An Evolution of Mathematical Tools

From Conceptualization to Formalization

Here's what we do when we build a *formal model* (or *do a computation*):

0. Identify a collection of objects/events in the real world. This is the *semantic mapping*, how math is linked with reality. The objects/events must have these properties:

> *unique*, not confused with different objects/events
> *stable and permanent*, not in flux or changing too rapidly to identify
> *discrete*, not lacking well defined borders
> *comprehendible*, not confusing or too ambiguous
> *relevant*, not outside of what we consider to be the objects in question
> *permitted*, not in violation of tacit understandings about how things are

1. Use unique labels to identify each of the things in the semantic mapping. The *value* of a label is the thing it identifies.

2. Limit our interest in the types of things in the real world to an abstract mathematical property, such as Truth or Count or Membership.

3. Use different labels to name different *abstract* things:

labels for things	*object labels*
labels for an entire set of things	*property labels*
labels for an arbitrary thing in the set	*variable labels*
labels to name properties of things indirectly	*function labels*
labels to name combinations of things	*relation labels*

4. Follow the *rules of symbol transformation* in manipulating the labels as if they were the things. However, the labels do not have to share any of the real world properties of the physical things.

In summary, we convert from physical to virtual, ignore the physical aspects of reality, manipulate the virtual (or digital) aspects using the rules of virtuality, and then return to physical reality with new knowledge. The rules of virtuality are listed in the handout *The Canons of Formal Symbol Systems*.

Virtuality has become so prevalent in the current Information Age that much of what we do is never part of physical reality in the first place. Computer Science, for example, is a discipline in which the only connection to physical reality is silicon hardware. Study of the physicality of hardware is a different discipline, Electrical Engineering.

Computational hardware is engineered (constructed) to behave as if it were a mathematical system called *Timed Boolean Logic*. "Timed" simply means that some parts of the physical hardware are used more than once to do the Boolean logic task for which they were constructed.

Formal Simplicity

The simplest interesting formal domain is **Propositional Calculus**, also known as **Boolean Logic**, and also known as **Boolean Algebra**.

In Boolean algebra, the labels for real world objects are sentences in human language which can be evaluated as either True or False. These sentences are composed of atoms (propositions) and logical connectives. The atoms are labels for things which exist and are in some way actual, or real, or True. The logical connectives are those parts of language which do *not* refer to something in physical reality.

More generally, Boolean algebra includes all of *decision theory*, which phrases a problem space as a sequence of binary decisions. (Note that more complex decisions can usually be phrased as a collection of binary decisions.)

The truth of a sentence, the direction of a decision, and the voltage in a digital circuit wire are all binary properties. Every object label in a Boolean problem has the *binary property*, either 0 or 1, and that is the only property of interest. Logic is the set of rules which maintains the invariance of the value of binary properties.

Labeling Reality

Let {a, b, c, ...} be simple Boolean variables identifying a set of decision events, each either 0 or 1.

The ways which we combine Boolean variables have been part of our language from antiquity. They are the familiar operators: {not, or, and, if, equal, if-then-else}. Combinations of Boolean operators and Boolean variables are called *sentences* or *expressions*.

Let {A, B, C, ...} be names of Boolean expressions.

Valid Boolean expressions are defined recursively,

a	simple variables are expressions
not A	negated expressions are expressions
A or B	expressions joined by disjunction are expressions

Since all other Boolean operators can be expressed in terms of **not** and **or**, they are all included in the above definition. This definition means that Boolean expressions are composable and decomposable; the recursive rules provide Constructors and Accessors.

Symbolic Complexity

The truth table (or Boolean property table) of a Boolean operator specifies how that operator combines the Boolean properties, or the values, joined by the operator. But here things start to get complex. Mainly, we can form a huge variety of expressions, and to find the Boolean value of any expression, we must examine a table with a size exponential in the number of variables

(2^n entries). So a logic circuit with 200 input variables requires an impractical 2^200 rows in the truth table.

So all decision problems which require a large number of decisions become intractably inconvenient to examine in the whole. Boolean algebra provides *transformation rules* which hold for all expressions, regardless of which Boolean value an expression has. For example,

not (not A) = A

is valid regardless of the expression A.

Axioms and Theorems

Transformation rules can be decomposed into axioms and theorems. The axioms are a minimal set of rules which preserve the Boolean property and serve as a basis for all other possible theorems. All the axioms of an axiomatic formal system can be compressed into one single, usually complex axiom. For Boolean logic, the simplest single axiom which supports all other Boolean transformations is the Kauffman/Flagg formalization:

(A or B) and (A or (not B)) = A

Similarly, the Resolution rule is a single rule which supports all Boolean transformation. The many rules of Natural Deduction provide another, more complex set of axioms. One particularly simple set of supporting axioms (Bricken's formalization of Boundary Logic) is the following:

A or True = True
not (not A) = A
A or (not (A or B)) = A or (not B)

This set is particularly useful for pattern-matching and automated deduction, since the patterns to be matched are relatively simple, and the right-hand-side of each equation is simpler than the left-hand-side solely through erasing something from the right-hand-side.

Proof

Boolean algebra axioms and theorems (valid transformations) provide a way to explore decision spaces without making the actual decisions. This is called *logical or algebraic proof*. However, the situation remains complex because now we must select which theorem to apply and where to apply it. Although the search space is more abstract, it is still intractable and inconvenient. Although Boolean algebra abstracts the physical properties of decisions, it is still a *real world* problem to use Boolean algebra efficiently.

Predicate Calculus

The next extension to mathematical technique is to permit more complex types of objects. Instead of being restricted to atoms with no internal complexity, predicate logic lets us form *complex objects* in two ways:

1. *Quantification*:

Property labels were introduced at the very beginning in order to identify a set of things we wish to talk about. Without them, we could not group any things together, we could only talk about specific particular things. (This is the difference between *Arithmetic* and *Algebra*.)

Quantification lets us address sets or collections of things directly within a logical expression. The **forall** quantifier permits transformation of property labels (entire sets) rather than only individual members of the set. The **exists** quantifier allows us to refer to the existence of at least one object in a set, a *generic object*, without saying which particular object we are talking about. The generic object is an arbitrary object; we choose one object from the set of interest, but we do not say exactly which object it is. Thus, the generic object can stand in place of any particular object in a collection.

2a. *Relations*:

Instead of non-decomposable atoms having a Boolean property, we permit arbitrary compositions of atoms to have that property. The arbitrary compositions, or *relations*, are specified by a table of associations. If the association is in the table, then the relation between two (or more) simple objects is True, otherwise it is False. Only simple objects or atoms can be in relationship to each other. Expressions cannot be in a relation, but a relation (such as *aRb*), is an expression and can be combined with other expressions by Boolean operators.

2b. *Functions*:

Technically functions are a structured subset of relations, since **F(a)=b** can be expressed as **aFb=True**. Functions let us name objects indirectly by naming other objects and the function which tells us how to get to the object in question. For instance, the function **add1** moves us from one integer to the next.

The two extensions are sufficient to characterize all of standard mathematics, and therefore all of computation.

Other Formal Systems

The art of mathematical modeling is to find properties of real world objects which are both understood mathematically (i.e. formal) and useful for physical description. Predicate calculus provides the framework for formally defining arbitrary properties, but without special computational techniques, each structure becomes difficult to use for computation. Other formal systems provide specialized techniques for speeding up particular types of computation.

Induction/Recursion: This fundamental technique lets us work with a general case of a set of expressions, so long as the entire set is in some way structured by an ordering relation. We can then analyze entire classes of expressions by looking at only the base case and the general recursive case. This then provides a programming style.

Relational Calculus: The techniques of computing with only relations have been developed by the relational database community. The query language SQL exemplifies this approach.

Lambda Calculus: Likewise, the techniques of computing using only functions have been developed primarily as programming languages by the functional programming community.

Combinators: The combinator calculus is an elaboration of lambda calculus which turns functional operations into string operations. The specific abstraction technique removes all variable labels.

String Rewrite Theory: When expressions are represented by strings of tokens, the rules of transformation can be formulated as changes in the strings, using matching and substitution for effecting changes. This is the primary technique of mathematical manipulation and of theorem proving.

Graph Theory: Almost all data structures can be expressed using graphs. A graph rewrite theory supports valid graph transformations. As well, the graph data structure provides information about the problem itself, such as the shortest path (fewest number of transformations) to get from one expression to an equivalent one. This extremely general technique is underdeveloped solely because graphs are two dimensional structures, while language has developed using one dimensional strings.

Probability Theory: When the property of interest is a composite of the individual properties of a sample of objects, that property is a *probability*, and the rules for manipulating this statistical object are expressed as probability theory.

Cellular Automata: This is an exotic formal system based on a regular array of autonomous cells acting on their local neighborhoods. These models have limited but growing applicability.

Fractals: This technique is based on recursive functions usually expressed on a set of two or higher dimensional objects. Again, fractals are new since they apply to spatial rather than linear structures.

Boundary Mathematics: This is a general technique which converts all mathematical string-based operations (logic, sets, numbers, and equations) into spatial operations. The zero concept is the Void, that is, no representation at all. As a consequence, Boundary Mathematics is far more efficient in both representation and in transformation than all other mathematical formalisms. As a spatial technique, it suffers from both unfamiliarity and from the cultural bias of expressing mathematical operations in string languages.