The Reasoner

Volume 17, Number 4
July 2023

thereasoner.org
ISSN 1757-0522

Contents

Editorial

Dear Reasoners,

most of us are or will be soon free to administrative and teaching duties for the summer break. So this is a perfect time to remind you how you can contribute to and get involved in The Reasoner.

Last year we inaugurated a new format which we term *Focussed issue* – a collection of solicited contributions highlighting the state of the art of a field. So far we have covered a variety of topics, spanning *Evidential Pluralism*, *History of Logical Reasoning and Infinitary Reasoning* and *Philosophy of Finance*. And an issue on *Rationality* is currently in the pipeline. Those issues are great opportunities to let the wider community know about emerging topics or innovative takes on consolidated topics, so why not edit one? We'd be particularly interested in the reasoning-related issues raised by machine learning and AI, from scientific reasoning, to individual and institutional decision-making.

Our *Dissemination corner* is another relatively recent feature which has been popular in the last couple of years. It allows the wider reasoning community to keep up with the developments of reasoning-related projects,read about your latest books, papers, open positions, and hopefully get inspired for the next successful reasoning-related project.

Last, but certainly not least, you may contribute by interviewing someone in your own field or by contributing regular "What's hot in..." columns.

Hykel Hosni
University of Milan

Features

Logical approaches to ignorance representation

Several current works in epistemic logic focus on finding a way to model the notion of ignorance. One of the difficulties in achieving this task is that there is no agreement on exactly which notion one is trying to model. Even from an epistemological perspective, there exist at least two different definitions of ignorance. The Standard View takes ignorance as the 'absence of knowledge,' while the so-called New View takes it as the 'absence of true belief' (see Le Morvan & Peels, 2016, "The nature of ignorance: two views." *The epistemic dimensions of ignorance*, Peels, R., Blaauw, M. (eds), Cambridge University Press, pages 12-32). Besides this debate on the definition of ignorance, one should observe that ignorance is a multi-layered notion that embodies various types. Some examples include disbelieving ignorance and suspending ignorance. One of the most elusive types of ignorance is considered to be deep ignorance, which is when an agent neither believes a proposition nor its negation, nor suspends judgment on it. Epistemologists aim to lay out a comprehensive taxonomy of types of ignorance, which underscores the importance of analysing the specific features of each type of ignorance. The objective of this short note is twofold: to provide a comparative analysis of the existing

formal approaches to ignorance and to investigate further developments of ignorance representation.

The first definition we consider is the one of van der Hoek & Lomuscio (2014, "A logic for ignorance", *Electronic Notes in Theoretical Computer Science*, 85(2): 117-133) , who provide a system for representing *ignorance whether*. Syntactically, they represent ignorance as a primitive modality (we call it I^w) definable via K as $\neg K\phi \wedge \neg K\neg\phi$. Semantically, they use standard Kripke semantics, in which I^w is defined as follows:

- $\mathcal{M}, w \models I^w\phi$ iff there exists w' such that Rww' and $\mathcal{M}, w' \models \phi$ and there exists w'' such that Rww'' and $\mathcal{M}, w \models \neg\phi$.

An important observation should be made concerning the use of I^w as an operator representing ignorance. It does not permit the distinguishing of two distinct types of ignorance: deep and suspending ignorance. In particular, suspending ignorance is the one in which an agent is aware of the content of the proposition but suspends judgment on the truth-value of the proposition. Deep ignorance means that they are unaware of the content. The semantic definition of $I^w\phi$ presupposes that there is an accessible world in which ϕ holds and an accessible world in which $\neg\phi$ holds. How should this be interpreted from the agent's point of view? An intuitive answer is that the agent considers ϕ as possibly true and possibly false, thus suspending judgment on ϕ. However, this shows that I^w does not represent deep ignorance. Another natural language interpretation of the definition of $I^w\phi$ can be that the agent considers neither ϕ nor $\neg\phi$ as a true proposition. This is the case for both suspending and deep ignorance. However, in this case, the semantics proposed for I^w is too unexpressive to distinguish deep and suspending ignorance.

The second definition was introduced by Steinsvold (2008, "A note on logics of ignorance and borders", *Notre Dame Journal of Formal Logic*, 49(4): 385-392), who considers ignorance as *unknown truth*, formalized as $\phi \wedge \neg K\phi$. We call his operator I^u, and one can define it on Kripke frames as follows:

- $\mathcal{M}, w \models I^u\phi$ iff $\mathcal{M}, w \models \phi$ and there exists w' such that Rww' and $\mathcal{M}, w' \not\models \phi$.

Similarly to the case of I^w one can question the natural language interpretation of the semantic clause for I^u. This leads to two possible conclusions: either the I^u operator cannot represent deep ignorance, or it cannot distinguish deep ignorance from other types of ignorance.

The third option for ignorance representation is provided by the authors, Kubyshkina & Petrolo (2021, "A logic for factive ignorance", *Synthese*, 198: 5917-5928). We dub this ignorance operator I^f and define it as follows:

- $\mathcal{M}, w \models I^f\phi$ iff for all $w' \neq w$ if Rww' then $\mathcal{M}, w' \not\models \phi$ and $\mathcal{M}, w \models \phi$.

Unfortunately, as it is the case for I^w and I^u, I^f does not permit one to represent deep ignorance, or, at least, it does not permit one to distinguish deep ignorance from other cases of ignorance.

The three operators, I^w, I^u, and I^f are the only proposals currently available in the literature. While epistemologists refine their analysis of the notion of ignorance by considering its various types, the current logical frameworks are unable to distinguish between these forms of ignorance. In particular, I^w, I^u, and I^f represent three types of ignorance, none of which constitutes an exclusive case of deep ignorance. Deep ignorance is characterized by an absence of an attitude towards both a proposition and its negation. Therefore, a crucial point in the formal representation of deep ignorance is the characterization of this absence. To solve this problem, we see at least two strategies. First, one can reinterpret ignorance operators on Kripke models supplemented with an awareness function, similarly to the work proposed by Fagin & Halpern (1988, "Belief, Awareness and Limited Reasoning", *Artificial Intelligence*, 34(1): 39-76) for knowledge representation. The second option is to interpret ignorance operators on Kripke semantics with possibly incomplete worlds. In this framework, the absence of an attitude towards a proposition can be expressed by neither validating this proposition nor its negation.

Ekaterina Kubyshkina
University of Milan
Mattia Petrolo
University of Lisbon and Federal University of ABC

Dissemination Corner

BRIO: Low-Level Analysis of Trust in Probabilistic and Opaque Programs

The project BRIO aims at developing formal and conceptual frameworks for the analysis of AI systems and for the advancement of the technical and philosophical understanding of the notions of Bias, Risk and Opacity in AI, with the ultimate objective of generally contributing to the development of trustworthy AI. Overviews of the BRIO project and of its main objectives can be found in The Reasoner, Vol.16 Num. 1, Vol. 16 Num. 3, and Vol. 16 Num. 5. One of BRIO's technical research directions concerns the formal analysis of reasoning and trust assessment with regard to probabilistic and opaque processes. One of the most challenging aspects of modern AI systems, indeed, is the fact that they do not operate in a deterministic way, namely they do not simply implement a function that associates a unique output to each input, and they are not transparent, in the sense that a general formal description of their behaviour might not be available.

A new line of work which can be subsumed under this research direction recently yielded the first results. This work aims at a low-level analysis of probabilistic processes in general, of opaque probabilistic processes in particular, and of the computational processes involved in the analysis of their behaviour and in the assessment of their trustworthiness. As opposed to the previous work conducted in the context of BRIO, the focus is not on the inferences that can be drawn by observing the external behaviour of programs, but on what lies below the surface of computation and on the mechanisms underlying the computational practices that we can follow to conclude that a program is worth of our trust. This glimpse into the internal workings of programs and of computational practices related to the assessment of trust is also meant as a first step towards the development of methods for explaining the doing of highly unpredictable machines such as probabilistic programs. The hope is hence to bring a formal contribution also to the development of explainable AI.

Several formalisms for the low-level analysis of computational processes have been introduced and abundantly employed in the theoretical computer science literature, but one of the most versatile is certainly λ-calculus. This calculus has been devised by Alonzo Church in the 1930s in order to provide a foundation to mathematical reasoning that would give to the notion of function, a fundamental mathematical entity that essentially relate to the notion of change and of procedure, the place that it deserves. The portion of this original system that is nowadays employed for the theoretical analysis of computational processes has been isolated by Church in 1936, after the original system was shown to be inconsistent due to the Kleene–Rosser paradox. This is how pure λ-calculus was born, a model of computation that precisely formalises the intuitive notion of computable function. Another formal framework which tightly tied to λ-calculus is type theory. The first versions of type theory have been developed by Russell between 1902 and 1908 in order to avoid Russell's paradox, and the system that Church defined in order to keep using λ-calculus for foundational purposes after the discovery of the Kleene–Rosser paradox was precisely a type theory which has come to be known, not surprisingly, as Church's theory of types.

One might wander now how type theory is relevant to the analysis of computational processes. Well, type theory, after its original employment in the context of the foundations of mathematics, became an extremely fruitful and versatile instrument to enrich programming languages in order to gain more control over computations and formal insights about the behaviour of programs. But, lest this short article becomes as opaque as the opaquest AI system, some more insights on the nature of types and on what good do they do for programming is due. A type is simply a high-level input/output description of what a program does. A program of type $\mathbb{N}\times\mathbb{N} \to \mathbb{N}$, for instance, expects as input a pair of natural numbers, as specified by $\mathbb{N} \times \mathbb{N}$, and yields as output a natural number, as specified by the occurrence of $\mathbb{N}$ to the right of $\to$. A computer program implementing addition, for instance, will precisely be of type $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$. The type of a program, nevertheless, does not only tell us explicitly what the program is supposed to do, but it also forces the program to do it: to only accept certain inputs and to always yield outputs of a certain kind. As types were used by Russell and Church to constrain the behaviour of functions in order to avoid paradoxes, they can be used to constrain computer programs in order to avoid undesired behaviours.

This is clearly very useful since λ-calculus can be used to formally encode and study computer programs and types can be used in combination with it—by using, for instance, simply typed λ-calculus—to restrict our attention to computer programs that behave well by definition with respect to our expectations. It is perhaps still rather obscure, though, what all this has to do with probabilistic processes and opacity. After all, λ-calculus is supposed to formalise functions. And functions, by definition, always yield the same result given the same set of inputs. In other words, functions are very essentially deterministic objects. While this is certainly true, λ-calculus in the last decades took a road that took it very far away from its original form. It started roaming in the depths of the realm of applied computer science, taking forms that are far more general and versatile that the original one. Probabilistic λ-calculus is one of the many offsprings of these peregrinations. As for types, they followed eagerly and adapted to many of the new, different forms of λ-calculus. The kind of analysis that can be conducted by λ-calculus and type theory for traditional, deterministic computer programs can then be extended also to probabilistic programs.

Opacity's story is a completely different one. Indeed, the basic idea of λ-calculus is to analyse processes through a complete syntactic representation of them. A fully transparent definition of programs is, in a sense, presupposed by it. Radically new ideas are hence required to conduct a low-level analysis of opaque probabilistic programs and of the computational processes involved in the assessment of their trustworthiness. These ideas are currently under development and borrow from the literature on computability theory. In particular, the notion of oracle seems to be extremely promising. In brief, an oracle can be described as the formal tool used to represent a black box which is able to solve a given problem in a single, unanalysed operation. Thus, so far as computing theory is concerned, oracles are, as their very name suggests, as obscure as it gets: a perfect starting point for the study of opaque programs.

Francesco Genco
University of Milan

GoA: The Geometry of Algorithms

The Geometry of Algorithms (GoA) is an ongoing 4-year (2020-2024) research project, funded by the French National Research Agency - ANR with a grant of 252,024 euros (ref. ANR-20-CE27-0004). The project is hosted by the Institute for the History and Philosophy of Science and Technology (IHPST) in Paris. Its primary aim is to answer to the question: what is an algorithm?

Algorithms have an increasingly significant role in our daily lives. We use algorithms not only for everyday tasks, such as web research, but also for more complex and delicate operations such as medical diagnosis. The general feeling is that we are entrusting algorithms with not only a significant amount of our decisions but also our lives. However, there is no real consensus among experts today about what an algorithm is.

The GoA project aims to clarify this question by studying the epistemological and logical foundations of the notion of an algorithm. The main issue is to examine whether this notion is well-founded from both a philosophical and formal perspective. Specifically, the project seeks to determine whether it is possible to assign the status of *scientific entities* to algorithms by providing them with a precise mathematical representation.

The starting point of the GoA project is the observation that it is not possible to answer the question "what is an algorithm?" by means of a stipulative definition, since the meaning of this notion has been embedded in our language practice long before the birth of computer science. Indeed, the word "algorithm" can be found in mathematical discourse, but also in our ordinary discourse long before the first computers. To assign a precise and formal definition to the notion of an algorithm, one needs to perform a preliminary work of *conceptual analysis*. In particular, following the approach pioneered by G. Kreisel, the idea is that before undertaking a rigorous formalization step, one needs to perform a preliminary epistemological analysis to ensure some degree of "informal rigour."

It was this sort of conceptual analysis that led, in the early 20th century, to the introduction of formal notions of general recursive functions, lambda-calculus, Post machines, and Tur-

ing machines by K. Gödel, A. Church, E. Post, A. Turing, and others based on the informal notion of "effective computability." This formalization eventually led to the development of the theory of computability and the Church-Turing thesis, which provides a mathematically precise definition of the notion of "effective computable function." Since an algorithm is broadly conceived as an effective procedure for computing, one might be tempted to conclude that the theory of computability is enough to settle what an algorithm is. However, there are two main obstacles to this view. Firstly, there are algorithms that are non-deterministic as they associate multiple solutions with the same input and, therefore, cannot be analyzed directly in terms of functions. Secondly, even if we restrict to functions, the theory of computability views them from an extensional perspective, not an intensional one. To show that a function is computable, it is enough to show that its graph (i.e., its set of inputs-outputs) can be computed by one of the formalisms mentioned above. For instance, the Fibonacci function can be formalized by means of a recursion scheme when general recursive functions are used. But the same function has to be defined differently in lambda calculus, as recursion is not a primitive operation of this computational model. The choice of one formalism over the other could lead to different computational costs. However, from the perspective of the Church-Turing thesis, what matters is that a function is computable by an algorithm, not the comparison of different algorithms for computing it. The theory of computation does not provide a general framework for discussing all possible ways of computing a function, regardless of the specific primitive operations tied to a particular computational formalism. Defining such a general framework and thus developing a general theory of algorithms is the goal of the GoA project.

The conceptual analysis undertaken by GoA is based on the analysis of computer science practice and on the idea that the notion of an algorithm has to be understood in conjunction with two other notions characterizing such practice, computation, and program.

A computation is understood here as a physical process – whether mechanical, electronic, quantum, etc. – that takes place every time a computer program is run on some physical device (e.g., a computer). Conceived in this way, a computation is deterministic by nature: although the laws describing the evolution of the (e.g., quantum) system may be non-deterministic (e.g., probabilistic), a computation determines only one of the possible evolutions of the system. Moreover, although a computation is linked to a physical theory, in fact, it can be described independently of it and formalized via the theory of dynamical systems, as done, e.g., in Sieg & Byrnes (1999, "An abstract model for parallel computations: Gandy's thesis", *The Monist*, 82(1): 150–164).

Unlike computations, programs need not be deterministic and have a modal dimension. For instance, computer practice reveals the existence of non-deterministic programs, such as probabilistic programs, concurrent programs, and quantum circuits. To capture this modal dimension, specific formal systems called *computational models* are used. These models are characterized by a set of basic programming instructions that give rise to multiple possible computations (i.e., "executions" of these instructions). A program (with respect to a given model) is a formal arrangement of these basic instructions specifying a number of possible computations.

The GoA project will utilize an abstract notion of model of computation, inspired by the work of Seiller, (2017, "Interaction graphs: graphings", *Annals of Pure and Applied Logic*, 168(2): 278–320), which corresponds to the action of a monoid M on a space. The elements of the monoid are generated by a finite set of instructions. This is where the geometric perspective underlying the GoA project emerges. Similar to the Erlangen program developed by F. Klein in the late 19th century, which aims to classify various types of geometry based on the action of a group of transformations on a space (along with the invariants generated by these transformations), the GoA project seeks to classify computer programs based on the action of a monoid of instructions on a space. Then, the algorithms can be characterized as more general structures implemented by the programs. Specifically, by using tools from the Curry-Howard correspondence between proofs and programs, and particularly from J.-Y. Girard's Geometry of Interaction, one can view a program as a *graphing*. In other words, a program is the geometrical realization of a graph on a measurable space X, where the vertices of the graph correspond to subspaces of X (e.g. sets of configurations of a machine) and the edges are elements of a monoid M (e.g., the instructions of a machine).

This brings us to the main original proposal of the GoA project: defining *algorithms as specifications*. Since programs are defined as (some kind of) graphs, the idea is to define algorithms as (i) finite labelled graphs, (ii) together with a specification of the labels. Point (i) concerns the syntactic structure of an algorithm. Specifically, the idea that a given program implements the syntactic structure of a certain algorithm will be witnessed by some notion of graph homomorphism in which two distinct edges with the same label are mapped to isomorphic images. This notion of graph homomorphism can be adapted to allow some edges to be mapped to whole subgraphs. In this way, certain algorithmic operations would be encoded by program subroutines, and this opens the way to attribute an *epistemic* character to an algorithm. This means that an algorithm can be described with respect to the operations that the users consider to be relevant in order to understand and communicate it, although these operations are not necessarily in a one-to-one correspondence with the primitive instructions of a specific programming language or model of computation. For instance, in Figure 1, the four programs all implement a given algorithm $\mathcal{A}$ in which the part in purple corresponds to a single labelled edge. But only the two programs on the right implement a more detailed algorithm $\mathcal{B}$ specifying the use of an inner while loop. Point (ii) concerns the kind of operation performed by each edge of the labelled graph and is formalized as a *specification* for each label. The idea is that to correctly implement an algorithm, a program should not only satisfy the above condition, but also interpret the labelled edge by a program (or an instruction) realising this specification. The GoA project aims to formalize such a specification by means of the logical system associated with a given set of programs (thus exploiting the Curry-Howard correspondence). Looking again at Figure 1, the specification should allow one to distinguish between the programs `gdc1` and `f`, as they differ on the operation realising the purple edge (one computes the remainder, the other computes a substraction). Similarly, it should allow one to distinguish between `gcd2` and `g`, but it should also allow one to check that `gcd1` and `gcd2` implement the same algorithm $\mathcal{A}$ (with specifications) since the purple chunks perform the same operations.

One aim of the GoA project is to show that the proposed for-

```
def gcd1(x, y):
  while(y):
    z = x
    x = y
    y = z % y
  return x
```

(a) Realises $\mathcal{A}$

```
def f(x, y):
  while(y):
    z = x
    x = y
    y = z - y
  return x
```

(b) Realises $\mathcal{A}$

```
def gcd2(x, y):
  while(y):
    z = x
    x = y
    while z>= y:
      z = z - y
    y = z
  return x
```

(a) Realises $\mathcal{A}$ and $\mathcal{B}$

```
def g(x, y):
  while(y):
    z = x
    x = y
    while z>= y:
      z = z \ y
    y = z
  return x
```

(b) Realises $\mathcal{A}$ and $\mathcal{B}$

Figure 1: Four programs to illustrate algorithms $\mathcal{A}$ and $\mathcal{B}$

malization of the notion of algorithm is more flexible and encompassing than existing proposals. To achieve this, the project compares the geometrical approach described earlier with other proposals based on the generalization of models of computation, in order to analyze how they explain traditional algorithmic techniques such as Euclid's constructions, and identify the algorithmic components of contemporary procedures such as those used in machine learning and deep learning.

Alberto Naibo
University Paris 1 Panthéon-Sorbonne
Mattia Petrolo
Centre for Philosophy of Science of the University of Lisbon
Thomas Seiller
CNRS, University Sorbonne Paris Nord

What's Hot in ...

Statistical Relational AI

In the last column in this series, which appeared now a full four months ago in the March issue, the use of logic in statistical relational AI was explored. We saw that logic was used in different ways in different approaches, either as (soft) constraints on possible worlds, or as definitions of predicates in terms of other predicates. However, the logical formalisms employed in these different ways was always a variant of of the classical, first-order predicate calculus. Sometimes it is extended by some fixed-point operator, as in probabilistic logic programming, and sometimes it is restricted to exclude existential quantifiction (as in many implementations of Markoc logic networks). Essentially, though, the logic employed is classical, and seems entirely unrelated to the many formalisms of reasoning under uncertainty that have proliferated over the past decades. Particularly curious is that there seems to be little resemblance to *probability logic*, logical frameworks that directly adress probability themselves.

Probability logic has already been brought to wider philosophical attention by Rudolf Carnap, who discusses them extensively in his influential 1950 monograph *The logical foundations of probability*, and they have been received into artificial intelligence at least since Joseph Halpern's fundamental 1990 paper on *An analysis of first-order logics of probability*. Halpern distinguished two fundamentally different types of probability logic, corresponding to a similar distinction made by Carnap earlier. *Type I logics* are evaluated over a single structure, much like a sentence of first-order logic would be, but its domain is additionally equipped with a probability measure over its elements. It can answer questions such as *What is the probability of a given sheep to be black?* evaluated as the probability of $\{x \in \omega \mid \omega \models \text{Black}(x)\}$ on the probability space of domain elements. *Type II logics*, on the other hand, are evaluated not over a probability space of domain elements, but over a probability space of possible worlds. In other words, rather than a measure on the domain of a single structure, such a logic would be evaluated on a measure on a space of structures. It would answer questions such as *Given that the three sheep observed have all been black, what is the probability that every sheep is black?* evaluated as the conditional probability

$$P\left(\omega \models \forall_x \text{Black}(x) \mid \omega \models \text{Black}(a) \wedge \text{Black}(b) \wedge \text{Black}(c)\right).$$

Halpern also found that these notions of probability could easily be combined, leading to a *Type III logic* whose models are probability distributions over worlds, each of whose domains is itself equipped with a probability measure.

Compare this to what we know about statistical relational artificial intelligence. For any specified domain, a statistical relational specification defines a probability distribution over the possible worlds with that domain. In this sense, statistical relational languages such as probabilistic logic programming or Markov Logic Networks *are* Type II probability logics. They are declarative, syntactical specifications whose semantics is given by a probability measure over possible worlds. From this point of view, the defining factor of statistical relational frameworks within Type II probability logics is that they arrive at a Type II formalism that they complement classical logic with an essentially separate probabilistic layer of either graphical models or simple probabilistic facts.

There is another key difference to traditional probability logics, though: While in traditional probability logic, a sentence is evaluated with respect to a measure on the possible worlds, which may or not be a model of the sentence, a statistical relational framework defines but a single probability measure for any domain. This reminds strongly of the traditional relationship between classical logic and logic programming, where a sentence of first-order logic can have any number of models, while the meaning of a logic program is defined as its one unique minimal model. This analogy is surely not incidental, but while there has been some interesting work on the relationship between maximum entropy models and Markov logic networks, much remains to be understood.

So, what does that mean for the value of more traditional probability logic for statistical relational artificial intelligence? Firstly, there seems to be a lot of untapped potential in utilising Type I probability logics in precisely those roles that classical logics know have: As definitions in directed approaches, and as soft constraints in undirected approaches to statstical relational modelling. Beyond various computational advantages, this would lift the expressivity of such frameworks from a Type II to a Type III probability logic, with all the same benefits as in the traditional setting of probability logic. There have been

some recent attempts in this direction in the context of frameworks built on Bayesian networks, as well as some work on integrating such statements into probabilistic logic programming. As far as I know there has not been any work on probabilistic constraints in undirected approaches.

Secondly, more traditional Type II probability logics could be used as a powerful logical language to interact with statistical relational models. For instance, current query languages for statistical relational models do not make any use of their probabilistic semantics. While the details vary, queries are usually as ordinary quantifier-free formulas. They can be supplemented by additional formulas as "evidence", and by a command word to indicate the type of query desired, e. g. the conditional probability or the single most likely truth values given the evidence. This could be supplemented very effectively with queries expressed in a full Type II probability logic.

Finally, Type II probability logics could be used to reason about statistical relational approaches. Just to mention a one avenue, there has recently been an upsurge in interest around the asymptotic behavious of statistical relational approaches on large datasets. Wouldn't it be great to have not just nebulous convergence results, but a complete axiomatisation of their limiting behaviour? And which formalism could be more suitable for this task than probability logic?

Felix Weitkämper
Computer Science, LMU Munich

Courses and Programmes

Programmes

MA in Reasoning, Analysis and Modelling: University of Milan, Italy.
APhil: MA/PhD in Analytic Philosophy, University of Barcelona.
Master Programme: MA in Pure and Applied Logic, University of Barcelona.
Doctoral Programme in Philosophy: Language, Mind and Practice, Department of Philosophy, University of Zurich, Switzerland.
Doctoral Programme in Philosophy: Department of Philosophy, University of Milan, Italy.
LogiCS: Joint doctoral program on Logical Methods in Computer Science, TU Wien, TU Graz, and JKU Linz, Austria.
HPSM: MA in the History and Philosophy of Science and Medicine, Durham University.
LoPhiSC: Master in Logic, Philosophy of Science and Epistemology, Pantheon-Sorbonne University (Paris 1) and Paris-Sorbonne University (Paris 4).
Master Programme: in Artificial Intelligence, Radboud University Nijmegen, the Netherlands.
Master Programme: Philosophy and Economics, Institute of Philosophy, University of Bayreuth.
MA in Cognitive Science: School of Politics, International Studies and Philosophy, Queen's University Belfast.
MA in Logic and the Philosophy of Mathematics: Department of Philosophy, University of Bristol.
MA Programmes: in Philosophy of Science, University of Leeds.
MA in Logic and Philosophy of Science: Faculty of Philosophy, Philosophy of Science and Study of Religion, LMU Munich.
MA in Logic and Theory of Science: Department of Logic of the Eotvos Lorand University, Budapest, Hungary.
MA in Metaphysics, Language, and Mind: Department of Philosophy, University of Liverpool.
MA in Mind, Brain and Learning: Westminster Institute of Education, Oxford Brookes University.
MA in Philosophy of Biological and Cognitive Sciences: Department of Philosophy, University of Bristol.
MA programmes: in Philosophy of Language and Linguistics, and Philosophy of Mind and Psychology, University of Birmingham.
MRes in Methods and Practices of Philosophical Research: Northern Institute of Philosophy, University of Aberdeen.
MSc in Applied Statistics: Department of Economics, Mathematics and Statistics, Birkbeck, University of London.
MSc in Applied Statistics and Datamining: School of Mathematics and Statistics, University of St Andrews.
MSc in Artificial Intelligence: Faculty of Engineering, University of Leeds.
MSc in Cognitive & Decision Sciences: Psychology, University College London.
MSc in Cognitive Systems: Language, Learning, and Reasoning, University of Potsdam.
MSc in Cognitive Science: University of Osnabrück, Germany.
MSc in Cognitive Psychology/Neuropsychology: School of Psychology, University of Kent.
MSc in Logic: Institute for Logic, Language and Computation, University of Amsterdam.
MSc in Mind, Language & Embodied Cognition: School of Philosophy, Psychology and Language Sciences, University of Edinburgh.
MSc in Philosophy of Science, Technology and Society: University of Twente, The Netherlands.
MRes in Cognitive Science and Humanities: Language, Communication and Organization: Institute for Logic, Cognition, Language, and Information, University of the Basque Country (Donostia San Sebastián).
Open Mind: International School of Advanced Studies in Cognitive Sciences, University of Bucharest.
Research Master in Philosophy and Economics: Erasmus University Rotterdam, The Netherlands.
Doctoral Programme in Philosophy: Language, Mind and Practice, Department of Philosophy, University of Zurich, Switzerland.
MA in Philosophy: Dept. of Philosophy, California State University Long Beach.