Explainability in Autonomous Agents: Interpretable Models, Abstraction, and Beyond

Paulius Dilkas

9th December 2019

1 Introduction

As black-box machine learning algorithms become more powerful, an important issue emerges: how much are we willing to trust a man-made tool without knowing how it works, and why it suggest the answer that it does? Thus, the topics of explainability and interpretability gather importance. Perhaps information is more interpretable when expressed in a richer language or representation. Perhaps possible interpretations become obvious after eliminating unnecessary detail. These ideas form the foundation of the work presented in this proposal.

We begin by considering abstraction in a relatively general setting: we review what has been accomplished so far and the broad strokes of the work envisioned in this proposal. Sections 3 to 5 then outline three specific projects that are under way. Section 6 then divides the work into executable work packages, and we use Sections 7 to 9 to briefly consider outcomes, risks, and impact.

2 Abstraction in First-Order Probabilistic Inference

Abstraction can be broadly defined as the process (and result) of omitting detail [39]. Sometimes the omitted information is irrelevant in answering the questions we are interested in, and sometimes an abstraction provides a simplified (and approximately correct) view of a situation that originally was too complex to be reasoned about. While areas such as planning and verification have benefited from abstraction in various ways [65], research into abstraction in FOPI has just begun and awaits significant contributions [2, 32, 33].

Our goal is to develop new types of abstractions, find efficient algorithms for constructing an abstraction from a given model, and investigate how abstraction can be integrated into both inference and learning algorithms. Simplification and abstraction can benefit us in both efficiency and interpretability, i.e, simpler models are likely to result in faster inference, while at the same time being easier to understand by the user. Finally, the quest for abstraction algorithms is likely to lead to a better theoretical understanding of what properties can be preserved by an abstraction, what error bounds can be established when abstraction approximates the answer, and upper and lower bounds on the complexity of performing abstraction and providing the desired guarantees.

Objectives.

- To further the theoretical understanding of abstraction in FOPI.
- To improve inference speed by investigating abstraction as a separate process as well as a component of inference.
- To make models learned from data simpler and more transferable.
- To increase the interpretability of FOPI models.

2.1 Literature Review

Logical approaches to reasoning have dominated the fields of AI and computing for many decades. They have resulted in expert systems that can use reasoning to make predictions and diagnose problems [30], logic programming languages that allow the user to declaratively describe the problem and trust the inference algorithm to find the answer efficiently [43], and automated (interactive) theorem proving and proof checking software that assists mathematicians in constructing correct proofs [63]. Probabilistic methods, on the other hand, have particularly flourished with the arrival of big data and access to more data in general, transforming areas such as natural language processing and pattern recognition [13].

While probabilistic models are great at handling uncertainty, their simplistic representations can be hard to interpret. On the other hand, systems based on logic have rich representations, but cannot handle uncertainty. *First-order probabilistic inference* (FOPI) (also known as *statistical relational AI*) attempts to bridge the gap between the two and suggests a range of representations[1] capable of handling probabilities as well as (parts of) first-order logic [13]. The models can be constructed manually or learned from data, and the process of computing the probability of a query—usually in the form of a random variable, possibly conditioned on other random variables—is called *inference*.

Most of these models are based either on adding probabilities to programming languages or on adding richer representational structure to probabilistic graphical models (PGMs). The former kind is called *probabilistic programming* [29] and is an active area of research with many implementations. A well-known example is *ProbLog* [58]—a language that extends the logic programming language Prolog by attaching a probability to every clause in the program. A prominent example of the latter is a *Markov logic network* (MLN) [60], which is simply a collection of first-order statements (formulas), with a weight attached to each statement. MLNs were proposed as a first-order generalisation of *Markov random fields*—an undirected PGM. We will use MLNs to provide concrete examples, but we do not commit ourselves to a specific FOPI model just yet.

As an area of research, FOPI is a subfield of AI and can be thought as the probabilistic cousin of automated reasoning. The problem of learning a FOPI model from data draws the attention of researchers not only from the machine learning community, but also data mining and knowledge discovery. Due to how many of the models are formulated, there is also significant overlap with research in programming languages and, specifically, logic programming.

In the last decade, we have seen applications of FOPI in a wide range of areas, ranging from toy problems in probability one might find in a textbook [19] all the way to genetics [66] and cancer research [11]. For instance, probabilistic relational structures have been integrated into recommendation systems [80] and stream mining software [5], and various FOPI models have been used to predict the remaining lifetime of hardware components [75] as well as patterns of criminal and terrorist activity [14]. Moreover, one type of probabilistic logic has been extended to explicitly deal with changes over time and has been used to model virtual spaces such as online chat rooms and massively multiplayer online games [71, 72].

2.1.1 State of the Art: Inference

Recall that a ProbLog program is a set of clauses, where each clause has an associated probability. The ProbLog inference rule [57, 67] for calculating the probability of an arbitrary query Q being true is

$$P(Q) = \sum_{F \models Q} \prod_{f \in F} P(f) \prod_{f \notin F} 1 - P(f).$$

Here, we are summing over all instantiations of variables (called *possible worlds*) that satisfy the query, where F denotes a set of clauses that are evaluated as true. For each world, we calculate its probability by multiplying probabilities associated with clauses or their negations. With this definition, the inference problem becomes an instance of weighted model counting [57].

Weighted model counting (WMC) is an extension of model counting, which is an extension of the *Boolean satisfiability problem* (SAT) [6]. SAT asks whether one can assign values to variables so that a given formula

[1]We will sometimes refer to FOPI models as *probabilistic relational models*, since relations play an important part in this richer representational structure.

evaluates to true. Model counting asks to count the number of ways that can be done. Weighted model counting further extends this problem by assigning a weight to each possible world (in whatever way is appropriate for the problem) and asks for the sum of the weights corresponding to all possible worlds where the formula (or query) is true. In the case of ProbLog, the weight of a world is the product of probabilities of all literals (whether evaluated/instantiated to true or false) [57]. The WMC instance is then compiled to some type of logical circuit for efficient inference. *Knowledge compilation* [15] is the state-of-the-art inference technique for PGMs as well as many FOPI models [57].

Inference for MLNs works in a similar way. We still sum over all possible worlds where the query is true, but the probability of a world x is now defined as

$$P(x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right),$$

where Z is the normalising constant more commonly known as the *partition function*, F is the number of formulas in the MLN, w_i is the weight of the ith formula, and $n_i(x)$ is the number of ways that formula i can be grounded in order to satisfy world x. Here, *grounding* a formula refers to replacing each variable with a value so that the formula evaluates to true.

A commonly used inference algorithm for MLNs relies on *probabilistic theorem proving* [28, 73] which is an example of a *lifted inference* algorithm, i.e., an algorithm that attempts to work directly with variables without having to consider every possible value [54]. The underlying problem, however, is still WMC, and is solved using a combination of techniques well established in the SAT community, e.g., unit propagation and clause learning [73].

2.1.2 State of the Art: Abstraction

Abstraction is an important tool in human cognition and a well-studied subject in cognitive science. For example, Gentner and Hoyos [26] investigate how children learn abstract patterns from observing several objects with a common property, while Bransford and Franks [3] show how the idea conveyed by a sentence is abstracted away from the particulars of its syntactic expression.

Abstraction is also well-known in the AI community, where the main goal of abstraction is to reduce the computational complexity of a task, while ensuring that the process of abstraction itself is reasonably efficient [65]. For instance, abstraction plays a key part in modern approaches to planning, where compound tasks are used to abstract away the details of how those tasks can be implemented [23]. More specifically, abstraction is essential in developing explainable AI [1], where it has been used to create interpretable abstractions of observed behaviour [53] and model the domain knowledge of the user as an abstraction of the system, thus producing explanations that are at the level of detail corresponding to the user's knowledge [70].

Model checking and verification benefit from abstraction as well, particularly in the area of software verification, where a complete model of the program might be too big to be handled by even the most efficient methods, in which case an abstract model could be developed. Depending on how it is created, sometimes properties of the system can be verified using the abstraction [9], while other times the abstract model might produce a false positive, i.e., signal about a possible problem where there is none. If the occurrence of a false positive is suspected, parts of the abstraction can be refined to provide the necessary level of detail, while keeping other parts as they were [8, 31].

Probabilistic abstractions have been used in the context of software verification, where probabilities can help the verification algorithm choose which part of the abstract model needs to be refined [81]. Meanwhile, in the probabilistic programming community, abstractions have been used to determine the required number of Monte Carlo samples in order to compute a probability within a required level of precision [46].

However, only recently has the general case of abstraction for FOPI models been formalised, and the work is mostly limited to defining several key properties that an abstraction may have and showing how those properties interact with each other [2]. While the work on probabilistic programming considers specific examples of abstractions [33] and presents an algorithm for performing predicate abstraction [32], significant work is required to achieve the full generality outlined in this proposal.

2.2 Our Solution

While some theoretical groundwork for abstraction in FOPI has recently been developed by Belle [2], there are many questions left to be answered:

1. How to efficiently create an abstraction of an already-existing model?
2. When is the correct time to stop? What is the right balance between simplicity and information?
3. What makes one abstraction preferable to another?
4. How to incorporate abstraction steps into learning a model from data?
5. How to provide guarantees about an abstraction? For example, we may want to bound the error of an answer to any query, or to ensure that all answers remain exact for a selected set of queries.

In order to answer these questions and develop the required algorithms and techniques, we can draw inspiration from the theory of abstraction for reasoning in formal systems developed by Giunchiglia and Walsh [27] and recent work on abstraction for structural equation models [64]. In particular, an abstraction is often defined as a transformation of the representation into a different form. One way to create such a transformation is via a composition of atomic operations. For example:

- In some cases, $a \to b$ and $b \to c$ can be simplified to $a \to c$.
- If a statement S is true with high probability, perhaps that probability can be rounded up to 1, eliminating the need to consider the case where S is false.
- If a statement is true for all values of a variable, barring a few exceptions, perhaps the exceptions can be discarded.

Consider a specific query Q. Applying such an abstraction rule may or may not change the answer to Q, depending on whether the removed information is relevant to the query. Even if the answer becomes less precise, it might be an acceptable approximation, given that the error is bounded to a reasonable degree. Either way, the abstraction can reduce the search space the inference algorithm has to explore in order to produce an answer.

It becomes clear that it is important to consider creating abstractions with respect to a specific set of queries. Question 5 can then be answered by considering how each abstraction rule affects different types of queries. Sometimes we may get a reasonable numerical upper bound on the error, while other times it may be too time-consuming (or impossible) to bound the error to any reasonable degree, forcing us to reject the abstraction rule altogether.

Thus, we will develop a comprehensive list of abstraction rules (transformations) and define a way to categorise all queries answerable by a FOPI model such that we could answer the following set of questions for each abstraction rule:

- What types of queries can no longer be answered exactly after applying the abstraction rule?
- What is the error bound? Can it be calculated in constant time?
- What is the complexity of applying the abstraction?

Questions 2 and 3 delve deeper into how an abstraction algorithm could work. If the set of rules is extensive enough, any model might eventually be oversimplified into something trivial. We need to measure two things: the amount of (relevant) information preserved by an abstraction, and the complexity of the model. The two metrics would provide a systematic way to answer both questions, while being easily adaptable to different needs (e.g., how much precision are we willing to sacrifice? What queries do we want to support?).

Lastly, we will integrate abstraction steps into both inference and learning algorithms, resulting in faster inference as well as simpler and more robust models.

2.2.1 Innovative Aspects

We aim to innovate by expanding the idea of abstraction in AI to new domains as well as enhancing both inference and learning of probabilistic relational models.

Abstraction in AI. The work on abstraction so far has focused almost exclusively on deterministic systems. We will extend previous work to target rich probabilistic models, resulting in new theoretical ideas and definitions as well as algorithms for transforming models into their abstractions.

FOPI. As FOPI subsumes both probabilistic inference (often #P-complete) and logical inference (NP-complete) [60], it is unlikely that a useful variation of FOPI can ever be tractable. Thus, improvements in inference speed are highly desirable. We will integrate abstraction steps into inference algorithms, making the algorithm recognise opportunities to reduce the complexity of the model before continuing the search. This is likely to yield significant benefits to inference speed.

Statistical relational learning. Abstraction can also be integrated into learning a FOPI model from data. This is likely to make the model more understandable to the user as well as increase the transferability of the model to previously unseen data.

3 Equivalence of Constants in Logic Programs

The overarching goal of this project is to formalise how logic programs act on constants and, in particular, to define what it means for constants to be equivalent. The main contributions are as follows:

- We show that a relational knowledge base is fundamentally defined by its induced equivalence relation over tuples of constants by establishing an isomorphism between logic programs that translate knowledge bases and refinement relations over equivalence classes.
- We characterise acyclic logic programs as morphisms in a category where relational knowledge bases act as objects. We show how this new category relates to the category of partitions of a set.
- We formalise the semantics of an arbitrary formula as a composition of functions between sets, where each atom is interpreted as a composition of two functions: a new type of function based on the idea of *arrangements with repetition* in combinatorics, and the predicate itself.

3.1 Related Work

While equivalence between logic programs has received a fair amount of attention [41, 51, 21], the idea to formally consider equivalence of constants seems to be new. However, informal observations that it is useful to identify sets of constants with which the program's behaviour is identical can be seen in recent literature on domain abstractions [33].

There have been other categorical approaches to logic programming where logic programs are characterised as indexed categories [37], indexed monoidal categories [10], and τ-categories [24]. We provide a fresh perspective by thinking of logic programs as transformations between Herbrand universes, and so programs become morphisms rather than categories.

Rajan and Muhammed [59] define the category of partitions of a set where objects are defined as all partitions of a set (except the identity partition that partitions the set into one cell), and there is an arrow from object A to object B if, for every cell a of partition A, one can find a cell b of partition B such that $a \subseteq b$. As equivalences are 'equivalent' to partitions by the fundamental theorem of equivalence relations [20], it is likely that one can establish a formal connection between the category of partitions and the category of knowledge bases and logic programs.

4 Using Constraint Programming to Generate Random Logic Programs

We present a novel approach to generate random logic programs using constraint programming. The constraint programming approach to the problem has a major advantage in that one can easily add additional conditions for the generated programs. Some of these conditions can be expressed using constraints native to the constraint solver Choco [55], while novel propagation and entailment checking algorithms have been developed for ensuring independence and/or conditional independence amongst predicates. A constraint programming approach also benefits from many years of research into efficient constraint propagation algorithms which makes it likely that our model can generate larger programs relatively quickly.

The main disadvantage of this approach is that we have no control over what probability distribution is being sampled. However, we can avoid generating programs that are very similar to each other by regularly restarting the solver, which makes it forget decisions that have already been made and guess/infer new ones. Preliminary experiments show that restarting after every solution might be time-consuming, but regular restarts are still feasible.

While the model generates logic programs, it can be easily extended to generate probabilistic logic programs by adding probabilities to clauses and an additional constraint for avoiding negative cycles in the program.

4.1 Related Work

Previous approaches. Logic programs have been generated before. Some approaches are quite restrictive, e.g., restricted to clauses with only two literals [50], or to clauses of the form $a \leftarrow \neg b$ [79], but others are more expressive [77], e.g., defining a program only by the (maximum) number of atoms in the body and the total number of rules [83, 82]. Our advantages over previous work are twofold: our model can generate different syntactic representations of the same underlying probability distribution (which can be a blessing or a curse depending on the needs of the user), and our model allows us to integrate additional constraints on the program in an efficient (i.e., non-brute-force) manner.

Inference algorithms and their evaluation. How confidently can we claim that an algorithm works well if it is only tested on a couple types of problems? Perhaps the 'inferior' algorithm is actually better in some specific circumstances. Perhaps there are clearly identifiable types of programs that make every algorithm default to exponential behaviour that could be easily overcome with the right strategy. At present, most inference algorithms for probabilistic (logic) programs are only evaluated on 1–4 different problems: sometimes just a single network [36, 44, 35], sometimes two [69, 4] or even four networks [74] coming from a range of areas such as social networks and citation/genetic/biological data sets. In order to better understand the strengths and weaknesses of these algorithms, along with real data they should also be evaluated on a range of synthetic problems that accurately represent the potential complexities that have to be handled by a well-designed solver.

Complexity and empirical hardness. Research in computer science has a long history of both theoretical and empirical approaches towards characterising the difficulty of a problem irrespectively of its solutions. In both cases, many attempts have been made to discover parameters that can be used to predict or ensure the difficulty of the problem. On the theory side, this has resulted in an entire subfield of complexity theory called parameterised complexity [17] and well-known parameters such as clique-width [12, 78] and treewidth [16]. On the empirical side, parameters were used to identify phase transitions—peaks in the empirical hardness of decision problems between problems that are so 'easy' that the answer is obviously 'yes' and problems that are so 'easy' that the answer is obviously 'no' [7]. This approach has been successfully applied on a wide range of problems, ranging from the travelling salesman [25] to voting systems [76]. More recently, empirical approaches to hardness estimation have used machine learning techniques to unveil a more detailed picture of the hardness landscape [40] and tackled optimisation problems by running billions of experiments to eliminate noise and randomness [45]. Being able to generate benchmarking instances for a problem is the crucial first step towards a better understanding of empirical hardness and will allow us to characterise which (probabilistic) logic programs are easy, and which ones are hard.

Data generation. As probabilistic logic programs have been used to generate random data [18], one could combine random data generation with random program generation to generate random relational data that is generalisable across generative models and only satisfies some basic independence conditions. The results could unveil important biases in how the programs are generated as well as biases inherent in encoding a probability distribution as a probabilistic logic program.

Learning. Learning the structure of a probabilistic logic program from data remains a challenging problem. While some approaches only consider ground programs [62, 61], and some consider the problem of identifying a near-optimal subset of clauses from an existing program [56], there have been no successful attempts to tackle the problem in its full generality. Generating suitable structures and testing them against the data would serve as a baseline algorithm for structure learning.

5 Predicate Invention for Probabilistic Logic Programs

Different syntactic representations of the same program can certainly differ in interpretability. This is especially important when the program is learned from data. Can they differ in inference speed? How much can a program be transformed without changing any of its answers to a set of relevant questions?

This project is most closely related to the initial topic of the proposal and examines one particular type of transformation, namely creating a new predicate that can replace a formula. We propose a number of transformations (depending on the underlying formula), prove their correctness, and examine their empirical implications and composability.

5.1 Related Work

This project is related to predicate invention as it is known in the context of inductive logic programming and meta-interpretive learning [48, 49], but there are several key differences. First, predicate invention is typically considered in the context of learning a first-order representation from data [38, 47], while my work invents new predicates in order to transform an already existing program. Second, situating the work in probabilistic logic means that my arguments for the correctness of the transformations are expressed primarily in a probabilistic rather than logical setting.

Besold et al. [68] identify an issue with predicate invention: the newly invented predicate is often left without an interpretable name. In our setting, however, this is less of a problem because the new predicate can be named after its underlying structure (as long as it is not too complicated), e.g., the new predicate meant to replace $\mathsf{father}(X, Y) \lor \mathsf{mother}(X, Y)$, for lack of a better automatically-identifiable alternative, could be called $\mathsf{fatherORmother}(X, Y)$.

The idea to use predicate invention for interpretability is similar to the idea of splitting a logic program into a 'top' part and a 'bottom' part [42, 34]. In both cases, the goal is to identify a high-level structure that is mostly independent from the low-level details, as observed from the perspective of either interpretability or inference.

6 Methodology

6.1 Theory Paper: On the Equivalence of Constants in Logic Programs (WP1)

While the main results of the paper have already been developed, they need to be revised to ensure a sufficient level of mathematical rigour, good (and well-explained) use of terminology, an inviting and understandable explanation, and a convincing motivation for why the results are not only interesting but also impactful and valuable. **Duration:** two months. **Deliverables:** a paper submitted for publication.

6.2 Using Constraint Programming to Generate Random Logic Programs (WP2)

While a custom propagation algorithm for independence has already been implemented, it needs to be extended to support an arbitrary number of predicates. The algorithm for conditional independence, while designed, still needs to be implemented and tested. As this would be useful for many applications, there should also be a constraint to ensure that the model does not generate two or more programs with different syntactic expressions that are also clearly logically equivalent. This could be achieved by restricting the format of each clause.

The model should be primarily evaluated on the basis of its speed as measured by the time it takes to produce the first program and the time between subsequent programs as a function of various parameters of the model. Having generated a set of programs, we can then further motivate the work by running multiple inference algorithms on a range of synthetic problems and outlining the types of programs that cause difficulty to all algorithms and the types of programs that are easy for some but difficult for others. **Duration:** six months. **Deliverables:** a software product as well as a paper describing the model, empirical results, and observations.

6.3 Predicate Invention: Implementation and Theory (WP3)

The predicate invention project, while under way, requires significantly more work on both the theoretical and the empirical aspects:

- A number of important theorems have been defined but still need to be proven.
- When looking for formulas that could be abstracted, the algorithm should take into account rules such as De Morgan's laws.
- While the theory is formulated to work on any number of predicates, the implementation is only able to handle two predicates at a time.

Duration: three months. **Deliverables:** a list of theorems with proofs and software that can execute the transformations suggested by the theorems.

6.4 Greedy Algorithms (WP4)

At this point, we can reason about applying multiple abstraction rules in sequence. We will aim to answer two key questions:

- How to choose which abstraction rule should be applied first?
- When is the correct time to stop simplifying things?

Exploring different answers to these questions will result in a set of abstraction algorithms. As we expect most abstraction rules to commute with one another, making the order the rules are applied in immaterial, we will focus on *greedy* algorithms, where we choose which abstraction rule should be applied on a model based solely on the current state of the model.

Each algorithm will take a model, a description of a set of queries that need to be supported, and an indication of how much loss in precision (if any) the user is willing to tolerate. We will explore a variety of heuristics that establish preferences over which abstraction rule should be applied first as well as termination conditions. For instance, we could always pick an abstraction rule that results in a model with highest entropy (similar ideas have been very successful in reinforcement learning [84]), or prefer rules that have little effect on precision. While a natural termination condition could be an inability to apply an abstraction rule without losing too much precision, the process could also be terminated because either the loss in precision or the expected running time of applying the abstraction is too high for the predicted benefits in inference speed. **Duration:** six months. **Deliverables:** implementations of the algorithms as well as one or more papers detailing the reasoning behind design decisions, observed empirical performance, and the differences in inference speed before and after running each algorithm.

6.5 Abstraction During Inference (WP5)

There is no reason to limit abstraction to the intermediate stage between constructing (or learning) a model and performing inference. Thus, we will investigate how the state-of-the-art inference algorithm for our chosen FOPI model can benefit from performing abstraction steps throughout its execution. For example, a common way to compute a conditional probability $P(Q \mid E)$ inside a model Δ using WMC is by calculating the WMC of $\Delta \wedge Q \wedge E$ and $\Delta \wedge E$ separately, and then dividing one by the other [6]. It is likely that the two computations would benefit from different abstractions. **Duration:** six months. **Deliverables:** an implementation of a new inference algorithm with abstraction, and a report describing its design and performance.

6.6 Abstraction in Learning (WP6)

A learned model is, in a way, an abstraction of the training data. Performing abstraction afterwards is suboptimal, since the algorithm would attempt to stay close to the learned model rather than the underlying data itself. While learning algorithms already measure goodness of fit, the measure will have to be combined with a meaningful measure of simplicity in order to provide a reasonable balance between the two. The abstraction rules themselves will need to be rethought in the context of data, as abstraction ought not to be added at the end as an extra step, but rather a fully integrated part of the learning process that concerns itself primarily with a meaningful and accurate representation of data. The abstraction ideas developed in previous work packages will help ensure that the learned models are both more explainable and transferable to new kinds of data. **Duration:** six months. **Deliverables:** an implementation of a learning algorithm with integrated abstraction, and a report describing the rationale for its design as well as observed performance.

7 Measurable Outcomes

- A thoroughly-described landscape of probabilistic logic programs, identifying programs that are challenging to some or all of the inference algorithms. (WP2).
- Demonstrated increase in exact and approximate inference speed (WP5).
- Demonstrated improvement in models learned from data in terms of simplicity and transferability without (significant) loss in precision (WP6).

8 Risks

Creating a new algorithm always carries a risk that the algorithm may not perform well compared to what has already been achieved. Fortunately, previous work, which features a very limited version of abstraction, shows promising results in terms of reduction in inference speed [32]. We are confident that we can observe similarly successful results with a much broader range of abstractions. In any case, the project will result in valuable theoretical contributions regardless of how well the ideas perform in practice.

Another risk is related to integrating abstraction into inference. Namely, it might take too much time to create the abstraction compared to the time saved during inference. Even if this turns out to be the case, the developed abstraction algorithms would still be useful for models that are constructed once and used to perform many inferences. However, the observed reductions in inference speed seem to reduce the complexity of the task [32], while at least some abstraction rules can definitely be implemented in linear time. This suggests that with a big enough model the gains in inference speed should eventually overtake the time taken to construct the abstraction.

9 Impact and National Importance

National Importance. Our proposal covers an EPSRC growth area for statistics and applied probability as well as several maintenance areas such as AI technologies, logic and combinatorics, and theoretical computer science. Furthermore, our work will contribute to areas such as big data (as more scalable inference will allow expressive probabilistic models to be used with more data) as well as other scientific disciplines (as both abstraction and richness of representation are important topics in social sciences and medicine). Moreover, abstraction for FOPI models is a new yet promising research area [32] which is likely to see many major developments.

Impact. The survey will highlight the weaknesses of current approaches and direct future research towards open problems. Furthermore, many of the basic ideas behind abstraction for a particular model are likely to be transferable to many others, perhaps even inspiring a unifying theory behind all representations. Moreover, making the models more efficient and explainable should also make them more attractive to a larger user base, both academic and industrial.

References

[1] Adadi, A., and Berrada, M. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access 6* (2018), 52138–52160.

[2] Belle, V. Abstracting probabilistic relational models. *CoRR abs/1810.02434* (2018).

[3] Bransford, J. D., and Franks, J. J. The abstraction of linguistic ideas. *Cognitive Psychology 2*, 4 (1971), 331 – 350.

[4] Bruynooghe, M., Mantadelis, T., Kimmig, A., Gutmann, B., Vennekens, J., Janssens, G., and Raedt, L. D. Problog technology for inference in a probabilistic first order logic. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings* (2010), H. Coelho, R. Studer, and M. J. Wooldridge, Eds., vol. 215 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 719–724.

[5] Chandra, S., Sahs, J., Khan, L., Thuraisingham, B. M., and Aggarwal, C. C. Stream mining using statistical relational learning. In *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014* (2014), R. Kumar, H. Toivonen, J. Pei, J. Z. Huang, and X. Wu, Eds., IEEE Computer Society, pp. 743–748.

[6] Chavira, M., and Darwiche, A. On probabilistic inference by weighted model counting. *Artif. Intell. 172*, 6-7 (2008), 772–799.

[7] Cheeseman, P. C., Kanefsky, B., and Taylor, W. M. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991* (1991), J. Mylopoulos and R. Reiter, Eds., Morgan Kaufmann, pp. 331–340.

[8] Clarke, E. M., Grumberg, O., Jha, S., Lu, Y., and Veith, H. Counterexample-guided abstraction refinement. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings* (2000), E. A. Emerson and A. P. Sistla, Eds., vol. 1855 of *Lecture Notes in Computer Science*, Springer, pp. 154–169.

[9] Clarke, E. M., Grumberg, O., and Long, D. E. Model checking and abstraction. *ACM Trans. Program. Lang. Syst. 16*, 5 (1994), 1512–1542.

[10] Corradini, A., and Asperti, A. A categorical model for logic programs: Indexed monoidal categories. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)* (1992), Springer, pp. 110–137.

[11] Côrte-Real, J., Dutra, I., and Rocha, R. On applying probabilistic logic programming to breast cancer data. In *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers* (2017), N. Lachiche and C. Vrain, Eds., vol. 10759 of *Lecture Notes in Computer Science*, Springer, pp. 31–45.

[12] Courcelle, B., Engelfriet, J., and Rozenberg, G. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci. 46*, 2 (1993), 218–270.

[13] de Salvo Braz, R., Amir, E., and Roth, D. A survey of first-order probabilistic models. In *Innovations in Bayesian Networks: Theory and Applications*, D. E. Holmes and L. C. Jain, Eds., vol. 156 of *Studies in Computational Intelligence.* Springer, 2008, pp. 289–317.

[14] Delaney, B., Fast, A. S., Campbell, W. M., Weinstein, C. J., and Jensen, D. D. The application of statistical relational learning to a database of criminal and terrorist activity. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA* (2010), SIAM, pp. 409–417.

[15] den Broeck, G. V., Taghipour, N., Meert, W., Davis, J., and Raedt, L. D. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011* (2011), T. Walsh, Ed., IJCAI/AAAI, pp. 2178–2185.

[16] Diestel, R. *Graph theory*, 3 ed., vol. 173 of *Graduate texts in mathematics.* Springer, 2005.

[17] Downey, R. G., and Fellows, M. R. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999.

[18] Dries, A. Declarative data generation with problog. In *Proceedings of the Sixth International Symposium on Information and Communication Technology, Hue City, Vietnam, December 3-4, 2015* (2015), H. Q. Thang, L. A. Phuong, L. D. Raedt, Y. Deville, M. Bui, T. T. D. Linh, N. Thi-Oanh, D. V. Sang, and N. B. Ngoc, Eds., ACM, pp. 17–24.

[19] Dries, A., Kimmig, A., Davis, J., Belle, V., and Raedt, L. D. Solving probability problems in natural language. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017* (2017), C. Sierra, Ed., ijcai.org, pp. 3981–3987.

[20] Dummit, D. S., and Foote, R. M. *Abstract algebra*, vol. 3. Wiley Hoboken, 2004.

[21] Eiter, T., and Fink, M. Uniform equivalence of logic programs under the stable model semantics. In Palamidessi [52], pp. 224–238.

[22] Elidan, G., Kersting, K., and Ihler, A. T., Eds. *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017* (2017), AUAI Press.

[23] Erol, K., Hendler, J. A., and Nau, D. S. Complexity results for HTN planning. *Ann. Math. Artif. Intell. 18*, 1 (1996), 69–93.

[24] Finkelstein, S. E., Freyd, P. J., and Lipton, J. Logic programming in tau categories. In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers* (1994), L. Pacholski and J. Tiuryn, Eds., vol. 933 of *Lecture Notes in Computer Science*, Springer, pp. 249–263.

[25] Gent, I. P., and Walsh, T. The TSP phase transition. *Artif. Intell. 88*, 1-2 (1996), 349–358.

[26] Gentner, D., and Hoyos, C. Analogy and abstraction. *Topics in Cognitive Science 9*, 3 (2017), 672–693.

[27] Giunchiglia, F., and Walsh, T. A theory of abstraction. *Artif. Intell. 57*, 2-3 (1992), 323–389.

[28] Gogate, V., and Domingos, P. M. Probabilistic theorem proving. *Commun. ACM 59*, 7 (2016), 107–115.

[29] Gordon, A. D., Henzinger, T. A., Nori, A. V., and Rajamani, S. K. Probabilistic programming. In *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014* (2014), J. D. Herbsleb and M. B. Dwyer, Eds., ACM, pp. 167–181.

[30] Hayes-Roth, F., Waterman, D., and Lenat, D. *Building expert systems*. Teknowledge series in knowledge engineering. Addison-Wesley, 1983.

[31] Henzinger, T. A., Jhala, R., Majumdar, R., and Sutre, G. Lazy abstraction. In *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002* (2002), J. Launchbury and J. C. Mitchell, Eds., ACM, pp. 58–70.

[32] Holtzen, S., den Broeck, G. V., and Millstein, T. D. Sound abstraction and decomposition of probabilistic programs. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (2018), J. G. Dy and A. Krause, Eds., vol. 80 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 2004–2013.

[33] Holtzen, S., Millstein, T. D., and den Broeck, G. V. Probabilistic program abstractions. In Elidan et al. [22].

[34] Ji, J., Wan, H., Huo, Z., and Yuan, Z. Splitting a logic program revisited. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA* (2015), B. Bonet and S. Koenig, Eds., AAAI Press, pp. 1511–1517.

[35] Kimmig, A., Costa, V. S., Rocha, R., Demoen, B., and Raedt, L. D. On the efficient execution of problog programs. In *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings* (2008), M. G. de la Banda and E. Pontelli, Eds., vol. 5366 of *Lecture Notes in Computer Science*, Springer, pp. 175–189.

[36] Kimmig, A., Demoen, B., Raedt, L. D., Costa, V. S., and Rocha, R. On the implementation of the probabilistic logic programming language problog. *TPLP 11*, 2-3 (2011), 235–262.

[37] Kinoshita, Y., and Power, A. J. A fibrational semantics for logic programs. In *Extensions of Logic Programming, 5th International Workshop, ELP'96, Leipzig, Germany, March 28-30, 1996, Proceedings* (1996), R. Dyckhoff, H. Herre, and P. Schroeder-Heister, Eds., vol. 1050 of *Lecture Notes in Computer Science*, Springer, pp. 177–191.

[38] Kramer, S. Predicate invention: A comprehensive view. *Rapport technique OFAI-TR-95-32, Austrian Research Institute for Artificial Intelligence, Vienna* (1995).

[39] Levy, A., and Bechtel, W. Abstraction and the organization of mechanisms. *Philosophy of Science 80*, 2 (2013), 241–261.

[40] Leyton-Brown, K., Hoos, H. H., Hutter, F., and Xu, L. Understanding the empirical hardness of *NP*-complete problems. *Commun. ACM 57*, 5 (2014), 98–107.

[41] Lifschitz, V., Pearce, D., and Valverde, A. Strongly equivalent logic programs. *ACM Trans. Comput. Log. 2*, 4 (2001), 526–541.

[42] Lifschitz, V., and Turner, H. Splitting a logic program. In *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994* (1994), P. V. Hentenryck, Ed., MIT Press, pp. 23–37.

[43] Lloyd, J. W. *Foundations of Logic Programming, 2nd Edition.* Springer, 1987.

[44] Mantadelis, T., and Janssens, G. Nesting probabilistic inference. *CoRR abs/1112.3785* (2011).

[45] McCreesh, C., Pettersson, W., and Prosser, P. Understanding the empirical hardness of random optimisation problems. In *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings* (2019), T. Schiex and S. de Givry, Eds., vol. 11802 of *Lecture Notes in Computer Science*, Springer, pp. 333–349.

[46] Monniaux, D. An abstract Monte-Carlo method for the analysis of probabilistic programs. In *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001* (2001), C. Hankin and D. Schmidt, Eds., ACM, pp. 93–101.

[47] Muggleton, S. Predicate invention and utilization. *J. Exp. Theor. Artif. Intell. 6*, 1 (1994), 121–130.

[48] Muggleton, S. H. Meta-interpretive learning: Achievements and challenges (invited paper). In *Rules and Reasoning - International Joint Conference, RuleML+RR 2017, London, UK, July 12-15, 2017, Proceedings* (2017), S. Costantini, E. Franconi, W. V. Woensel, R. Kontchakov, F. Sadri, and D. Roman, Eds., vol. 10364 of *Lecture Notes in Computer Science*, Springer, pp. 1–6.

[49] Muggleton, S. H., Lin, D., and Tamaddoni-Nezhad, A. Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Machine Learning 100*, 1 (2015), 49–73.

[50] Namasivayam, G., and Truszczynski, M. Simple random logic programs. In *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings* (2009), E. Erdem, F. Lin, and T. Schaub, Eds., vol. 5753 of *Lecture Notes in Computer Science*, Springer, pp. 223–235.

[51] Oikarinen, E., and Janhunen, T. Modular equivalence for normal logic programs. In *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings* (2006), G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, Eds., vol. 141 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 412–416.

[52] Palamidessi, C., Ed. *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings* (2003), vol. 2916 of *Lecture Notes in Computer Science*, Springer.

[53] Penkov, S., and Ramamoorthy, S. Explaining transition systems through program induction. *CoRR abs/1705.08320* (2017).

[54] Poole, D. First-order probabilistic inference. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003* (2003), G. Gottlob and T. Walsh, Eds., Morgan Kaufmann, pp. 985–991.

[55] Prud'homme, C., Fages, J.-G., and Lorca, X. *Choco Documentation.* TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017.

[56] Raedt, L. D., Kersting, K., Kimmig, A., Revoredo, K., and Toivonen, H. Revising probabilistic prolog programs. In *Inductive Logic Programming, 16th International Conference, ILP 2006, Santiago de Compostela, Spain, August 24-27, 2006, Revised Selected Papers* (2006), S. Muggleton, R. P. Otero, and A. Tamaddoni-Nezhad, Eds., vol. 4455 of *Lecture Notes in Computer Science*, Springer, pp. 30–33.

[57] Raedt, L. D., Kersting, K., Natarajan, S., and Poole, D. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2016.

[58] Raedt, L. D., Kimmig, A., and Toivonen, H. ProbLog: A probabilistic prolog and its application in link discovery. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007* (2007), M. M. Veloso, Ed., pp. 2462–2467.

[59] Rajan, A. R., and Muhammed, P. A. A. Normal category of partitions of a set. *arXiv preprint arXiv:1509.02888* (2015).

[60] Richardson, M., and Domingos, P. M. Markov logic networks. *Machine Learning 62*, 1-2 (2006), 107–136.

[61] Riguzzi, F. ALLPAD: approximate learning of logic programs with annotated disjunctions. *Machine Learning 70*, 2-3 (2008), 207–223.

[62] Riguzzi, F., et al. Learning ground problog programs from interpretations. In *Proceedings of the 6th International Workshop on Multi-relational Data Mining (MRDM07)* (2007), pp. 105–116.

[63] Robinson, J. A., and Voronkov, A., Eds. *Handbook of Automated Reasoning (in 2 volumes).* Elsevier and MIT Press, 2001.

[64] Rubenstein, P. K., Weichwald, S., Bongers, S., Mooij, J. M., Janzing, D., Grosse-Wentrup, M., and Schölkopf, B. Causal consistency of structural equation models. In Elidan et al. [22].

[65] Saitta, L., and Zucker, J.-D. *Abstraction in artificial intelligence and complex systems*, vol. 456. Springer, 2013.

[66] Sakhanenko, N. A., and Galas, D. J. Probabilistic logic methods and some applications to biology and medicine. *Journal of Computational Biology 19*, 3 (2012), 316–336.

[67] Sato, T. A statistical learning method for logic programs with distribution semantics. In *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995* (1995), L. Sterling, Ed., MIT Press, pp. 715–729.

[68] Schmid, U., Zeller, C., Besold, T. R., Tamaddoni-Nezhad, A., and Muggleton, S. How does predicate invention affect human comprehensibility? In *Inductive Logic Programming - 26th International Conference, ILP 2016, London, UK, September 4-6, 2016, Revised Selected Papers* (2016), J. Cussens and A. Russo, Eds., vol. 10326 of *Lecture Notes in Computer Science*, Springer, pp. 52–67.

[69] Shterionov, D. S., Kimmig, A., Mantadelis, T., and Janssens, G. DNF sampling for problog inference. *CoRR abs/1009.3798* (2010).

[70] Sreedharan, S., Srivastava, S., and Kambhampati, S. Hierarchical expertise level modeling for user specific contrastive explanations. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.* (2018), J. Lang, Ed., ijcai.org, pp. 4829–4836.

[71] Thon, I., Landwehr, N., and Raedt, L. D. A simple model for sequences of relational state descriptions. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II* (2008), W. Daelemans, B. Goethals, and K. Morik, Eds., vol. 5212 of *Lecture Notes in Computer Science*, Springer, pp. 506–521.

[72] Thon, I., Landwehr, N., and Raedt, L. D. Stochastic relational processes: Efficient inference and applications. *Machine Learning 82*, 2 (2011), 239–272.

[73] Venugopal, D. Advances in inference methods for Markov logic networks. *IEEE Intelligent Informatics Bulletin 18*, 2 (2017), 13–19.

[74] Vlasselaer, J., den Broeck, G. V., Kimmig, A., Meert, W., and Raedt, L. D. Anytime inference in probabilistic logic programs with tp-compilation. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015* (2015), Q. Yang and M. J. Wooldridge, Eds., AAAI Press, pp. 1852–1858.

[75] Vlasselaer, J., and Meert, W. Statistical relational learning for prognostics. In *Proceedings of the 21st Belgian-Dutch Conference on Machine Learning* (2012), pp. 45–50.

[76] Walsh, T. Where are the really hard manipulation problems? the phase transition in manipulating the veto rule. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009* (2009), C. Boutilier, Ed., pp. 324–329.

[77] Wang, K., Wen, L., and Mu, K. Random logic programs: Linear model. *TPLP 15*, 6 (2015), 818–853.

[78] Wanke, E. k-nlc graphs and polynomial algorithms. *Discrete Applied Mathematics 54*, 2-3 (1994), 251–266.

[79] Wen, L., Wang, K., Shen, Y., and Lin, F. A model for phase transition of random answer-set programs. *ACM Trans. Comput. Log. 17*, 3 (2016), 22:1–22:34.

[80] Yang, S., Korayem, M., AlJadda, K., Grainger, T., and Natarajan, S. Application of statistical relational learning to hybrid recommendation systems. *CoRR abs/1607.01050* (2016).

[81] Zhang, X., Si, X., and Naik, M. Combining the logical and the probabilistic in program analysis. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2017, Barcelona, Spain, June 18, 2017* (2017), T. Shpeisman and J. Gottschlich, Eds., ACM, pp. 27–34.

[82] Zhao, Y. *Answer set programming: sat based solver and phase transition.* PhD thesis, Hong Kong University of Science and Technology (Hong Kong), 2004.

[83] Zhao, Y., and Lin, F. Answer set programming phase transition: A study on randomly generated programs. In Palamidessi [52], pp. 239–253.

[84] Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008* (2008), D. Fox and C. P. Gomes, Eds., AAAI Press, pp. 1433–1438.