ELSEVIER

Artificial Intelligence 91 (1997) 281-307

Artificial Intelligence

A computational approach to George Boole's discovery of Mathematical Logic

Luis de Ledesma [a,*], Aurora Pérez [a,1], Daniel Borrajo [b,2], Luis M. Laita [a,3]

[a] *Facultad de Informática, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Madrid, Spain*
[b] *Escuela Politécnica Superior, Universidad Carlos III de Madrid, 28911 Leganés, Madrid, Spain*

Received December 1995; revised September 1996

Abstract

This paper reports a computational model of Boole's discovery of Logic as a part of Mathematics. George Boole (1815-1864) found that the symbols of Logic behaved as algebraic symbols, and he then rebuilt the whole contemporary theory of Logic by the use of methods such as the solution of algebraic equations. Study of the different historical factors that influenced this achievement has served as background for our two main contributions: a computational representation of Boole's Logic before it was mathematized; and a production system, BOOLE2, that rediscovers Logic as a science that behaves exactly as a branch of Mathematics, and that thus validates to some extent the historical explanation. The system's discovery methods are found to be general enough to handle three other cases: two versions of a Geometry due to a contemporary of Boole, and a small subset of the Differential Calculus. © 1997 Published by Elsevier Science B.V.

1. Introduction

In 1847, George Boole found that, by adequately representing Logic, it became a branch of Algebra in a precise sense: all known results in Logic (and some unknown) could be obtained by the use of standard mathematical techniques. Our objective has been to provide a coherent detailed account of this surprising discovery, both from the

* Corresponding author. E-mail: ledesma@fi.upm.es.
[1] E-mail: aurora@fi.upm.es.
[2] E-mail: dborrajo@grial.uc3m.es.
[3] E-mail: laita}@fi.upm.es.

PII S0004-3702(97)00017-9

historical and the computational point of view. Boole's discovery was surprising because even though there had been attempts to mathematize Logic, nobody had reached a full formalization, presumably because no one regarded Logic from a strict mathematical point of view and applied genuine mathematical tools, as Boole did (see Section 2.2).

We have studied how, plausibly, Boole arrived at his knowledge of the Method of Separation of Symbols (MSS). This was the contemporary name for a method which is a precursor of today's abstract mathematical formalization. Also, we have studied how Boole extensively used the method in several branches of Calculus and Probability. The practice of applying the method had been and remained common among some mathematicians of the 18th and 19th centuries, both on the Continent and in the U.K., but always within recognized branches of Mathematics (see Section 2).

We concluded that Boole's discovery in Logic was just one more application of the method, but this time to a non mathematical discipline: Logic. Our next step was to devise a computational representation of Logic as it was presented in Boole's Logic writings, in order to write a program that could discover the algebraic character of Logic by examining it under viewpoint of the MSS.

Two facts were crucial for Boole's discovery:

(i) First, the change in representation from a philosophical view of Logic as an enormous complex structure of syllogisms and conditional statements, to a view of it as (possibly) fitting the general scheme for any branch of Algebra. Simplistically, the scheme consists of a set of very simple objects that are subjected to operations on numbers.

(ii) The second fact was the successful application of the MSS to Logic, which meant that some (interesting) subset of properties of the operations on numbers held also for Logic. This made it possible to use the powerful tools of Mathematics on this new domain of Logic.

Our work on representation is based on fact (i). In all his Logic texts, Boole provided descriptions in terms that were susceptible to the MSS (see the remark below, and Sections 2.2 and 5.4). We have tried to model this part of the history by designing a representation which is given as input to a computational system which we named BOOLE (see Sections 4 and 5.4). The system execution applies the method to find that Logic is actually a branch of Algebra.

The first version of the BOOLE system dealt with the task in fact (ii) above (applying MSS to Logic). It is a frame-based production system whose productions, the successive states of its working memory, and its outputs are supposed to follow Boole's account of his discovery as revealed by his writings on Logic and the writings about him by his contemporaries. See, for instance, the "Memoir of Augustus De Morgan" [10], "Home side of a scientific mind" [12], or Boole's first biography [16].

But the whole of Boole's writings (on Logic as well as on Mathematics) provides another idea that motivated the following step in our work. In Boole's view, the process of finding whether the MSS successfully applies to a branch of science is always the same (see, for instance, [21]). We have exploited this idea in the design of BOOLE2, which incorporates the features of the previous version that refer to the MSS; this is the generic part of the system. However, BOOLE2 can deal not only with Logic, but with

other branches of science. Currently, it handles Logic, two different versions of a subset of Geometry that was proposed by a contemporary of Boole (see the remark below), and the first principles of Differential Calculus. The result is that properties similar to those in Logic hold for the first version of Geometry and the Calculus, although not for the second version of Geometry because of a non-commutativity. Thus, BOOLE2 reasons from the input representation of a science and verifies that some properties hold or not hold in that science (see Section 3).

This work can be summarized as follows. We construe the historical discovery of the algebraic nature of Logic as the sum of two sudden changes. The first one is the representation of Logic. The second is the application of the MSS. BOOLE2 is designed to follow the ideas expressed in Boole's writings. We have tried to make the system reproduce not only the result of the discovery, but also Boole's account of the process that led to it, and so BOOLE2 reasons in a linguistic way to reproduce the historical discovery in Logic. Other discoveries are reached also by trying to parallel Boole's results in several branches of Mathematics. BOOLE2 constitutes a computational explanation of the discovery in Logic, and opens a path for the study of discoveries in abstract sciences such as Mathematics or Logic.

Remarks. In this paper, the terms *symbolize* and *symbolization* are equivalent to *formalize* and *formalization* in the sense of verifying that a science subject matter is an instance of some formal system. We use *symbolize* rather than *formalize* in order to follow Boole, whose writings employ phrases such as "the use of symbols", "symbolic expression", "algebra of symbols", "symbolic language", "let us use the symbols", or "the symbolical forms ... are sufficient for the basis of a Calculus".

Also, the terms *operation* and *combination* depart here from common usage, because we wish to reflect the conception of sciences in Boole's time. Thus, an *operation* means something close to the modern idea of operator. For instance, differentiation is an operation of Calculus; taking the trace of a point moving through a segment is an operation of Geometry ("transference"); and selecting a class of individuals from a universe is an operation of Logic ("class"). The effects of the operations in these three examples are the well known concepts of derivative, segment, and predicate (identified with a set of individuals). On the other hand, we use the term *combination* (of operations) much in the sense of today's algebraic operation. For instance, in Logic two successive acts of selecting individuals first from class x and then from class y ("class succession") yield the class of the individuals which are both x and y. Similarly, in Geometry the trace of a point subjected to two movements through two different directions is a parallelogram.[4] Finally, two successive differentiations yield a second derivative.

2. The actual historic discovery

Boole was an able developer of a methodological tradition begun in France during the second half of the 18th century, and continued in Britain during the first half of

[4] This original presentation of elementary Geometry is due to Gregory [14].

the 19th century. Arbogast, Lagrange and others worked in France [11, Chapter 2]. Murphy, Greatheed, Hershel and others were British mathematicians who used the method [30], but in our opinion it was Gregory who more clearly expressed the philosophy of the method, and Boole who developed it most fully [6,21]. This tradition involved the methodological idea of *separating symbols of operations from their subjects of application, and operating with these symbols as with algebraic entities.* The method was called the *Method of Separation of Symbols* (MSS) and it was at the heart of Boole's discovery and development of Logic as an algebraic discipline.

2.1. An example of the application of the Method of Separation of Symbols

An example from one of Gregory's articles [13] serves to illustrate the MSS: Gregory determines the symbolic laws used in Newton's binomial,

$$(a+b)^n = a^n + na^{n-1}b + \frac{n(n-1)}{1*2}a^{n-2}b^2 + \cdots + b^n$$

and notes that according to Euler only the following three laws of combination of symbols are necessary for the general application of the binomial development:

- Commutative law: $ab = ba$.
- Distributive law: $c(a+b) = ca + cb$.
- Index law: $a^m a^n = a^{m+n}$.

Therefore, the development of the binomial, which in principle refers to numbers, is valid when the symbols a, b, n in $(a+b)^n$ represent other entities, provided they fulfill these three laws. Gregory states that since it can be proved that the operations of Differential Calculus and of the Calculus of Finite Differences are subject to these laws, it can be assumed that Newton's binomial development is valid for such Calculi, which means that it is not necessary to repeat the proof for each particular case. Let us consider an example application from Gregory's article: the determination of the nth derivative of a product of functions $u \cdot v$. The known equality

$$\frac{\mathrm{d}(u \cdot v)}{\mathrm{d}x} = u\frac{\mathrm{d}v}{\mathrm{d}x} + v\frac{\mathrm{d}u}{\mathrm{d}x}$$

may be written as follows,

$$\frac{\mathrm{d}(u \cdot v)}{\mathrm{d}x} = \left(\frac{\mathrm{d}'}{\mathrm{d}x} + \frac{\mathrm{d}}{\mathrm{d}x}\right)(u \cdot v),$$

where $\mathrm{d}'/\mathrm{d}x$ is an operation upon v but not upon u, and $\mathrm{d}/\mathrm{d}x$ is an operation upon u but not upon v. Gregory states:

> ... these operations, from their nature, are distributive and, as they are independent of each other, they must be commutative, hence they come under the circumstances to which the binomial theorem applies.

So, the nth derivative may be considered as a power, the result being:

$$\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^n (u \cdot v) = \left(\frac{\mathrm{d}'}{\mathrm{d}x} + \frac{\mathrm{d}}{\mathrm{d}x}\right)^n (u \cdot v)$$

$$= \left\{\left(\frac{\mathrm{d}'}{\mathrm{d}x}\right)^n + n\left(\frac{\mathrm{d}'}{\mathrm{d}x}\right)^{n-1}\left(\frac{\mathrm{d}}{\mathrm{d}x}\right) + \frac{n(n-1)}{1*2}\left(\frac{\mathrm{d}'}{\mathrm{d}x}\right)^{n-2}\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^2 + \cdots\right\}(u \cdot v)$$

$$= u\frac{\mathrm{d}^n v}{\mathrm{d}x^n} + n\frac{\mathrm{d}u}{\mathrm{d}x}\frac{\mathrm{d}^{n-1}v}{\mathrm{d}x^{n-1}} + \frac{n(n-1)}{1*2}\frac{\mathrm{d}^2 u}{\mathrm{d}x^2}\frac{\mathrm{d}^{n-2}v}{\mathrm{d}x^{n-2}} + \cdots.$$

Gregory says that the result is also valid when n is fractional or negative. Thus, Gregory "separates" the symbols for derivatives from the subjects of application of those derivatives, and operates with the former as with algebraic entities, applying to them Newton's binomial formula in this particular case.

Even though Gregory's contribution to the method is usually recognized as substantial, only Boole dared to try it on a previously non mathematical discipline: Logic. The historical and detailed explanation of this process has been a substantial part of our research, which this Section 2 briefly summarizes. We then introduce a program that works according to our reconstruction of Boole's discovery.

The historical development of the method has been studied in [18, 19, 30] as part of the history of Symbolic Calculi. More detailed accounts on Boole are in [6, 21–23, 29].

2.2. *Boole and the birth of Mathematical Logic*

Prior to Boole, there had been attempts to mathematize Logic by Leibniz (1646–1716), Saccheri (1667–1733), Lambert (1728–1777), and others, but the results were very limited. Leibniz, for instance, produced an abstract calculus of terms, attributes, and classes, in which the idea of membership in a class was related to logical concepts. Nevertheless, his calculus could not take account of negation and conjugation between conjunction and disjunction, among many other limitations [17, Chapter V]. Boole cited Leibniz once [5, p. 240] in a context of the principle of contradiction, but did not cite Leibniz's mathematical approach. This approach was not actually known until the first half of the 20th century.

Boole's awareness of the power of Mathematics as a calculus of symbols began early. In 1835, at the age of twenty, he gave an address on Newton which already contained ideas on the necessity of a systematic use of symbolism, and on the possibility of separating the use of symbols from their interpretation. In 1839, Boole met Gregory, who had recently founded the *Cambridge Mathematical Journal*. Boole submitted several papers for publication and Gregory saw with astonishment that Boole had developed by himself -influenced only by his readings of Lagrange and other French mathematicians- an approach to Mathematics very similar to the one developed at Cambridge. Gregory acquainted Boole with the properties of the MSS. Boole later acknowledged that Gregory was one of his inspirers, but he surpassed Gregory by making the bold guess that the Symbolic Calculus, and specifically the MSS, could be applied outside Mathematics, particularly to Logic.

Boole's interest in Logic dated from much earlier: "... I was induced by the interest which it (the controversy) inspired, to resume the almost forgotten thread of *former*

inquires" [3, p. 1]. He renewed his interest in Logic because of the controversy between the philosopher William Hamilton and the mathematician Augustus De Morgan. However, there is no evidence that Gregory had been interested in Logic.

Boole's exposition of the first principles of Logic at the start of his book "The Mathematical Analysis of Logic" [3] reveals his awareness of the interdependence between Logic and Language:

> That which renders Logic possible, is the existence in our minds of general notions, our ability to conceive of a class, and to designate its individual members by a common name. The theory of Logic is thus intimately connected with that of Language. A successful attempt to express logical propositions by symbols, the laws of whose combinations should be founded upon the laws of the mental processes which they represent, would, so far, be a step toward a philosophical language ... [3, pp. 4–5].

Thus, Logic is a language with signs and rules for combining signs. The mind forms the idea of a class, to which a name (a label or a sign) is given. Once signs have been assigned to classes belonging to a universe of discourse, the necessary next step is to determine the laws which the signs follow. These laws are found by investigating the mental processes which carry out these acts of classification.[5] The goal of determining laws then led to the discovery of Mathematical Logic, which Boole formulated in "The Mathematical Analysis of Logic" [3] as follows:

> Further, let us conceive a class of symbols x, y, z, possessed of the following character. The symbol x, operating upon any subject comprehending individuals or classes, shall be supposed to select from that subject all the X's [individuals of some class] which it contains ... When no subject is expressed, we shall suppose 1 (The Universe) to be the subject understood, so that we shall have $x = x(1)$ ... the product xy will represent, in *succession*, the selection of the class Y, and the selection from the class Y of such individuals of the class X as are contained in it, the result being the class whose members are both X's and Y's ... The result of an act of election is independent of the grouping or classification of the subject. Thus it is indifferent whether from a group of objects considered as a whole, we select the class X, or whether we divide the group into two parts, select the X's from them separately, and then connect the results in one aggregate conception. We may express this law mathematically by the equation $x(u + v) = xu + xv$ ... It is indifferent in what order two successive acts of election are performed ... The symbolic expression of this law is $xy = yx$. The result of a given act of election performed twice or any number of times in succession is the result of the same act performed once ... supposing the same operation to be n times performed, we have $x^n = x$ ... (These) laws are sufficient for the basis of a calculus. From the first of these it appears that elective symbols are *distributive*, from the second that they are *commutative*; properties which they possess in common with symbols of *quantity*,

[5] Similar ideas are found in "The Laws of Thought" [5], his second book on Logic, and "The Claims of Science" [4], a lecture delivered at Cork. These, together with "The Mathematical Analysis of Logic", are the only published works of Boole on the subject.

> and in virtue of which, all the processes of common algebra are applicable to the present system ... The third law $x^n = x$ we shall denominate the index law. It is peculiar to elective symbols and will be found of great importance ...

This quotation explains that Logic is a calculus governed by algebraic laws (distributive, commutative, and index), which is the starting point for the use of the MSS. Boole's entire book [3] completely develops an Algebraic Logic based on precisely those three laws. This special Algebra is elementary, its main procedure being the solution of systems of algebraic equations. However, it is powerful: it even surpasses somewhat the level of Boole's contemporary Logic (Boole proved the existence of combinations of premises in which there is absolutely no medium of comparison). Also, his Algebra should not be confused with *Boolean Algebra*, a different discipline, built up by Jevons, Peirce, Venn, and others after Boole's death.

Boole's discovery was the realization that Logic is a science amenable to symbolization, meaning that its basic operations follow the same laws of combination of symbols as some of the families in Symbolic Algebra.

3. Overview of BOOLE2

Our first program BOOLE1 succeeded in applying the MSS to Logic, and it reached the same conclusions as Boole at the beginning of his "Mathematical Analysis of Logic" [7].

Since we believed that—in Boole's view—ascertaining whether a science is symbolizable (whether the MSS is applicable to it) is always the same, we undertook the task of generalizing BOOLE1's heuristics to make the program applicable to other sciences as well. Gregory's Geometry[6] was chosen as a first test case.

We emphasize that, even though the system does not currently handle a large number of case studies, it does capture an interesting level of detail for several, and it is potentially able to handle further cases. The separation-of-symbols heuristic has been consistently used since the 19th century under the name of *formalization* or *abstraction*. Whenever a portion of knowledge is symbolically represented and its symbols (*separated* from the particular meaning they have in their domain) are found to behave in the same way as those of, say, Logic, Algebra, Graph Theory, etc., a separation-of-symbols process has been carried out. In this sense, *any science* could be a candidate input for the system. There would always be an output, *it is or is not symbolizable*, if we could devise an appropriate representation of the science. Due to their low degree of formalization, Boole's contemporary sciences are more suitable than others to the spirit of the present study.

Our subsequent version, BOOLE2, uses a uniform and plausibly humanoid way of representing and reasoning and is able to discover that Logic, Geometry, and a subset of Differential Calculus are symbolizable [8], and that the generalization of Gregory's Geometry is not symbolizable because of its lack of commutativity. BOOLE2's repre-

[6] A brief description can be found in the remark of Section 1.

Given: A science in terms of:
- Set of operations of the science
- Set of combinations of the science

Return:
- `True` if the science is symbolizable, or `False` otherwise
- Laws that are fulfilled

Fig. 1. Formalization of BOOLE2 discovery task.

sentation and reasoning tools are intended to handle any science as a candidate for symbolization, provided its basic contents can be made accessible to the system. The generic part of the system concerns the MSS and is designed to receive some specific science as input: operations, combinations and the possible laws thereof. Thus the system becomes applicable to domains besides Logic.

For simplicity of exposition, most examples in this paper are drawn from Logic, but there are parallel examples in the other domains that BOOLE2 has handled.

BOOLE2's starting point is the knowledge and goals prior to the discovery of a science being symbolizable or not, and it decides whether the discovered laws allow incorporating the science to the set of symbolizable sciences. The input is a description of a science in terms of its operations and combinations; the output is a record of its algebraic properties and whether it is symbolizable. Fig. 1 is a formalized depiction of the discovery task.

BOOLE2 is designed to operate in a human-like manner and to be consistent with Boole's account of his reasoning process. The program's problem space consists of these elements:

- state space: sets of instances of operations, combinations and laws of a science,
- initial state: set of generic initial operations, and set of combinations of a science,
- goal: find out whether a state exists in which some combinations fulfill the MSS requirements; that is, determine whether the science is symbolizable,
- operators:
  - apply a combination to two operations,
  - check whether a combination fulfills a law,
- heuristics: a set of techniques used by a human agent in a discovery process. For instance, when a combination has been performed with operations $o1$ and $o2$, try to perform the same combination with $o2$ and $o1$ to test for commutativity. Or, in a different context, when two mathematical expressions are tested for equality, try a factoring process.

Fulfilling the initial goal depends on reaching a number of subgoals, that in turn need other subgoals, and so on. The achievement of each goal simulates one step of the discovery process. Fig. 2 shows the top part of the subgoaling structure of BOOLE2's reasoning.[7] This heuristic-guided backward process is an interpretation of the seemingly insightful reasoning method of Boole.

[7] The numbers in parentheses indicate the number of instances of those laws that have to appear in order to conclude that a science is symbolizable.

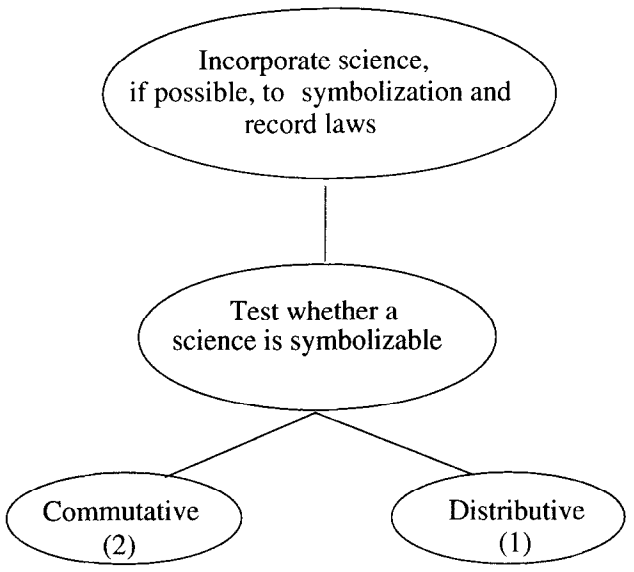

Fig. 2. Top part of the subgoaling structure generated by BOOLE2 when reasoning.

For example, the initial state in the case of Logic is:

- Set of simple operations: there is only one simple operation, called *logic-class*. Initially, three instances of this concept are generated: x, y, and z, which describe three simple generic operations in Logic. Each one represents the generic *act of election* idea of Boole expressed in the quotation of Section 2.2.
- Set of combinations: *succession* and *aggregation*, which are also described in Section 2.2 and correspond roughly to conjunction and disjunction in classical logics.

4. Representation in BOOLE2

A key issue in BOOLE2 is representing the input science. Most aspects are described in this section, but there are also science-dependent procedures and heuristics which are described below in Section 5. We have chosen a frame-based representation for concepts and a production system for actions, both implemented using a frame-based tool for generating production systems, FRULEKIT [31]. Fig. 3 shows, as an example, an overview of the frames hierarchy for the science Logic.

The hierarchy of frames has three parts: (a) static science-independent, (b) static science-dependent, and (c) dynamic. The static science-independent part is common to all sciences; its frames correspond to generic descriptions of concepts, without implying that every science will possess an instance of every frame. The static science-dependent part is specific to a science, but also consists of frames and instances that are created before execution of the production system. Finally, the dynamic instances are created by the production system. The next subsections discuss how to represent operations and combinations.

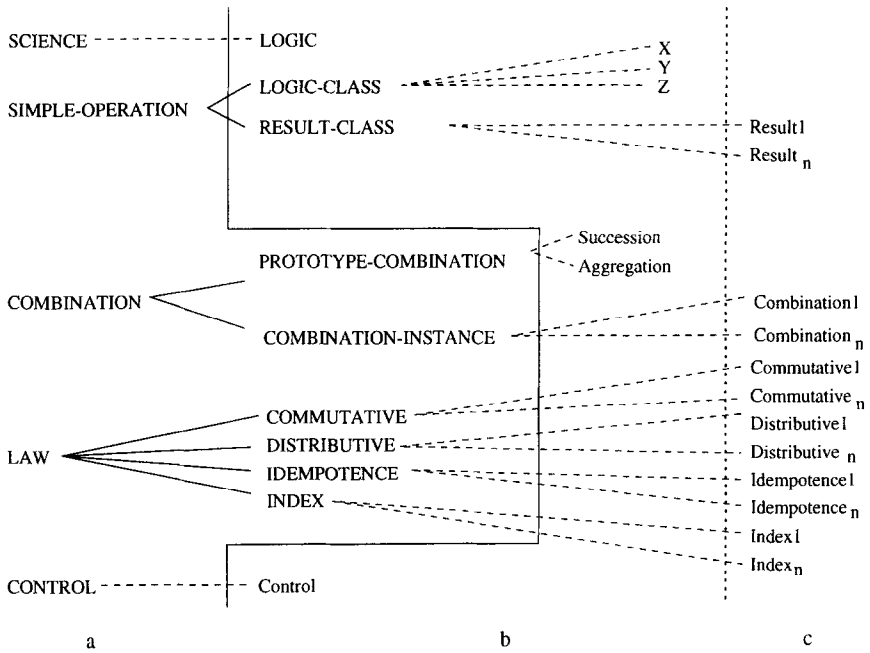

Fig. 3. Hierarchy of frames in BOOLE2 for Logic. (a) is the static science-independent part, (b) is the static science-dependent part, and (c) is the dynamic part.

Frame **simple-operation**	
Slot	*Comments*
`name`	identifier of the operation
`science`	science to which it belongs
`notation`	linguistic representation of the operation

Fig. 4. Representation of the frame `simple-operation`.

4.1. Representing operations

The operations of any science are defined in terms of subclasses of a predefined frame called `simple-operation`. In Logic, the subclass of `simple-operation` is the simple operation `logic-class`. Similarly, in Geometry there is `transference`, and in Calculus, `(partial) differentiation`. Fig. 4 shows the definition of this frame and its slots. The first two slots are self-explanatory. The slot `notation` describes a linguistic representation of the operation that parallels the name of the operation. The notation is explained in detail in Section 4.2.2.

Frame **transference**	
is-a **simple-operation**	
Slot	*Comments*
`magnitude`	number of dimensions of the transference
`direction`	directions of the transference

Fig. 5. Representation of the frame `transference`.

Frame **combination**	
Slot	*Comments*
`name`	name of the combination
`science`	science to which it belongs
`arithmetic-name`	name given to its arithmetic role

Fig. 6. Representation of the frame `combination`.

Besides inheriting all slots from their respective superclasses, all subclasses of the hierarchy can define new local slots. For instance, the frame `transference` inherits the slots of `simple-operation` and defines two new ones, as shown in Fig. 5. Initially, generic examples of the simple operation of the science are generated.

Also, for each science, the so-called result operations (e.g., `result-class` in Logic, and `result-transference` in Geometry) are defined as simple operations. Result operations are used to represent the outcome of combining two simple operations, unlike the other simple operations which are defined at the initial state.

4.2. Representing combinations

The second input to BOOLE2 is the combinations for a given science. Combinations require more representation features, since one must describe what the system should do when combining two operations (application of the first operator of the problem space). The static character of combinations (what they are) is defined by the frames, whereas the dynamic character (what they do) is defined in terms of rules and functions.

4.2.1. Definition of combinations

Fig. 6 shows the static definition of the frame `combination`. The `arithmetic-name` is the name that the combination will receive in Algebra, once the system discovers that the corresponding science can be symbolized, and, therefore, that its formulae can be translated into Algebra and transformed using algebraic tools. The allowed slot values are *times* and *plus*.

Frame **prototype-combination**	
is-a **combination**	
Slot	*Comments*
`laws`	set of laws that the combination fulfills
`notation-generator-1`	generator of notation for each operation
`notation-generator-2`	an alternative way of generating notations
`result-generator`	generator of the resulting notation

Fig. 7. Representation of the frame `prototype-combination`.

Frame **combination-instance**	
is-a **combination**	
Slot	*Comments*
`simple-operation1`	the first simple-operation that is combined
`simple-operation2`	the second simple-operation that is combined
`simple-operation3`	sometimes there is a third optional operation
`result`	result of combining two simple-operations

Fig. 8. Representation of the frame `combination-instance`.

For each science, two subclasses of this frame are defined:

- `prototype-combination`. Represents the combinations themselves; its definition is in Fig. 7. For example, in Logic the two instances of the frame `prototype-combination` have values for the slot `name` of *succession* and *aggregation*, respectively. The `generator` slots refer to how the combination applies to two operations (how it works); they are explained in Section 4.2.2. The system fills the slot `laws` when discovering laws that the combination fulfills.
- `combination-instance`. Represents the successive applications of a combination to different pairs of operations. For example, if BOOLE2 performs in Logic the succession of x and y, it generates an instance of the frame `result-class`, and an instance of the frame `combination-instance` that represents the application of this combination. Among other things, this allows the program to easily avoid repeating the combination on the same pair of operations in the same order. Fig. 8 shows the description of the frame. The slots `simple-operation*` represent the operations that were combined in the corresponding instance of the combination.

  The value of the slot `result` is the notation that results from applying the combination. Section 4.2.2 explains this in more detail.

```
Function Succession-Notation-Generator(operation1,operation2)
    name1 := operation-name(operation1);
    name2 := operation-name(operation2);
    If operation2 is a result-class
    Then Return simple-operation-notation(operation2)
    Else If name1 = name2
        Then Return ((ALL THAT ARE name1))
        Else Return ((name2 THAT ARE name1)
                     (name2 THAT ARE NOT name1))
```

Fig. 9. Function that computes notations for performing the succession of operations in Logic.

4.2.2. Definition of notations

One must also define how the combinations of a science can be computed. The notations described by Boole and Gregory for Logic and Geometry, and the opening chapters of Calculus textbooks, motivated our representation of their respective operations and combinations. For instance, based on Boole's description of performing the two combinations of Logic, we generated a set of notations in English which are similar, both in form and meaning, to Boole's own sentences. Thus, to perform a combination, two functions need to be defined to describe the notations of operations:

- Notations for representing the input operations (`notation-generator`). We describe here only the notations generated for succession in Logic, but equivalent work has been done for aggregation and for combinations in Geometry and Calculus. Fig. 9 shows the output of the notation generator function when applying succession to two operations in Logic.
  For example, suppose BOOLE2 applies succession to the operations x and y. It calls the function `Succession-Notation-Generator` with the frames representing x and y (instances of `logic-class`) as arguments. Since the frame for y is not an instance of `result-class`, and since x and y are not the same, the function generates the following notation for y:

  $$((Y \text{ THAT ARE } X)\ (Y \text{ THAT ARE NOT } X))$$

  while x is just represented by its name.
- Notations for representing the result (`result-generator`). This function takes as input both the name of the first operation and the notation generated for the second operation by `notation-generator`; it returns the notation that results from applying the combination. For example, Fig. 10 shows the result-generator function for succession. The result of applying succession to the two operations *name* and *notation* consists of all the *notation* elements that have *name* as the first element, or that have *name* as the last element and the atom *ARE* as the next-to-last. For instance, suppose the notation for the second operation is:

  $$N = ((Y \text{ THAT ARE } X)\ (Y \text{ THAT ARE NOT } X))$$

```
Function Succession-Result-Generator(name1,notation)
    Result := {};
    For n in notation do
        If name1 = first-element(n) OR
           (name1 = last-element(n) AND
            previous-to-last-element(n) = ARE)
        Then Result := Result ∪ {n};
    Return Result
```

Fig. 10. Function that computes the resulting notation after performing a succession of operations in Logic.

```
Rule Succession
    If o1 is a logic-class with n1 := name AND
       o2 is a logic-class with n2 := name AND
       there is no instance of combination-instance with:
            name = succession;
            simple-operation1 = o1;
            simple-operation2 = o2
    Then n := Succession-Notation-Generator(o1,o2);
         r := Succession-Result-Generator(n1,n);
         Create a new instance of combination-instance with:
                name = succession;
                simple-operation1 = o1;
                simple-operation2 = o2;
                result = r;
         Create a new instance of result-class with:
                notation = r
```

Fig. 11. Rule that performs the succession of two operations in Logic.

The output of Succession-Result-Generator(x, N) would be:

$((Y$ THAT ARE $X))$

4.2.3. Definition of rules for the combinations

BOOLE2 needs production rules to actually compute the combinations. These rules implement the operator "apply a combination to two operations" of the problem space as defined in Section 3. The operator variables to be instantiated are: combination, operation1, and operation2. The combination rules are all very similar. The left-hand side finds two operations (initial or the result of a previous combination) and a combination, such that the combination has not yet been applied to the two operations in this order. The right-hand side generates notations for the operations and the result, and defines two new instances: the corresponding result-* frame (the * stands for the simple-operation of the science involved), and a combination-instance frame. Fig. 11 illustrates the succession of two classes in Logic.

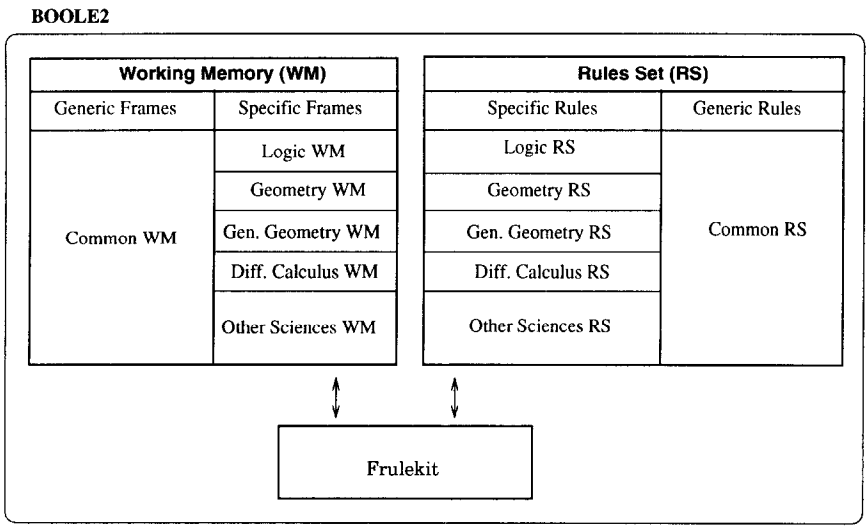

Fig. 12. Architecture of BOOLE2.

5. BOOLE2 as a production system

BOOLE2 is a production system with three classical components: working memory, rule set, and control. Its generic knowledge consists of a set of rules and frame descriptions common to any science, while its specific knowledge is composed of a set of rules and frame descriptions for each particular science. This partition into generic and specific is advantageous, since it allows creating a *discoverer* for each science, given the inputs for that science. Fig. 12 shows the system architecture.

The initial facts in working memory describe the input science prior to any test of its algebraic character. The intermediate and final states represent the plausible mental states of the scientist during his or her reasoning. The generic production rules determine whether a given science can be symbolized or not, while the specific rules describe how to perform combinations in the science. The control mechanism guides BOOLE2 towards the same kind of reasoning that is found in G. Boole's writings. Even though the science-specific knowledge is contained in BOOLE2, conceptually it has the role of an input.

Working memory in BOOLE2 consists of a set of frame descriptions and a set of frame instances. For example, Fig. 3 shows the hierarchy of frames in Logic. Working memory was already discussed in Section 4. The next two sections describe the production rules and control.

5.1. Rule set

The rules can be classified into the following groups:

Recording rules. These collect all laws that are found to hold in a given science.

```
Rule Symbolizable
   If s is a science AND
      c1 is a prototype-combination AND
      cl1 is commutative with combination = c1 AND
      c2 is a prototype-combination AND
      cl2 is commutative with combination = c2 AND
      d is distributive with combination1 = c1;
                             combination2 = c2;
   Then Modify s with symbolizable = True;
        Modify c1 with arithmetic-name = Times;
        Modify c2 with arithmetic-name = Plus
```

Fig. 13. Rule that decides whether a science is symbolizable.

General rules. These control the abstract reasoning for considering a science symbolizable. They are all science independent since they embody knowledge of the MSS. For instance, Fig. 13 shows an example of the rule that states whether a science is symbolizable or not. It tests whether there are two combinations that are each commutative and jointly distribute. If so, it concludes that the science can be symbolized, and it *renames* the combinations with their respective algebraic (arithmetic) names: *times* and *plus*. The renaming will allow a future system (under development) to operate with any symbolizable science in the algebraic problem space.

Law rules. These check the common laws for the sciences, such as *commutativity*, *distributivity*, *idempotence*, and *index-idempotence*. When a law rule fires, the system creates an instance of the frame corresponding to that law (see Fig. 3, part (c)), which asserts that the given combination fulfills the law. Other rules discover that a given combination does not fulfill a law, as in the case of the Generalized Geometry. The law rules are science-independent, except for the equality of notations test, which we discuss below. For example, the rule that tests commutativity looks for one combination-instance, the result r of applying a combination n to operations op1 and op2, and another combination-instance, the result $r1$ of applying n to the same combinations in reverse order, such that $r1$ is *equal* to r.

The science-dependent boolean function *equal-notation* returns `True` if two notations are equal in the given science. For instance, in Logic, the function `equal-notation` returns `True` if:

- The two notations are identical, or
- The difference between the sets *notation1* and *notation2* is empty. The difference is computed by removing notation elements that are equal. Two notation elements, n_1 and n_2, are equal if they are identical ($n_1 = n_2$), or if the first element of n_1 equals the last element of n_2, the first element of n_2 equals the last element of n_1, and *NOT* does not appear in either of the notation elements.
  For instance, of the following notation elements, only 1, 2 and 3 are equal.

Notation Element 1: (*X* THAT ARE *Y*)
Notation Element 2: (*X* THAT ARE *Y*)
Notation Element 3: (*Y* THAT ARE *X*)
Notation Element 4: (*X* THAT ARE NOT *Y*)
Notation Element 5: (*Y* THAT ARE NOT *X*)

The following two notations are equal:

Notation 1: ((*X* THAT ARE *Y*) (*X* THAT ARE NOT *Y*))
Notation 2: ((*X* THAT ARE NOT *Y*) (*Y* THAT ARE *X*))

As another example, the equality of two notations (formulae) in the Differential Calculus requires that the successive application of 17 mathematical transformation operators (or a subset of them) to one notation yield the second notation.

Combination rules. These science-dependent rules perform combinations over the operations of a science; these rules were already described in Section 4.2.3.

Forced combination rules. These science-dependent rules act as problem space heuristics by selecting which combination of two operations should be tried. These rules peform the combination, they do not just cede control to a combination rule. For example, in Logic the heuristic rule `forced-succession` guides the system to combine the two operations *o2* and *o1*, if it already has performed the opposite combination of *o1* and *o2*. If the results of both combinations are the same, the commutativity rule will then fire. This is human-like reasoning which Boole employed in his writings. Moreover, this is the likely way that one would try to prove that a given combination is commutative.

Continuation rules. These heuristic rules lead to uniformly applying the combinations. They prevent using the same combination twice in a row when other combinations can be applied instead, by giving priority to rules that did not fire during the last cycle.

5.2. Control mechanism

Execution is controlled by means of an agenda mechanism. Rules are organized in six priority levels with the more abstract rules at higher priority, and the more detailed rules at lower priority. In order of decreasing priority, the rule types are: recording rules, general rules, law rules, forced combination rules, combination rules, and continuation rules.

Beginning with the highest priority, the rules are matched[8] to select a rule that can fire. If none, then the next lower priority rules are matched, and so on. When a rule fires, control returns to highest priority rules. The agenda mechanism can be overriden with the forced combination rules and the continuation rules.

[8] FRULEKIT uses a Rete net for efficiently matching the rules.

5.3. Execution of the program

BOOLE2 is initiated by loading (into working memory and rules) the science-independent part of the program, followed by loading the science-dependent part (refer to Appendix A). A set of instances of the simple operations of the science is defined so that they can be combined. Guided by the control mechanism, the first applicable combination rule fires, which leads to firing other combination, law and control rules. Execution ends after the general rules have produced a definitive answer to the question of symbolizability, and the recording rules have filed the discovered laws.

5.4. Recapitulation on the input science in BOOLE2

We now recapitulate the previous subsections in order to provide a unified vision of how an input science is represented in BOOLE2, using Logic for illustration.

For Boole and his contemporaries, the theory of Logic was similar. Except for minor differences, Logic consisted of Aristotelian syllogism and the theory of the conditional. Texts other than Boole's were purely philosophical: they used mostly natural language and, for the sake of a clearer and shorter look, some occasional mathematical symbols, but these symbols lacked algebraic meaning and no mathematical operations involved them (see, for instance, [15] or [9]). In his "Mathematical Analysis of Logic", Boole did not try to mathematize the whole edifice of Logic. Instead, he chose one fundamental operation, the act of election, and two ways of combining it, succession and aggregation (see Section 2.2). This is not an oversimplification, since the acts of election (the equivalent to predicates in modern terms) and their combinations are sufficient to recreate and enlarge classical Logic. This captures essentially the entire content of Boole's book. Operations and combinations are the starting point for the reasoning reported in his book, which we therefore model in BOOLE2.

The act of election in Logic is represented in BOOLE2 as a subclass of the predefined `simple-operation` frame, and its representation (or notation) reflects the so-called linguistic way of reasoning, which becomes apparent when the notation is used to compute the effect of combinations. A choice of different notations is allowed, to reflect particular deductive strategies that are found in Logic (see Sections 4.1, and 4.2.2).

Two `combination-instance` frames represent the combinations of Logic, succession and aggregation. Their `result-generator` functions rely on a sophisticated examination of the English sentences that represent the operations to be combined (Section 4.2).

Some of the heuristics that lead to discovering the algebraic character of Logic are domain independent. However, testing for equality, which is a basic tool in Boole's Logic and is a law rule in BOOLE2, is a science-dependent heuristic. It is implemented as the `equal-notation` boolean function, and it is the final step of any logical reasoning that ends in the discovery of a law (see Section 5.1).

6. Future research

There are two open questions on which we are beginning work, corresponding to the two traditional AI approaches:

Cognitive approach. We intend to consider further sciences as inputs to the system. This approach has a cognitive focus because to use a new science, one must first represent that science as a suitable input to the system, either by studying how scientists represent the concepts of their field or by designing a wholly new representation. It would be interesting to match the representations used in specific sciences against the common representation entailed by the MSS and embedded in BOOLE2. Once the system discovers that a science is symbolizable, that science could be input to a future system that is sketched as follows.

Engineering approach. The goal would be to automate Boole's Logic. Since Logic is abstractly equivalent to Algebra, it can be manipulated in terms of algebraic properties and reasoning. Therefore, one could build an algebraic reasoning system for proving theorems in Logic. Since there are many potentially symbolizable sciences, as tested by BOOLE2, the same "algebraic theorem prover" could be applied to reason in different sciences using the common structure defined in BOOLE2. For example, proving a theorem in one of those sciences would be equivalent to solving a system of linear equations.

7. Related work

The main difference between BOOLE2 and other systems dealing with the discovery of historical laws, such as BACON, GLAUBER, STAHL and DALTON [24,25], AM and EURISKO [26–28], KEKADA [20,33], ARE [32] or CDP [1, pp. 46–57], is that BOOLE2 is exclusively guided by theory, instead of experimentation, i.e., it is not a data-driven discovery system. BOOLE2's input is not experimental data, but an abstract representation of some science. Further, many of its heuristics stem from theory, i.e., from the precise symbolic in which Boole reported his discovery of mathematical Logic.

8. Conclusions

Boole's work, as we understand it, bears on four significant fields:

- From the viewpoint of Scientific Discovery, Boole's achievement was finding that Logic is symbolizable, and that its symbolic operations belong to the family of operations that obey commutative, distributive and (special) index laws.
- As concerns the History of Mathematics, we have first understood the knowledge that Boole probably had, and the scientific and human historical circumstances. This understading has informed our acount of his historic discovery using the same terms and background that he reported.
- In terms of Cognitive Science, our contribution has focused on human intelligence, seeking to abstract a theory of an intelligent process. Boole applied the method to some truly difficult problems in the Differential Calculus and the Calculus of Finite Differences. He seemed convinced that the method was a symbolical calculus applicable not only to other branches of Mathematics, but also to other

branches of knowledge, if they had any promise of being based on similar laws. Boole's boldness was to imagine that human thought could be expressed in terms of operators that he called "elective" (since they represent "the act of election of a set of individuals from a universe of discourse"), and to check whether these operators could be combined in laws similar to algebraic laws. It is not by mere chance that he entitled his main logical treatise "The Laws of Thought", and that in his first public appearance, in an 1835 lecture on Newton, well before he met Gregory, he stated that his aim was to deal with Newton's mind rather than with Newton's biography [2, p. 1].

- Fourth, in terms of AI, we have found another example of an AI system that resembles how humans work in representing and solving problems. Boole's writings report the use of abstraction and generalization (between Algebra and Logic), goal-driven reasoning, and deduction from a given generalization to "classify" a science as symbolizable. The kind of representation that appears to be used by Boole in the process of discovery is very important: he explained the idea of operation in Logic in a linguistic way, i.e., in terms of natural language sentences. This representation was powerful enough to yield truly general rules. These reasons motivated the use of such representations and methods in our computer programs.

Our main research contributions can be summarized as follows:

- We have performed a detailed study of George Boole's discovery of Logic as an algebraic discipline by means of his application of the MSS that, at that time, had been used only within Mathematics.
- This study informed our computational model BOOLE2 of the steps recounted by Boole in his Logic writings. The science-dependent part of BOOLE2 is a computational representation of the subset of Logic that Boole chose to start his reasoning. The execution of BOOLE2 with Logic as input suggests that the system can proceed from the starting point—the logical definition of the act of election and the logical rules for combining instances of this act—to the conclusions reached by Boole himself. This shows that the knowledge given to the program suffices for a plausible explanation of how George Boole might have made his discovery and, therefore, that BOOLE2 itself is an approximation to a theory of this scientific discovery.
- BOOLE2's methods suggest that the process is general enough to be applied to other sciences, by just entering their initial description. Evidence for this generality is provided by BOOLE2's execution with Gregory's Geometry and a small subset of the Differential Calculus, which are found symbolizable, and a generalization of Gregory's Geometry, which is not.

Acknowledgments

We owe thanks to Professor Herbert A. Simon for encouraging us in the past to undertake the task of creating the BOOLE systems and for many subsequent and extremely fruitful discussions prior to this paper. We also thank Dr Raúl E. Valdés-Pérez and anonymous reviewers, whose helpful comments have contributed to improve our work.

Luis de Ledesma, Aurora Pérez and Luis M. Laita were supported in part by the Spanish Government and the Government of Madrid, projects DGCYT-PB910036 and 375/92.

Appendix A

Below is a detailed trace of the execution of BOOLE2 on Logic (but simplified by removing unimportant steps). Commentary is provided. Traces for other input sciences are available at `http://grial.uc3m.es/~dborrajo/boole/`.

```
> (load-boole 'logic)
```

This initial function loads the science-independent part of the system and the specific part corresponding to Logic. It also stores in working memory the instances X, Y, Z of generic simple operations of the science.

```
----------> Firing production AGGREGATION-1

****** Making the aggregation of Z and Z.
        The input notation is:
                ((ALL THAT ARE Z)) and
                ((ALL THAT ARE Z)
        The result is: ((ALL THAT ARE Z))
```

This rule is the first from the list of rules ready to fire. The rules at higher priority levels (recording, general, laws, and forced-combinations) can not yet be matched. This rule aggregates the simple operations Z and Z. To apply the aggregation, the rule first generates a notation for the two input operations, and then it generates a notation for their aggregation. The notation for an input simple operation is `(ALL THAT ARE operation-name)`, whereas the result notation is the union of the input notations, in which repeated or subsumed notation elements have been removed. Finally, a new simple-operation Z + Z is added to the working memory.

```
----------> Firing production IDEMPOTENCE

****** Verified the idempotence of the combination AGGREGATION
```

Since the result of aggregating Z and Z is equal to Z, the rule relative to idempotence fires.

```
----------> Firing production AGGREGATION-2

****** Making the aggregation of X and Z
****** for making succession with Z.
```

```
        The input notation is:
                ((X THAT ARE Z) (X THAT ARE NOT Z)) and
                ((ALL THAT ARE Z))
        The result is: ((ALL THAT ARE Z) (X THAT ARE NOT Z))
```

The system is still at the combination priority level. Control has been transferred to the aggregation-2 rule by the continuation rules. For each combination, there may be more than one rule, corresponding to different ways of the intended use of the combination. For the example case of aggregation, there are two different ways to generate notations for the input operations. The first way has appeared before, and this rule corresponds to the second way. This rule is used when BOOLE2 plans to apply succession to the result of the aggregation of two operations, to test distributivity.

```
----------> Firing production FORCED-AGGREGATION-1

****** Making the forced aggregation of Z and X.
        The input notation is:
                ((ALL THAT ARE Z)) and
                ((ALL THAT ARE X))
        The result is: ((ALL THAT ARE X) (ALL THAT ARE Z))
```

Since the system has earlier aggregated X and Z, the heuristic rule FORCED-AGGREGATION-1 matches, and applies the aggregation to Z and X, in order subsequently to test the commutativity of aggregation.

```
----------> Firing production COMMUTATIVE

****** The commutative holds for the combination AGGREGATION
```

Since the aggregation of X and Z has the same resulting notation as the aggregation of Z and X, aggregation is commutative. Note that checking whether two notations are the same is not trivial step.

```
----------> Firing production SUCCESSION-1

****** Making the succession of Z and Z.
        The input notations are:
                Z and
                ((ALL THAT ARE Z))
        The result is: ((ALL THAT ARE Z))
```

Succession works like aggregation: it generates input notations, and then applies succession to generate a result notation. The input notations are the operation-name of the first operation, and `(ALL THAT ARE operation-name-2)` when the two operations are the same.

```
----------> Firing production IDEMPOTENCE

****** Verified the idempotence of the combination SUCCESSION
```

Since the result of Z succession Z is Z, the rule for idempotence fires for the combination succession.

```
----------> Firing production SUCCESSION-2

****** Making the succession of Z
****** and ((ALL THAT ARE Z) (X THAT ARE NOT Z)).
        The input notations are:
                Z and
                ((ALL THAT ARE Z) (X THAT ARE NOT Z))
        The result is: ((ALL THAT ARE Z))
```

This rule is similar to SUCCESSION-1 but it need not generate a notation for the second operation since it is just a previous result (instead of one of the operations initially generated) and so a notation already exists for it.

```
----------> Firing production AGGREGATION-1

****** Making the aggregation of Z and Y.
        The input notation is:
                ((ALL THAT ARE Z)) and
                ((ALL THAT ARE Y))
        The result is: ((ALL THAT ARE Y) (ALL THAT ARE Z))

----------> Firing production FORCED-AGGREGATION-1

****** Making the forced aggregation of Y and Z.
        The input notation is:
                ((ALL THAT ARE Y)) and
                ((ALL THAT ARE Z))
        The result is: ((ALL THAT ARE Z) (ALL THAT ARE Y))

----------> Firing production AGGREGATION-2

****** Making the aggregation of X and Z
****** for making succession with Y.
        The input notation is:
                ((X THAT ARE NOT Y) (X THAT ARE Y)) and
                ((Z THAT ARE NOT Y) (Z THAT ARE Y))
        The result is: ((Z THAT ARE NOT Y) (Z THAT ARE Y)
                        (X THAT ARE NOT Y) (X THAT ARE Y))
```

```
----------> Firing production SUCCESSION-1

****** Making the succession of Z and Y.
        The input notations are:
                Z and
                ((Y THAT ARE Z) (Y THAT ARE NOT Z))
        The result is: ((Y THAT ARE Z))

----------> Firing production FORCED-SUCCESSION-1

****** Making the forced succession of Y and Z.
        The input notations are:
                Y and
                ((Z THAT ARE Y) (Z THAT ARE NOT Y))
        The result is: ((Z THAT ARE Y))
```

For similar reasons as in FORCED-AGGREGATION-1, once the succession of Z and Y is obtained, the succession of Y and Z must be performed.

```
----------> Firing production COMMUTATIVE

****** Commutativity holds for the combination SUCCESSION

----------> Firing production FORCED-SUCCESSION-2

****** Making the forced succession of Y and X.
        The input notations are:
                Y and
                ((X THAT ARE Y) (X THAT ARE NOT Y))
        The result is: ((X THAT ARE Y))

----------> Firing production FORCED-AGGREGATION-2

****** Making the forced aggregation of YX and YZ.
        The input notation is:
                ((Z THAT ARE Y))
        The result is: ((Z THAT ARE Y) (X THAT ARE Y))

----------> Firing production FORCED-SUCCESSION-3

****** Making the forced succession of Y and X+Z.
        The input notations are:
                Y and
                ((Z THAT ARE NOT Y) (Z THAT ARE Y)
```

```
                (X THAT ARE NOT Y) (X THAT ARE Y))
        The result is: ((Z THAT ARE Y) (X THAT ARE Y))

----------> Firing production DISTRIBUTIVE-1

****** The distributive holds for the
****** combinations AGGREGATION and SUCCESSION

----------> Firing production SYMBOLIZABLE

****** Science LOGIC is symbolizable

----------> Firing production INCORPORATE-SCIENCE

****** We add science LOGIC to symbolizable sciences


----------> Firing production INDEX-IDEMPOTENCE

----------> Firing production RECORD-IDEMPOTENCE

----------> Firing production RECORD-INDEX-IDEMPOTENCE

----------> Firing production RECORD-DISTRIBUTIVE

----------> Firing production RECORD-COMMUTATIVE

----------> Firing production INDEX-IDEMPOTENCE

----------> Firing production RECORD-IDEMPOTENCE

----------> Firing production RECORD-INDEX-IDEMPOTENCE

----------> Firing production RECORD-COMMUTATIVE
```

BOOLE2 records the laws of LOGIC using its record rules, and then summarizes its work as follows:

```
The science LOGIC is symbolizable.
Its laws are:
  COMMUTATIVE, with respect to the combination SUCCESSION
  INDEX IDEMPOTENCE, with respect to the combination SUCCESSION
  IDEMPOTENCE, with respect to the combination SUCCESSION
  COMMUTATIVE, with respect to the combination AGGREGATION
  DISTRIBUTIVE, the SUCCESSION with respect to the AGGREGATION
```

```
INDEX IDEMPOTENCE, with respect to the combination AGGREGATION
IDEMPOTENCE, with respect to the combination AGGREGATION

----- All combinations made (in order of creation) -----
Z+Z ==> ((ALL THAT ARE Z))
X+Z (Z) ==> ((ALL THAT ARE Z) (X THAT ARE NOT Z))  'RESULT1'
Z+X ==> ((ALL THAT ARE X) (ALL THAT ARE Z))
Z*Z ==> ((ALL THAT ARE Z))
Z*RESULT ==> ((ALL THAT ARE Z)) Z*RESULT1
Z+Y ==> ((ALL THAT ARE Y) (ALL THAT ARE Z))
Y+Z ==> ((ALL THAT ARE Z) (ALL THAT ARE Y))
X+Z (Y) ==> ((Z THAT ARE NOT Y) (Z THAT ARE Y)
             (X THAT ARE NOT Y) (X THAT ARE Y))    'RESULT2'
Z*Y ==> ((Y THAT ARE Z))
Y*Z ==> ((Z THAT ARE Y))                           'RESULT3'
Y*X ==> ((X THAT ARE Y))                           'RESULT4'
RESULT+RESULT ==> ((Z THAT ARE Y) (X THAT ARE Y))  RESULT4+RESULT3
Y*RESULT ==> ((Z THAT ARE Y) (X THAT ARE Y))       Y*RESULT2
```

References

[1] *Working Notes of the AAAI Spring Symposium on Systematic Methods of Scientific Discovery*, Stanford University, Stanford, CA (1995).

[2] G. Boole, *An Address on the Genius and Discoveries of Sir Isaac Newton* (Gazete Office, Mechanics Institution, Lincoln, 1835).

[3] G. Boole, *The Mathematical Analysis of Logic, Being an Essay towards a Calculus of Deductive Reasoning* (Macmillan, Barclay and Macmillan, 1847).

[4] G. Boole, *The Claims of Science, Especially as Founded in its Relation to Human Nature* (Taylor, Walton and Maberley, London, 1851).

[5] G. Boole, *An Investigation of the Laws of Thought, On Which Are Founded the Mathematical Theories of Logic and Probabilities* (Walton and Maberley, London and Cambridge, 1854).

[6] L. de Ledesma and L.M. Laita, G. Boole. From differential equations to mathematical logic, in: *Actas de la Reunión Matemática en Honor de A. Dou*, Madrid (Publicaciones de la Universidad Complutense de Madrid, 1989) 341–352.

[7] L. de Ledesma, L.M. Laita, A. Pérez and D. Borrajo, Descubrimiento científico e inteligencia artificial, in: *Real Academia de Ciencias Exactas, Físicas y Naturales, Proceedings of the Segundo Curso de Conferencias Sobre Inteligencia Artificial* (1993) 115–132.

[8] L. de Ledesma, A. Pérez, D. Borrajo and L.M. Laita, La lógica de Boole como consecuencia del método de separación de símbolos de Gregory Boole. Estudio histórico y su emulación por computador, in: *Contra los titanes de la rutina. Contre les titans de la routine. Proceedings of the Congreso Hispano-Francés Sobre la Historia y la Filosofía de la Matemática* (Comunidad de Madrid/CSIC, 1994) 287–300.

[9] A. De Morgan, *Formal Logic or The Calculus of Inference Necessary and Probable* (Taylor and Walton, London, 1847).

[10] S. De Morgan, *Memoir of Augustus De Morgan* (Longman, Essex, 1882).

[11] S. Bachir Diagne, *Boole: L'Oiseau de Nuit en Plein Jour* (Belin, Paris, 1989).

[12] M. Everest, Home side of a scientific mind, *The Dublin University Magazine* **91** (1878) 103–114, 173–183, 327–336, 456–460; Reprinted in *Mary Everest Collected Works*.

[13] D.F. Gregory, Demonstrations, by the method of separation of symbols of theorems in the differential calculus and calculus of finite differences, *Cambridge Math. J.* **1** (1838) 232–244.
[14] D.F. Gregory, On the elementary principles of the application of algebraic symbols to geometry, *Cambridge Math. J.* **2** (1839) 1–9.
[15] W. Hamilton, *Lectures on Metaphysics and Logic*, Vol. 4 (W. Blackwood and Sons, Edinburgh and London, 1862).
[16] R. Harley, G. Boole, *British Quarterly Review* (July 1886); Reprinted in: R. Harley, *Studies in Logic and Probability. Boole's Collected Logical Works*, pp. 425–472.
[17] W. Kneale and M. Kneale, *The Development of Logic* (Clarendon Press, Oxford, 1961).
[18] E. Knobloch, Symbolik und Formalismus im mathematischen Denken des 19. und beginnenden 20. Jahrhunderts, in: *Mathematical Perspectives, Essays on Mathematics and its Historical Development* (J.M. Dauben, New York, 1981) 208–216.
[19] E. Koppelman, The calculus of operations and the rise of abstract algebra, *Archive for the History of Exact Sciences* **8** (1971).
[20] D. Kulkarni and H.A. Simon, The processes of scientific discovery: the strategy of experimentation, *Cognitive Sci.* **12** (1988) 139–175.
[21] L.M. Laita, The influence of Boole's search for a universal method in analysis on the creation of his logic, *Annals of Science* (1977) 163–176.
[22] L.M. Laita, Influences on Boole's logic, The controversy Hamilton–De Morgan, *Annals of Science* (1979) 45–65.
[23] L.M. Laita, Boolean algebras and its extralogical sources, *History and Philosophy of Logic* (1980) 37–60.
[24] P.W. Langley, J.M. Zytkow, H.A. Simon and G.L. Bradshaw, The search for regularity: four aspects of scientific discovery, in: R.S. Michalski, T.M. Mitchell and J.G. Carbonell, eds., *Machine Learning* **2** (Morgan Kaufmann, Los Altos, CA, 1986) 425–469.
[25] P.W. Langley, J.M. Zytkow, H.A. Simon and G.L. Bradshaw, *Scientific Discovery. Computational Explorations of the Creative Processes* (MIT Press, Cambridge, MA, 1987).
[26] D.B. Lenat, Automated theory formation in mathematics, in: *Proceedings IJCAI-77*, Cambridge, MA (1977).
[27] D.B. Lenat, The nature of heuristics, *Artificial Intelligence* **19** (1982) 189–249.
[28] D.B. Lenat, Eurisko: A program that learns new heuristics and domain concepts, *Artificial Intelligence* **21** (1983) 61–98.
[29] D. MacHale, *G. Boole, His Life and Work* (Boole Press, Dublin, 1985).
[30] M. Panteki, Relationships between algebra, differential equations and logic in England, 1800–1860, Ph.D. Thesis, Council for National Academic Awards, UK (1991).
[31] P. Shell and J.G. Carbonell, FRuleKit: A frame-based production system, User's manual, Internal paper (1989).
[32] W.M. Shen, Functional transformations in AI discovery systems, *Artificial Intelligence* **41** (1990) 257–272.
[33] J. Shrager and P. Langley, *Computational Models of Scientific Discovery and Theory Formation* (Morgan Kaufmann, Los Altos, CA, 1990).