

# Modeling Versus Formalization

Cezar Ionescu      Nicola Botta

September 19, 2008

## Abstract

Reading the programme of the conference, we were somewhat surprised to discover that the Working Group for which the present article is meant to serve as a background paper is supposed to “explicitely focus on the potential and limitations of mathematical formalization in the social sciences”. The reason for our puzzlement is that the titles of all WGs prominently figure “model” or “modeling”, but none of them mentions “formalization”. We asked ourselves: is a model the same as a formalization? The terms formalization and model are defined in logic: can these definitions be extended to mathematics or natural sciences? What can we say about the limits of formalization in logic, mathematics and the natural sciences? What sort of formalization is practiced in the social sciences, and what are its limits? The reader is invited to examine our (tentative!) answers to these and similar questions.

## 1 Introduction

Let us start by examining an early multi-agent model, developed, to the best of our knowledge, around 1595, and presented at the beginning of Act 5 Scene 5 of *Richard II* by William Shakespeare (some would add, again, to the best of our knowledge) from the point of view of the modeler, the now-deposed king Richard. Richard starts by stating his intention (quotations after the third Arden Shakespeare Edition, [14]):

I have been studying how I may compare  
This prison where I live unto the world;

The downfall of Richard has been swift and, to him, completely surprising. He is trying to understand how this could have happened, to make sense of his condition. He is attempting to understand something very complex, “the world”, through something much simpler and at hand: “this prison” (actually, as is apparent in the context, his cell). He goes on:

And, for because the world is populous  
And here is not a creature but myself,  
I cannot do it.

This is a criticism of the means available. The prison cell seems too simple to adequately represent the world, and in particular an important aspect of it: the fact that it contains a great number of people. Richard is not satisfied with the single human being present in the cell, himself, as a model of all other human beings. He doesn't want to study, say, the anatomy of human beings, but their interactions: he is interested in the social aspect of the world, as opposed to the physical. He is looking for a multi-agent model. Otherwise, the complaint that the means are not adequate to the task is a common one, among modelers as well as among critics of models.

Yet I'll hammer't out.

continues Richard, and, indeed:

My brain I'll prove the female to my soul,  
My soul the father, and these two beget  
A generation of still-breeding thoughts;  
And these same thoughts people this little world,

he does find something in his cell: the material (thus, inferior, "female") engine of the brain, animated by the immaterial, immortal, soul, can create thoughts, which are entities that keep on reproducing ("still-breeding"). The numerical aspect is thus resolved, and an important structural similarity is established: the thoughts populate the inner world of Richard as the people populate the outside world, they live, they breed, and, we shall see, they die.

Now, Richard puts his model to work. Direct examination of his thoughts leads to the conclusion that, because of their inhomogeneity, thoughts are in permanent competition and "no thought is contented". This corrupts even the transmission of Divine wisdom, "the word":

The better sort,  
As thoughts of things divine, are intermixed  
With scruples and do set the word itself  
Against the word, as thus: 'Come, little ones';  
And then again:  
'It is as hard to come as for a camel  
To thread the postern of a small needle's eye.'

Other examples are drawn from his current experience:

Thoughts tending to ambition, they do plot  
Unlikely wonders – how these vain weak nails  
May tear a passage through the flinty ribs  
Of this hard world, my ragged prison walls,  
And, for they cannot, die in their own pride.  
Thoughts tending to content flatter themselves  
That they are not the first of Fortune’s slaves  
Nor shall not be the last, like silly beggars  
Who sitting in the stocks refuge their shame  
That many have and others must sit there;  
And in this thought they find a kind of ease,  
Bearing their own misfortunes on the back  
Of such as have before endured the like.

Richard now draws the first conclusion from his model: the strife among his thoughts translates to a corresponding inconsistency and discontent in his behavior:

Thus play I in one person many people,  
And none contented. [...]

Now Richard can use himself as a representative human being, and generalize to *all* human beings:

But whate’er I be,  
Nor I nor any man that but man is  
With nothing shall be pleased till he be eased  
With being nothing.

This is the understanding that Richard has reached. The divine order in which he had believed without question is not only not realized in this, we could say, sinful world, but in fact not realizable: thoughts are “in humours like the people of this world” and “no thought is contented”. This constant state of strife and change explains, perhaps, his downfall, and maybe in this understanding Richard too finds “a kind of ease”.

We have used this example, somewhat tongue-in-cheek, to highlight some typical features of models. A correspondence is established between an object of study (the social world) and a simpler otherwise more accessible one (the inner world of thoughts). The simpler object is then examined independently of the original one, and the conclusions reached are then

transferred to that initial object. The transfer is a delicate procedure, in need of justification: notice that Richard is not satisfied with the structural correspondence established in the beginning, but also checks the plausibility of the “thoughts – people” interpretation by introducing his own person into the argument.

In the context of examining the limits of formalization, this example has an additional quality, since it is a model that is *not* a formalization. In a certain sense, it already establishes a limit to formalization: an aesthetic one. But, before examining formalization in the context of social science, we need to take a look at it on its “home ground”, as it were, that is, in Logic.

## 2 Formalization and model in logic

In logic, to formalize something is to set up a formal system together with an interpretation to that something.

A formal system is, essentially, an idealized entity for generating, from “valid” objects and according to precisely specified rules, other “valid” objects. The objects in question are usually represented symbolically, but the formal system as such is considered distinct from its symbolic representations.

As an example, consider the following formal system:

1. 0 is a valid object.
2. If  $x$  is a valid object, then  $S x$  is also a valid object.
3. Nothing is a valid object, unless it is either 0 or has been constructed according to the second rule.

It is easy to see that a valid object cannot be both 0 and constructed according to the second rule: for the second rule constructs only objects which begin with an  $S$ , and 0, we can see by inspection, does not.

It is almost as easy to see that all valid objects end with a 0, for 0 certainly ends with a 0, and if an object ends with a 0, then the rule constructs another object which ends with a 0. This argument, by structural induction, uses the third rule: there are no “spurious” valid objects we haven’t been told about.

An interpretation is, roughly speaking, a systematic way of translating the objects of the formal system in terms of the “something” that is supposed to be formalized. We can interpret the objects of our example as natural numbers: 0 stands for the natural number 0, and  $S x$  stands for the natural number obtained by adding 1 to the interpretation of  $x$ . It is clear that the formal system generates only objects that can be interpreted as natural

numbers, and that for any natural number there exists an object generated by the system which is interpreted by that natural number.

Thus, we can say that we have formalized the set natural numbers. But this is a very poor formalization indeed. First of all, the interpretation is certainly not unique. Indeed, consider a singleton  $\{*\}$ . We can interpret 0 to “mean”  $*$ ,  $S$  to be the unique endo-function on  $\{*\}$ , the identity function, and  $S x$  to be the result of applying the identity function to the interpretation of  $x$ . Then all objects generated by our system stand for the same element of the singleton set:  $*$ , and  $S$  is the unique function from a singleton set to itself. Therefore, we can with equal justification assert that we have formalized a singleton set.

In order to make our formal system do a better job for natural numbers, we have to embed it into a much more interesting formal system: that of first-order logic with equality. The objects that the new formal system generates will be *sentences*, instead of just stand-ins for natural numbers, and “valid” object will mean “true sentence”. The sentences involved use the familiar logical connectives and quantifiers ( $\neg$ , *and*, *or*,  $\rightarrow$ ,  $\leftrightarrow$ ,  $\forall$ ,  $\exists$ ). The valid objects we start with are meant to express our understanding of natural numbers, for example, we have

$$\forall x \neg (S x = 0)$$

which, if we interpret equality in the usual fashion, allows us to exclude the singleton interpretation. We could still have an interpretation with two elements: 0 and 1, taking  $S$  to be the constant function 1. In order to eliminate this interpretation, we add a sentence stating that  $S$  is injective:

$$\forall x (\forall y \neg (x = y) \rightarrow \neg (S x = S y))$$

We generate new, true sentences, from such given sentences with the help of the usual rules of predicate logic. For example, if we have generated a sentence  $p$  and another one  $p \rightarrow q$ , we can generate  $q$ , and so on. There is one new rule which allows the use of induction *within* the formal system, in much the same way that we have used structural induction *outside* the system in order to establish that all objects generated by our simple formal system end with 0. This rule says that if we have generated a sentence of the form  $P 0$ , and another of the form  $P x \rightarrow P (S x)$ , we are entitled to generate the sentence  $\forall x (P x)$ .

This system does a pretty good job of formalizing natural numbers. We still have “non-standard” interpretations (see Skolem’s theorem, below), but even these are interesting and related to natural numbers (and, besides, quite useful in programming languages such as Haskell).

It is moreover easy to “go on formalizing”. Addition, for example, can be represented by trying to make sure that all true sentences about addition

can be generated by the system. In order to do that, we need to add as valid sentences the following two:

$$\begin{aligned} &\forall x (x + 0 = x) \\ &\forall x (\forall y (x + S y = S (x + y))) \end{aligned}$$

The kind of valid objects we have thrown in in order to jumpstart the generation process are called *axioms*, while the objects which are generated on the basis of these axioms are usually called *theorems*. This terminology, which is quite natural when the objects are meant to represent sentences, reveals that the historical origins of formal systems are in the study of logic. A translation of the elements of the formal system is an *interpretation* if the axioms and the generated theorems come out true under the translation.

What is a model, then? A model is the target of the interpretation: in other words, that which we set out to formalize (if we were successful). Notice the inversion: we start out with something, for example informal arithmetic. We want to create a formal representation of it, something that might very well be called “model” according to everyday usage. Then, assuming that we are successful, the formal representation acquires priority, and the thing we started out with becomes relegated to being just a model. Goldblatt calls this “a typical ploy in mathematics” and continues “a formal representation becomes an actual definition” ([8]). When this terminology was introduced, this move was justified by the fact that the formal systems *were* considered more important. Model theory, introduced (or at least christened) around 1954 by Tarski as the study of the “mutual relations between sentences of formalized theories and mathematical sentences in which these sentences hold” ([15]), represented then a kind of Hegelian synthesis between “material logic” (thesis) and “formal logic” (anti-thesis).

With this understanding of formalization, we can make the following two points. The first is that there are well known limits to this formalization, expressed by the so-called limitative theorems of Gödel, Church and Skolem, which apply to formal systems adequate for primitive recursive arithmetic. A certain amount of technical machinery is needed to state these theorems, much more than we can present here. We can just point to relevant literature ([3]) and lift a short summary from DeLong’s popular text [7]:

1. Gödel’s first incompleteness theorems: There exists a predicate such that there is no correct and complete formal system for it.
2. Church’s theorem: There exists a predicate such that there is no correct formal system which contains a decision procedure for both the predicate and its negation.
3. Skolem’s theorem: There is no consistent, categorical formal system having the natural numbers as its intended interpretation.

These are among the most important results in logic ever, but as far as the limitations of formalization applied in the social sciences are concerned, they do not seem to be more relevant than the aesthetical limitation mentioned in the previous section. Of rather more immediate concern seems to be the second point announced above, which is this: formalization as described seems to be conspicuously absent: nobody seems to do it! There is, apparently, a kind of pragmatic limitation to formalization, even in mathematics, not to mention the natural sciences. This is what we'll discuss in the next section.

### **3 Formalization and modeling in mathematics and natural science**

A formal system such as that of first-order logic provides us with a firm standard of proof. The rules of the game are simple and can be checked mechanically, requiring no cleverness at all: indeed, a basic requirement is that they can be implemented on a computer. Thus, any disagreement about whether a particular argument establishes a claim from given assumptions can be resolved by translating the assumptions into valid objects of the formal system, and attempting to generate a translation of the claim by translating the steps of the suspicious argument into legal moves of first-order logic. If this works, the argument is valid. If it doesn't, one might attempt to "repair" it, by obtaining the translation of the claim by other moves and translating these back into steps of the argument, or one might be able to show that in fact no argument will be able to establish the claim from the assumptions. In any case, a mechanical check will always be sufficient to tell us whether a formalized argument is, in fact, a proof or not, and, as we know from the limitative theorems, this is as far as we'll ever get to the leibnizean dream of resolving philosophical debate by calculation.

Yet, even in logic books formal proofs are only given for relatively simple statements, and the more one advances into substantive matters, the fewer formal proofs show up. For example, in the chapter on first-order set theory of a popular textbook ([16]), we find

The one thing we won't be doing very much is constructing formal proofs of set-theoretical claims. This may disappoint you. Many students are initially intimidated by formal proofs, but come to prefer them over informal proofs because the rules of the game are so clear-cut. For better or worse, however, formal proofs of substantive set-theoretic claims can be hundreds or even thousands of steps long.

The problem is that formal proofs suffer from the "forest versus trees" syndrome: they are placed squarely on the side of the trees, and present an almost insurmountable obstacle to anyone interested in the forest. Here is

how Mac Lane explains the situation ([11]):

[...] There are good reasons why Mathematicians do not usually present their proofs in fully formal style. It is because proofs are not only a means to certainty, but also a means to understanding. Behind each substantial formal proof there lies an idea, or perhaps several ideas. The idea, initially perhaps tenuous, explains why the result holds. The idea becomes Mathematics only when it can be formally expressed, but that expression must be so couched as to reveal the idea; it will not do to bury the idea under the formalism.

The problem is analogous to that of writing a program in assembly language on a modern computer. The individual instructions in the program are easy to understand, but the sheer number of instructions necessary to perform even simple tasks (say, compute the factorial of a natural number entered from the keyboard) makes understanding the program an enormous undertaking. It is extremely difficult to make sense of an assembly-language program just from looking at its components, without having the benefit of comments or some guidance from the programmer who wrote it, and it's not that easy even with those. The situation is the same for a derivation in a formal system.

In both cases, the solution is the same: a higher-level language is proposed, and its connections to the lower-level language are made as precise as necessary for the translation to that lower-level to be trivial. In the case of programming, "trivial" means mechanizable, implementable as a compiler. In the case of mathematics, "trivial" means rather "easy for an expert, but very, very boring". In both cases, the tools for "raising the level of abstraction" are similar: definitions of new concepts, special notation, short cuts for the rules of inference, etc. These all allow a bootstrapping process to start up, so that new concepts are defined in terms of the already defined ones, new rules are stated that can be reduced immediately to the previously introduced one, but perhaps not so immediately to the rules of a formal system. There is a gradual loss of the contact with the lowest language, and in this loss lies a frequent possibility of error: the programmer, or the mathematician, come to rely on their intuition of what the definitions, notations, rules are *supposed* to represent, and not on what they have been *stated* to be, with the same kind of result: the program crashes, the proof is flawed.

Accordingly, formalization in mathematics consists of "giving formal expression to a mathematical idea", as Mac Lane puts it, but this formal expression need not be actually written (one could say implemented) in an existing formal system: the requirement is that it is sufficiently clear how such an implementation could be done if necessary. If one wants to distinguish this kind of activity from the formalization practiced in logic, one can perhaps refer to semi-formalization or mathematization.

Moving further away from the level of formal systems, we come to the natural sciences. Actually, the expression “formalization” is not used in this context frequently, and when it is, it usually denotes a process rather more similar to the mathematization described above. This time, the ideas to be translated are specific to the discipline to be studied, for example, in the case of physics, one might think of ideas of mass, force, fluid, temperature, or time, and the language in which the translation is to be done is that of mathematics (for example, partial differential equations, Hilbert spaces, or differential geometry are taken for granted). The translation is not always as trivial as in the case of mathematics, and usually includes conditions to be verified not just by intuition, but also by means of experiments and summarized as experimental protocols. Again, the point is not that these translations could not, *in principle*, be continued to the level of a formal system: of course they could, if we accept that the mathematical ideas interposed, as it were, between the physical and the logical could be so translated. This is not an impossibility in principle, but one in practice: a lot of work would need to be done for no apparent gain.

Finally, let us examine what “modeling” means in the context of mathematics and the natural sciences. We are interested, in particular, in the notion of “mathematical model”. Here is Mac Lane, once again ([12]):

The term (*mathematical model*) can be a precise one: a mathematical model proposes to replace a given empirical situation by an axiomatic structure, where specified quantities, numbers, or concepts are assumed to satisfy certain axioms (such as equations). Then, the consequences deduced from these axioms can be applied to the given empirical situation once the input quantities can be accurately assessed.

This usage of “model” corresponds quite well to the everyday one, and represents a reversal of the usage seen in the context of logic. If one were to follow the latter, one would say that the empirical situation is a model of the mathematical structure which was the result of the process of mathematization. This kind of usage of model, meant to denote the representation of an idea in a language closer to the level of formal systems, is also seen in mathematics. The logic textbook cited above refers, for example, to the set theoretical representation of relations as subsets of cartesian products “a model of relations”, while Goldblatt refers to it as “a formalization of the intuitive idea” of relation ([8], p. 18). Goldblatt is not entirely consistent in this usage, since he also refers to a “set-theoretic *model* of the intuitive idea of a function” only two pages later, and also uses “definition” and “characterization” in similar contexts. It should not be concluded that this is the result of a hopeless confusion of notions; rather, the various terms are used to alert the reader to where the emphasis will be placed in the con-

text: on the more formal representation, on the less formal ideas, or on the translation from the one to the other.

## 4 Formalization and programming

By far the largest group of users of formal systems is that of programmers. Indeed, there are at least two ways of viewing the relationship between programs and formal systems:

1. A program can be interpreted as a valid object to be processed according to the rules of a formal system. An order relation is given on the valid objects which can be generated from the program (for example, an object generated on the basis of another one is considered “smaller”) and the output of the program is the smallest object so generated (if it exists). For example, a program might be represented as a lambda-term ([9]), or an input string to a universal Turing machine. In the first case, the order between lambda-terms is given by  $\beta$ -reduction, and the smallest valid object is one which cannot be so reduced anymore. In the second case, the smallest object is the string left on the tape of the machine when it enters its halting state.
2. A program can be seen as giving the rules of a formal system for modifying an initial state of the computer to a final state via a sequence of “legal” intermediate states. Again, an order on these states is assumed (intended to represent the temporal evolution of the computer). For example, a program that implements the reduction rules of the lambda calculus is such a formal system, and the states are represented by lambda-terms. Similarly, a program that implements a universal Turing machine transforms pairs of internal state and tape state until a pair is reached for which the internal state is the halting one.

These two points of view correspond (very) roughly to denotational and operational semantics, respectively (see, for example, [17]).

Therefore, a programmer implementing, say, a product ordering system for a company can be said to formalize the customer’s ideas of ordering the products of the company. That would certainly be the case, except for one (big!) problem: the interpretation of the formal system is nowhere near systematic enough to satisfy the logical criteria of formalization. In fact, at present, in most so-called “real-world” cases, the connection between the formal entity, the program, and the ideas it is supposed to represent is much looser than in mathematics or in the natural sciences. This connection is represented in “requirement documents”, informal “UML diagrams”, and other similar devices, all leaving considerable room for interpretation.

Whether the program does in fact correspond to the intended domain is usually determined by agreement between the programmer and the customer, and resolving disagreements is usually a matter for the legal system rather than for leibnizean computation.

However, the situation is slowly improving, and in the current state it gets better with increasing distance from the “real world”. In the best case, we are able to create an intermediate mathematical layer, which acts as a buffer between the client domain and the program. In this layer, we formulate “specifications”: mathematical expressions of the relationships that must hold between the various elements of the program if it is to be an adequate formalization of the client domain. This intermediate layer must allow writing high-level, understandable specifications, whose connection to the client domain must be as clear as possible: thus, the intermediate layer must be close to the client domain. At the same time, the intermediate layer, having to deal directly with the elements of the program, must also be close to the formal system level, in order to allow the kind of systematic interpretation which is necessary for a formalization.

Thus, it is no surprise that this kind of layer can be found at the moment in those cases where the client domain has already been mathematized, so that most elements needed for our buffer are already present (this is the case, for example, of numerical methods), or when the client domain is itself part of the computational world, so that the distance between it and the formal system level is very short (for example, models of parallel computing [4], or specifications of operating system kernels [6]). We can also understand why “real-world” formalization is so difficult: the interaction between customers and programmers is essentially interdisciplinary, and creating a common language to bridge across the disciplines is a notoriously difficult task even when dealing with trained scientists.

As mentioned in the previous section, in order to deal with the complexity of the formal entities that the programs represent, a great deal of effort has been made in order to “raise the abstraction level”, by means of introducing definitions, higher-level programming languages, systematic ways of describing the intended meaning of the program elements, and so on. This activity, going through phases known in the profession (and usually also outside) under names such as “structured programming”, “object-oriented software design”, “agile developement”, not to mention a host of three-letter acronyms, has been pursued in a largely non-mathematical way, so that the connections between the new elements (“objects”, for instance) and the old ones (“subroutines”) are not always as clear as they could be. Nevertheless, much progress has been made, and more is currently underway, which is directly inspired by and takes advantage of the mathematical ideas that can be seen to address similar problems. We mention in this context the development of functional programming languages such as Haskell ([1]) whose elements are very close to their mathematical interpretations. For exam-

ple, Haskell functions can be interpreted as arrows in a suitable category, in which the objects are the interpretations of Haskell datatypes. All the mathematical machinery used to reason within that category is thus available for use when working with Haskell programs (for an introduction to this subject see [2]).

In conclusion, programming is the field where the pragmatic limit of formalization is most evident. The difficulties, similar to the ones encountered in mathematics, for example, are increased by the interdisciplinary nature of the customer-programmer interaction. On the flip side, precisely because of this, progress made in this field can be very interesting for us when it comes to formalization in the social sciences.

## 5 Formalization in the social sciences

We are going to examine formalization in the social sciences in a simple context: that of multi-agent models of socio-economic interactions.

From the very beginning, we have to remark that in this field we do not have a theory to guide us. For example, we cannot give convincing answers to the questions raised in the programme of this conference. Of course, in more limited contexts such as classic producer-consumer interactions in economics, we can rely on a number of mathematical results, but there is no comparable information about analysing, for instance, political interactions such as coalition building and lobbying.

Instead, what we see is an attempt to achieve mathematization via exploratory programming: we set up software multi-agent models and experiment with them, trying to build into them the features of the interactions we consider relevant and observing the results. This usage is very different from that of most models in the natural sciences. For example, simulations of flows are based on a mathematical theory of fluid motions. Their role is to accelerate or automate extracting the consequences, according to that theory, of the situation represented at the start of the simulation, perhaps in order to optimize the design of some engineering device or to choose between different control strategies of given processes. A similar role is often played by computer programs in mathematics, such as in the famous proof of the four color theorem. On the other hand, the role of the multi-agent programs in the social sciences is similar to that of laboratory experiments designed not to test the consequences of a theory, but to obtain the data (one could even say, the inspiration) needed to build a theory.

The problem with this approach is that we are attempting to link the complex ideas of the client domain directly to the formal representation given by the program. The difficulty can be illustrated by the following analogy: what inferences can we base on a simulation of a bridge on a computer screen which we have made with no knowledge of statical mechanics?

The problem is compounded by the difficulty of demonstrating program correctness. In examining the output of multi-agent models, we have to consider the possibility of error: what is correct behavior, and what is just a “bug”?

There is no easy solution to this problem, but awareness of the difficulties can help by at least making us cautious (forewarned is forearmed, as the saying goes). The cautious approach is, in this case, that of introducing the mathematical layer described in the previous section, in order to specify the correct behavior of the program.

In the introduction of this mathematical layer lie, we believe, potential benefits that go far beyond the avoidance of programming errors: among the concepts we need to come up with for the connection between the ideas of interactions between social entities and the formal program elements we hope to find the building block of a mathematical theory of these interactions.

For example, it is a commonplace among programmers that every component of an implementation should be specified and tested in isolation before embedding it into a complex system and attempting to use it there. This suggests a “bottom-up” (or “inside-out”) approach to multi-agent modeling: first, get the agents right, then study their interactions. This approach is currently the most popular, and it involves identifying the notion of an agent with that of a state machine or a dynamical system (for a textbook exposition see [19]). The problem is that when the agents involved are not homogenous, operate at very different scales or otherwise deviate from the “canonical” form, it is no longer clear how to model their interactions. Moreover, current implementations do not easily allow one to view a set of agents as an agent (for example, none of the platforms presented in a recent survey [18] has this facility). Surely a flaw if one wants to study the relationship between clubs and clubs of clubs.

Alternatively, we can choose to focus on the interactions between agents directly, leading to a more “top-down” (or “outside-in”) approach to multi-agent models, in a generalization of the stochastic systems presented in [13]. In this case, we consider a set of agents  $A$ , a set of messages  $M$ , and a totally ordered time set  $T$  (say,  $\mathbb{N}$  or  $\mathbb{R}$ ). The interactions between the agents can then be mathematized as a communication relation  $K \subseteq A \times A \times M \times T$ , the interpretation being that  $(a_1, a_2, m, t) \in K$  (also written, as usual,  $K(a_1, a_2, m, t)$ ) if agent  $a_1$  sends agent  $a_2$  the message  $m$  at time  $t$ . With the help of the communication relation we can define a host of derived notions. As an example, an agent  $a$  can be said to die at time  $t$  if  $\forall t' > t, a', m$  we have  $\neg K(a, a', m, t') \wedge \neg K(a', a, m, t')$ . A message  $m$  can be considered lethal if  $\forall a (\exists a' K(a', a, m, t) \Rightarrow a \text{ dies at } t)$ , and so on. Further, we can define morphisms between communication networks in the usual way: a morphism from  $K \subseteq A \times A \times M \times T$  to  $K' \subseteq A' \times A' \times M' \times T'$  is a triple of functions  $f : A \rightarrow A', g : M \rightarrow M', h : T \rightarrow T'$  such that  $h$  is monotonous and  $\forall a_1, a_2, m, t$  we have

$$K(a_1, a_2, m, t) \Rightarrow K'(f a_1, f a_2, g m, h t).$$

Consider a communication relation  $K' \subseteq A' \times A' \times M' \times T'$ . We can call the entities in  $A'$  clubs of agents in  $A$  if we can find a way of defining a communication network  $K'' \subseteq P A \times P A \times M \times T$  which is isomorphic to  $K'$ . That is, we can study the interactions of agents in  $A'$  by studying the interactions of sets of agents in  $A$ .

At present, neither view has been developed to a satisfactory degree. We have made progress in describing the interactions between heterogenous dynamical systems ([10]), and we are beginning to use communication networks to specify the overall behavior to be expected from interactions of dynamical systems. It might be that a theory capable of addressing the subtle interactions between social entities will have to be built out of entirely different elements, but we believe that at least some of those elements will come out of similar efforts of crossing the bridge from the ideas of social entities to the formal expressions of computer programs.

Not all efforts of mathematizing the ideas of social entities and their interactions go via experimental programming. We have mentioned above the work of Peyton-Young, other examples in microeconomics can be found in a recent textbook of Samuel Bowles [5]. Still, these attempts amount to a less than convincing theory.

The reason for the discomfort with which we view these attempts is that we find ourselves confronted with the same pragmatic limitation that we saw in the case of exploratory programming: the link between the mathematical formalization or model and the concepts it is supposed to represent is tenuous. The conditions under which one could use such a model in the way one uses a similar one in the natural sciences are not clear. In particular, there is no equivalent of the experimental protocols mentioned in the context of mathematization in the natural sciences. Therefore, we find it difficult to translate the elements from the social domain to the mathematical one and, conversely, to interpret the results of the mathematical investigation in social terms. We might perceive the similarities between the two, but without a better link we remain at the level of a metaphor, rather than of a formalization.

Since the difficulty is similar to the one we encountered previously, it is reasonable to hope that it might be solved in the same way: by introducing intermediate levels of abstractions, by finding the “right” mathematical concepts to bridge the gap between the more formal and the conceptual. In crossing from computer programs to “real-world” domains, the most promising approaches use tools from category theory, intuitionistic type-theory and relational calculus. It would be interesting to attempt to lift the level of abstraction of modern microeconomic models by expressing specifications for them using these tools, with a view to bringing them closer to the realm of the social.

## 6 Conclusions

Formalization defined in the logical sense is subject to strict limitations which are the subject of the so-called limitative theorems of Gödel, Church, or Skolem. Outside of symbolic logic, however, the main limit of formalization seems to be a pragmatic one: the distance between the formal and the conceptual is too big. This holds in mathematics, where the formal is usually the syntactic level of formal system, and the conceptual is the world of mathematical ideas; it also holds in the natural sciences, where the formal is represented by the mathematical ideas, and the conceptual is given by the specific science; and, most prominently, in building software, where the formal is the program to be written, and the conceptual is the “real-world” requirement of the client. This even holds in the case of the model of Richard II, where the formal is given by the world of his thoughts, and the conceptual is the social world of his contemporaries.

Not surprising, then, this pragmatic limitation seems also to be the most relevant when it comes to using mathematical formalization within social science. The distance between the concepts with which we operate in the social world, and the formal entities with which we usually attempt to connect them, be they elements of computer programs or of a mathematical framework, is simply too big for the connection to be compelling.

The problem of this kind of gap is addressed in similar ways in all cases: by introducing intermediate layers which are sufficiently close to both sides of the divide, so that the connections between these layers and either side *is* compelling. In mathematics we raise the level of abstraction of the syntactic level by means of definitions, using careful notation or streamlining the rules of inference. In physics, we might establish the connections by means of experimental protocols, and convince ourselves by carefully designed experiments of the predictive properties of the mathematical formalization. Richard introduces his own person into the discussion, in order to make the link between his thoughts and the world. As pointed out by Mac Lane in the case of mathematics, the intermediate layer functions here also as an *explanation* of the connection.

It is our belief that the situation in the social sciences is most similar to the one seen in computer programming, and that it can therefore be addressed using similar tools, the most successful of which are nowadays based on category theory (high-level programming languages like Haskell, for example), intuitionistic type theories (giving rise to dependently-typed systems such as Agda or Coq), and relational calculus (used to specify the requirements of programs and to model the relationships of client domain entities).

But whether we use these tools or others, the gap must be shortened. In the absence of a better connection between the formal and the social, our mathematical models are going to come second to Shakespeare’s even

in terms of scientific quality, not just aesthetical appeal.

## References

- [1] R. Bird. *Introduction to Functional Programming using Haskell*. International Series in Computer Science. Prentice Hall, second edition edition, 1998.
- [2] R. S. Bird and O. de Moor. *Algebra of Programming*. International Series in Computer Science. Prentice Hall, 1997.
- [3] G. Boolos, J. Burgess, and R. Jeffrey. *Computability and Logic*. Cambridge University Press, 5th edition edition, 2007.
- [4] N. Botta and C. Ionescu. Relation-based computations in a monadic bsp model. *Parallel Computing*, 33(12):795–821, 2007.
- [5] S. Bowles. *Microeconomics: Behavior, Institutions, and Evolution*. The Roundtable Series in Behavioral Economics. Princeton University Press, 2006.
- [6] I. D. Craig. *Formal Models of Operating System Kernels*. Springer, 2006.
- [7] H. DeLong. *A Profile of Mathematical Logic*. Dover Publications, 1998.
- [8] R. Goldblatt. *Topoi, The Categorical Analysis of Logic*. Dover Publications, Inc., 2006.
- [9] J. R. Hindley and J. P. Seldin. *Lambda-Calculus and Combinators, an Introduction*. Cambridge University Press, 2008.
- [10] C. Ionescu. Modeling vulnerability and monadic dynamical systems. Phd. thesis, Freie Universitaet, Berlin, forthcoming.
- [11] S. Mac Lane. *Mathematics: Form and Function*. Springer, 1986.
- [12] S. Mac Lane. *A Mathematical Autobiography*. AK Peters, Ltd., 2005.
- [13] H. Peyton Young. The evolution of conventions. *Econometrica*, 61(1):57–84, 1993.
- [14] W. Shakespeare. *King Richard II*. Third Series. Arden Shakespeare, 2002.
- [15] A. Tarski. Contributions to the theory of models i. *Indagationes Mathematicae*, 16:572–581, 1954.

- [16] R. D. Tennet. *Semantics of Programming Languages*. Prentice Hall, 1991.
- [17] R. D. Tennet. *Semantics of Programming Languages*. Prentice Hall, 1991.
- [18] R. Unland, M. Klusch, and M. Calisti, editors. *Software Agent-Based Applications, Platforms and Development Kits*. Birkhauser, 2005.
- [19] M. Wooldridge. *An Introduction to Multi-Agent Systems*. John Wiley & Sons, 2002.