Logic programming of some mathematical paradoxes

M.A. Nait Abdallah
Department of Computer Science
University of Western Ontario
London, Ontario, Canada

Abstract

Using the notions of logic field and ion defined in [7, 9], we give an algorithmic analysis, in terms of logic programming, of three paradoxes: Protagoras, Newcomb and the Hangman. We show that each one of these paradoxes points out a *programming* mistake to be avoided in logic programming.

1 Introduction

The question we discuss in this paper is: how does logic programming relate to fundamental problems of expression and reasoning in mathematics, i.e. how it relates to *mathematical paradoxes.*

A careful analysis of Russell paradox from the point of view of combinatory logic, leads to the fixpoint combinator $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$. The logical paradox then resolves into an infinite computation. The Y combinator turns out to be interpreted as the least fixpoint operator in Scott-Strachey denotational semantics [3, 15], thus giving a semantics for recursive procedure calls in programming.

Not all paradoxes are so easily analyzed in terms of *combinatory* logic, however, as constructs other than *abstraction* and *application* may intervene. Some of these other paradoxes are especially important from the point of view of a *formal logic of knowledge* [2]. The following question then arises: can we learn as much about *knowledge-oriented programming languages*, from a computational study ot these paradoxes, as we have learned about *imperative and functional programming languages* from a computational study of Russell paradox? In order to examine this question, we shall consider in this paper three such paradoxes, namely *Protagoras paradox*, *Newcomb's paradox* and the *Hangman paradox*. We shall not attempt to give comprehensive references about these paradoxes, as they have been discussed in innumerable books and articles.

Our method of investigation for solving the above problems is based on programming language semantics. Such a *prescriptive* use of semantics has been advocated in [1].

The general approach we adopt is that of *predicate logic*, as most of mathematics is based on predicate logic. More specifically, we shall use a *logical extension* of Horn-clause programming [8, 9]. This extension is based on the two concepts of *ions* and *logic fields*, and may be seen as an amalgamation of algebraic semantics [5] and Horn-clause programming [4]. It uses equation-solving in the set of logic programs.

Intuitively the notion of *(generic) procedure* corresponds to what is available in ordinary programming languages: the possibility to handle a λ-abstracted form of a piece of code. Here pieces of code are sets of Horn clauses, and the variables to be abstracted may be any one of the following: object variables, function variables, predicate variables, or procedure variables.

The equations on logic programs to be solved (called here, following [5], *schemes* or *rewriting systems*) correpond to the evaluation of a given program in the context of a given set of (recursive) procedure definitions, or to the evaluation of those procedure values that allow to solve a given goal (checking theories against facts, see below). Pairs of the form (logic program, system of equations) are called *logic fields* [9].

The paper is organized as follows.

We first give an outline of our theoretical framework. We define ions, procedures, rewriting systems and logic fields. We also define derivations on ions, and give an example.

We then describe an algorithmic approach to Protagoras, Newcomb's, and the Hangman paradoxes. Each one of these paradoxes is given a full formalization in the framework of logic fields, and its mechanisms are analyzed in computational terms. We argue that each one of these paradoxes stresses the importance of a feature of logic fields: Protagoras indicates the importance of *ions*, Newcomb's illustrates procedure parameter transmission, and the Hangman illustrates the *non-determinacy of rewriting systems.*

The yield of our study of mathematical paradoxes is as follows. Protagoras unfolds into a set of computations that can be represented by a *finite automaton.* Newcomb's paradox turns out to arise from an ill-defined notion of prediction, and one of its components, the *dominance argument,* is based on the incorrect assumption that some derivation is successful. The Hangman comes out both from an incorrect management of procedure variable assignments, and a set of unjustified but hidden assumptions.

In other words, each one of these paradoxes points out some programming mistake which should be avoided in logic programming.

2 Logic fields and ions

Logic fields were first introduced in [9]. A more detailed exposition can be found in [8, 9, 12], where [8] describes how to divide a classical logic program into the main program P (see below) and the rewriting system Σ (see below), and [9, 12] describe the formalism. For the sake of brevity, we shall only recall here some definitions.

2.1 Logic fields and ions

Definitions. Let Ξ be a set of *procedure variables*, R be a set of *constant relation* symbols, Ψ be a set of *relation variables*, F be a set of *constant function* symbols, Φ be a set of *function variables*, and V be a set of *variables of individuals.*

A *logic field* is a pair (Σ, P) where P is a program with no occurrence of function or relation variables, and Σ is a rewriting system. A *program* is a set of definite clauses and procedure calls. A *definite clause* is an implicitly universally quantified well-formed formula of the form:

$$A \leftarrow B_1 \wedge \ldots \wedge B_m \qquad m \geq 0$$

where each one of A, B_1, ..., B_m is an atom or an ion. *Atoms* (or *atomic formulae*) are defined in the usual way. A *procedure call* is an expression of the form $\xi(\vec{\alpha};\vec{\beta})$, where ξ is a procedure variable, $\vec{\alpha}$ is a list of actual input parameters, and $\vec{\beta}$ is a list of actual output (name) parameters. We designate by $\Pi(\Xi, R \cup \Psi, F \cup \Phi, V)$ the set of all logic programs constructed in this way.

A *rewriting system* over $\Pi(\Xi, R \cup \Psi, F \cup \Phi, V)$, where Ξ is the set of *procedure* variables, R (resp. Ψ) is the set of constant (resp. variable) relation symbols, F (resp. Φ) is the set of constant (resp. variable) function symbols, and V is the set of variables of individuals, is a system Σ of n equations:

$$\Sigma: \quad \xi_i = \forall\vec{\varphi_i}\exists\vec{\chi_i}.T_i \qquad i = 1, \ldots, n$$

where $\xi_i \in \Xi$ is a procedure variable, $\forall\vec{\varphi_i}\exists\vec{\chi_i}.T_i$ is a set of procedures $\forall\vec{\varphi_i}\exists\vec{\chi_i}.Q_i$, where $T_i \subseteq \Pi(\Xi, R \cup \Psi_i, F \cup \Phi_i, V)$ is a subset whose elements are logic programs such that $\Phi_i = \Phi \cap (\vec{\varphi_i} \cup \vec{\chi_i})$, $\Psi_i = \Psi \cap (\vec{\varphi_i} \cup \vec{\chi_i})$. A *procedure* is a closed expression of the form $\forall\vec{\varphi}\exists\vec{\chi}.Q$, where Q is a logic program.

An *ion* is a pair $((\Sigma, P), g)$ where (Σ, P) is a logic field and g is a conjunction of atoms or ions; the g part can be seen intuitively as the "goal component" of the ion.

If a logic program consists of a single procedure call $\{\xi(\vec{\alpha};\vec{\beta}) \leftarrow\}$, then it will be abbreviated as $\xi(\vec{\alpha};\vec{\beta})$. Also, if we are in the context of one single rewriting system Σ, ions $((\Sigma, P), g)$ may be abbreviated as (P, g), the rewriting system thus remaining implicit.

2.2 Derivations on ions

We define derivations steps on ions as follows.

An *SLD step* is any instance of the following ion rewriting rule:

$$\frac{g \rightarrow^P g'}{((\Sigma, P), g) \rightarrow ((\Sigma, P), g')}$$

where $g \rightarrow^P g'$ means that g yields g' in one SLD-step in the context of logic program P, more precisely, in the clausal context of the occurrence of g in P [7, 9].

A *rewriting step* (or *procedure call step*) is any instance of the following ion rewriting rule:

$$\frac{P \rightarrow^\Sigma P'}{((\Sigma, P), g) \rightarrow ((\Sigma, P'), g)}$$

where $P \rightarrow^\Sigma P'$ means that logic program P yields logic program P' in one rewriting step in the context of rewriting system Σ. For the needs of this paper, rewriting steps shall be submitted to the following *unique procedure call rewriting rule*: *Whenever in a given derivation, a given procedure variable is rewritten into some specific procedure, in all later steps of this derivation, this same procedure variables can only be rewritten into the* same *procedure value.*

A derivation *succeeds* if it yields an *empty ion*, i.e. one whose "goal component" is empty.

Since ions are pairs $((\Sigma, P), g)$, where g is itself a conjunction of atoms or ions, the question arises about what SLD-steps are permitted in the case of *nested ions*, for example $((\Sigma, P), g_1 \wedge ((\Sigma_2, P_2), g_2))$. In this case, for the needs of this paper, we shall take as a convention that only definite clauses belonging to P_2 will be applicable to g_2 for generating SLD-steps. More details on this question are given in [12].

2.3 Example

An example of a derivation with procedure calls is as follows. Let (Σ, P) be the following deterministic logic field.

$$\begin{aligned}
\Sigma = \{\ \xi_1 = \exists\chi.\{\ & \chi(0,x) && \leftarrow \\
& \chi(s(x),s(y)) && \leftarrow\ \chi(x,y)\} \\
\xi_2 = \forall\varphi_1\exists\chi_1.\{\ & \chi_1([x]) && \leftarrow \\
& \chi_1([xy \mid z]) && \leftarrow\ \varphi_1(x,y) \wedge \chi_1([y \mid z])\}\}
\end{aligned}$$

with logic program:

$$\begin{aligned}
P = \{\ & 1.\ \ \xi_1(;le) \leftarrow \\
& 2.\ \ \xi_2(le;ordered) \leftarrow \\
\}&
\end{aligned}$$

In rewriting system Σ, procedure ξ_1 *exports* the binary predicate χ, procedure ξ_2 *imports* binary predicate ϕ_1 end *exports* unary predicate χ_1. Logic program P acts like a pipe: it feeds the predicate exported by ξ_1 (and called here *le*) to ξ_2, which will return the unary predicate *ordered*.

Then we obtain the following derivation:

1.	$((\Sigma, P), ordered([0\, s(0)])$	
2.	$((\Sigma, P_1), ordered([0\, s(0)])$	rewriting step for ξ_2
3.	$((\Sigma, P_1), le(0, s(0)) \wedge ordered([s(0)])$	$2, P_1(2.2)$
4.	$((\Sigma, P_1), le(0, s(0)))$	$3, P_1(2.1)$
5.	$((\Sigma, P_2), le(0, s(0)))$	rewriting step for ξ_1
6.	$((\Sigma, P_2),\ )$	$5, P_2(1.1)$

where P_1 is obtained from P, and P_2 from P_1, by means of a rewriting step. More precisely P_1 and P_2 have values:

$$P_1 = \{\ 1.\quad \xi_1(; le) \leftarrow$$
$$2.1\quad ordered([x]) \leftarrow$$
$$2.2\quad ordered([x\ y|z]) \leftarrow le(x, y) \wedge ordered([y|z])$$
$$\}$$

$$P_2 = \{\ 1.1\quad le(0, x) \leftarrow$$
$$1.2\quad le(s(x), s(y)) \leftarrow le(x, y)$$
$$2.1\quad ordered([x]) \leftarrow$$
$$2.2\quad ordered([x\ \ y|z]) \leftarrow le(x, y) \wedge ordered([y|z])$$
$$\}$$

The derivation yields the empty ion $((\Sigma, P_2),\ )$ and thus is a successful derivation.

3 Protagoras paradox

Protagoras was a law teacher who had a bright, but pennyless student. In order to allow this student to complete his studies, he signs a contract with him, stating that the student would have to pay him only once he (the student) would have won his first lawsuit in court. The student successfully completes his studies to become a lawyer. But he then refuses to take any client, and also refuses to pay his teacher, arguing on the basis of their contract that he has not won any lawsuit yet. The teacher finally takes his student to court, reasoning that he would certainly get his money back, whether he wins (decision of the court is in his favour), or loses (the contract then applies). The student is not worried in the least, arguing that he does not have to pay in any case, whether he loses (the contract then says he does not have to pay yet), or wins (the court decided he did not have to pay).

In our formalization, we shall assume the following about this paradox: (i) it is a fully stated and fully logical problem (e.g. no police is available). (ii) the contract itself is never challenged in court, (iii) the student is *logically honest* in the sense that he will do his best to avoid paying, but he shall pay when logically forced to do so, (iii) the teacher will do everything he can to get his money, and is ready to sue as many times as necessary.

3.1 The logic field

The statement of the paradox will be expressed in terms of a logic field (Σ, P) defined as follows. The rewriting system Σ defines the following set of procedures :

Court describes the enforcing of the court's decision, **Contract** describes the clauses of the contract, **Student** describes the student's point of view, **Teacher** describes the teacher's point of view. Finally **History** records the events so far i.e., the history of the lawsuit; as the solution of the case is likely to take some time, this procedure will be defined by a non-deterministic equation, thus permitting updates.

The main program P defines the "rules of the game".

More precisely, Σ is defined as being the following set of equations:

$$
\begin{array}{lcl}
Court & = & \{\ doesnotpay(x,t) \leftarrow wins(x,t) \\
 & & \quad pays(x,t) \leftarrow loses(x,t) \\
 & & \} \\
Contract & = & \{\ pays(x,t) \leftarrow haswonatrial(x,t) \\
 & & \quad doesnotpay(x,t) \leftarrow \neg haswonatrial(x,t) \\
 & & \} \\
Student & = & \{\ doesnotpay(student,t) \leftarrow \\
 & & \qquad (Contract \cup \{loses(student,t) \leftarrow\}, doesnotpay(student,t)) \\
 & & \qquad \wedge\, loses(student,t) \\
 & & \quad doesnotpay(student,t) \leftarrow \\
 & & \qquad (Court \cup \{wins(student,t) \leftarrow\}, doesnotpay(student,t)) \wedge \\
 & & \qquad \wedge\, wins(student,t) \\
 & & \quad pays(x,t) \leftarrow \neg doesnotpay(x,t) \\
 & & \} \\
Teacher & = & \{\ pays(student,t) \leftarrow \\
 & & \qquad (Court \cup \{loses(student,t) \leftarrow\}, pays(student,t)) \wedge \\
 & & \qquad \wedge\, loses(student,t) \\
 & & \quad pays(student,t) \leftarrow \\
 & & \qquad (Contract \cup \{wins(student,t) \leftarrow\}, pays(student,t)) \wedge \\
 & & \qquad \wedge\, wins(student,t) \\
 & & \} \\
History & = & \{\ \{\ goestocourt(student,k) \leftarrow \\
 & & \qquad \varphi_k(student,k) \leftarrow\ : \\
 & & \qquad k = 1,\ldots,n \text{ and } \varphi_k \in \{loses, wins\} \\
 & & \quad \}\ :\ \ n = 1,2,3,\ldots \\
 & & \}
\end{array}
$$

The main program P is given by:

$$
\begin{array}{l}
\{\ 1.\ \ pays(student,t) \leftarrow (Student, pays(student,t)) \\
\quad 2.\ \ haswonatrial(x,t) \leftarrow goestocourt(x,t') \wedge wins(x,t') \wedge (t' \leq t) \\
\quad 3.\ \ History \leftarrow \\
\}
\end{array}
$$

The *court decision making process* itself is not formalized by a procedure, since no information is given in the statement of the paradox. We assume that it is described by an oracle giving, each time the student and the teacher go to court, a decision that is either in favour or against the student.

3.2 The computation

The sequences of facts to be solved are :

$$F_i = goestocourt(student, i) \wedge \varphi_i(student, i) \wedge \psi_i(student, i)$$

where $\varphi_i \in \{wins, loses\}$ and $\psi_i \in \{pays, doesnotpay\}$. (As we said earlier, the exact value of the actual sequence depends on the successive decisions of the court.) In other words the ion to be solved is $((\Sigma, P), \bigwedge_{i \in N} F_i)$, where $N = \{1, 2, \ldots, n\}$ is some initial subset of the natural numbers, corresponding to the number of times n the teacher, trying to get his money, takes the student to court. The resolution of this ion leads to a regular language over the alphabet $\{w, f\}$ denoted by the regular expression f^*ww^*f, which gives the set of "paying sequences", where letter w means that the *student wins*, and f that the *student loses*. Words of this language correspond to sequences of court decisions.

The behaviour of the system is given by the *prefix language* of f^*ww^*f, i.e. by the language $ww^*+ff^*w^*+f^*ww^*f$. In other words, each string from this language gives a possible behaviour of the system. Only strings belonging to f^*ww^*f, however, will be considered as satisfying by the teacher, since they represent the only computations ending with his getting paid.

The transition table of the 4-state finite automaton corresponding to these languages is as follows:

	start	*use contract*	*use court decision*
student wins	use court decision	use court decision	use court decision
student loses	use contract	use contract	pay teacher

4 Newcomb's paradox

The paradox involves a Being who has the ability to predict the choices you will make.

There are two boxes. Box 1 contains 1,000 gold coins. Box 2 contains either 1 million gold coins or no money. You have the choice between two actions: taking what is in both boxes or taking only what is in the second box. If the Being predicts you will take what is in both boxes, (or you will base your choice on some random event), he does not put the 1 million coins in the second box. If he predicts you will take only what is in the second box, he puts 1 million gold coins in the second box. You know these facts, he knows that you know them and so on. The Being makes his prediction of your choice, does (or does not) put the 1 million coins in the second box, and then you choose. It is also assumed that you are a greedy person, and that you want to get as much gold as possible. What do you do?

There are two plausible arguments for reaching two different decisions. The *expected utility argument* is as follows. Whatever I am doing, the Being has already predicted, and thus it is better to take only the second box and thus get 1 million gold coins, instead of only 1,000 coins if I took both. On the other hand the *dominance argument* says that the Being has already made his prediction, and has acted accordingly. He has, or has not put the 1 million gold coins in the second box, and whatever I choose to do cannot change this. Therefore it is better to take both boxes, and do better by 1,000 coins, whether the second box contains 1 million coins or not.

We now provide an analysis, in terms of logic fields, of this paradox. We shall see that the paradox is ill-stated, as no precise definition of *choice* and *prediction* is given. More technically, Newcomb's paradox illustrates non-determinacy and parameter-passing in procedure calls in logic fields computations.

In our formalization, we shall assume the following about this paradox: *It is a fully logical problem i.e., the Being makes no mistake*, and thus, no probabilities are involved.

4.1 The logic field

We shall use a logic field (Σ, P) in order to express the above paradox. The rewriting system Σ defines the following two procedures : β (or *History*) describes the prediction made by the Being, and ξ describes the box contents.

The main program P defines the "rules of he game", i.e. here the description of the possible choices to be made.

The ion to be solved in order to express the solution of the paradox in terms of logic fields is : $((\Sigma, P), choose(i))$, i.e., which box is to be taken ? The resolution of this ion leads to the following results.

4.2 Newcomb's paradox under the randomness assumption

To clarify ideas about Newcomb's paradox we first define some other logic field (Σ, P) which does not formalize Newcomb as stated above, but a slight variant of it. More precisely, in the following we shall make the following *randomness assumption* (formalized by the definition of β given below): *the Being has made some* guess *and acted accordingly.* That guess has nothing to do with my own acts, and whatever I choose to do cannot change that guess (*Dominance argument*). The Being's guess may be seen as a *free* prediction.

In that case, rewriting system Σ is defined as being the following set of equations:

$$\begin{array}{rcl} \beta & = & \{ \quad \{ \quad \sigma(both) \leftarrow \quad \}, \\ & & \quad \{ \quad \sigma(second) \leftarrow \quad \} \\ & & \} \end{array}$$

$$\begin{array}{rcl} \xi & = & \forall\gamma \exists B_1 B_2 . \\ & & \{ \quad B_1(1,000) \leftarrow \\ & & \quad B_2(0) \leftarrow (\gamma, \sigma(both)) \\ & & \quad B_2(1,000,000) \leftarrow (\gamma, \sigma(second)) \\ & & \} \end{array}$$

The main program P is given by:

1. $c(both, x + y) \leftarrow B_1(x) \wedge B_2(y)$
2. $c(second, y) \leftarrow B_2(y)$
3. $choose(i) \leftarrow c(i, z) \wedge c(j, w) \wedge i \neq j \wedge (w \leq z)$
4. $\xi(\beta\ ;\ B_1,\ B_2) \leftarrow$

4.2.1 The computation

The ion to be solved is : $((\Sigma, P), choose(i))$ i.e., which box is to be taken ? The resolution of this ion leads to the following results. We have two successful derivations. The first one is obtained in the case the Being has predicted that I would take both boxes. The derivation yields the empty ion $((\Sigma, P_2),\ )$, and thus is a successful derivation is reached by taking *both* boxes and obtaining a *gain* of 1,000 coins.

We have a second successful derivation in the case the Being has predicted I would take only the second box:

$((\Sigma, P), choose(i)) \rightarrow^* \square$, where $\beta = \{\sigma(second) \leftarrow\}, i = both$, and $gain = 1\ million + 1,000$

where $\square$ is an abbreviation of the fact that the derivation succeeds.

We also have two finitely failed derivations, namely:

- $((\Sigma, P), choose(i)) \rightarrow^* ff$, where $\beta = \{\sigma(both) \leftarrow\}$, $i = second$, and $gain = 0$.
- $((\Sigma, P), choose(i)) \rightarrow^* ff$, where $\beta = \{\sigma(second) \leftarrow\}$, $i = second$, and $gain = 1million$.

where ff is an abbreviation of the fact that the derivation finitely fails.

Thus we may conclude from the previous derivations that if the Being makes a *random guess* (procedure β) before putting any money in the boxes (procedure ξ), and if my goal is to get as much money as possible, then I am better off by 1,000 coins if I pick *both* boxes. Notice that the only difference between this argument and the *dominance argument* discussed earlier in the exposition of the paradox, is that here, the Being's prediction is a random guess.

In the next step of the discussion, we shall examine what happens when we establish a link between β, i.e. the *prediction*, and $choose(i)$, i.e. the *choice*. This will give a formalization of the dominance argument itself.

4.3 Newcomb's paradox under the accuracy assumption

We now modify the previous logic field as follows. We assume that the prediction made by the Being is dependent on the action I choose to perform. This may be called the *accuracy assumption*: *The Being will predict some act if I do perform that act later on.* This yields the following new definition of β $(History)$.

$$\begin{array}{rcl} \beta & = & \{ \quad \{ \quad \sigma(both) \leftarrow ((\Sigma, P), choose(both)) \quad \}, \\ & & \quad \{ \quad \sigma(second) \leftarrow ((\Sigma, P), choose(second)) \quad \} \\ & & \} \end{array}$$

The remaining of the logic field is unchanged.

4.3.1 The computation

The resolution of the ion $((\Sigma, P), choose(i))$ leads to the following results. We have two infinite, and thus unsuccessful, derivations:

$$((\Sigma, P), choose(i)) \rightarrow^* ((\Sigma, P), (\beta, ((\Sigma, P), choose(both)))) \rightarrow^* \ldots$$

where $\beta = \{\sigma(both) \leftarrow ((\Sigma, P), choose(both))\}$, $i = both$, and $gain = 1,000$, and

$$((\Sigma, P), choose(i)) \rightarrow^* ((\Sigma, P), (\beta, ((\Sigma, P), choose(second)))) \rightarrow^* \ldots$$

where $\beta = \{\sigma(second) \leftarrow ((\Sigma, P), choose(second))\}$, $i = both$, and $gain = 1million + 1,000$.

In both cases, we have a vicious circle: I shall choose, say, both boxes, if the Being predicts that '*I shall choose both boxes*', if the Being predicts that "*the Being predicts that* 'I shall choose both boxes'", etc ...

We also have two finitely failed derivations:

- $((\Sigma, P), choose(i)) \rightarrow^* ((\Sigma, P), (\beta, ((\Sigma, P), choose(both))) \wedge (0 \leq 1,000)) \rightarrow^* ff$ where $\beta = \{\sigma(both) \leftarrow ((\Sigma, P), choose(both))\}$, $i = second$, and $gain = 0$.
- $((\Sigma, P), choose(i)) \rightarrow^* ((\Sigma, P), (\beta, ((\Sigma, P), choose(second))) \wedge (1million + 1,000 \leq 1million)) \rightarrow^* ff$ where $\beta = \{\sigma(second) \leftarrow ((\Sigma, P), choose(second))\}$, $i = second$, and $gain = 1million$.

where ff means that the derivation finitely fails.

The only way to obtain successful derivations under the accuracy assumption, is to somehow try to modify the logic field in order to make the infinite derivations successful, i.e. stop the infinite recursion.

Thus, we see that Newcomb's paradox under the *accuracy assumption*, is a combination of non-determinism plus recursiveness.

The *dominance argument* assumes that the recursion present in the above two infinite derivations is already *terminated* (i.e. solved), and thus assumes that the correct formalization of the paradox is the logic field we gave earlier by using the *randomness assumption*. Thus it appears that the meaning of the notion of "*prediction*" in the framework of the dominance argument is unclear; it is obviously not expressed by our accuracy assumption. In other words the dominance argument assumes that the Being has predicted our action, but otherwise discards any link between that prediction and our action, thus making that prediction *computationally* equivalent to a random choice. *This makes the dominance argument logically unsound.*

4.4 Newcomb's paradox as an instance of Schrödinger's Cat

The *expected utility argument* uses another way to solve the aforementioned recursion. It says "Whatever I am doing, the Being has predicted." It links the explanation substitution with the possible success of the goal $choose(i)$, by feeding the value of i back to the "prediction" β of the Being. The notion of *physical determinism* vanishes, and the Being's prediction β becomes a *function* of the box choice i I am making. In other words, my choice determines the Being's prediction and the box contents. The content of the boxes remains completely undefined until I open them. This is reminiscent of Schrödinger's Cat. In terms of logic fields, this can be done as follows.

$$
\begin{array}{rcl}
\beta & = & \forall i. \\
 & & \quad \{ \quad \{ \quad \sigma(both) \leftarrow (i = both) \quad \}, \\
 & & \qquad \{ \quad \sigma(second) \leftarrow (i = second) \quad \} \\
 & & \quad \}
\end{array}
$$

$$
\begin{array}{rcl}
\xi & = & \forall i \forall \gamma . \exists B_1 B_2 . \\
 & & \quad \{ \quad B_1(1,000) \leftarrow \\
 & & \qquad B_2(0) \leftarrow (\gamma(i), \sigma(both)) \\
 & & \qquad B_2(1,000,000) \leftarrow (\gamma(i), \sigma(second)) \\
 & & \quad \}
\end{array}
$$

The main program P is given by:

1. $c(both, x+y) \leftarrow (\xi(both, \beta; B_1, B_2), B_1(x) \wedge B_2(y))$
2. $c(second, y) \leftarrow (\xi(second, \beta; B_1, B_2), B_2(y))$
3. $choose(i) \leftarrow c(i, z) \wedge c(j, w) \wedge i \neq j \wedge (w \leq z)$
4. $\xi(\beta; B_1, B_2) \leftarrow$

4.4.1 The computation

The resolution of ion $((\Sigma, P), choose(i))$ leads to an optimal *gain* of 1 million gold coins, and it will be obtained if I pick the second box only.

5 The Hangman paradox

"A judge decrees on Sunday that a prisoner shall be hanged on noon on the following Monday, Tuesday or Wednesday, that he shall not be hanged more than once, and that he shall not

know until the morning of the hanging the day on which it will occur. By familiar arguments it appears that the decree cannot be fulfilled and that it can."[6].

The analysis in terms of logic fields of this paradox turns out to be an *effective* (i.e. *computable*) formalization of those of Quine [14] and Montague [6].

5.1 A formalization in terms of logic fields

We shall assume that $h(x)$ stands for " The prisoner is hanged at time x ". The days of the week Sunday, Monday, Tuesday and Wednesday shall be abbreviated by their initials. The sequence of times is given by the sequence of days $\{\mathbf{S}, \mathbf{M}, \mathbf{T}, \mathbf{W}\}$. If $t \in \{\mathbf{S}, \mathbf{M}, \mathbf{T}, \mathbf{W}\}$ we denote by $[\mathbf{S}, t]$ the smallest initial subsequence containing t. A *history* is defined as a sequence of events. At any given time (day), only two mutually exclusive events are possible: "the prisoner is hanged" or "the prisoner is not hanged". No event occurs on Sunday.

5.1.1 The rewriting system Σ

The rewriting system Σ of logic field (Σ, P) will contain three procedure definitions: the prisoner K, the decree δ, and the possible sequence of events $History$.

The *prisoner* will suppose he is at some time t, and assume that the sequence of events is following some course γ where t occurs. He knows that he shall be hanged at some later time x if and only if, whenever he assumes he assumes he is at time t of history γ, he is able to prove that he will actually be hanged at time x. Obviously time t occurs earlier in history γ than time x, and the prisoner will only be able to use in his proof that part of γ that took place until time t : that part of γ we will denote by $\gamma\,|_{[\mathbf{S},t]}$.

$$
\begin{array}{rcl}
K & = & \forall t \forall \gamma. \\
 & & \{ \quad 1. \quad \textit{is-history}(\gamma) \leftarrow \textit{occurs-in}(t, \gamma) \\
 & & \quad\; 2. \quad h(x) \leftarrow \textit{is-history}(\gamma) \wedge (\gamma\,|_{[\mathbf{S},t]}, h(x)) \\
 & & \}
\end{array}
$$

As far as our syntax is concerned, the *ion* $((K(y, \gamma), q)$ stands for the statement " The prisoner knows, in the case the course of events is equal to γ, at time $y \in \gamma$, that q holds". For example, $((K(\mathbf{M}, Q), h(\mathbf{T}))$ means that the prisoner, on the basis of that portion of events from Q that took place so far, knows on Monday that he will be hanged on Tuesday. Notice that we omit, in our ion notation, rewriting system Σ, as no other rewriting system occurs in our problem.

The *punishment* defined in the decree consists of two components: first, the prisoner shall be hanged at time $t \in \{\mathbf{S}, \mathbf{M}, \mathbf{T}, \mathbf{W}\}$; second at any given time he shall not know that he will be hanged on the following day. This is expressed by the following procedure, where γ is the "history" parameter, and p is the punishment predicate defined in the decree.

$$
\begin{array}{rcl}
\delta & = & \forall \gamma \exists p. \\
 & & \{ \quad 1. \quad p(\mathbf{M}) \leftarrow h(\mathbf{M}) \wedge \neg(K(\mathbf{S}, \gamma), h(\mathbf{M})) \\
 & & \quad\; 2. \quad p(\mathbf{T}) \leftarrow h(\mathbf{T}) \wedge \neg(K(\mathbf{M}, \gamma), h(\mathbf{T})) \\
 & & \quad\; 3. \quad p(\mathbf{W}) \leftarrow h(\mathbf{W}) \wedge \neg(K(\mathbf{T}, \gamma), h(\mathbf{W})) \\
 & & \}
\end{array}
$$

The possible *sequences of events* are defined by the following non-deterministic equation:

$$History = \{Q_0, Q_1, Q_2, Q_3, Q_4, \}$$

where the programs Q_i are defined by:

$$\begin{aligned} Q_0 &= \emptyset \\ Q_1 &= \{h(\mathbf{M})\} \\ Q_2 &= \{\neg h(\mathbf{M}), h(\mathbf{T})\} \\ Q_3 &= \{\neg h(\mathbf{M}), \neg h(\mathbf{T}), h(\mathbf{W})\} \\ Q_4 &= \{\neg h(\mathbf{M}), \neg h(\mathbf{T}), \neg h(\mathbf{W})\} \end{aligned}$$

More intuitive names for the Q_i's would be respectively: *Is-hanged on Monday, Is-hanged on Tuesday, Is-hanged on Wednesday* and *Is-not-hanged at all.* But since we want to submit the paradox to a formal treatment, we shall stick to the Q notation.

5.1.2 The main program

The main program is defined by:

$$\begin{aligned} P = \{ \; & 1. \quad History \leftarrow \\ & 2. \quad \delta(History, p) \leftarrow \\ \} & \end{aligned}$$

The formal definition of predicate $occurs\text{-}in(t, \gamma)$ used in procedure K, and which says whether a given day t occurs in a given history γ, should be included in the main program P. Such a formalization is straightforward, however, and is omitted for reasons of clarity.

5.2 The computation

The intuitive argument leading to the paradox is some kind of backward "induction": one first shows that the prisoner cannot be hanged on Wednesday, then by the same argument one eliminates Tuesday and Monday. Therefore he cannot be hanged. But the hangman comes, say, on Tuesday without being expected, and the decree is fulfilled.

Using our logic programming model, we show that this intuitive argument contains in fact quite a few hidden assumptions, and is indeed flawed. Indeed, we find three successful ions:

$$\begin{aligned} ((\Sigma, P_2^{1,Q_3}), \neg p(\mathbf{W})) &= ((\Sigma, P_2^{Q_3} \cup H_1), \neg p(\mathbf{W})) \\ ((\Sigma, P_2^{2,Q_2}), \neg p(\mathbf{T})) &= ((\Sigma, P_2^{Q_2} \cup H_2), \neg p(\mathbf{T})) \\ ((\Sigma, P_2^{3,Q_1}), \neg p(\mathbf{M})) &= ((\Sigma, P_2^{Q_1} \cup H_3), \neg p(\mathbf{M})) \end{aligned}$$

where we define $P_i^{j,Q_k} = P_i^{Q_k} \cup H_j$, where $P_i^{Q_k}$ is the value obtained from P after the i-th rewriting step, with Q_k as the assumed sequence of events, and the set of additional hypotheses H_j is defined in each case. More precisely,

$$\begin{aligned} P_2^{1,Q_3} = \{ \; & 1.1 \quad \neg h(\mathbf{M}) \leftarrow \\ & 1.2 \quad \neg h(\mathbf{T}) \leftarrow \\ & 1.3 \quad h(\mathbf{W}) \leftarrow \\ & 2.1 \quad p(\mathbf{M}) \leftarrow h(\mathbf{M}) \wedge \neg(K(\mathbf{S}, Q_3), h(\mathbf{M})) \\ & 2.2 \quad p(\mathbf{T}) \leftarrow h(\mathbf{T}) \wedge \neg(K(\mathbf{M}, Q_3), h(\mathbf{T})) \\ & 2.3 \quad p(\mathbf{W}) \leftarrow h(\mathbf{W}) \wedge \neg(K(\mathbf{T}, Q_3), h(\mathbf{W})) \\ & 3. \quad ((K(\mathbf{T}, Q_3), h(\mathbf{W})) \leftarrow ((\Sigma, P_2^{Q_3}), \neg h(\mathbf{M})) \wedge ((\Sigma, P_2^{Q_3}), \neg h(\mathbf{T})) \\ \} & \end{aligned}$$

$$P_2^{2,Q_2} =$$

$$\{ \quad 1.1 \quad \neg h(\mathbf{M}) \leftarrow$$
$$1.2 \quad h(\mathbf{T}) \leftarrow$$
$$2.1 \quad p(\mathbf{M}) \leftarrow h(\mathbf{M}) \wedge \neg(K(\mathbf{S}, Q_3), h(\mathbf{M}))$$
$$2.2 \quad p(\mathbf{T}) \leftarrow h(\mathbf{T}) \wedge \neg(K(\mathbf{M}, Q_3), h(\mathbf{T}))$$
$$2.3 \quad p(\mathbf{W}) \leftarrow h(\mathbf{W}) \wedge \neg(K(\mathbf{T}, Q_3), h(\mathbf{W}))$$
$$3. \quad ((K(\mathbf{T}, Q_3), h(\mathbf{W})) \leftarrow ((\Sigma, P_2^{Q_3}), \neg h(\mathbf{M})) \wedge ((\Sigma, P_2^{Q_3}), \neg h(\mathbf{T}))$$
$$4. \quad (K(\mathbf{M}, Q_2), h(\mathbf{T})) \leftarrow (K(\mathbf{M}, Q_2), \neg h(\mathbf{M})) \wedge (K(\mathbf{M}, Q_2), \neg h(\mathbf{W}))$$
$$5. \quad (K(\mathbf{M}, Q_2), \neg h(\mathbf{W})) \leftarrow ((\Sigma, P_2^{1,Q_3}), \neg p(\mathbf{W}))$$
$$\}$$

In other words, ion $((\Sigma, P \cup H_1), \neg p(\mathbf{W}))$ succeeds with $History = Q_3$, ion $((\Sigma, P \cup H_2), \neg p(\mathbf{T}))$ succeeds with $History = Q_2$, and ion $((\Sigma, P \cup H_3), \neg p(\mathbf{M}))$ succeeds with $History = Q_1$. Thus the only logic field where all three goals $\neg p(\mathbf{W})$, $\neg p(\mathbf{T})$, and $\neg p(\mathbf{M})$ succeed is $(\Sigma, P \cup H_3)$, where H_3 is given by:

$$((K(\mathbf{T}, Q_3), h(\mathbf{W})) \leftarrow ((\Sigma, P_2^{Q_3}), \neg h(\mathbf{M})) \wedge ((\Sigma, P_2^{Q_3}), \neg h(\mathbf{T}))$$
$$(K(\mathbf{M}, Q_2), h(\mathbf{T})) \leftarrow (K(\mathbf{M}, Q_2), \neg h(\mathbf{M})) \wedge (K(\mathbf{M}, Q_2), \neg h(\mathbf{W}))$$
$$(K(\mathbf{M}, Q_2), \neg h(\mathbf{W})) \leftarrow ((\Sigma, P_2^{1,Q_3}), \neg p(\mathbf{W}))$$
$$(K(\mathbf{S}, Q_1), \neg h(\mathbf{T})) \leftarrow ((\Sigma, P_2^{2,Q_2}), \neg p(\mathbf{T}))$$
$$(K(\mathbf{S}, Q_1), h(\mathbf{M})) \leftarrow (K(\mathbf{S}, Q_1), \neg h(\mathbf{M})) \wedge (K(\mathbf{S}, Q_1), \neg h(\mathbf{W}))$$
$$(K(\mathbf{S}, Q_1), \neg h(\mathbf{W})) \leftarrow ((\Sigma, P_2^{1,Q_3}), \neg p(\mathbf{W}))$$

This logic field is quite different from the initial logic field (Σ, P). In other words, the intuitive argument does not address the situation (Σ, P) described by the decree, but a different situation described by $(\Sigma, P \cup H_3)$. Therefore, there is, *stricto sensu*, no paradox.

6 Conclusion

A precise computational analysis of three mathematical paradoxes has been made. Each one of these paradoxes sheds light on some important aspect of the formal logic of knowledge.

Protagoras paradox stresses the importance of the notion of *ion*. Indeed, apart from tautologies, no statement is true except in the context of some axiomatic theory. Furthermore, there may be several coexisting, or even competing, such theories [11]. Protagoras paradox occurs because the notion of an ion is missing.

Newcomb's paradox illustrates *non-determinacy* and *parameter-passing* in procedure calls in logic field computations. This paradox occurs because these mechanisms have been ignored.

The Hangman illustrates non-determinacy and the importance of the coherent management of procedure variable rewritings provided by our framework. The intuitive argument in the Hangman paradox rests on a logical patchwork made up of the three ions mentioned in the above discussion: $((\Sigma, P_2^{1,Q_3}), \neg p(\mathbf{W}))$, $((\Sigma, P_2^{2,Q_2}), \neg p(\mathbf{T}))$, and $((\Sigma, P_2^{3,Q_1}), \neg p(\mathbf{M}))$, where the values of the three different programs P_2^{1,Q_3}, P_2^{2,Q_2}, and P_2^{3,Q_1} were also given. If the procedure variable $History$ is seen as a variable which gets assigned during the computation, it appears that this variable has been badly treated here, as it is not uniformly assigned throughout the argument. For example, we have clause (5) of P^3:

$$(K(\mathbf{M}, Q_2), \neg h(\mathbf{W})) \leftarrow ((\Sigma, P_2^{1,Q_3}), \neg p(\mathbf{W}))$$

where a property that holds in the case the course of events is Q_3 is used to deduce some fact the prisoner would know in the case the course of events is Q_2. Thus we are here simply mixing "temporal lines".

In other words, each one of these paradoxes points out some programming mistake which should be avoided in logic programming.

References

[1] E. A. Ashcroft, W. W. Wadge. *Prescription for semantics*, ACM TOPLAS 4 (2), (1982) pp. 238-293

[2] N. Asher, J. Kamp. *The knower's paradox and representational theories of attitudes*, in Theoretical Aspects of Reasoning about Knowledge, J. Y. Halpern ed, Morgan Kaufmann (1986), pp. 131-147

[3] H. Barendregt. *The type free lambda calculus*, in Handbook of Mathematical Logic, J. Barwise ed, North Holland (1977), pp. 1091-1142.

[4] van Emden M.H. and Kowalski R. *The semantics of logic as a programming language*, J. ACM 23, (1976) pp. 733-742

[5] Guessarian I. *Algebraic semantics*, Springer LNCS 99, Berlin (1981)

[6] D. Kaplan, R. Montague. *A paradox regained* , Notre Dame Journal of Formal Logic 1, (1960), pp. 79-90

[7] Nait Abdallah M. A. *Ions and local definitions in logic programming*, Springer LNCS 210 (1986), pp. 60-72

[8] Nait Abdallah M. A. *Procedures in logic programming*, Springer LNCS 225 (1986), pp. 433-447

[9] Nait Abdallah M. A. *AL-KHOWARIZMI: A formal system for higher order logic programming*, Springer LNCS 233 (1986), pp. 545-553

[10] Nait Abdallah M.A. *Logic programming with ions*, Springer LNCS 267 (1987), pp. 11-20

[11] Nait Abdallah M.A. *Heuristic logic and the process of discovery*, Proc. Fifth International Conference of Logic Programmaing, Bowen and Kowalski ed., Vol 2. MIT Press (1988)

[12] Nait Abdallah M.A. *A logico-algebraic approach to the model theory of knowledge*, (Theoretical Computer Science, to appear)

[13] R. Nozick. *Newcomb's problem and two principles of choice*, in Essays in Honor of Carl G. Hempel, N. Rescher ed., Humanities Press (1969)

[14] W.V.O. Quine, *On a so-called paradox*, Mind 57, (1953), pp. 65-67

[15] J. Stoy, *The Scott-Strachey approach to programming language semantics*, MIT Press (1977)