigation in arrays is more flexible as the position of the scanning device is set directly through the index; navigation on a random access memory is yet more flexible as the scanning device is set directly through a direct or an indirect reference, abstracting away the computational steps required to navigate across the tape. Tables and arrays are a compromise between graphs in which the scanning protocol is fully determined and random access media in which it is abstracted away altogether.

Scanning operations in Turing Machines are local to tape and grid cells. The symbols of an alphabet are also local, in the sense that they are always fully contained in individual cells, a cell can contain only one symbol, symbols are read and written on single atomic operations, and algorithms are discrete sequences of these operations. These are formal properties which permit us to describe algorithms in a very precise way; however, if we relax these properties and think of symbols of the alphabet in abstraction from the individual cells, and scanning protocols in abstraction from local and sequential constraints, then a large class of external representations can be thought of as a particular selection of medium, alphabet and notation of a Turing computation. We suggest that this abstraction could be achieved by thinking of the scanning device as an active retina, as in the Whisper system [Funt, 1980], in such a way that a number of cells could be inspected and interpreted simultaneously by active processors associated to each cell, and the symbols on the grid, abstracted as wholes, would be the objects manipulated by the Turing Machine proper. This architecture is illustrated in Figure 2.2.

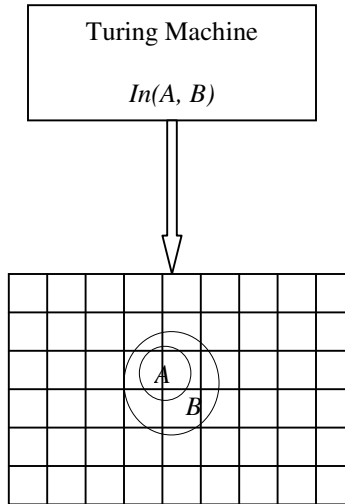


Figure 2.2 A Turing Machine with an abstract scanning device

An example of the kind of computation that could be carried out by this architecture is the process of reasoning with Euler circles, in which circles representing sets of objects or classes are manipulated by an abstract process which implements a sound inference method. Here, circles are not constrained to individual cells of grid, do overlap between each other and, nevertheless, are inspected and modified by the human interpreter in the course of producing the proof. This process, as most algorithms employed by people, is very complex and its full algorithmic characterisation might never be specified, but we can focus on the properties of the media, alphabet and notation of external representations, and regard the algorithm used by people to interpret them as an abstraction.

Looking at human computational processes from the point of view suggested here might have some implications for the debate on the propositional versus the analogical views of knowledge representation. The propositional view of knowledge representation, on the one hand, is concerned with the meaning of logical or natural language expressions, but not necessarily with how effectively these kinds of representations are interpreted by people. Propositional representations are interpreted through processes in which external symbolic manipulation and scanning protocols reveal little about the nature of the algorithm, as much of the work is done internally by the interpretation process. Furthermore, questions about effectiveness cannot even be raised because such a view adopts a particular choice of media and notation beforehand. In the analogical view and the study of diagrammatic representations, on the other hand, the nature of the algorithm followed by the interpreter is revealed in the external acts, and the medium becomes a concrete reference for understanding the process. In this sense, the interpretation process of propositional representations is more abstract than the process involved in the interpretation of diagrams, which have a much more concrete character. However, the borderline between the two views is difficult to demarcate because all computational processes have both an internal and external aspect. In general, information expressed through diagrams can be expressed also, at least in principle, through propositional representations; these kinds of representations do not differ in meaning, but one might be interpreted much more effectively than the other depending on the choice of medium and notation. In a similar vein, Larkin and Simon [Larkin and Simon, 1987] distinguish between informational and computational equivalence of representations. Effective computations will require less computational resources than the corresponding computations using different kinds of representations, and in that sense, will be easier for people to use and understand with limited computational resources.

2.2 The specificity of graphics

From the previous discussion we can see that effectiveness is a relational notion which involves the comparison of different representations for the same kind of knowledge and inferential task. In order to obtain a cognitive perspective on the properties that make representational systems effective, we consider the cognitive theory of graphical and linguist reasoning of Stenning and Oberlander [Stenning and Oberlander, 1995] which was developed on the intuition that graphical representations limit abstraction and thereby aid processibility. The limitation requires that a certain amount of information must be specified in a system

concretely, in contrast to systems of representation that allow arbitrary abstractions. This property of graphical systems of representation is referred to as *specificity*. The theory characterises three kinds of representational systems according to the restriction of 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). Stenning and Oberlander assume that in the same way as logical and natural language, 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 of the represented world are satisfied by a single model in the intended interpretation, but LARS and UARS can have several models. Hence, MARS are expressions that can never be ambiguous or contain symbols standing for incomplete information. For instance, the representation of a sum in decimal notation on a grid is a MARS because different numerals denote different numbers, and the number representing the sum is either correct or incorrect in relation to the numbers being added. The knowledge of this algorithm for the sum is nowhere in the representation but only in the mind of person performing the computation, and for the purpose of the model, it is an abstraction. So, the process of adding two numbers on a piece of paper involves some abstraction, but the interpretation of the symbols in the representation is known directly and unambiguously by the interpreter. Since the abstraction involved is minimal, MARS are interpreted very effectively. MARS are complete representations denoting fully determined states of affairs.

In Stenning and Oberlander's theory it is also important to consider how the interpreter knows the meaning of basic symbols and composite expressions in the representation. For this, they use the notion of *representational key* which consists of the knowledge that has to be made explicit to the users of a representation so they can understand it, and to know this is to know the notation of the representation. If we ask how many models there are for " $111 + 11 = 11111$ ", we need to know that the notation is monadic, and that the representation states that three and two are five; so there is one model, and the string is a MARS. (If the notation is binary or decimal the expression can also be interpreted but is false and has no model.) The difference between MARS, LARS and UARS depends also on the structure of the representational key. In the case of MARS, representational terms have a denotation that is fully determined if the notation is known. In the case of LARS, the interpretation of a symbol or a composite term can have a number of possible interpretations and then represents an abstraction. To implement abstraction in representations there must be symbols standing for more than one individual or state of affairs; the simplest example is to include symbols on a representation standing for variables. Consider the expression " $X + 1 = Y + 1$ ", and suppose that in the key is stated that "X" and "Y" stand either for one or two; then the expression represents more than one of states in the world, and in fact it has two models: in one it asserts that two is equal to two, and in the other that three is equal to three. As a consequence, the interpretation of this second kind of representation needs to consider the alternatives, and then it is more expensive computationally than a MARS.

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, such as "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. Stenning and Oberlander review the logical equivalencies between MARS and complete sentences of monadic propositional logic which describe a situation comprehensively (i.e., a sentence in conjunctive normal form exhausting the combinatorial possibilities of predicates and constants of the logical language, assuming that there is no lexical ambiguity). However, if a logical sentence describes a situation partially there might be several situations that are compatible with the statement and the logical representation will be equivalent to a LARS, as the sentence can be thought of abstracting over all such situations; if there were lexical or syntactical ambiguity in a representational structure, it would also be a LARS for similar reasons. As MARS can be computed effectively they are also compared with Levesque's *vivid knowledge-bases* [Levesque, 1988] 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 the subsumption of predicates in taxonomy, they are roughly equivalent to LARS, as they permit a restricted form of abstraction.

The theory of graphical specificity is designed to relate the effectiveness of graphics to the properties of MARS, LARS and UARS, but these can also be thought of as applying to linear representations, as in our examples. According to Stenning and Oberlander the semantic properties of a representation, especially in relation to restricted abstraction, have a *syntactic reflex* which makes some representations easier to perceive and understand. For instance, a tabular representation of the semantics of a logical language in which columns are labelled by predicate names, rows by constants and the cells are filled with “1” or “0”, is much easier to interpret than the corresponding comprehensive sentence of monadic predicate logic. It is the particular nature of this syntactic reflex that makes graphics more effective. However, the question of how the syntactic reflex is linked to a semantic property is illustrated only by examples. The tabular representation for the semantics of logical language in our example has no empty cells: all cells have one out of two possible values, and there is never more than one symbol in each cell. Another way to put this is that the information conveyed for all propositions is displayed simultaneously in an orderly, synoptic fashion. Stenning and Oberlander use syllogistic reasoning with Euler circles as the running example to illustrate the theory. In this task, the diagrams representing the premises of a syllogism (Figure 1.3) are aggregated into a composite figure in a single holistic operation from the result of which the conclusion of the syllogism can be read off.

Enforcement of a particular level of abstraction in a representation depends on the system having a suitably chosen syntactic reflex. Euler circles for syllogistic reasoning have been criticised as an effective method of reasoning because premises can have several graphical representations, and all combinations of diagrams representing the premises must be taken into account when producing a conclusion. Regions in such diagrams have a definite interpretation and each diagram is a MARS, but several concrete diagrams have to be inspected to assess whether there is a valid conclusion. In the Stenning and Oberlander version of the Euler circles reasoning system, on the other hand, a notational device (a mark) is introduced to assert that there are some types of individuals whose existence is necessarily implicated by the premises, but unmarked regions mean that individuals of the types represented by such regions might or might not exist, expressing incomplete information. Hence, the system allows a conclusion to be derived from a very small number of diagrams — between one and three — by the addition of a notational device through which abstraction is expressed. Another way to put this is that a graphical LARS permits reasoning that minimises the number of cases that have to be considered, as each case represents an abstraction. The syntactic reflex for this task is the superposition of the diagrams representing both of the premises in a single holistic operation giving a synoptic view of all the relevant information for the reasoning task. As this superposition is additive, it naturally expresses conjunctive information directly; disjunctive information embodied in abstractions, on the other hand, needs to be expressed by means of notational devices interpreted according to the representation key. Stenning and Oberlander also argue that Johnson-Laird’s mental models [Johnson-Laird, 1983] are graphical LARS, although the syntactic reflex in the latter model is less natural than the superposition of diagrams in Euler circles.

2. 3 Computational simulation of diagrammatic reasoning

Having reviewed the notion of effectiveness of a representation from a computational and a cognitive perspective, we turn now in this section to the question of how diagrammatic information has been used for problem-solving and theorem-proving in AI, and to what extent such programs model human diagrammatic reasoning. A good survey of philosophical and historical issues, foundational and theoretical approaches, computational models and applications can be found in [Glasgow et al., 1995]. According to Chandrasekaran [Chandrasekaran, 1997], so-called diagrammatic information is used in AI in three different fashions, which he refers to as predicate extraction and projection, reasoning and simulation. In the first, a diagram is just a pool of information which is inspected by (predicate extraction) or modified by (predicate projection) a symbolic problem-solving system in the course of completing a task. Diagrams in this approach codify large amounts of the information that would have to be made explicit by axioms in a data-base otherwise. In this sense, a diagram codifying some information can be thought of as a vivid knowledge-base, since the test of whether something follows from it could be implemented with the help of algorithms extracting and interpreting information from diagrams in a very efficient fashion. Probably, the first antecedent in this line is Gelernter’s geometric proving system which was able to prove geometric theorems symbolically but used diagrams to prune the search space [Gelernter, 1963]. Problem-solving systems can also read and modify diagrams to represent partial states of a computation, and predicate extraction and projection can interact in complex ways in course of solving a problem, as in the case of the Hyperproof system for teaching logic [Barwise and Etchemendy, 1991]. Predicate extraction and projection can be also used for defining and interpreting graphical languages [Pineda, 1989, Klein and Pineda, 1990] and to establish the set of

interpretations that a diagram can have in relation to a conceptual scheme [Reiter and Mackworth, 1987]. The second kind of task is characterised by the aim of developing or analysing proof-theoretic methods in which premises and conclusions are represented through diagrams. Stenning and Oberlander's version of Euler circles [Stenning and Oberlander, 1995], Shin's model of Venn diagrams [Shin, 1995], Wang and Lee's system for reasoning about graphical concepts [Wang and Lee, 1993] and Jamnik's work on inductive proofs of arithmetic theorems [Jamnik et al., 1999] are instances of this category. The third kind of system in Chandrasekaran's taxonomy produces a simulation of the process to be modelled. An instance in this class is Funt's Whisper system which uses a "retina" to "visualise" unstable objects collapsing in a blocks world [Funt, 1980].

Now, we turn to the question of whether these kinds of systems capture important properties of human diagrammatic reasoning. To assess this we suggest three levels of resemblance between computational and human problem-solving. In the first, a similarity between the data-structures used by the reasoning program and the external expressions representing premises and conclusions of an argument or proof would be expected. In the second, similarity between the proof procedures to come from premises to conclusions in the external proof and the proof-procedures in the computational implementation would be required. In the third level, a resemblance between the architecture of the computational machine and the process that is likely employed by people would be expected. Of course, this third level can be stated only very vaguely. Let us consider Chandrasekaran's taxonomy in relation to these three levels of similarity. Systems relying on predicate extraction and projection shed little light in this respect, unless there is a clear description of the syntax and semantics of diagrams; in such a case, the first level of similarity would be achieved. Reasoning systems would achieve the second level if the computational transformations on the representational structures resemble directly what people would do in working out the solution to the same task. A system to reason with Euler's circles within this level, for instance, would apply procedures for combining the graphical representations of the premises into that of the conclusion by graphical superposition, read the conclusion from the diagram thus produced, search for diagrams invalidating the conclusion and make sure that there is none, in the same way people produce such proofs with a pencil and a piece of paper. These systems, if asked to explain the proof, should be able to produce a sequence of drawings representing the proof procedure at a level of abstraction that is intuitive from the point of view of the human interpreter³. To illustrate the third degree of similarity consider a qualitative simulation produced through inference rules applied sequentially and contrast this with a similar qualitative process model through a visualisation which can profit from a large parallel or distributed computation as in Funt's Whisper system. The latter would be a better cognitive model according to the third level. Below, in Section 4 and 5, we study these levels of similarity through a case study in diagrammatic inductive proofs of arithmetic theorems.

3. Reasoning from a system: the pragmatics of theorem-proving

Thus far, we have focussed on the relation between a given representational system and a particular reasoning problem. In practice, faced with a reasoning problem there is usually a crucial prior step of determining which representation system to use. We call this the problem of *representation selection*. Past theories of diagrammatic reasoning have tended to focus either on the semantics of diagrams or on the advantages of graphics over other kinds of representations, but have not brought these together well [Gurr et al., 1998] to help with understanding the complex issues that influence the optimal selection of representations.

One aspect of this is familiar in the literature of cognitive science, popularised especially by Don Norman (cf. [Norman, 1994]). A typical example is the solution of the "Towers of Hanoi" problem using coffee-cups instead of the usual discs-on-sticks. A situation is set up so that coffee-cups (when inverted) can only be stacked smaller on top of larger — otherwise they fall inside each other. This reduces dramatically the number of possible ways the path to a solution can go astray. In effect, it reduces the search-space of the problem and thus makes solving it much quicker and easier.

Such a situation can be seen as analogous to graphics, except that in the Towers of Hanoi there is no semantics: the problem is simply to rearrange a given configuration using minimal effort, which can be

³ For instance, the Graflog system is able to solve technical drafting problems by the application of qualitative drafting rules and produce a graphical explanation of the graphical problem-solving task directly reflecting drafting actions performed by people solving similar kinds of problems [Pineda, 1992].

compared to a purely syntactic transformation. Some purely syntactic constraints are introduced which limit, and hence in a sense facilitate, that transformation. A reasoning problem of the kind addressed in this paper involves, additionally, an interpretation and some further constraints that derive from this (e.g. that transformations should be truth-preserving). Diagrams turn out to be especially useful when the syntactic constraints coincide with what is spatially possible, and when the interpretation is defined such that these also enforce semantic requirements — in other words, when we have, in the sense introduced above, a particularly well-chosen syntactic reflex of the desired semantics.

We can see this happening when, for example, we use Euler circles to solve the syllogism depicted in Figure 1.3. We select a representation for the premises in which we interpret circles as sets and graphical containment as set-inclusion. When combined in the only spatially possible way that preserves truth under the interpretation, these yield a diagram that represents A as included in C. Hence we can simply “read off” the conclusion, which we have arrived at via what Shimojima calls a “*free ride*” [Shimojima, 1996]. The problem has been solved essentially by selecting a representation in which simply representing the problem provides “for free” a representation of the conclusion.

There is indeed [Gurr et al.,1998] more to this than appears at first glance, because in some systems (and even in this system, with some other syllogisms) the conclusion is a good deal less obvious; but the ride, if not free, is still less expensive than in a sentential system with full abstraction. The cost of rides is related to (but not, as we see later, determined by) the limitations that the system places on abstraction.

A danger in taking cheap rides is that they may go in the wrong direction. Indeed they may go completely off the rails. There are many examples of graphical “proofs” which produce false conclusions, and some of these may be due to adopting quite inappropriate conventions of inference. One cannot define arbitrary graphical moves and expect them to produce meaningful results. Prior discusses a closely related point in natural language inference, mocking the notion of a “runabout inference ticket” [Prior, 1960]. He proposes a language with a new connective, *tonk*, having the following associated rules of inference:

$$\begin{array}{ccc} A & & A \text{ tonk } B \\ A \text{ tonk } B & & B \end{array}$$

Clearly, successive application of these rules allows anything to be derived from anything. Prior suggests that this shows connectives have to get their meaning from some pre-existing natural language concept (e.g. conjunction) that precludes the definition of rules like these. But one can also argue [Haack, 1978 pp. 31-2] that any rules of inference are acceptable so long as they do not lead to inconsistency; i.e. that syntactic moves always have to be constrained, but perhaps relatively broadly, by semantic considerations. Graphical inferences may not directly resemble those drawn in natural language, but can still be constrained to be valid; which allows e.g. for the sorts of completeness results derived for particular graphical systems by Shin [Shin, 1995]. Close attention to the “systematicity” of representational mappings [Gurr et al., 1998] should help to keep us on track and reinforce our resistance to suspect inference tickets, for instance in strengthening the syntactic reflex by avoiding the use of transitive graphical relations to represent intransitive domain relations.

The risk remains that our ride will take us to a valid destination, but other than where we want to go. Graphical proofs can very often, perhaps always, be interpreted in more than one way. Consider the proof, given above, of the theorem of the sum of odd numbers. As discussed in section 6 below, the same diagram (our Figure 1.4) can be used to show that $1+2+\dots+(n-1)+n+(n-1)+\dots+2+1=n^2$, by considering successive diagonals, rather than *Ls*, across the square of dots. Thus the emergence of different patterns allows us to see proofs of different theorems. For practical purposes in reasoning, therefore, something must always direct the *attention* of the diagram user in particular ways. This re-emphasises the importance of the *representational key*, introduced above, which defines how a particular diagram is being *used* for a particular purpose in a given situation. A diagrammatic proof exists only when a diagram, an interpretation and a theorem are all brought into appropriate relationships with each other. There is apparently no abstract way to delineate the class of theorems that any diagram could be used to prove, since the ways in which it can be related to an interpretation are ultimately arbitrary. (Though we note that the class may be limited e.g. by stipulating that the interpretation be one that avoids abstraction and defines a LARS, or even a MARS.)

A final, but for the concerns of this paper highly critical aspect of cheap rides, is that they may only take us part of the way to the desired conclusion. As discussed in much more detail later, we may find that beyond a certain stage a *change* of interpretation is needed to take us to the final conclusion. We need to interpret the array of dots as *Ls* for one part of the argument, but then we must see that it can be equally viewed as a square; the re-interpretation involved in the Pythagoras proof at the end of section 6 is even more dramatic. Similarly, we may reinterpret with a different level of generality, e.g. to make a figure represent either a specific square or any possible square. Major issues arise at the junctions between these interpretations. What constrains the range of interpretations that can be imposed on the figure? How can we tell which directions it is safe or profitable to go in? Might we jump to interpretations that will lead us to invalid conclusions? Might some options take us to the same conclusion but at much greater cost? Perhaps, by viewing the dot-array now as *Ls*, now as diagonal lines, we can derive an interesting relation between odd numbers and symmetrical sequences; but this is of no use if we want to relate either to n^2 . Goals may direct our focus quite sharply.

This reminds us that even in the simplest cases there is typically more than one way to get to the same conclusion, alternative rides that may have different costs. Thus the syllogism solved above with Euler circles could also have been solved with a Venn Diagram, or indeed with some sentential apparatus. One can argue that all of these, even the sentences if constrained as a "vivid" representation (cf. above discussion of Stenning and Oberlander), constitute LARSs. From the specificity point of view, they may even have just the same level of abstraction, and therefore be equivalent. The evaluation of these different syntactic reflexes must be sought in issues concerning the human perceptual faculty, which are very much more difficult to render explicit and formal. Formal systems can only distinguish diagrams up to a certain level — the level at which they can be called *counterparts* [Hammer, 1995 p.38] — which is intuitively the level at which they are equivalent under the current interpretation function. A simple example would be an instance of the proof of the sum of odds that constructs *Ls* from the lower left instead of the upper right: the proof may be formalised so as not to discriminate between these options. Counterparts can in other cases differ visually a good deal more than this, but are still defined to be semantically identical. A proof using different such counterparts may easily differ in its accessibility to the human reasoner; it has somehow different cognitive costs. It seems unlikely that people have a specific cognitive bias say towards constructions from the top-right; but if they did have then it would matter in choosing the best form of representation to use. One could define a formalism which made this distinction, but in general it is hard to know which distinctions to formalise.

From the pragmatic point of view, a reasoning task should be approached by selecting a representation which has the minimal necessary level of abstraction, and which offers the cheapest rides available in the right direction. Whereas the first of these requirements may be addressed formally, there are no algorithmic means of assessing the latter. We are facing here the problem of reasoning *from* or *to* a representation, rather than *within* a representation system already selected. As noted above, deciding how to represent a problem may be the most important step in finding a solution, yet it is the one that eludes much current theorising. Diagrammatic representation systems have a number of "metalogical" properties that relate to their suitability for use with various types of problem; for example, "self-consistency" [Gurr et al., 1998]. It is suggested that people can learn simply to *see* when systems, e.g. Euler circles as opposed to Venn diagrams, have certain such properties. However, little is known about the extent to which this happens, or how best to teach these kinds of sensitivities. We can here simply re-emphasise that problem-solving strategies will in general have to address much more clearly the selection and use of possibly a sequence of appropriate representations.

The costing of inferential rides is also a particularly difficult aspect to relate to computational problem-solving systems. The third level of resemblance between human and computer problem-solving, as mentioned in the last section, would seem to demand at least an approach to devising programs that mirror the costs for humans of working with given representations. By and large, however, the first step in the computational representation of a problem is to turn it into something so far removed from human perceptual experience that no comparison seems possible. For example, a graphic is often turned into a set of ground clauses. These may well at one level constitute a LARS with processing properties very similar to the graphic, but the syntactic reflex is so divergent as to completely preclude perceptual costing. This issue has a pervasive influence on the following discussion of how to move away from the "traditional AI", logicist approach to representation and reasoning — motivating especially the discussion of "Whisper" in section 5.

4. Reasoning within a system: A case study on diagrammatic inductive proofs.

After reviewing some of the pragmatic issues concerning the relation of signs to the interpreters in which theorem-proving systems are embedded, we move to the traditional AI point of view which is centred on the formal representations and manipulations upon them by explicit computational processes. For this purpose we use the theorem of the sum of the odd numbers presented in Figure 1.4 as a case study. We first summarise and discuss a proof of this theorem and an inductive theorem-proving system that has been developed by [Jamnik et al., 1999] within the traditional AI logicist perspective. Despite its formal approach, issues about the syntax and semantics of diagrams representing the theorem and proof procedure are not explicitly developed in this work, and questions regarding whether the external representation plays an essential role in proof need to be answered. To look closely into these issues we introduce a second approach in which explicit syntax and semantics for graphics with the corresponding theorem-prover are provided. Diagrams in this latter view correspond to well-formed expressions of a graphical language, and the new proof procedure permits us to visualise better the essential aspects of the graphical induction, as valid inference steps of the theorem-prover correspond to diagram transformations that can be perceived directly. However, as this second approach relies on a representational language of a logical kind in which explicit relational operators denote graphical relations that human interpreters can perceive directly, this theorem-prover still stands within the AI traditional logicist view. In order to illustrate how a theorem-proving system can take advantage of the property of “directness” of diagrams and theorem-provers, in Section 5 a third approach is presented and discussed, in which the intuition that diagrams have a well-defined syntax and semantics is preserved, but in which the underlying representation is a grid filled with dots, an analogical representation.

4.1 An inductive diagrammatic theorem-prover system

In this section the inductive diagrammatic theorem-proving system “Diamond”, developed by [Jamnik et al., 1999] is presented and discussed. Interestingly enough, the stated goals in this work were to simulate human diagrammatic reasoning in computers, capturing the intuitive notion of truthfulness that humans find easy to see and understand, and also to investigate the relation between formal algebraic proofs and the more “informal” diagrammatic proofs. The procedure consists of three main steps: (1) expressing the theorem through a graphical interactive interface with a set of geometric operations, (2) inducing diagrammatic patterns through inductive learning techniques and (3) verifying the proof by translating it into the corresponding algebraic one. For step (1) a number of geometric operations for decomposing a shape into its constituent parts in a systematic fashion are performed by the human user expressing the proof; for instance, the square representing a square of size n can be decomposed into an “L” shape of size $2n - 1$ and a square of size $n - 1$. This latter square can be decomposed in turn, until the original square representing the square of n is decomposed into a sequence of “L” shapes representing a sequence of odd numbers. According to Jamnik, these operations capture the inference steps of the proof. The operators that perform this decomposition are provided by the Diamond system and they constitute the “graphical vocabulary” of the theorem-prover. Geometrical operations are given an operational semantics and the system is able to prove correct decompositions of diagrams. In the second step of the process, an inductive constructive procedure, the so-called *constructive ω -rule*, is used to produce the representation of a recursive geometric pattern out of a number of proof instances. The resulting recursive scheme can be used to decompose an arbitrary square, and in the intended interpretation it represents the generalisation to be made when one realises that the diagram is a proof of the theorem because it holds for any arbitrary sequence of “L” shapes making up a square. The final step of the procedure consists in verifying that the resulting scheme is correct. For this, it is required to show that there is a proof in a meta-theory which corresponds to the diagrammatic proof, where a symbolic proof-tree must be constructed which has the representation of the theorem at its roots and axioms at its leaves. The theorem and proof are expressed through a symbolic language, and the proof itself is produced by an inductive argument in this language. Mappings relating arithmetic to diagrammatic representations are also defined, and expressions of the symbolic language can be interpreted as arithmetic expressions.

4.2 Graphical syntax and semantics for graphical inductive proofs.

In order to improve our understanding of the problem, we have developed an approach to modelling the inductive diagrammatic proof, in which diagrams are thought of as well-formed expressions of a language with explicit syntax and semantics, rather than as objects to be decomposed by a sequence of geometrical operations, and where the validity of the diagrammatic inductive argument can be assessed by direct

inspection, as is usually done with Euler circles or Venn diagrams. As a reference for this discussion, an arithmetical inductive proof of the theorem, a propositional one, is shown in Figure 4.1:

Theorem:

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

Proof:

- (1) $1 + 3 + 5 + \dots + (2n-1) = n^2$
- (2) $1 + 3 + 5 + \dots + (2n-1) + (2(n+1)-1) = n^2 + (2(n+1)-1)$
- (3) $1 + 3 + 5 + \dots + (2n-1) + (2(n+1)-1) = n^2 + 2n+1$
- (4) $1 + 3 + 5 + \dots + (2n-1) + (2(n+1)-1) = (n+1)^2$

FIGURE 4.1 Mathematical induction on the theorem of the sum of the odd numbers.

To appreciate better the propositional nature of this proof procedure, consider how it would be implemented as a traditional symbolic process (i.e., an AI theorem-proving system). The steps of such a program would be as follows:

- (a) Receive as its input the theorem in eq. (1)
- (b) Apply the inductive hypothesis to produce eq. (2)
- (c) Reduce the eq. (2) to eq. (4)

To carry out step (c), a heuristic search process would have to be applied involving several steps, including eq. (3). The ellipsis “...” in expression (1) to (4) states that the pattern in which it is embedded represents an abstraction: that the pattern occurs an unspecified number of times. Now consider the analogous diagrammatic formulation of the theorem and proof in Figure 4.2:

Theorem:



Graphical inductive proof:

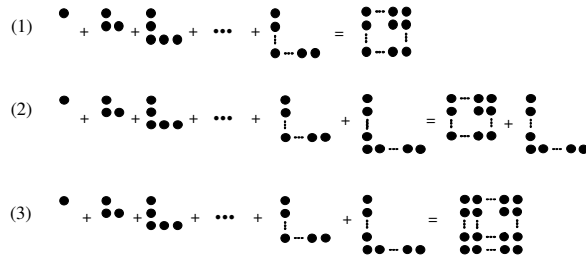


FIGURE 4.2 Diagrammatic inductive proof of the theorem of the odd numbers

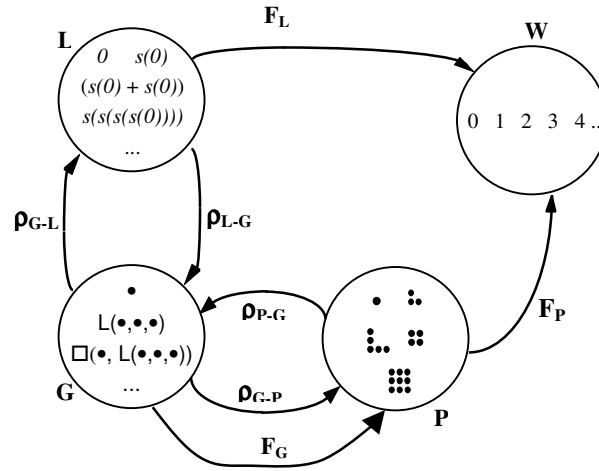


Figure 4.3 Multimodal system of representation

This illustrates a protocol for inspecting the diagram that can be applied by a human reasoner in the course of making the proof, although the actual external representation need not be other than the diagram in 1.4. Notice that this proof also uses ellipsis as a notational device, in both the vertical and horizontal dimensions. We believe that this proof procedure captures better the diagrammatic intuition underlying the inductive graphical reasoning. If a graphical theorem-proving system were designed to perform this proof, it would proceed as follows:

- (a) Receive as its input the original theorem in graphical eq. (1)
- (b) Apply the inductive assumption producing graphical eq. (2)
- (c) Perform a graphical manipulation process to reduce graphical eq. (2) to graphical eq. (3)

To model the inductive process, we need to show that the graphical expressions used in the proof do belong to a language and have the intended semantic interpretation; we also need to show how the steps of the proof can be performed as direct transformations on the diagrams, what operations perform such transformations, and what is the correlation of the diagrammatic proof with the corresponding arithmetic proof. For this we use the multimodal representational system illustrated in Figure 4.3, which has been developed over the last few years [Pineda, 1989, 1996], [Klein et al., 1990], [Santana, 1999] and [Pineda and Garza, 1999], and which follows closely the spirit of Montague's general semiotic programme [Dowty et al., 1985]. The reason to adopt this formalism is that although it was originally conceived to capture compositionally the semantics and translation relations between arbitrary natural languages, it can also be applied to more general systems of signs including diagrams [Pineda and Garza, 1999]. In general, Montague's semantics provides us with a framework for defining meaning preserving translation rules relating expressions of source and target languages. For our case study, if the arithmetic and diagrammatic proofs do mean the same, there must be meaning preserving translations between diagrams and their corresponding arithmetic expressions. An additional reason to use this framework is the strict separation employed by Montague between syntactic rules and semantic operations for the description of unambiguous languages. Syntactic rules are thought of as formal structures, abstracted away from any external materialisation, but they are defined in terms of syntactic operations which separate the external shape of symbols in the external expression from the actual syntactic structure of the expression, a flexibility that is essential when thinking of drawings as expressions of a formal language. Note that this cannot be achieved with a standard production system in which the shape of an expression, a string of characters, is symmetrical to its syntactic structure.

The circles labelled **P** and **L** in Figure 4.3 represent the sets of expressions of the graphical language and the language of arithmetic respectively. The circle labelled **G** stands for an *interlingua* between **P** and **L** which, on the one hand, captures the geometrical structure of **P** and, on the other, has a well-defined syntax and semantics that can be related to the language of arithmetic and can be used for the specification of a computational implementation. The functions ρ_{L-G} , ρ_{G-L} , ρ_{P-G} and ρ_{G-P} stand for the translation mappings

between the corresponding languages. The last two translation relations define the “generation” and the “perceptual interpretation” of diagrams. The set \mathbf{W} stands for the world, which in this case is the set of natural numbers and together with the functions \mathbf{F}_P and \mathbf{F}_L constitutes a multimodal system of interpretation. The ordered pairs $\langle \mathbf{W}, \mathbf{F}_P \rangle$, and $\langle \mathbf{W}, \mathbf{F}_L \rangle$ define, respectively, the model \mathbf{M}_P for the diagrammatic or pictorial language, and the model \mathbf{M}_L for the language of arithmetic. The functions ρ_{G-P} and ρ_{P-G} define homomorphisms between \mathbf{G} and \mathbf{P} as basic and composite terms of these two languages can be mapped into each other. The interpretation of expressions of \mathbf{G} in relation to the world can be computed by two routes: the translation into \mathbf{P} through ρ_{G-P} and the model \mathbf{M}_P or, alternatively, the translation to \mathbf{L} through ρ_{G-L} and the model \mathbf{M}_L . The mapping ρ_{G-P} is illustrated in Figure 4.4.

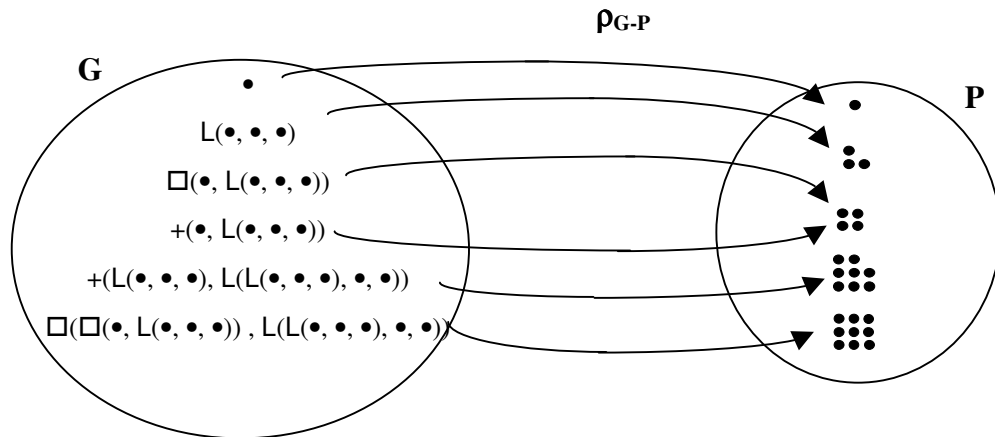


Figure 4.4 Mappings between G and P

Expressions of \mathbf{G} also have an interpretation in relation to the diagrammatic world \mathbf{P} . This second interpretation captures the intuition that diagrams can be thought of not only as representational objects, but also as entities of a world that can be represented. Under this interpretation, expressions of \mathbf{G} are representations of diagrams in \mathbf{P} , and operators of \mathbf{G} denote geometrical algorithms that check whether the operator’s arguments conform to the shape of operator’s value. \mathbf{F}_G in Figure 4.3 stands for this second interpretation function and the ordered pair $\langle \mathbf{P}, \mathbf{F}_G \rangle$ is the model \mathbf{M}_G for the graphical language in relation to the world of diagrams.

The language \mathbf{G} defines “•” as a constant symbol, and its operators are the symbols “L”, “+” and “□”, in addition to the equality sign. The “L” operator has three arguments which are an object of type L (a single dot at the origin is also of type L), and two dots which extend both edges of the L (upwards and rightwards), and produces an object of type L that is two units longer than the one from which it is built (the first argument). The “□” operator takes an object of type *square* (a single dot is also a square) and an object of type L that lies next to the square, on the left, and produces a square whose side is the same as the legs of the component L. The “+” operator takes two objects, a sequence of consecutive L’s (a single dot is considered the basic sequence of L’s) and an L on the left of the first argument, and produces a composite object of type L^* as illustrated in the fifth expression in Figure 4.4. The translation function ρ_{G-P} can be thought of as a drafting interpreter which draws the picture on the screen or a piece of paper which corresponds to a well-formed expression of \mathbf{G} . On the other hand, graphical expressions of \mathbf{P} can be translated into the corresponding expressions of \mathbf{G} through the application of ρ_{P-G} , which is like “seeing the diagram”.

Expressions of \mathbf{G} have also a translation into the language \mathbf{L} , as illustrated in Figure 4.5. The mapping ρ_{G-L} defines a recursive translation from expressions of \mathbf{G} into \mathbf{L} in a simple fashion. Constants and operators of \mathbf{G} have an associated symbol in \mathbf{L} , and the translation of composite expressions is achieved by applying the translation of the functor to the translation of the arguments. As can be seen in the figure, the translation of “•” is $s(0)$. The translation of both “+” and “L” is a function that, when applied to the translations of the arguments, produces the corresponding expression of \mathbf{L} , which is a sum. The translation of “□” is a function that maps square expressions of \mathbf{G} into successor expressions of \mathbf{L} . The system also includes the equality sign to allow the expression of axioms and theorems. The last expression in Figure 4.5, for instance, expresses that a sequence of the two smallest L’s of the system is equal to the square of size two,

and it is translated into the equation $1 + 3 = 4$. The interpretation of symbols of \mathbf{L} is given directly in terms of the model defined with the natural numbers and the interpretation function \mathbf{F}_L in the standard way: it is the language of arithmetic. The system of multimodal interpretation as a whole defines an interpretation that corresponds to the interpretation a human interpreter would make when looking at the symbols if he or she knows how to interpret the notation. The full formalisation of the syntax, semantics and translation relations between all three languages of the scheme is presented in Appendix 1.

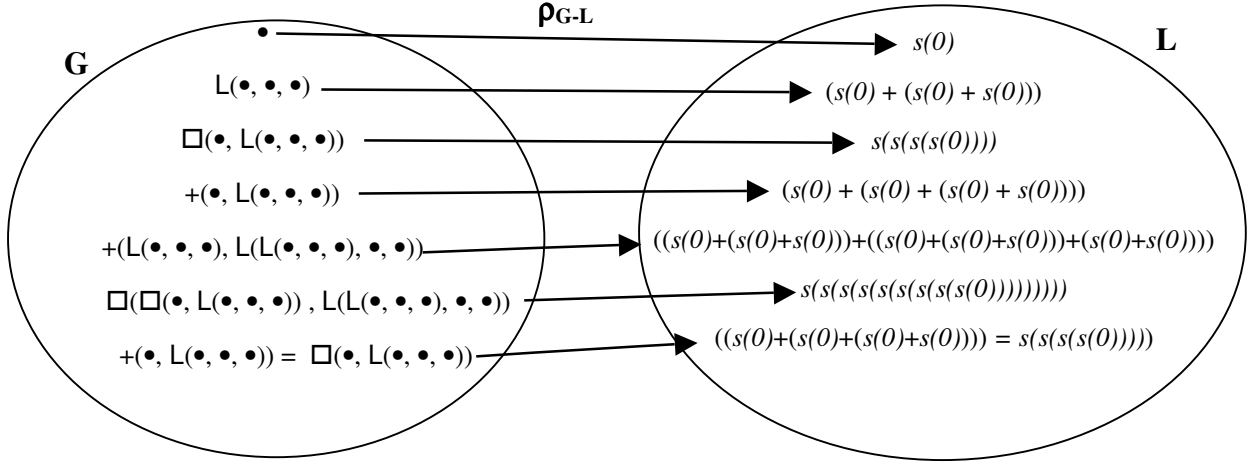


Figure 4.5 Translation between G and L

With this machinery in place, we can proceed to define a theorem-proving system which is able to produce the proof in Figure 4.2, but first we consider the question of how the theorem can be expressed or input to the theorem-prover. This can be achieved by typing the theorem directly as an expression of \mathbf{G} , or alternatively, by expressing the diagram directly (e.g. through the picture in 1.4) and interpreting it through a graphic interactive facility or through vision and learning processes, or through a combination of these two. In the traditional logicist view of AI the first alternative is a matter of course, but if our goal is to capture the essence of diagrammatic reasoning, the issue is the relation between seeing the diagram, noticing that it represents a theorem, and also a proof of the theorem which happens to be valid. Additionally, understanding this relation probably cannot be done without taking into account pragmatic issues related to reasoning *with* and reasoning *from* a system. Here, in order to clarify the process of proving a theorem when it has already been learned, we assume that the expression of \mathbf{G} representing the theorem is available directly. Later on, in Section 6, we discuss some aspects of the relation between expressing, learning and proving the theorem using diagrammatic means.

The main steps of the proof as well as their corresponding translations into \mathbf{L} are shown in Figure 4.6. In the proof procedure, the transformation rules applied to expressions of \mathbf{G} reflect the graphical transformations of the diagrammatic proof in Figure 4.2. Once the inductive hypothesis has been obtained from the original theorem, the square on the right hand side of the graphical equation (2) in Figure 4.6 is decomposed into a square and an L, as shown in the right hand side of equation (3). This substitution can be expressed as a diagrammatic production rule of the diagrammatic theorem-proving system as follows:

$$\square^n \Rightarrow \square^{n-1} + L^n$$

where

$$L^1 = \bullet_{x,y} \text{ and } L^n \text{ stands for an L of size } 2n-1, n>1;$$

$$\square^1 = \bullet_{x,y} \text{ and } \square^n \text{ stands for a square of size } n, n>1.$$

The transformation from eq. (3) to eq. (4), which completes the proof, is achieved by the elimination of the same term whenever it appears on both sides of an equation, in a normal symbolic fashion. As these production rules preserve “shape”, they can be considered sound inference rules in a strict logical sense, and the system could be proved sound and complete in relation to these axioms, as has been done for Venn diagrams by Shin [Shin, 1995].

(1)	$\bullet + L(\bullet, \bullet, \bullet) + L(L(\bullet, \bullet, \bullet), \bullet, \bullet) + \dots + L^n = \square^n$ $\Rightarrow s(0) + s(s(s(0))) + s(s(s(s(0)))) + \dots + s^{2n-1} = s^{n^2}$
(2)	$\bullet + L(\bullet, \bullet, \bullet) + L(L(\bullet, \bullet, \bullet), \bullet, \bullet) + \dots + L^n + L^{n+1} = \square^n + L^{n+1}$ $\Rightarrow s(0) + s(s(s(0))) + s(s(s(s(0)))) + \dots + s^{2n-1} + s^{2n+1} = s^{n^2} + s^{2n+1}$
(3)	$\bullet + L(\bullet, \bullet, \bullet) + L(L(\bullet, \bullet, \bullet), \bullet, \bullet) + \dots + L^n + L^{n+1} = \square^{n+1}$ $\Rightarrow s(0) + s(s(s(0))) + s(s(s(s(0)))) + \dots + s^{2n-1} + s^{2(n+1)-1} = s^{(n+1)^2}$

Figure 4.6 Proof in G of the theorem of the sum of the odd numbers and its translation into L

The power of the representational system can also be assessed by noticing that the translation rules between **G** and **P** ensure that the proof is isometric to the corresponding diagrammatic one, and the translations between **G** and **L** ensure that the proof corresponds also to the standard arithmetic proof. Through these translations, and bearing in mind that all three languages have a model relative to the same world, the isomorphism between diagrammatic and algebraic proofs is established.

4.3 Comparing the two approaches: Expressing and proving a theorem

The two models presented above emphasise different aspects of diagrammatic reasoning; while the first is more concerned with the diagrammatic expression of the theorem, the induction and verification stages are in no sense diagrammatic. The representational structures and graphical proof procedures of the second approach, on the other hand, resemble better the drawings and drafting transformations required to visualise the proof by people with pencil and paper; consequently, it captures better the intuitive notion of truthfulness that humans find easy to see and understand, and also makes explicit the relation between diagrammatic and algebraic proof. In particular, the effectiveness in understanding the diagrammatic depends on size and possible moves in the search space of the proof process, as graphical transformation can be seen as an abstraction of a large sequence of algebraic transformations. The graphical proof in Figure 4.2 and its representation in the language **G** in Figure 4.6, for instance, is produced by only two transformations while the arithmetical counterpart requires a larger number of more primitive symbolic transformations, some of which are shown in Figure 4.1, that depend on the form of the algebraic expressions⁴.

However, this second approach assumes that the expression representing the theorem is given, and many questions about how the theorem can be expressed by graphical means still remain. In particular, the mapping ρ_{P-G} has not been specified, and depending on particular research questions, several ways to look at it are possible. In particular, if we think of Diamond in terms of our multimodal system of representation, the geometrical operations used to specify a theorem through the graphical interface and the inductive learning technique can be thought of as a particular implementation of such a translation function. In such a specification ρ_{P-G} would be a function mapping a sequence of geometrical operations to the recursive description of the general pattern. However, alternative settings to express the theorem graphically without relying on inductive learning can be conceived. For instance, a graphical interface supporting graphical cursors with square and L shapes, associated to the operators “ \square ” and “L”, as in Diamond, but augmented

⁴ In a related investigation involving 3-D diagrammatic reasoning we have studied how to produce isometric views of polyhedra from their orthogonal projections using similar kinds of graphical languages and graphical operators. Although a full implementation of that system is still pending, current results show that the search-space for the problem-solving task is small [Garza and Pineda, 1998].

with a symbol for expressing ellipsis, could map graphical input sequences into expressions of \mathbf{G} directly. With such a kind of interactive or interpretation technique, the diagram in Figure 4.7, for instance, could be interpreted as the theorem. This kind of interface is likely to have a very specific character as it would have to be designed and built taking into account the syntactic structure of the target representational structure, \mathbf{G} in our case, and a large number of pragmatic assumptions about the nature of the task. In the case of Diamond, for instance, the human-user needs to know that the input expected by the system is a sequence of graphical patterns to be interpreted as concrete instances of the theorem to be proved.

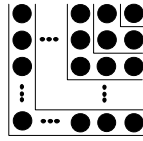


Figure 4.7 Expressing the theorem

Another way to conceive $\rho_{\mathbf{P-G}}$ is to think of the drawing as a bidimensional expression on a grid to be interpreted using a graphical grammar and parser [Wittenburg, 1998]. However, the use of this latter approach can be conceived better within the context of a much more complex computer vision system in which the symbolic representation of the theorem would be induced not only by bottom-up processes acting upon the external input but also by top-down processes projecting conceptual expectations about arithmetic theorems into the diagram. Considering the particular combination of graphical interactive input facilities and the inductive learning techniques of Diamond in terms of $\rho_{\mathbf{P-G}}$ suggests that the effective use of grammars involves vision and learning processes in very cumbersome ways, in addition to the diagrammatic process. To appreciate this better, suppose that the input sequence in Diamond were not provided by a human-user but by a fact-finding process in a learning system designed to learn arithmetic theorems from a particular diagram. If such a program were able to learn not only the theorem of the odds, but other theorems represented by the same diagram, as will be discussed below, it would be a very interesting discovery and learning system. Furthermore, it would only make use of the diagrammatic theorem-prover once the theorem had been discovered. In the light of this reflection we speculate that Diamond's diagrammatic inferences are more concerned with learning theorems than with proving them, as should be expected given that a non-demonstrative inference such as learning induction is very different from mathematical induction which is demonstrative.

We conclude the comparison between the two approaches by mentioning that both use diagrams for predicate extraction, the first to collect the input for its inductive learning process, and the second to carry out the proof with very few graphical transformations, using very effectively the geometrical algorithms associated with the geometrical operators of the representational language. However, as both of the approaches investigated here are propositional in character, the question of whether it is possible to simulate human diagrammatic reasoning directly requires further investigation. Next, in Section 5 we present a refined system in which the language \mathbf{G} is eliminated when a computational process for reasoning directly with the expressions of \mathbf{P} is specified, and examine the consequences of such a system.

5. Diagrammatic theorem proving

5.1 A generalised scanning device and visualisation

An antecedent for the following discussion is Funt's system for solving problems about mechanical relations among rigid bodies [Funt, 1980]. This program used a *retina* to inspect a diagram represented as an array (i.e., a memory buffer, in which different colours of pixels represent different bodies). The retina is constructed from a number of processors arranged in rings and aligned by rays (wedges) coming from the centre of the circular structure, as shown in Figure 5.1. Each processor in the array was able to read the colour of the pixel behind it, and to communicate with its immediate neighbours both in the ring and the wedge. All processors could also communicate with a central processor, called the supervisor, through a common bus. The retina's basic functionality was used to implement a number of "perceptual operations" such as: finding the centre of area of a body; simulating the rotation of a body; scaling a body; finding the collision point between two bodies; finding the nearest processor to the retina's centre satisfying a given condition; and detecting similarity between two different bodies. Particularly interesting is the visualisation of rotation: to rotate a body by a given angle, each processor reading the colour of the body sends a message to the next in the ring

after erasing its own colour, until the sought angle is reached. Collisions can be easily identified: if a processor receives the colour of a body that is being rotated when it is holding the colour of a different body, there is a collision between the two bodies at the point. In Whisper it was also possible to position the centre of the retina on an arbitrary part of a diagram, to inspect the diagram closely, as the processors in the centre of the retina had a better resolution than the processors in the outer rings.

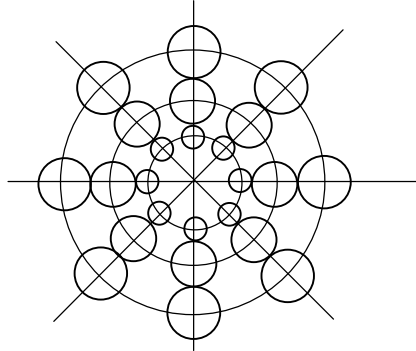


Figure 5.1 Whisper's retina

A program with knowledge about instabilities of rigid bodies, called the High-Level Reasoner (HLR), in Whisper, was able to use the retina's primitives to produce a simulation of the collapsing process. The relation between the Whisper's HLR, the retina and the diagram, is shown in Figure 5.2.

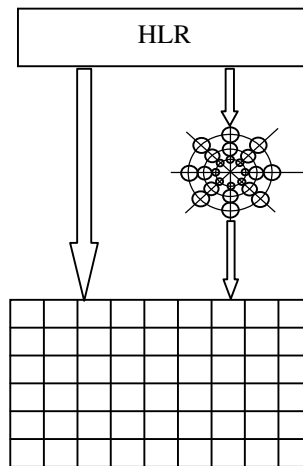


Figure 5.2 The architecture of Whisper

This architecture can be contrasted with the architecture of a Turing Machine described in section 2, as shown in Figure 2.2. The high-level reasoner would correspond to the algorithm used internally by the Turing Machine where the abstract process takes place, the retina would correspond to a generalised scanning device and the diagram would correspond to the external representation. Whisper's direct link between the HLR and the diagram can be thought of as the hand of the agent in which the computation as a whole is embedded, which performs the action of modifying the diagram.

The problem-solving strategy was to simulate the dynamics of rigid bodies when they were subjected to mechanical forces such as gravity. Through the simulation, Whisper was able to draw a new diagram after a change had taken place, and use it as the new object seen by the retina. The simulation was a qualitative process (i.e., one that did not use numerical methods to solve differential equations describing the process) and did not use a symbolic externalised language for representing and reasoning about the diagram. The claim

is not that the states of affairs expressed diagrammatically cannot be expressed logically, but only that although the actual scanning device operates on symbols in the grid, in the same way that linear strings of symbols are normally scanned in a Turing Machine to model logical (symbolic) computations, the objects “scanned” by the high-level reasoner, the abstract process, are symbols representing individual objects of the scene domain. The individualisation of a bunch of pixels into a shape is achieved by the perceptual primitives implemented in the retina, but in a process that is controlled by the HLR. The fact that shapes are produced by local processes acting on the grid cells is not inconsistent with thinking of shapes and diagrams as constants and composite terms of well-defined languages. To achieve the simulation, symbols denoting the shapes represented diagrammatically must be defined internally in the HLR of Whisper, and an account of this level of description can be given compositionally. Compositional semantics provides us with a tool to describe a type of expression (such as diagrams) and its interpretation process at a very high level of abstraction, but does not commit us to the view that the application of semantic rules is psychologically real.

In Chandrasekaran’s terminology, diagrams are used in Funt’s program through predicate extraction and simulation. The process that implements such manipulations can be thought of as an abstraction, but the crucial fact is that the definition of the algorithm depends on the particular properties of the computational retina. The process of reasoning about the dynamics of rigid bodies can be implemented as a different algorithm, even a logical one as in qualitative physics, but this particular algorithm for implementing the simulation depends on the particular choice of media and notation, and the properties of the generalised scanning device.

5.2 Visualisation and diagrammatic proofs

The question that we raise here is whether it is possible to think of the diagrammatic proof of the theorem of the sum of the odds in Figure 4.2 in such a way that the diagram is not used by predicate projection but by a process in which the external representation is used directly, as in Whisper’s simulations. The novelty consists in employing a diagram for representing abstract objects such as natural numbers and proofs, rather than rigid bodies. The proof would be diagrammatic because it would be a simulation of the proof in which the process applying the inductive hypothesis and the simplification of the diagram would act on the concrete representation of the theorem on the grid, and the transformation would take place in the retina’s parallel processors. However, to show that such a process is possible we have to provide a representational system in which diagrams can be interpreted as natural numbers, and the relation between diagrammatic and arithmetic expressions is also established systematically. In particular, we require to develop a multimodal system of representation similar to the one used in our proof of the theorem in Section 4.2, but one in which the relation between the diagrammatic language and the language of arithmetic is direct, and the language **G** is removed altogether from the representational system; such a multimodal representational system is illustrated in Figure 5.3.

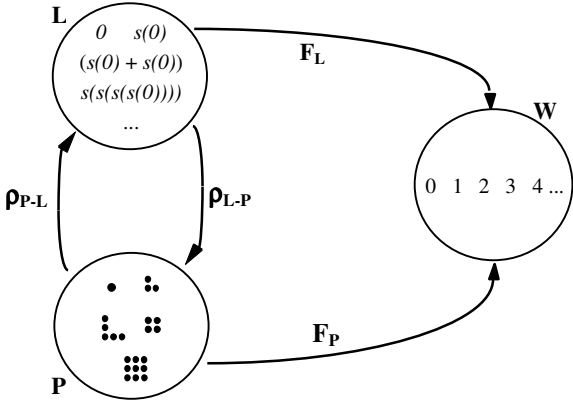


FIGURE 5. 3 Refined multimodal system of representation.

We can now relate this diagram to the architecture of the Turing Machine in Figure 2.2, and the architecture of Whisper in Figure 5.2. **L** and **P** are the language of arithmetic and a pictorial language as before. The

translation relations ρ_{L-P} and ρ_{P-L} establish an isomorphism between the languages and permit meaning preserving translation between basic and composite expressions of both languages⁵. The abstract process that uses expressions of either of these two languages is embedded in the algorithmic box of the Turing Machine, or in the corresponding HLR box of Whisper’s similar architecture.

Now we proceed to describe a computational process to prove the diagrammatic proof of the theorem of the sum of the odds in Figure 4.2 through visualisation with a Turing Machine with a grid and a generalised scanning device. Knowledge of mathematical induction is kept as an abstract process in the HLR. The HLR understands the notation and knows the theorem already as discussed before. The original diagrammatic representation on the grid is any instance of the diagram. It is interesting to note that all instances of the theorem — any square full of dots — have the same logical properties in relation to the notation, and one can define an equivalence relation between diagrams, similarly to Shin’s counterpart relation for defining a class of Venn diagrams that differ in shape but mean the same in relation to the domain [Shin, 1995; Hammer, 1995]. Then, a relation holding in the concrete case would hold for the whole of the class, and that would prove the theorem. However, under this assumption the proof of the theory would consist in demonstrating graphically that all diagrams in the class do belong to such a counterpart relation, and an inductive argument would still be required.

We consider now how the simulation can be produced on the retina’s array of processors. For simplicity we assume that the retina’s shape is also a grid, and there is a processor for inspecting every grid of the diagram. Also, each processor can communicate with its eight surrounding neighbours in the array. To simulate the process, several procedures can be designed. We exemplify one possible implementation through a process consisting of three main parts: (1) read the square, (2) visualise the theorem and (3) apply the inductive hypothesis. To perform (1), a perceptual primitive of the retina is used by the HLR system to detect the square, and each processor inspecting cells on the square’s spatial extension stores a “1” in its local memory, and the square of size n is read by the HLR; the local memory of all processors in the retina is switched off. In (2) a sequence of numbers in “L notation” is visualised in the retina and compared with the visualisation of a number in “square notation”. To do the former, the HLR visualises a sequence of n Ls in the retina (e.g., by instructing the processors in the top row of the square to switch on and propagate the message through its lower side, and similarly for the processors on the right side of the square, which would propagate to the left; when one processor receives a message from the top and right side, it is instructed to switch on but to stop the process) and compares this with the object recognised in (1). Then, it visualises a square of size n in the retina, and compares that with the object recognised in (1). If both of these tests succeed, then the n th instance of the theorem is recognised. To apply the inductive hypothesis, the HLR system increases the parameter n by one, and repeats (2). If the operation succeeds, the inductive proof by visualisation is completed. Note that in this implementation the induction is visualised, but the external representation is never altered.

Processing expressions of \mathbf{P} with a computational retina in the context of the multimodal system of representation provides an illustration of how a distributed process can be given a compositional semantics. Each processor computes a simple function with local information and yet the distributed computation as a whole can be given a meaning: the graphical symbols and transformations are interpreted as the process of performing a mathematical inductive proof.

It is interesting that Stenning and Oberlander have argued that diagrams in a graphical argument stand in an ordered relation that can be thought of as an “animation”. Animation introduces temporal specificity, and temporal states of the proof that can be easily visualised have to be proximate in this order. It has also been argued that this kind of continuity of structural description underlies people’s ability to solve problems through “visualisation” [Hinton, 1979, 1980].

⁵ Note that the fact that \mathbf{L} is a “propositional representation” does not mean that it has no external aspect, as symbols written on a piece of paper are always external. The abstract process performing the proof with expressions of \mathbf{P} is as “internal” as the process acting on expressions of \mathbf{L} , and expressions of both \mathbf{P} and \mathbf{L} are written on the external tape of the Turing Machine, and can be inspected with a generalised scanning device.

To conclude this section, we consider what kind of representational system the square representing the theorem should be, according to the theory of graphical specificity. As a first approximation, one can say that the square is a MARS because the square denotes a specific square number, and the Ls specific odd numbers, as in Jamnik's inductive system where the squares input by the human user are meant to represent concrete instances of the theorem. However, if all dots in an L shape representing an odd number are thought of as a bidimensional monadic notation for numerals, the dependencies of objects in different cells in the Ls and squares have to be stated using notational keys. The key for the Ls states that *any* L shape represents an odd number, the number of dots in the L, hence an abstraction, and similarly for the squares, as illustrated in Figure 5.4.

Ls NOTATION		SQUAREs NOTATION	
Syntax	Semantics	Syntax	Semantics
●	1	●	1
● ●●	3	●● ●●	4
● ●● ●●●	5	●●● ●●● ●●●	9
● ● ⋮ ●...●●	$2n-1$	●...●● ●●● ⋮ ●...●●	n^2

Figure 5.4 Bidimensional monadic notations

Consequently, the tabular form filled in with dots with these notational keys is best thought of as a LARS. It is not a UARS, because there are no assertional keys. Furthermore, if the notational keys for the interpretation of Ls and squares permit such limited abstraction over an infinite number of cases, then a single square representing an instance of the theorem stands for all instances of the theorem. Accordingly, reasoning with this concrete object abstracts over all the cases that would have to be considered to assess the validity of the theorem otherwise. However, the syntactic reflex producing the efficacy of the representation is very likely to be related to the perceptual modality, like the visualisation supported by the retina and its perceptual operations. Furthermore, as a particular diagram abstracts over all diagrams, the inductive step (3) in the proof by visualisation is not required. For people to verify the theorem, it is only necessary to visualise *any* square of dots under both of the notations. We speculate that this is why people are able to verify the truth of the theorem of the sum of the odds so effectively.

6. Pragmatics and notational change: Learning and graphical proof

In the practice of mathematics and logic, proving theorems that are assumed to be true is the main concern. In the algebraic proof of the theorem of the sum of the odds, for instance, the hard work consists in developing an argument by mathematical induction. How we get to know the theorem is an issue which is not even considered. Such questions about creativity are not for mathematicians or logicians. In the diagrammatic setting, on the other hand, the situation is reversed. Verifying the truth of the theorem is trivial if the notation is known and the diagram is interpreted as a LARS, as one knows that a single diagram abstracts over all instances of the proof and, consequently, it represents the proof. However, learning this abstraction requires an inductive learning inference, and to model this latter process is by no means trivial. Interestingly enough, in the Diamond system, although the intention is to model diagrammatic proofs, the hard work consists in expressing and learning the theorems, and the proofs themselves are not even claimed to be diagrammatic. And this is right, because in the case of this graphical theorem, learning the theorem is the issue. There are other graphical reasoning situations in which the graphical proof procedure does elucidate facts that are hard to appreciate at first sight, as in syllogistic reasoning by graphical means, but even in those situations the

question of why the concrete sequence of diagrams employed in the proof stands for the proof, and not just for an instance of the proof, should be answered.

To address this question, we can exploit two perspectives already discussed in section 3. One is to take a view entirely *within* a system and the other is to adopt a perspective *from* the system. The Diamond system, for instance, sits within the first perspective and adopts the view that learning induction can be used to produce the generalisation from a number of concrete cases. However, the question of whether a concrete diagram represents the proof is not even raised, and indeed, Diamond learns the theorem but no diagram in its setting represents the theorem. However, the process looks rather different if we take the second perspective and reason about the notations of the representational system. To state the notational key for L shapes we do not say that the basic L of size one (the top right dot) represents the number one, and that the L of size 2 represents the number three and so on, and assert an infinite number of notational keys. We just say that any L represents an odd number, which is two times the side of the L plus one; similarly for the square. And there are facts that follow from the choice of notation and the abstraction it embodies. One such fact is that a sequence of Ls aligned together make up a square, and as this syntactic fact is general, it has a semantic consequence that is also general. When we realise a relation between the two notations, all semantic consequences of the notations are involved in the relation. We could have stated a third representational key to the effect that *any* pattern of Ls arranged as *any* square represents *some* relation between the number represented by such a pattern of Ls and the number represented by such a square. In the case at hand, the sequence of Ls represents the sum of the numbers represented by the Ls in the pattern, and the relation is equality. To learn the abstraction that any square represents the theorem is to learn this *assertional* key out of the *notational* keys that were given when the system of representation was originally introduced.

To stress that a diagram represents a proof only if it is interpreted with a particular notation in mind, notice that the same diagram can be read as representing different proofs. In the case of the squares, suppose that we define a notation that we call monadic diagonal notation such that a diagonal of n dots represents the number n . Then, a square of size n represents the theorem $1 + 2 + \dots + n + (n - 1) + \dots + 2 + 1 = n^2$, which also appears in Nelsen's book [Nelsen, 1993]. But in order to read the square as this and no other theorem, the notational key stating the monadic diagonal notation has to be stated, and the corresponding assertional key has to be induced to see the drawing as a LARS, and then as a representation of the theorem. The same external representation but with different representational keys expresses a different abstraction.

There is no reason why a string like " $1 + 2 + \dots + n + (n - 1) + \dots + 2 + 1 = n^2$ " can express a theorem while the graphical counterpart cannot. The ellipsis sign " \dots " is a notational device that, in the context of the other symbols, expresses the abstraction, but the abstraction is never in the external representation: it is always in the mind. To know that a collection of marks on a piece of paper expresses a theorem is to know the notation; the notational keys, which are given, and the assertional keys, which are realised when the interpreter realises the theorem. As Wittgenstein once put it: "The sentence is 'elliptical', not because it leaves out something that we think when we utter it, but because it is shortened" [Wittgenstein, 1953, §20].

Learning systems conceived from within a system of representation cannot reason about the properties of its own syntactic structure or semantics, the shape of the representation medium and notation, as these are given from above the system. Nevertheless, when an agent needs to learn the solution to a problem, rather than verifying one, her task can focus on expressing the problem in different ways and trying to read the solutions from these representations. Good choices of notation and medium will make it easier to express the problem and find the solution. To learn arithmetic theorems, the learner has a few choices of notation and media at hand, from which the search space for the learning problem can be built. If a problem can be expressed easily in a representation, the likelihood of finding the solution easily would be high, and vice versa. It would be cumbersome to compute the sum of, say, the first ten odd numbers if one is restricted to use decimal notation and a linear tape, even with external aids like a pencil and a piece of paper. To realise that the sum of the first n odd numbers is n^2 from an arithmetic representation is only likely when one has performed a few such sums and noticed the pattern, but for the diagrammatic case, when a graphical pattern is looked at with the notations in mind, the theorem is read off from the representation directly. If a person is asked what is the sum of the first ten odd numbers, but is also asked to express the problem in an L shape monadic notation, as soon as she writes down the sum and observes, from the resulting shape, that the expression can be reinterpreted with a different notation, the solution can be read directly from this interpretation shift. If the interpreter is a computational device, changing the interpretation means changing

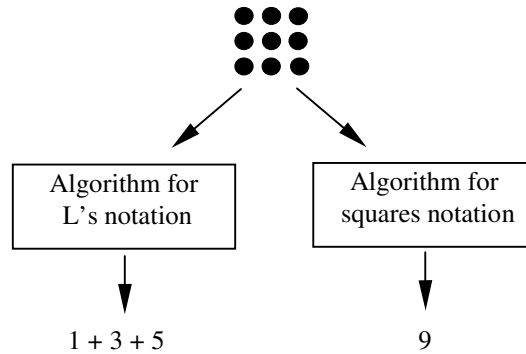


Figure 6.1 Change of algorithm during the interpretation

the algorithm whereby the “tape” of the Turing Machine is read. This is illustrated in Figure 6.1. The visualisation discussed in Section 5.2 might look trivial if the aim is to verify the theorem, but if the same process is performed as a part of a learning task, before the relation between the notations has been realised, the task is much more interesting. The theorem is recognised when the diagram is reinterpreted as a construction representing an abstraction. For people, learning the abstraction that the figure represents the theorem and proving the theorem are really one and the same thing.

To end this discussion we consider now the proof of the Theorem of Pythagoras. First a comment on the proof in Figure 1.2, which is the one that appears as proposition 47 in Euclid’s Geometry. The figure is not really a proof but just a diagram that is used as a reference for the geometric argument, a symbolic one, which is the real proof. Here, we can simply say that the diagram is used for predicate extraction and projection, especially if the diagram is constructed along the development of the argument. We turn now to the proof presented in Figure 1.1, which is repeated for clarity below in Figure 6.2. Bronowski has speculated that this was the proof through which the theorem was first discovered by Pythagoras [Bronowski, 1981]. As can be seen, the construction is performed out of a right triangle which is rotated and translated four times to form a square on the hypotenuse. The length of the side of the square inside the construction is the difference between the lengths of the two right sides of the triangle. In the last state, the triangles are rotated, and from the resulting configuration two squares emerge. Each of these squares is on each of the right sides.

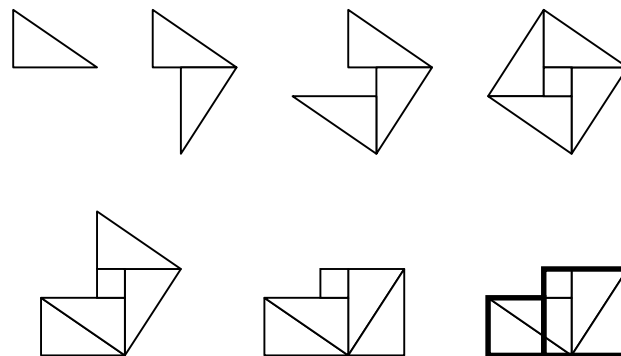


Figure 6.2 The Theorem of Pythagoras

The intuition of the validity of the proof is overwhelming, and nevertheless, some people would argue that it is not formal, and it does not constitute a valid argument. In a recent discussion on the Internet about another diagrammatic proof of the theorem, for instance, it was argued that the diagrammatic sequence would constitute an individual instance of the proof. This issue can be addressed on the light of the present discussion. First, we consider whether the triangle in the initial state of the construction stands for itself, an individual concrete object, or stands rather for any triangle, for the class of all triangles; similarly for the squares involved in the proof. The answer depends on the representational key. If people are looking at the

diagram without a notation in mind, the diagram is not a representation at all. If a child looks at the sequence in Figure 6.2 without thinking of geometry, the picture does not represent anything. If the notation is that the triangle stands for a concrete individual object, a notational key expressing complete information, then the whole construction stands for a concrete instance of the proof, as there is no abstraction involved. In that reading, the initial triangle and the whole of the construction is a MARS. In particular, the assertion that the area of the square emerging as the square on the hypotenuse is the same as the area of the two squares on the right sides of the triangle is a particular assertion holding only for the particular construction. However, if the key says that the initial triangle stands for *any* (right-angled) triangle, then it represents the class of all such triangles, and the particular geometric properties of the triangle are also part of the abstraction. In this latter situation, the graphical representation, the diagram, is a LARS. In this reading, realising that the area of *any* square on the hypotenuse of *any* right triangle is the same as the area of *any* two squares on the right sides of the same triangle, the statement of the theorem, is to learn an assertional key. To learn the abstraction, this assertional key has to be induced from the notational key and a general geometrical property: the fact that the transformations between the state in which the square of the hypotenuse emerges, and the state in which the squares on the right sides emerge, preserve the area. Realising this abstraction, seeing the sequence of figures as a representation of the theorem, is learning the theorem; and people really do see the theorem. But suppose for the sake of argument that some people look at the diagram as a MARS (unlikely as it might seem), say during the process of understanding the proof: once the truth of the theorem is revealed, then there has to be an interpretation shift, a change of the notational key, and from then on the diagram is read as a LARS. If the theorem is first learnt through this proof, learning and proving the theorem are again one and the same thing.

The shapes of this proof can be produced through a simple production system. In the shape-grammar formalism introduced in the context of design [Stiny, 1975], for instance, a grammar has been developed which produces configurations out of right triangles, and produces all the triangle shapes of the proof [Knight, 1981]. However, since this grammar manipulates triangles only, the squares emerging in the essential stages of the construction are just contingent side effects, which are not a part of the syntactic structure of the corresponding shape-grammar sentence. These squares are not produced from *within*, and the triangle's grammar cannot define the search space for the proof process. However, an agent looking at the diagram sequence can recognise the proof because she is reasoning not within but *from* the system. To recognise both the square on the hypotenuse and the inner square which emerge in the same state of the construction is a free ride. However, to realise that the squares on the right sides emerge in the last stage of the sequence is not so easy. It is easy to see that the L shape in the last stage of the sequence has the same area as the square on the hypotenuse, but it is not so easy to see that the L can be reinterpreted as two squares lying one beside the other, and that their sides correspond to the right sides of the triangle. To perform this latter interpretation shift, and see the theorem as a whole, is at best a cheap ride, or possibly even an expensive one. This is because the last reinterpretation requires the agent looking at the construction to restructure five geometrical figures — the four triangles and the little square — into the two new squares. As this reinterpretation does not appear in the diagram, it has to be visualised. We are fortunate that our retina has such a preference for square notations. This visualisation is rather stable but sometimes notations can be brought about by rather arbitrary forces, as in the well-known duck-rabbit picture popularised by Wittgenstein [Wittgenstein, 1953, part II, §10], shown in Figure 6.3. For some people, what is revealed in these visualisations might be accompanied by a feeling of emotion.

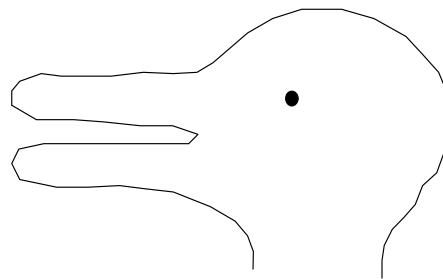


FIGURE 6.3 A notational change

ACKNOWLEDGEMENTS

The authors gratefully acknowledge support to Luis Pineda from the Institute for Applied Mathematics and Systems (IIMAS) at the National University of México (UNAM) and Conacyt grant 400316-5-27948-A, to Luis Pineda and Gabriela Garza from the Institute for Electrical Research (IIE), México, and to John Lee from the Human Communication Research Centre of the UK Economic and Social Research Council (ESRC).

REFERENCES

- [Boolos and Jeffrey., 1990] George S. Boolos, Richard C. Jeffrey. 1990. *Computability and Logic*. Third edition. Cambridge University Press.
- [Barwise and Etchemendy, 1991] John Barwise and John Etchemeny. 1991. Visual information and valid reasoning. *Visualization in Teaching and Learning Mathematics*. Zimmerman W. and Cunningham (eds.) Mathematical Association of America. pp. 9 – 24.
- [Bronowski, 1981] J. Bronowski. 1981. *The Ascent of Man*. (pp. 155 – 188). BBC Corporation, London.
- [Chandrasekaran, 1997]. B. Chandrasekaran. 1997. Diagrammatic Representation and Reasoning: some Distinctions. *Working notes on the AAAI-97 Fall Symposium Reasoning with Diagrammatic Representations II*. MIT, November 1997.
- [Dowty et al., 1985] David R. Dowty, Robert E. Wall and Stanley Peters. 1985. *Introduction to Montague Semantics*. D. Reidel Publishing Company, Dordrecht, Holland.
- [Funt, 1980] Brian V. Funt, Problem-solving with diagrammatic representations. 1980. *Artificial Intelligence* 13: 210 – 230.
- [Garza and Pineda, 1998] E. G. Garza and L. A. Pineda, 1998. Synthesis of Solid Models of Polyhedra Views using Logical Representations. *Expert Systems with Applications*, Vol. 14, No. 1. Pergamon Press.
- [Gelernter, 1963] H. Gelernter. 1963. Realization of a geometry theorem-proving machine. In Feigenbaum, E. And Feldman, J. (eds.). *Computers and Thought*, pp. 134 – 152. McGraw Hill.
- [Glasgow et al., 1995]. Janice Glasgow, N. Hari Narayanan and B. Chandrasekaran (eds.). 1995. *Diagrammatic Reasoning: cognitive and computational perspectives*. The AAAI Press, Menlo Park, California.
- [Gurr et al., 1998] Corin Gurr, John Lee and Keith Stenning. 1998. Theories of Diagrammatic Reasoning: Distinguishing Component Problems. *Minds and Machines* 8: 533 – 557.
- [Haack, 1978] S. Haack. 1978. *Philosophy of Logics*. Cambridge University Press.
- [Hammer, 1995] Eric M. Hammer. 1995. *Logic and Visual Information*, CSLI Publications, Centre for the Study of Language and Information, Leland Stanford Junior University.
- [Hinton, 1979] G. Hinton. 1979. Some demonstrations of the effect of structural descriptions in mental imagery, *Cognitive Science*, 3: 231 – 250.
- [Hinton, 1980] G. Hinton. 1980. Frames of reference and mental imagery. In J. Long and Baddeley (eds.). *Attention and Performance*. Hillsdale, NJ: Erlbaum.
- [Jamnik et al., 1999] Mateja Jamnik, Alan Bundy and Ian Green. 1999. On Automating Diagrammatic Proofs of Arithmetic Arguments. *Journal of Logic Language and Information* 8, 3: 297–321.
- [Johnson-Laird, 1983] Johnson-Laird, P.N. 1983. *Mental Models: towards a cognitive science of language, inference, and consciousness*. Cambridge: Cambridge, University Press.
- [Klein and Pineda, 1990] Ewan Klein, Luis Pineda. 1990. Semantics and Graphical Information. *Human-Computer Interaction, Interact'90*. pp. 485-491. Diaper, Gilmore, Cockton, Shackel (Eds). IFIP, North-Holland.
- [Knight, 1981] T Weissman Knight. 1981. Languages of designs: from known to new. *Environment and Planning B*, 8:213 – 238.

- [Larkin and Simon] J. Larkin and H. Simon. 1987. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science* 11:65-99.
- [Levesque, 1988] H. J. Levesque. 1988. Logic and the complexity of reasoning. *Journal of Philosophical Logic*, 17: 355 – 389.
- [Nelsen, 1993] R.B. Nelsen. 1993. *Proofs without Words: Exercises in Visual Thinking*. The Mathematical Association of America.
- [Norman, 1994] Donald A. Norman. 1994. *Things That Make Us Smart : Defending Human Attributes in the Age of the Machine*. Perseus Press.
- [Pineda, 1989] Luis Pineda. 1989. Graflog: a Theory of Semantics for Graphics with Applications to Human-Computer Interaction and CAD Systems. PhD thesis, University of Edinburgh, U.K.
- [Pineda, 1992] Luis Pineda. 1992. Reference, Synthesis and Constraint Satisfaction. *Computer Graphics Forum*. Vol. 2, No. 3, pp. C-333 - C-334.
- [Pineda and Garza, 1999] . Luis Pineda and Gabriela Garza. A Model for Multimodal Reference Resolution. To be published in *Computational Linguistics*.
- [Prior, 1960] A. N. Prior. 1960. The runabout inference ticket. *Analysis* 21.
- [Reiter and Mackworth, 1987] Raymond Reiter and Alan K. Mackworth. 1987. The Logic of Depiction. *RCBV-TR-87-18: Technical Reports on Research in Biological and Computational Vision*. Department of Computer Science, University of Toronto.
- [Santana, 1999] Julio Sergio Santana Sepúlveda. 1999. The Generation of Coordinated Natural and Graphical Explanations in Design Environments, PhD thesis, Universidad de Salford.
- [Shimojima, 1996] Shimojima, A. 1996. Operational constraints in diagrammatic reasoning. In J. Barwise and C. Allwein (eds) *Logical Reasoning with Diagrams*. New York: Oxford University Press. Pp. 27- 48.
- [Shin, 1995] Sun-Joe Shin. 1995. *The Logical Status of Diagrams*. Cambridge University Press.
- [Stiny, 1975] G. Stiny. 1975. *Pictorial and Formal Aspects of Shape and Shape Grammars*. Basel, Birkhauser Verlag.
- [Stenning and Oberlander, 1995] Keith Stenning, Jon Oberlander, 1995. A cognitive Theory of Graphical and Linguistic Reasoning: Logic and Implementation, *Cognitive Science* 19: 97–140.
- [Wang and Lee, 1993]. Dejuan Wang and John Lee. 1993. Visual reasoning: Its formal semantics and applications. *Journal of Visual Languages and Computing*, 4: 327–356.
- [Wittenburg, 1998] Kent Wittenburg 1998. Visual Language Parsing: If I had a Hammer In Harry Bunt, Robbert-Jan Beun and Tijn Borghuis (eds.), *Multimodal Human-Computer Communication: Systems, Techniques and Experiments*, pp. 231-249. LNAI 1374, Springer-Verlag.
- [Wittgenstein, 1953] L. Wittgenstein. 1953. *Philosophical Investigations*. Oxford: Basil Blackwell.

APPENDIX 1

Specification of the Multimodal System of Representation

A. Definition of language G

In this section the syntax and semantics of the language G are presented.

A.1. Syntax of language G

The types of G are as follows:

- (1) *dot*, *dotPair*, *L*, *square*, L^* , *integer* and *orderPair* are types.
- (2) t is a type (truth values).
- (3) If a and b are any types, then $\langle a, b \rangle$ is a type.
- (4) Nothing else is a type.

The constants (or basic expressions) of type *dot* are $\bullet_{x_1, y_1}, \bullet_{x_2, y_2}, \dots$, where $x_1, y_1, x_2, y_2, \dots \in \mathbf{N}$ and the subscript indicates the position of the dot according to the grid as shown in Figure A.1. These constants are also of types *L*, *square* and L^* (i.e., a basic L, a basic square and a basic sequence of consecutive L's are dots). Constants of types *integer* and *orderPair* are the numerals and the pairs of numerals as usual; *position* is a constant of type $\langle s, \text{orderPair} \rangle$ where s is either *dot*, *dotPair*, *L*, *square* or L^* , and similarly *size* is a constant of type $\langle s, \text{integer} \rangle$. There is an additional constant *minL* of type $\langle L^*, \text{integer} \rangle$ which denotes the size of the smallest L of an object of type L^* . Let C_s^G be the set of constants of type s and E_s^G the set of well-formed expressions of type s for any type s .

A syntactic rule forming a composite expression contains: (1) the types of the input expressions and the constraints on the parameters' value which are accessible through the predicates *position*, *size* and *minL*; (2) a *syntactical operation* ($F_\bullet, F_\square, F_L, F_+, F_-$ or F_{pred}) used in the rule which specifies how the input expressions must be combined or transformed to produce the output expression, and (3) the type of the output expression and the constraints on the parameters' value mentioned above. The specification of the syntactic rules of G is based on the definition of the graphical objects in Figure A.1 as follows:

Constants

S0_G. If $\alpha \in C_s^G$ then $\alpha \in E_s^G$ for any type s .

dotPair construction rule

S1_G. If $\beta \in E_{dot}^G = \bullet_{x, y-n}$ and $\delta \in E_{dot}^G = \bullet_{x-n, y}$, for some integers $x, y > 1, 0 < n < x, y$, then $F_\bullet(\beta, \delta) \in E_{dotPair}^G$, where $F_\bullet(\beta, \delta) = \bullet_\bullet(\beta)(\delta)$ such that $size(F_\bullet(\beta, \delta)) = n+1$ and $position(F_\bullet(\beta, \delta)) = (x, y)$.

L construction rule

S2_G. (a) If $\beta \in E_L^G = \bullet_{x, y}$ then $size(\beta) = 1$ and $position(\beta) = (x, y)$.

(b) If $\beta \in E_L^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x, y)$, and $\delta \in E_{dotPair}^G$ such that $size(\delta) = n$ and $position(\delta) = (x, y)$, for some integers $x, y > 1, 1 < n \leq x, y$, then $F_L(\beta, \delta) \in E_L^G$ where $F_L(\beta, \delta) = L(\beta)(\delta)$ such that $size(F_L(\beta, \delta)) = n$ and $position(F_L(\beta, \delta)) = (x, y)$.

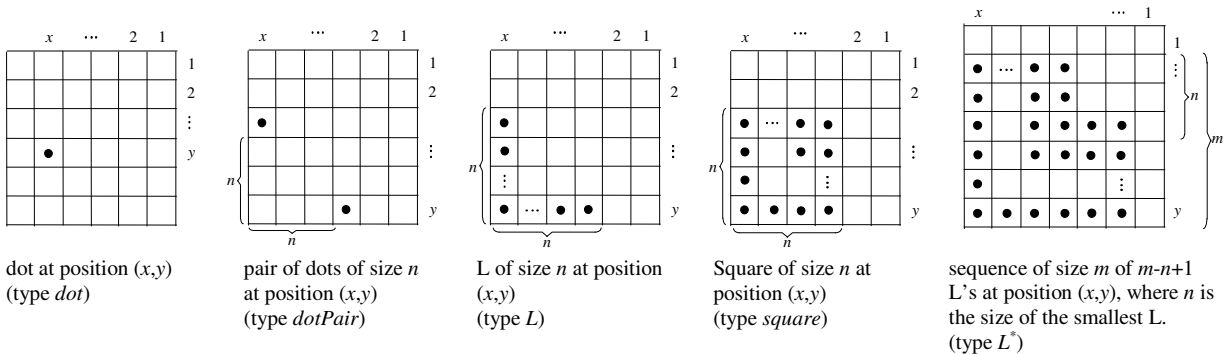


FIGURE A.1 Objects of graphical types referred to by language G .

square construction rule

- S3_G. (a) If $\beta \in E_{square}^G = \bullet_{x,y}$, then $size(\beta) = 1$ and $position(\beta) = (x,y)$.
- (b) If $\beta \in E_{square}^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1,y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = n$ and $position(\delta) = (x,y)$, for some integers $x,y > 1$, $1 < n \leq x,y$, then $F_{\square}(\beta,\delta) \in E_{square}^G$ where $F_{\square}(\beta,\delta) = \square(\beta)(\delta)$ such that $size(F_{\square}(\beta,\delta)) = n$ and $position(F_{\square}(\beta,\delta)) = (x,y)$.
- (c) If $\beta \in E_{square}^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1,y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = n$ and $position(\delta) = (x,y)$, for some integers $x,y > 1$, $1 < n \leq x,y$, then $F_{+}(\beta,\delta) \in E_{square}^G$ where $F_{+}(\beta,\delta) = +(\beta)(\delta)$ such that $size(F_{+}(\beta,\delta)) = n$ and $position(F_{+}(\beta,\delta)) = (x,y)$.

L* construction rule

- S4_G. (a) If $\beta \in E_L^G$ such that $size(\beta) = n$ and $position(\beta) = (x,y)$ for some integers $x,y > 1$, $1 < n \leq x,y$, then $\beta \in E_{L^*}^G$ such that $size(\beta) = minL(\beta) = n$ and $position(\beta) = (x,y)$.
- (b) If $\beta \in E_{L^*}^G$ such that $size(\beta) = m-1$, $minL(\beta) = n$ and $position(\beta) = (x-1,y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = m$ and $position(\delta) = (x,y)$, for some integers $x,y > 1$, $n > 0$, $x,y \leq m > n+1$, then $F_{+}(\beta,\delta) \in E_{L^*}^G$ such that $size(F_{+}(\beta,\delta)) = m$, $minL(F_{+}(\beta,\delta)) = n$ and $position(F_{+}(\beta,\delta)) = (x,y)$.

Equality construction rule

- S5_G. If $\beta \in E_{s_1}^G$ and $\delta \in E_{s_2}^G$, for $s_1, s_2 \in \{dot, L, L^*, square\}$, then $F_{=}(\beta,\delta) \in E_b^G$ where $F_{=}(\beta,\delta) = \beta = \delta$.

Predicate construction rule

- S6_G. If $\alpha \in E_{<a,b>}^G$ and $\beta \in E_a^G$, for any types a, b , then $F_{pred}(\alpha,\beta) \in E_b^G$ where $F_{pred}(\alpha,\beta) = \alpha(\beta)$.

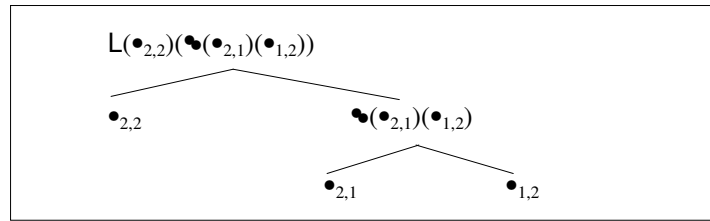
An instance of a syntactic derivation of **G** is illustrated in Figure A.2.

A.2. Semantics of language G

The semantics of **G** is given in a model-theoretic fashion. A model M_G for **G** is an ordered pair $\langle \mathbf{P}, \mathbf{F}_G \rangle$ such that the domain \mathbf{P} is a set of individuals (drawings of **P**), and \mathbf{F}_G is a function assigning a denotation to each constant of **G**. Let D_x be the set of possible denotations for expressions of type x , such that $D_{dot}, D_{dotPair}, D_L, D_{L^*}, D_{square}, D_{integers}, D_{orderPair}$ are subsets of \mathbf{P} , $D_t = \{\text{true}, \text{false}\}$ and, for any types a and b , $D_{<a,b>} = D_b^{D_a}$ (the set of all functions from D_a to D_b). The interpretation function \mathbf{F}_G assigns to each constant of type a a member of D_a and is defined as follows: $\mathbf{F}_G(\bullet_{x,y}) = \bullet$ —in position (x,y) —, for any integers $x,y > 0$; \mathbf{F}_G assigns to the numerals and order pairs of numerals the normal denotation; and for the constants $position$, $size$ and $minL$ \mathbf{F}_G assigns a function from objects of the proper type to the value of the corresponding parameter.

Following [Dowty et al., 1985], we adopt the notational convention by which the semantic value or denotation of an expression α with respect to a model M is expressed as $[[\alpha]]^M$. The semantic rules for the interpretation of **G** are as follows:

Constants interpretation rule



- | | | |
|----|---|--|
| 1) | $\bullet_{2,1}$ | rule S0 _G |
| 2) | $\bullet_{1,2}$ | rule S0 _G |
| 3) | $\bullet(\bullet_{2,1})(\bullet_{1,2})$ | from 1) and 2) by rule S1 _G |
| 4) | $\bullet_{2,2}$ | rule S0 _G |
| 5) | $L(\bullet_{2,2})(\bullet(\bullet_{2,1})(\bullet_{1,2}))$ | from 3) and 4) by rule S2 _G |

FIGURE A.2 Syntactic derivation of an expression of **G**.

M0_G. If $\alpha \in C_s^G$ then $[[\alpha]]^M = \mathbf{F}_G(\alpha)$ for any type s .

dotPair interpretation rule

M1_G. If $\beta \in E_{dot}^G = \bullet_{x,y-n}$ and $\delta \in E_{dot}^G = \bullet_{x-n,y}$, for some integers $x, y > 1$, $0 < n < x, y$, then $[[F_{\blacktriangleright}(\beta, \delta)]]^M$ is the drawing containing two dots in positions $(x, y-n)$ and $(x-n, y)$, respectively.

L interpretation rule

- M2_G. (a) $\beta \in E_L^G = \bullet_{x,y}$ then $[[\beta]]^M$ is a point in position (x,y) .
 (b) If $\beta \in E_L^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x,y)$, and $\delta \in E_{dotPair}^G$ such that $size(\delta) = n$ and $position(\delta) = (x,y)$, for some integers $x, y > 1$, $1 < n \leq x, y$, then $[[F_L(\beta, \delta)]]^M$ is the drawing containing the drawings $[[\beta]]^M$ and $[[\delta]]^M$.

square interpretation rule

- M3_G. (a) If $\beta \in E_{square}^G = \bullet_{x,y}$ then $[[\beta]]^M$ is a point in position (x,y) .
 (b) If $\beta \in E_{square}^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = n$ and $position(\delta) = (x,y)$, for some integers $x, y > 1$, $1 < n \leq x, y$, then $[[F_{\square}(\beta, \delta)]]^M$ is a drawing containing the drawings $[[\beta]]^M$ and $[[\delta]]^M$.
 (c) If $\beta \in E_{square}^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = n$ and $position(\delta) = (x,y)$, for some integers $x, y > 1$, $1 < n \leq x, y$, then $[[F_{+}(\beta, \delta)]]^M$ is a drawing containing the drawings $[[\beta]]^M$ and $[[\delta]]^M$.

L* interpretation rule

- M4_G. (a) Like rule M2_G.
 (b) If $\beta \in E_{L^*}^G$ such that $size(\beta) = m-1$, $minL(\beta) = n$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = m$ and $position(\delta) = (x,y)$, for some integers $x, y > 1$, $n > 0$, $x, y \leq m > n+1$, then $[[F_{+}(\beta, \delta)]]^M$ is a drawing containing the drawings $[[\beta]]^M$ and $[[\delta]]^M$.

Equality interpretation rule

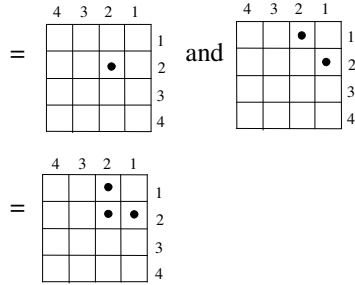
M5_G. If $\beta \in E_{s_1}^G$ and $\delta \in E_{s_2}^G$, for $s_1, s_2 \in \{dot, L, L^*, square\}$, then $[[F_{=}(\beta, \delta)]]^M = \text{true}$ if $[[\beta]]^M = [[\delta]]^M$, false otherwise.

Predicate interpretation rule

M6_G. If $\alpha \in E_{\langle a, b \rangle}^G$ and $\beta \in E_a^G$, for any types a, b , then $[[F_{pred}(\alpha, \beta)]]^M = [[\alpha]]^M ([[\beta]]^M)$.

Consider, for instance, the formal expression $L(\bullet_{2,2})(\blacktriangleright(\bullet_{2,1})(\bullet_{1,2}))$ –which can be expressed informally in a relational form as $L(\bullet_{2,2}, \blacktriangleright(\bullet_{2,1}, \bullet_{1,2}))$, or alternatively in the intuitive form $L(\bullet, \bullet, \bullet)$. The semantic interpretation of this expression is as follows:

$$\begin{aligned}
 & [[L(\bullet_{2,2})(\blacktriangleright(\bullet_{2,1})(\bullet_{1,2}))]]^M \\
 &= [[\bullet_{2,2}]^M \text{ and } [\blacktriangleright(\bullet_{2,1})(\bullet_{1,2})]]^M \\
 &= [[\bullet_{2,2}]^M \text{ and } ([[\bullet_{2,1}]]^M \text{ and } [[\bullet_{1,2}]]^M)] \\
 &= \begin{array}{|c|c|c|c|} \hline 4 & 3 & 2 & 1 \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \bullet & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \end{array} \begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \end{array} \text{ and } ([[\bullet_{2,1}]]^M \text{ and } [[\bullet_{1,2}]]^M) \\
 &= \begin{array}{|c|c|c|c|} \hline 4 & 3 & 2 & 1 \\ \hline \square & \square & \bullet & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \end{array} \begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \end{array} \text{ and } \left(\begin{array}{|c|c|c|c|} \hline 4 & 3 & 2 & 1 \\ \hline \square & \square & \bullet & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \end{array} \begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \end{array} \text{ and } \begin{array}{|c|c|c|c|} \hline 4 & 3 & 2 & 1 \\ \hline \square & \square & \square & \bullet \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \square & \square & \square & \square \\ \hline \end{array} \begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \end{array} \right)
 \end{aligned}$$



B. Definition of language L

In this section the syntax and semantics of L are formally defined.

B.1. Syntax of language L

The types of the language L are as follows:

- (1) *integer* is a type.
- (2) t is a type (truth values).
- (3) If a and b are any types, then $\langle a, b \rangle$ is a type.
- (4) Nothing else is a type.

The only constants of L is 0 and it is of type *integer*. Let C_s^L be the set of constants of type s and E_s^L the set of well-formed expressions of type s , for any type s . The syntactic rules of L are as follows:

Constants

S0_L. If $\alpha \in C_{integer}^L$, then $\alpha \in E_{integer}^L$.

Successor construction rule

S1_L. If $\alpha \in E_{integer}^L$ then $s(\alpha) \in E_{integer}^L$.

Sum construction rule

S2_L. If $\beta, \delta \in E_{integer}^L$ then $F_+(\beta, \delta) \in E_{integer}^L$ where $F_+(\beta, \delta) = (\beta + \delta)$.

Equality construction rule

S3_L. If $\beta, \delta \in E_{integer}^L$ then $F_=(\beta, \delta) \in E_t^L$ where $F_=(\beta, \delta) = (\beta = \delta)$.

As an illustration of a well-formed expression of L consider the syntactic tree in Figure A.3 (which corresponds to the translation of the expression of G in Figure A.2).

B.2. Semantics of language L

The semantics of L is also defined in a model-theoretic fashion. A model M_L for L is an ordered pair $\langle W, F_L \rangle$ such that the domain W is a set of individuals (the natural numbers), and F_L is a function assigning a denotation to each constant of L . Let D_x be the set of possible denotations for expressions of type x , such that $D_{integer} = W$, $D_t = \{\text{true}, \text{false}\}$. The interpretation function F_L assigns a member of D_a to each constant of type a and is defined as follows: $F_G(0) = 0$.

The semantic rules of L are as follows:

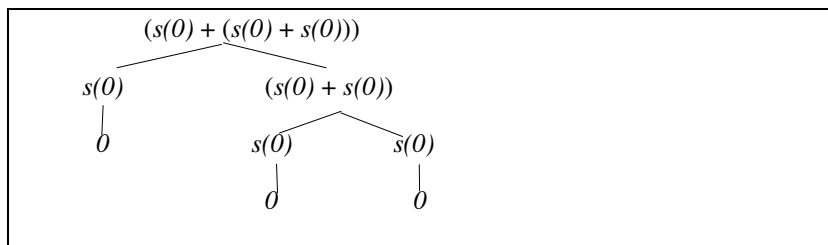


FIGURE A.3 Syntactic derivation of an expression of L .

Constants interpretation rule

M0_L. If $\alpha \in C_{integer}^L$, then $[[\alpha]]^M = \mathbf{F}_L(\alpha)$.

Successor interpretation rule

M1_L. If $\alpha \in E_{integer}^L$ then $[[s(\alpha)]]^M = [[\alpha]]^M + 1$.

Sum interpretation rule

M2_L. If $\beta, \delta \in E_{integer}^L$ then $[[F_+(\beta, \delta)]]^M = [[\beta]]^M + [[\delta]]^M$.

Equality interpretation rule

M3_L. If $\beta, \delta \in E_{integer}^L$ then $[[F_=(\beta, \delta)]]^M = \text{true}$ if $[[\beta]]^M = [[\delta]]^M$, false otherwise.

The semantic interpretation of the expression in Figure A1.3 is as follows:

$$\begin{aligned} & [[s(0) + (s(0) + s(0))]^M \\ &= [[s(0)]]^M + [[s(0) + s(0)]]^M \\ &= [[s(0)]]^M + [[s(0)]]^M + [[s(0)]]^M \\ &= 1+1+1 \\ &= 3. \end{aligned}$$

C. Definition of language P

In this section the syntax of the language **P** is presented. The semantics of **P** can be defined along the lines of Section B.2.

C.1. Syntax of language P

The types of the language **P** are as follows:

- (1) *dot*, *dotPair*, *square*, *L* and *L** are types.
- (2) If *a* and *b* are any types, then $\langle a, b \rangle$ is a type.
- (3) Nothing else is a type.

Let C_s^P be the set of constants of type *s* and E_s^P the set of well-formed expressions (drawings) of type *s* for any type *s*. The only constant of **P** is “•” and it belongs to C_{dot}^P , C_L^P and C_{square}^P . There are no predicates for expressing the position, size or any other property of the symbols, as these properties can be read directly from the drawing. The constant “•” of type *dot* has a position on the grid, and placing the dot in a particular place expresses implicitly its position in the object language. Constants and composite expressions of the types *L*, *L** and *square* have additional properties like position and size which are expressed similarly; assigning an order pair or an integer value to any of these properties cannot be done in the object language, but referring to these values can be done in the statement of syntactic, semantic and translation rules.

The syntactic rules are defined as follows:

Constants

S0_P. If $\alpha \in C_s^P$ then $\alpha \in E_s^P$ for any type *s*.

dotPair construction rule

S1_P. If $\beta, \delta \in E_{dot}^P$ are at positions $(x, y-n)$ and $(x-n, y)$, respectively, for some integers $x, y > 1$, $0 < n < x, y$, then $F(\beta, \delta) \in E_{dotPair}^P$ is of size $n+1$ at position (x, y) , where $F(\beta, \delta)$ is the drawing of β and δ in the corresponding positions.

L construction rule

S2_P. (a) If $\beta = \bullet$ is at position (x, y) , for $x, y > 0$, then $\beta \in E_L^P$ and it is of size 1.

(b) If $\beta \in E_L^P$ is of size $n-1$ at position (x, y) , and $\delta \in E_{dotPair}^P$ is of size n at position (x, y) , for some integers $x, y > 1$, $1 < n \leq x, y$, then $F(\beta, \delta) \in E_L^P$ is of size n at position (x, y) .

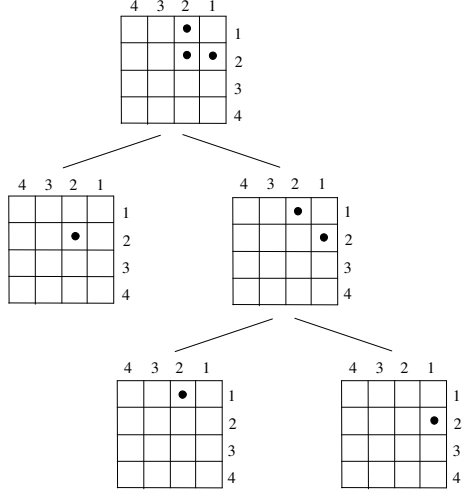


FIGURE A.4 Syntactic derivation of an expression of \mathbf{P} .

square construction rule

- S3_P. (a) If $\beta = \bullet$ is at position (x,y) , for $x,y > 0$, then $\beta \in E_{square}^P$ and it is of size 1.
 (b) If $\beta \in E_{square}^P$ is of size $n-1$ at position $(x-1,y-1)$, and $\delta \in E_L^P$ is of size n at position (x,y) , for some integers $x,y > 1$, $1 < n \leq x,y$, then $F(\beta,\delta) \in E_{square}^P$, and it is of size n at position (x,y) .

L^* construction rule

- S4_P. (a) Like rule S2_G.
 (b) If $\beta \in E_{L^*}^P$ is of size $m-1$ at position $(x-1,y-1)$ where the smallest L of the sequence is of size n , and $\delta \in E_L^P$ is of size m at position (x,y) , for some integers $x,y > 1$, $n > 0$, $x,y \leq m > n+1$, then $F(\beta,\delta) \in E_{L^*}^P$ and it is of size m at position (x,y) where the smallest L of the sequence is of size n .

An instance of a syntactic derivation of \mathbf{P} is illustrated in Figure A.4.

D. Translation from language G into language L

The relation between basic types of \mathbf{G} and \mathbf{L} is shown in Figure A.5.

The translation function ρ_{G-L} assigns an expression (basic or composite) of \mathbf{L} to each expression (basic or composite) of \mathbf{G} . This function is defined by the following translation rules:

Constants translation rule

T0_{G-L}. If $\alpha \in C_s^G$ then $\rho_{G-L}(\alpha) = s(0)$ for $s \in \{dot, L, square\}$.

dotPair translation rule

T1_{G-L}. If $\beta \in E_{dot}^G = \bullet_{x,y-n}$ and $\delta \in E_{dot}^G = \bullet_{x-n,y}$, for some integers $x,y > 1$, $0 < n < x,y$, and $\rho_{G-L}(\beta)$, $\rho_{G-L}(\delta)$ are β' , δ' , respectively, then $\rho_{G-L}(F_{\bullet}(\beta,\delta)) = (\beta' + \delta')$.

L translation rule

- T2_{G-L}. (a) If $\beta \in E_L^G = \bullet_{x,y}$ then $\rho_{G-L}(\beta) = s(0)$.
 (b) If $\beta \in E_L^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x,y)$, and $\delta \in E_{dotPair}^G$ such that $size(\delta) = n$ and $position(\delta) = (x,y)$, for some integers $x,y > 1$, $1 < n \leq x,y$, and $\rho_{G-L}(\beta)$, $\rho_{G-L}(\delta)$ are β' , δ' , respectively, then $\rho_{G-L}(F_L(\beta,\delta)) = (\beta' + \delta')$.

square translation rule

T3_{G-L}. (a) If $\beta \in E_{square}^G = \bullet_{x,y}$, then $\rho_{G-L}(\beta) = s(0)$.

- (b) If $\beta \in E_{square}^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = n$ and $position(\delta) = (x, y)$, for some integers $x, y > 1$, $1 < n \leq x, y$, and $\rho_{G-L}(\beta)$, $\rho_{G-L}(\delta)$ are β' , δ' , respectively, then $\rho_{G-L}(F_{\square}(\beta, \delta))$ is the result of expressing the sum of β' and δ' as a successor number (e.g., if $\beta' = s(0)$ and $\delta' = (s(0) + (s(0) + s(0)))$ then $\rho_{G-L}(F_{\square}(\beta, \delta))$ is $s(s(s(0)))$).
- (c) If $\beta \in E_{square}^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = n$ and $position(\delta) = (x, y)$, for some integers $x, y > 1$, $1 < n \leq x, y$, and $\rho_{G-L}(\beta)$, $\rho_{G-L}(\delta)$ are β' , δ' , respectively, then $\rho_{G-L}(F_{+}(\beta, \delta)) = (\beta' + \delta')$.

L* translation rule

T4_{G-L}. (a) Like rule T2_{G-L}.

- (b) If $\beta \in E_{L^*}^G$ such that $size(\beta) = m-1$, $minL(\beta) = n$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = m$ and $position(\delta) = (x, y)$, for some integers $x, y > 1$, $n > 0$, $x, y \leq m > n+1$, and $\rho_{G-L}(\beta)$, $\rho_{G-L}(\delta)$ are β' , δ' , respectively, then $\rho_{G-L}(F_{+}(\beta, \delta)) = (\beta' + \delta')$.

Equality translation rule

T5_{G-L}. If $\beta \in E_{s_1}^G$ and $\delta \in E_{s_2}^G$, for $s_1, s_2 \in \{dot, L, L^*, square\}$, and $\rho_{G-L}(\beta)$, $\rho_{G-L}(\delta)$ are β' , δ' , respectively, then $\rho_{G-L}(F_{=}(\beta, \delta)) = (\beta' = \delta')$.

As an instance of the application of these rules the translation of the example in Figure A.2 is shown below.

$$\begin{aligned} & \rho_{G-L}(L(\bullet_{2,2})(\blacklozenge(\bullet_{2,1})(\bullet_{1,2}))) \\ &= (\rho_{G-L}(\bullet_{2,2}) + \rho_{G-L}(\blacklozenge(\bullet_{2,1})(\bullet_{1,2}))) \\ &= (\rho_{G-L}(\bullet_{2,2}) + (\rho_{G-L}(\bullet_{2,1}) + \rho_{G-L}(\bullet_{1,2}))) \\ &= (s(0) + (s(0) + s(0))). \end{aligned}$$

E. Translation from language L into language G

The translation rules from **L** into **G** are as follows:

Constants translation rule

T0_{L-G}. If $\alpha \in C_s^L$, then α does not have a translation.

Successor and sum translation rules

T1_{L-G}. TRANSLATION TO EXPRESSIONS OF TYPE *square*

- (1) $\rho_{L-G}(s(0)) = \bullet_{x,y}$, for any x, y .
- (2) If $\alpha \in E_{integer}^L$ such that $[[\alpha]]^M = n^2$, and $\beta, \delta \in E_{integer}^L$ such that $[[\beta]]^M = (n-1)^2$ and $[[\delta]]^M = 2n-1$, then $\rho_{L-G}(\alpha) = \square(\rho_{L-G}(\beta))(\rho_{L-G}(\delta))$, where $size(\rho_{L-G}(\beta)) = n-1$, $size(\rho_{L-G}(\delta)) = n$, $position(\rho_{L-G}(\beta)) = (x-1, y-1)$ and $position(\rho_{L-G}(\delta)) = (x, y)$.

Type of G	Type of L
<i>dot</i>	<i>integer</i>
<i>dotPair</i>	<i>integer</i>
<i>L</i>	<i>integer</i>
<i>L*</i>	<i>integer</i>
<i>square</i>	<i>integer</i>

FIGURE A.5 Relation between basic types of **G** and **L**.

- (3) If $\alpha \in E_{integer}^L$ such that $[[\alpha]]^M = n^2$, and $\beta, \delta \in E_{integer}^L$ such that $[[\beta]]^M = (n-1)^2$ and $[[\delta]]^M = 2n-1$, then $\rho_{L-G}(\alpha) = +(\rho_{L-G}(\beta))(\rho_{L-G}(\delta))$, where $size(\rho_{L-G}(\beta)) = n-1$, $size(\rho_{L-G}(\delta)) = n$, $position(\rho_{L-G}(\beta)) = (x-1, y-1)$ and $position(\rho_{L-G}(\delta)) = (x, y)$.

Notice that through this rule the translation of an expression of **L** can be expressed as an number of different expressions of **G**.

T2_{L-G}. TRANSLATION TO EXPRESSIONS OF TYPE *dotPair*

- (1) If $\alpha \in E_{integer}^L$ such that $[[\alpha]]^M = 2$ then $\rho_{L-G}(\alpha) = \bullet_{(x,y-n)}(\bullet_{x-n,y})$.

T3_{L-G}. TRANSLATION TO EXPRESSIONS OF TYPE *L*

- (1) $\rho_{L-G}(s(0)) = \bullet_{x,y}$.
(2) If $\alpha \in E_{integer}^L$ such that $[[\alpha]]^M = 2n-1$ and $\beta \in E_{integer}^L$ such that $[[\beta]]^M = 2$, then $\rho_{L-G}((\alpha+\beta)) = L(\rho_{L-G}(\alpha))(\rho_{L-G}(\beta))$ where $size(\rho_{L-G}(\alpha)) = n$, $size(\rho_{L-G}(\beta)) = n+1$, $position(\rho_{L-G}(\alpha)) = position(\rho_{L-G}(\beta)) = (x, y)$.

T4_{L-G}. TRANSLATION TO EXPRESSIONS OF TYPE *L**

- (1) Like rule T3_{L-G}.
(2) If $\alpha \in E_{integer}^L$ such that $[[\alpha]]^M = 2pq+p^2$ (i.e, the sum of consecutive odd numbers)⁶, and $\beta \in E_{integer}^L$ such that $[[\beta]]^M = 2(p+q)+1$ (i.e., the next greater consecutive odd number) for some $p > 2, q \geq 0$, then $\rho_{L-G}((\alpha+\beta)) = +(\rho_{L-G}(\alpha))(\rho_{L-G}(\beta))$, where $size(\rho_{L-G}(\alpha)) = p+q$, $minL(\rho_{L-G}(\alpha)) = q+1$, $size(\rho_{L-G}(\beta)) = p+q+1$, $position(\rho_{L-G}(\alpha)) = (x-1, y-1)$ and $position(\rho_{L-G}(\beta)) = (x, y)$.

Equality translation rule

T5_{L-G}. If $\beta, \delta \in E_{integer}^L$ then $\rho_{L-G}((\beta=\delta)) = \rho_{L-G}(\alpha) = \rho_{L-G}(\beta)$.

F. Translation from language G into language P

As each object in **P** can be drawn on a piece of paper, the function ρ_{G-P} assigns to each constant of **G** a drafting procedure which corresponds to the geometry of the operators and constants of **G**.

Constants translation rule

T0_{G-P}. $\rho_{G-P}(\bullet_{x,y})$ is a drawing containing a dot in position (x, y) .

dotPair translation rule

T1_{G-P}. If $\beta \in E_{dot}^G = \bullet_{x,y-n}$ and $\delta \in E_{dot}^G = \bullet_{x-n,y}$, for some integers $x, y > 1, 0 < n < x, y$, then $\rho_{G-P}(F\bullet(\beta, \delta))$ is the drawing containing the drawings $\rho_{G-P}(\beta)$ and $\rho_{G-P}(\delta)$.

L translation rule

- T2_{G-P}. (a) If $\beta \in E_L^G = \bullet_{x,y}$ then $\rho_{G-P}(\beta)$ is as defined in T0_{G-P}.
(b) If $\beta \in E_L^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x, y)$, and $\delta \in E_{dotPair}^G$ such that $size(\delta) = n$ and $position(\delta) = (x, y)$, for some integers $x, y > 1, 1 < n \leq x, y$, then $\rho_{G-P}(F_L(\beta, \delta))$ is the drawing containing the drawings $\rho_{G-P}(\beta)$ and $\rho_{G-P}(\delta)$.

square translation rule

- T3_{G-P}. (a) If $\beta \in E_{square}^G = \bullet_{x,y}$ then $\rho_{G-P}(\beta)$ is as defined in T0_{G-P}.
(b) If $\beta \in E_{square}^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = n$ and $position(\delta) = (x, y)$, for some integers $x, y > 1, 1 < n \leq x, y$, then $\rho_{G-P}(F_{\square}(\beta, \delta))$ is the drawing containing the drawings $\rho_{G-P}(\beta)$ and $\rho_{G-P}(\delta)$.
(c) If $\beta \in E_{square}^G$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = n$ and $position(\delta) = (x, y)$, for some integers $x, y > 1, 1 < n \leq x, y$, then $\rho_{G-P}(F_+(\beta, \delta))$ is the drawing containing the drawings $\rho_{G-P}(\beta)$ and $\rho_{G-P}(\delta)$.

⁶ $2pq+p^2$ is the sum of p consecutive positive odd numbers where $2q+1$ is the smallest, for some $p > 0, q \geq 0$

L^* translation rule

T4_{G-P}. (a) Like rule T2_{G-P}.

(b) If $\beta \in E_{L^*}^G$ such that $size(\beta) = m-1$, $minL(\beta) = n$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^G$ such that $size(\delta) = m$ and $position(\delta) = (x, y)$, for some integers $x, y > 1$, $n > 0$, $x, y \leq m > n+1$, then $\rho_{G-P}(F_+(\beta, \delta))$ is the drawing containing the drawings $\rho_{G-P}(\beta)$ and $\rho_{G-P}(\delta)$.

Consider, for instance, the expression $L(\bullet_{2,2})(\blacktriangleright(\bullet_{2,1})(\bullet_{1,2}))$. Its translation to **P** is shown in Figure A.6.

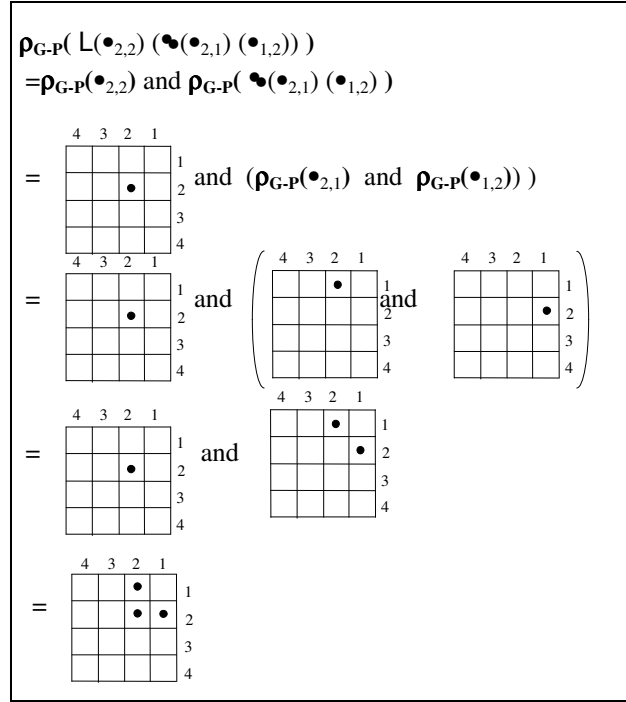


FIGURE A.6 Translation into **P** of an expression of **G**.

G. Translation from language P into language G

To conclude the specification of the multimodal system of representation the definition of ρ_{P-G} is presented. Expressions of **P** (drawings) are translated into expressions of **G**, as follows:

Constants translation rule

T0_{P-G}. If $\alpha \in E_{dot}^P$ is the drawing of a dot (\bullet) in position (x, y) $\rho_{P-G}(\alpha)$ is $\bullet_{x,y}$

dotPair translation rule

T1_{P-G}. If β and $\delta \in E_{dot}^P$ are dots at positions $(x, y-n)$ and $(x-n, y)$ respectively, for some integers $x, y > 1$, $0 < n < x, y$, $F(\beta, \delta) \in E_{dotPair}^P$ and $\rho_{P-G}(\beta) = \beta'$ and $\rho_{P-G}(\delta) = \delta'$ then $\rho_{P-G}(F(\beta', \delta')) = F\blacktriangleright(\beta', \delta')$. (i.e., $\blacktriangleright(\beta')(\delta')$ such that $size(F\blacktriangleright(\beta', \delta')) = n+1$ and $position(F\blacktriangleright(\beta', \delta')) = (x, y)$).

L translation rule

- T2_{P-G}. (a) If $\beta \in E_L^P$ is at position (x,y) , for $x, y > 0$ and it is of size 1, then $\rho_{P-G}(\beta)$ is $\bullet_{x,y}$
- (b) If $\beta \in E_L^P$ is of size $n-1$ at position (x,y) , and $\delta \in E_{dotPair}^P$ is of size n at position (x,y) , for some integers $x, y > 1$, $1 < n \leq x, y$, (i.e., $F(\beta, \delta) \in E_L^P$ is of size n at position (x,y)), $\rho_{P-G}(\beta) = \beta'$ and $\rho_{P-G}(\delta) = \delta'$ then $\rho_{P-G}(F(\beta, \delta)) = F_L(\beta', \delta') \in E_L^G$ (i.e., $L(\beta')(\delta')$ such that $size(F_L(\beta', \delta')) = n$ and $position(F_L(\beta', \delta')) = (x,y)$).

square translation rule

- T3_{P-G}. (a) If $\beta = \bullet$ is at position (x,y) , for $x, y > 0$, where $\beta \in E_{square}^P$ and it is of size 1, then $\rho_{P-G}(\beta)$ is $\bullet_{x,y}$
- (b) If $\beta \in E_{square}^P$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^P$ such that $size(\delta) = n$ and $position(\delta) = (x,y)$, for some integers $x, y > 1$, $1 < n \leq x, y$, and $\rho_{P-G}(\beta) = \beta'$ and $\rho_{P-G}(\delta) = \delta'$ then $\rho_{P-G}(F(\beta, \delta)) = F_{\square}(\beta', \delta') \in E_{square}^G$ (i.e., $\square(\beta')(\delta')$ such that $size(F_{\square}(\beta', \delta')) = n$ and $position(F_{\square}(\beta', \delta')) = (x,y)$).
- (c) If $\beta \in E_{square}^P$ such that $size(\beta) = n-1$ and $position(\beta) = (x-1, y-1)$, and $\delta \in E_L^P$ such that $size(\delta) = n$ and $position(\delta) = (x,y)$, for some integers $x, y > 1$, $1 < n \leq x, y$, and $\rho_{P-G}(\beta) = \beta'$ and $\rho_{P-G}(\delta) = \delta'$ then $\rho_{P-G}(F(\beta, \delta)) = F_{+}(\beta', \delta') \in E_{square}^G$ (i.e., $+(\beta')(\delta')$ such that $size(F_{+}(\beta', \delta')) = n$ and $position(F_{+}(\beta', \delta')) = (x,y)$).

L* translation rule

- T4_{P-G}. (a) Like rule T2_{P-G}.
- (b) If $\beta \in E_{L^*}^P$ is of size $m-1$ at position $(x-1, y-1)$ where the smallest L of the sequence is of size n , and $\delta \in E_L^P$ is of size m at position (x,y) , for some integers $x, y > 1$, $n > 0$, $x, y \leq m > n+1$, $\rho_{P-G}(\beta) = \beta'$ and $\rho_{P-G}(\delta) = \delta'$, then $\rho_{P-G}(F(\beta, \delta)) = F_{+}(\beta', \delta') \in E_{L^*}^G$ (i.e., $+(\beta')(\delta')$ such that $size(F_{+}(\beta', \delta')) = m$, $minL(F_{+}(\beta', \delta')) = n$ and $position(F_{+}(\beta', \delta')) = (x,y)$).

Note that for the translation of a square, rules (b) and (c) are available. Each one of these options reflects one of the notations through which a square can be interpreted (as a square number or as the sum of odd numbers), and the use of one or the other will map a given expression of **P** into a different expression in **G**. The choice of notation, as discussed in Sections 5 and 6, is a decision that has to be taken not within but reasoning about the system. Note as well that rules for equality cannot be expressed in **P** and the fact that an array of dots can express an equality is also realised by reasoning from or about the representational system.