Systems AI: A Declarative Learning Based Programming Perspective

Parisa Kordjamshidi[a,b], **Dan Roth**[c], **Kristian Kersting**[d]
[a] Tulane University, CS Dept., USA
[b] Florida Institute for Human and Machine Cognition, USA
[c] University of Pennsylvania, Dept. of CIS, USA
[d] TU Darmstadt, CS Dept. & Centre for Cognitive Science, Germany
pkordjam@tulane.edu, danroth@seas.upenn.edu, kersting@cs.tu-darmstadt.de

Abstract

Data-driven approaches are becoming dominant problem-solving techniques in many areas of research and industry. Unfortunately, current technologies do not make it easy to use them for application experts who are not fluent in machine learning technologies nor for machine learning experts who aim at testing ideas on real-world data and evaluate those as a part of an end-to-end system. We review key efforts made by various AI communities to provide languages for high-level abstractions over learning and reasoning techniques needed for designing complex AI systems. We classify the existing frameworks based on the techniques as well as the data and knowledge representations they use, provide a comparative study of the way they address the challenges of programming real-world applications, and highlight some shortcomings and future directions.

1 Introduction

Multiple research disciplines, from cognitive sciences to biology, finance, physics, and the social sciences, as well as many companies, believe that data-driven and "intelligent" solutions are necessary in order to solve many of their key problems. Unfortunately, current machine learning (ML) and AI technologies are not sufficiently democratized—they do not provide domain experts who are not AI experts easy ways to develop applications; rather, they provide cumbersome solutions along multiple dimensions. Moreover, even for AI experts who aim at evaluating their new ideas and algorithms, there is a need for heavy experimentation and evaluation efforts. This is due to the fact that AI has entered to the era of serious usage of basic research to solve real-world problems.

Building complex AI systems requires extensive programming skills and the ability to work with various reasoning and learning paradigms and techniques at a rather low level of abstraction. It also requires extensive experimental exploration for model selection, feature selection, and parameter tuning due to lack of theoretical understanding that could be used to automatically abstracting these subtleties. Conventional programming languages and software engineering paradigms have also not been designed to support challenges faced by users of AI Systems, such as dealing with messy, real-world data at the right level of abstraction.

Thus, there is a need for innovative paradigms that seamlessly support embedded trainable models, abstracting away most low level details, and that facilitate reasoning with respect to these at the right level of abstraction. We need to enrich existing frameworks with capabilities for *learning based programming* [Roth, 2005] and for designing complex AI systems: (1) Easy interaction with raw, heterogeneous data; (2) High-level and intuitive abstractions for specifying the requirements of the application; (3) Intuitive means to specify the data and domain knowledge and express uncertainties; (4) Access to various learning, inference and reasoning techniques; (5) Ability to reuse, combine and chain models and perform flexible inference on complex models/pipelines.

Indeed, there have been various AI proposals to address these problems, most notably Probabilistic (logic) programming (P(L)P), Logical Programming (LP), Constrained Conditional Models (CCM) and Statistical relational learning/AI (SRL/StarAI). Most of them aim at learning over probabilistic structures and exploiting knowledge in learning. Recently, Deep Learning (DL) tools have created easy to use abstractions for programming model configurations of deep neural architectures. However, even with these frameworks, one still needs deep mastery of ML and AI techniques and methodologies in order to engineer AI systems; this knowledge far exceed what most application programmers have.

To help close this gap and facilitate making progress with AI technologies, we survey[1] efforts made in this direction. We emphasize the need to keep some fundamental declarative ideas such as first-order query languages, knowledge representation and reasoning techniques, database management systems (DBMS), and deductive databases (DDB), but place them within ML formalisms, integrated with programming languages and software development techniques as a way to address complex real-world problems that require both learning and reasoning models. Our goal is to highlight the importance of integrating multiple "old" ideas into a new paradigm of *Declarative Learning Based Programming* and the necessity of performing principled research to investigate the fundamental requirements from languages, representations and

[1] The short survey is necessarily incomplete; we apologize to all those we failed to cite.

computational models that will facilitate this paradigm.

In the next sections, first we describe the requirements of AI applications, what is addressed, and what is missing. Then we sketch the ideal characteristics of learning based programs and what abstraction layers of various paradigms.

2 AI Applications Requirements

To discuss aforementioned (1)–(5) as well as the existing research and key techniques addressing them, we will use designing an intelligent model solving a simple entity-mention-relation (EMR) extraction task as running example: **Given** an input text as "*Washington works for Associated Press.*", **find** a model that is able to extract the entity types such as people, organizations and locations as well as their relationships, and generate the following labels, $[Washington]_{person}$ $[works\ for]_{worksFor}$ $[Associated\ Press]_{organization}$.

2.1 Interaction with heterogeneous data

For real-world applications, organizing and using data is an essential starting point of a learning based program. The EMR task, e.g., can be helpful to put raw data from emails into a structured database for easy access in other tasks.

Heterogeneous and unstructured data. Many real-world systems need to operate on heterogeneous and unstructured data such as text and images. To put the data in some structure, we need information-extraction modules that could be complex intelligent components themselves. In the EMR task, a primary step to operate on the sentence before any learning is chunking, that is splitting the sentence into a number of phrases such as [Washington][Works For][Associated Press][.]. This is a challenging learning task by itself but also provides a primary structure that classifiers can operate on.

Some of the current research has tried to combine information extraction modules with relational DB systems and use standard querying languages for retrieving the information [Krishnamurthy *et al.*, 2009]. Some systems are designed for processing textual data and provide a regular expression language to directly query from text [Broda *et al.*, 2013]. To facilitate working on unstructured data, there has been efforts in designing unified data structures for processing textual data and preparing tools that can operate on those data structures. A well-known example of such universal data structure is UIMA that can be augmented with NLP tools that provide lexical, syntactic and semantic features [Sammons *et al.*, 2016]. They focus on providing a specific internal representation of raw data (text here) but do not allow for declaration of a data model with an arbitrary structure. Though some of these information extraction systems are equipped with very well designed and efficient query languages such as SystemT [Krishnamurthy *et al.*, 2009].

In one hand, such systems do not address learning and inference, i.e., their functionality is independent from designing learning models. However, they indeed could be used as *sensors* for information extraction in designing learning models. On the other hand, there are many ML tools such as WEKA and newer python and deep learning packages. They provide easy access to learning algorithms, but typically support a flat data structure in a specific format only; one cannot work easily with raw nor structured data. Here is the obvious gap in the existing systems for working on raw data and applying machine learning.

Relational and graph data. In many applications we are facing data with complex structures. Organizing, manipulating and efficient querying from data has been well addressed by relational database management systems based on relational representations and standard query languages. However, these systems are traditionally independent from learning based models, as our EMR task illustrates. We want to learn to extract the entities and relationship and put them into a database for efficient and easy usage.

Providing ML capabilities on top of relational databases has been followed e.g. in the DeepDive [Zhang *et al.*, 2017] while first order logical SQL expressions are grounded and form a Markov logic network or some other relational probabilistic model [De Raedt *et al.*, 2016] to be trained and used for inference and predictions over relational data. Or, one may use black box classifiers based on relational features on top of a datalog-style as e.g. implemented in kLog [Frasconi *et al.*, 2014], a relational logic based learning framework. The possibility of programming for the objective functions by SQL in DBMS environment and forming learning objectives was followed in the LogicBlox [Aref *et al.*, 2015] and RELOOP [Kersting *et al.*, 2017] systems. Another example is the Saul [Kordjamshidi *et al.*, 2015] language, which is equipped with in-memory graph queries which can be directly used in learning models as features or for constructing learning examples. Moreover, the queries can form global structural patterns between inputs as well as outputs [Kordjamshidi *et al.*, 2016; Kordjamshidi *et al.*, 2017].

Feature extraction. One central goal of interaction with data in learning based programs is to be able to define and extract features from various data sources for learning models. Typically, feature engineering includes a) the capability of obtaining low level sensory properties of learning examples, for example the length of a phrase, or the lemma of its contained words, b) the capability of selecting, projecting or joining properties for complex and structured data, c) feature induction, d) feature selection, and e) mapping features from one representation to another. This basically implies that feature extraction is a component that needs to deal with the two aforementioned issues of interaction with raw data putting them into structure and querying from the obtained structure. While there has been research on each of them, a unifying framework as well as programming environment enabling ML is still missing. Specifically, feature extraction approaches can be deterministic such as logical query languages on relational data or can be information extraction tools as described above. For example, we can have all phrases in a relational database and query for all pairs of phrases that have a specific distance between them in the sentence. In NLP, Fextor [Broda *et al.*, 2013] is a tool that provides an internal representation for textual data and provides a library to make queries, like asking the pos-tag of a specific word,

relying on its fixed internal representation. Prior to Fextor, Fex [Cumby and Roth, 2003] viewed feature extraction from a first order knowledge representation perspective. Their formalization is based on description logic where each feature extraction query is answered by logical reasoning.

2.2 High-level abstractions

While the goal of conventional programming is to automate tasks that are step-by-step instructions, the main goal of AI are programs that make intelligent decisions and solve real-world problems. Recall the very first AI problem solvers. They were expert systems modeling the way experts reason and make decision based on a set of logical rules. Programming languages like Lisp and Prolog made programming them easy even for non-expert users, or at least targeted this idea. Only the domain knowledge is programmed in a declarative form; the way the rules are used in a logical reasoning process is hidden from the programmer. This declarative programming paradigm highlights the 'what' and not the 'how'.

Traditional declarative programming often considers programs as the theories of formal logic. But declarative programs could be any high-level specifications of 'what' needs to be done where the 'how' is left to the language's implementation. This being said, all current tools and languages aim at obtaining the right level of abstraction and being declarative in that sense. We distinguish between two types of abstractions, a) *data and domain abstractions* and b) *computational and algorithmic abstractions*.

Current ML[2] and deep learning tools[3,4] have made a huge progress towards being more declarative and independent from algorithms, at least for standard ML problems. Using classical ML libraries, the programmer needs to provide feature vectors and specify a number of parameters only. The programs are written independent from the training algorithms. However, keeping the high-level declarations becomes harder when the data becomes complex and structured as we go beyond predicting a single variable in the output. We need to use additional domain knowledge beyond data items.

Depending on the type of the techniques, various abstractions has been made based on both data and computations: a) Data abstractions based on the dependency structure of variables, b) Data and domain abstractions in terms of logical model of the domain knowledge and c) Computational abstractions based on mathematical functions that form the objective of learning and inference, d) A mixture of data and computational abstractions by representing the model as a procedural partial program. We describe these various perspectives and related implementations.

Dependency structure of the variables. Probabilistic programming languages facilitate the specifications of (in)dependencies. The user declares random variables and their dependency structure and other related parameters such as distributions and density functions. The structure is specified and used declaratively and, ultimately, independent of underlying algorithms for inference and parameter estimation. Reconsider our EMR task. We specify the phrases as random variables after we already have obtained an appropriate representation for them. Then we specify the dependency between each word and its label, or the labels of each word and its adjacent word. Given the data, we can then train the parameters and query probabilities of each label or do MAP inference, etc. Examples of such languages[5] are InferNet [Minka *et al.*, 2014], Figaro [Pfeffer, 2016], AutoBayes [Fischer and Schumann, 2003], and BUGS [Gilks *et al.*, 1994]. Some of them are Turing-complete and support general purpose programs using probabilistic execution traces (Venture [Mansinghka *et al.*, 2014], Angelican [Wood *et al.*, 2014], Church [Goodman *et al.*, 2008], and Pyro[6].

Logical representation of the domain knowledge. Traditional expert systems emphasize world knowledge. We use the term knowledge for the type of information that goes beyond single data items and expresses the relationships between classes of objects in the real-world. This is the kind of information that, for example, first order logical formalisms are able to express. In our EMR example, while the specific linguistic features of each word/phrase is a part of our information about each instance, we can have some higher level knowledge over the sets of phrases. For example, we know that 'if an arbitrary phrase is a person it can not be a location' and that 'if an arbitrary phrase is a person and another arbitrary phrase is a location the relationship can not be married'.

Though the domain knowledge can convey more information compared to a set of data items, it is not always straightforward to account for it in classical learning approaches. Statistical relational learning models and probabilistic logical languages [De Raedt *et al.*, 2016] tackle this issue. Some probabilistic logical models provide the computational models of logical reasoning, using resolution, unification, etc., while the data items are also represented in the same framework, and learning models can be trained based on the data. A typical example is Problog [De Raedt *et al.*, 2007]. Logical representation of the domain knowledge has been used in other frameworks under the umbrella term of SRL models such as MLNs [Domingos and Richardson, 2004] and BLOG [Milch *et al.*, 2005], and PSL models [Broecheler *et al.*, 2010]. However, the logical representations in these frameworks are all grounded and generate data instances that form underlying probabilistic graphical models of various kind. In contrast to the above-mentioned probabilistic logical models, these models do not necessarily consider logical reasoning. Representing domain knowledge along with the data instantiated in deductive databases such as Datalog [Gottlob *et al.*, 1989] and expressing uncertainties in the data has been considered in probabilistic databases [Suciu *et al.*, 2011]. An example of a deductive database with representing uncertainties is ProPPR, [Wang *et al.*, 2014] which has been augmented to learn the probabilities of the facts in the database using neural techniques in TensorLog [Cohen *et al.*, 2017].

[2] scikit-learn.org
[3] www.tensorflow.org
[4] openai.com
[5] probabilistic-programming.org
[6] pyro.ai

Programming the mathematical objective functions. Typical examples of this type of abstraction are deep learning tools. The programmer does not concern specifying the structure of the data and the dependencies between variables but the architecture of the model based on mathematical operators, activation functions, loss functions, etc. [Abadi *et al.*, 2016]. Given the architecture of the operations, which is a computational graph in contrast to a dependency graph, the program would know how to compute the gradients and what procedure to run for training and prediction. If we design the EMR model in this paradigm, we will have a vector representation of each phrase beforehand and then specify the architecture of our model based on multiplications, summations and activations. The idea of mathematical abstractions has been used in other paradigms even in probabilistic programming tools such as WOLFE [Riedel *et al.*, 2014]. Such abstractions have been used in the context of designing structured output prediction models such as SSVM (`www.cs.cornell.edu/people/tj/svm_light/svm_struct.html`) or with search-based inference techniques [Daumé III *et al.*, 2014] where the loss and predict procedures can be written in a few lines of code. However, the end-to-end program has a sequential and imperative rather than declarative form.

Procedural representation of the domain knowledge. When we compare the declarative and imperative paradigms, we refer to the way we express the training and prediction specifications/procedures. However, teaching a machine to perform a task with a sequence of steps may require one to express the procedure of the task as parts of the background knowledge. The imperative task definition is different from an imperative program that hard codes the information about the task when training an objective function. For the same reason, here in this paper, even defining a task procedure subject to the learning is referred to as declaring the procedural domain knowledge. The procedure of a task, which is expressed in an imperative form, could be taken as the declaration of a specific learning model and be connected to some formal semantics with an underlying computation different from the deterministic sequential execution of a set of instructions. We call this also declarative programming since parts of the domain knowledge is expressed procedurally while its execution is not deterministic and depends on the trained models. While this might be terminology only, we believe this perspective is important to broaden the scope of declarative knowledge representation in our context. Given this view, we can call differentiable programs [Bosnjak *et al.*, 2017] also learning based programs. This will be more clarified when we talk about model composition in Sect. 2.5. An example of an imperative program based learning for the EMR task could be a basic if-then-else structure to form a pipeline of decision making, like, if phrase x is a person then check phrase y; if phrase y is a location then check the relationship between x and y; and so on. This specifies a procedure for decision making, though the decisions are based on learning functions. Nevertheless it guides the formulation of a global objective function for learning.

2.3 Representing uncertainty

Most data is uncertain due to noise, missing information or inherent ambiguities. Moving from traditional AI's logical perspective to models that support randomness and probabilities has been triggered by this fact. Statistical and probabilistic learning techniques inherently address the issue of uncertainty and this is reflected in the probabilistic programming and SRL languages [Domingos and Richardson, 2004; Milch *et al.*, 2005; De Raedt *et al.*, 2016]. Conventional programming languages by no means address the issue of uncertainty—a main reason why they can not directly solve real-world problems and help intelligent decision making.

Dealing with uncertainty in a generic problem solving programming paradigm has been addressed in a very limited way. Considering uncertainty when programming for problem solving with Turing-complete capabilities can be seen in the implementations of *probabilistic logical programming* languages [De Raedt *et al.*, 2007; Sato and Kameya, 1997; Eisner, 2008] as well as *probabilistic programming* considering randomness in the execution traces [Goodman *et al.*, 2008; Mansinghka *et al.*, 2014]. In these frameworks, researchers have used a Turing-complete language in the background, which enables performing any arbitrary task, and have enriched it with uncertainty representation to find the best possible output when lacking evidence for finding the exact output of the program. However, the way the uncertainties are interpreted is mapped to a specific formal semantics in the existing languages. Therefore different inference techniques based on various formalisms are often not supported.

The recent idea of differentiable programming also addresses this by using a different type of underlying algorithm, namely that of recurrent neural networks and neural Turing machines [Graves *et al.*, 2014]. Based on this type of techniques, in [Bosnjak *et al.*, 2017] e.g., the sketch of an imperative program is given while the uncertain components of the program are trained given a set of input/output examples. We believe as a future direction there is a need to address the uncertainty and incompleteness in the data and knowledge in its various forms and be able to use various computational models and underlying algorithms.

2.4 Wide range of algorithms

One of the challenges of designing learning based AI systems is the lack of theoretical evidence about the effectiveness of various inference and learning approaches. This issue leaves the programmer with the huge space of experimentation using trail and error. While automatic exploration is an ideal goal and first steps for are promising, see e.g. [Thornton *et al.*, 2013; Pfeffer *et al.*, 2016], representations that can be connected to a variety of algorithms without much engineering is still essentially unexplored. The current programming frameworks mostly support a specific class of algorithms for training and inference. For example, probabilistic programming languages and SRL frameworks are based on inference and learning in probabilistic graphical models, either directed on undirected ones or generic factor graphs. Probabilisitc soft logic considers a PGM too with more scalable algorithms and efficient solutions by forming a convex optimization problem in a continuous space to do inference. LBJava, RELOOP and

Saul map the inference problems under the domain's logical constraints to form integer linear programs and use efficient off the shelf techniques in that area to solve inference. In LBJava and Saul, learning independent models provides the possibility to exploit any arbitrary ML algorithm in the training phase and perform global inference during the prediction.

2.5 Model composition

As we move towards engineering and using AI systems for more and more complex real-world tasks, the ability to reuse, combine and chain models and perform flexible inference on complex models or pipelines of decision making becomes an essential issue for learning based programming. When designing complex models, one main question is how we compose individual models and build more complex ones based on those in the current formalisms. Reconsider our EMR task. We can design a model for classifying entities and a separate model for classifying the relationships. The final EMR model will use them as its building blocks.

The composition language can be a unified language and consistent with basic ML building block declarations. For example, we can form a global objective using the structured output prediction models and do collective classification to solve this problem. However, if we have heterogeneous underlying models based on different techniques, then forming a global objective will not be straightforward and we will have various possibilities for combining models. This issue brings up the question, whether the current tools naturally support composition or should this be an additional language on top of forming learning objectives. If we look at the above mentioned frameworks, we see that the first set of tools for classical ML do not support declarative composition. They rely on the ML and programming expertise of the users to program imperatively the model composition.

Composition in probabilistic programming. Probabilistic programming covers the aspect of composition inherently. Since all involved variables can be declared consistently in one framework and the dependency structure could express the decomposition and composition semantics for learning and inference. However, the way that we compose complex models is limited to expressing more global dependencies and the same dependency structure is used for both training and prediction time. This is not expressive enough for building complex models and pipelines of decision making. For example we might need to check if certain conditions hold and compose arbitrary parts, etc.

Composition in CCMs. When designing constrained conditional models (CCMs) we need to program the two components of local learning declarations and global constraints specifications. The composition can be done consistently as far as it can be formulated by imposing global constraints and building global models in that way. The current implementations based on CCMs [Kordjamshidi *et al.*, 2015; Rizzolo and Roth, 2010] can model pipelines and model composition by considering the learning models as first class objects where their outputs can be used to form new learners and new layers of abstractions. Although, in the frameworks that are designed as libraries of general-purpose languages, the compositions can be made by the programmer, we believe a composition language with well-defined semantics will provide a better way to design complex models end-to-end.

Composition in deep neural models. The deep learning tools rely on general purpose programming environments and the ML and programming skills of users to compose models imperatively. They provide a way for designing single models, though CapsNets [Sabour *et al.*, 2017] recently made a first step towards learning compositions in deep networks.

Diffrentiable programs seen as compositions. While the composition of the trained models is helpful in designing and programming complex models, one new issue arises. Can we parameterize them and in turn learn the composition itself? This is a less established line of research and it is not clear how the structure of the program can be represented and what will be the parameters of the program. Differentiable programs could be seen as an important step into this direction. On one hand, they are related to program synthesis in the sense that we learn a program from inputs and outputs, on the other hand it is related to the idea of learning the parameters of the composition of learning based components. We can provide the structure or the scheme of the program and learn parts of it. This latter perspective should be distinguished from program synthesis though, because the target programs that we learn are not deterministic programs like sorting; they are never complete and just estimate an output given a partial structure and making inference. They are unlikely to be learnable in a full deterministic structure.

3 Declarative Learning based Programming: An Integration Paradigm

While existing paradigms indeed address some of the capabilities (1)-(5), there is still a need to integrate more aspects to design truly smart AI systems that can constantly collect weak signals independently of specific tasks, and relate them on-the-fly to solve a task without supervision [Roth, 2017].

We here argue for a paradigm that highlights the aspect of *learning from data* as a pivot, tries to extend the capabilities of designing intelligent systems around this concept, and address the above-mentioned challenges accordingly by allowing programming to construct complex configurations based on basic learning building blocks.

Abstractions that are independent from computations. The idea of learning based programming which has been initiated in [Roth, 2005] is arguing for the necessity of data abstractions and hiding the algorithmic details and even hiding high-level algorithmic abstractions. Learning is a mapping from one data abstraction layer to another given the data instances, starting from raw data. Therefore user needs to specify the intended abstractions for an application in hand and the system should figure out how to perform the actual mappings. While this abstraction is similar to what is argued

for in logical formalisms, here we are not limited to logical predicates –concept-learners can be represented by arbitrary functions– and the mapping computations are not limited to logical reasoning mechanisms –heterogeneous learners can take the data and learn the mappings.

LBJava [Rizzolo, 2011] was a first implementation of the idea based on CCM [Chang *et al.*, 2012] computational model. Learners are first class objects and the domain knowledge which is also represented in terms of data abstractions can be used by learners to make global and consistent mappings. Saul [Kordjamshidi *et al.*, 2015] has been proposed with a similar computational model and the possibility of join training of global models. Saul is in the form of a library without the data-driven compilation step and it comes with an explicit support for the representation of the data as a graph for relational feature engineering. The data graph representation helps to specify domain concepts and their relationships. Some concepts are connected to sensors abstracting away from raw data and some are concept learners.

Abstractions that facilitate algorithmic coverage. Most of the frameworks mentioned in the previous sections have a limited coverage of algorithms. While some of these are more flexible than others in supporting heterogeneous computational building blocks, training complex configurations with global learning is addressed with one class of techniques in each framework such as probabilistic inference, integer linear programming or dynamic programming and search. When the learning models abstractions are based on the data abstractions in addition to domain knowledge and generic problem specification, then in principle these representations can be connected to various computational models. In contrast, the representations based on computational abstractions (such as deep learning methods) are more bounded to the type of underlying techniques for computations.

Abstractions that help in closing the loop of data to knowledge. Intelligent systems need to evolve over time as they receive more data and knowledge and they find better abstractions of the data, as illustrated e.g. by NELL [Mitchell *et al.*, 2015], the never ended language learning. Representations of the learning models based on the data and knowledge will naturally support feeding the current models (which will be trained concepts) to obtain new abstraction layers. This will also naturally support the model compositions. There will be a direct connection between how we compose models and how we compose real-world concepts. Such abstractions will help closing the loop of moving from data to knowledge and exploiting it to generate new data.

Abstractions that help learning the programs. While the goal of ML is to write programs that can learn to do a task or make a decision, a more ambitious goal would be to learn even those programs structure from the data. From the classical ML perspective this is connected to structure learning, for example, learning the dependency structure of a Bayesian network, or learning features (feature induction) and global constraints by analyzing the data [Bessiere *et al.*, 2017]. In the programming languages community, this problem is very close to program induction from the inputs and outputs. From the classical AI perspective this is also related to inductive logic programming and program induction, see e.g. [Muggleton and De Raedt, 1994], and in the case of not specifying the domain predicates this will be connected to predicate invention [Stahl, 1995]. More expressive than rules are logical programs. They can be seen as a set of rules augmented by global formal semantics for unification, entailment, etc. and will be treated all together rather than independently.

However, a logical programming or a classical structured programming language, even if it is Turing equivalent, still will not be able to solve an AI-complete problem. If we represent the structure of a learning model with these paradigms, we still need to think about the parameters addressing incompleteness and uncertainty for solving problems intelligently.

Other issues from AI and learning based systems perspective. While we focus on the issues related to appropriate and easy to use abstractions and coverage of various formalisms for learning based programs, there will be many issues to be addressed for the AI systems that will be platforms for such declarative languages. The community will need to take a similar considerations into account as database management systems when designing AI systems. There will be necessary to have learning based management systems that can deal with security and privacy of data as well as learning models, scalability of learning and inference, distributed and parallel implementations, concurrency etc. Moreover, there will be new issues such as fairness and explainability to be addressed in AI and learning based management systems.

4 Conclusion

Triggered by the emerging research area of Systems AI—the computational and mathematical modeling of complex AI systems—we provided an overview on *declarative learning based programming* languages as a central component of such a mission. We discussed the related works that can guide designing such a language covering a) the type of abstraction that the make over the data and computations, b) the type of techniques that they cover for learning and reasoning/inference c) the way they address the interaction with data and the issue of incompleteness and uncertainty d) the way that those facilitate designing complex models by composition of simpler models. Most importantly, we pointed to what is missing and the necessity of joining forces to develop an integrated framework for Systems AI.

References

[Abadi *et al.*, 2016] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[Aref *et al.*, 2015] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and implementation of the LogicBlox system. In *Proc. of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1371–1382, 2015.

[Bessiere *et al.*, 2017] Christian Bessiere, Luc De Raedt, Tias Guns, Lars Kotthoff, Mirco Nanni, Siegfried Nijssen, Barry O'Sullivan, Anastasia Paparrizou, Dino Pedreschi, and Helmut Simonis. The inductive constraint programming loop. *IEEE Intelligent Systems*, 32(5):44–52, 2017.

[Bosnjak *et al.*, 2017] M. Bosnjak, T. Rocktäschel, J. Naradowsky, and S. Riedel. Programming with a differentiable forth interpreter. In *Proc. of 34th ICML*, pages 547–556, 2017.

[Broda *et al.*, 2013] Bartosz Broda, Paweł Kedzia, Michał Marcińczuk, Adam Radziszewski, Radosław Ramocki, and Adam Wardyński. Fextor: A feature extraction framework for natural language processing: A case study in word sense disambiguation, relation recognition and anaphora resolution. In *Computational Linguistics*, pages 41–62. Springer, 2013.

[Broecheler *et al.*, 2010] Matthias Broecheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic similarity logic. In *Conference on Uncertainty in Artificial Intelligence*, 2010.

[Chang *et al.*, 2012] Ming-Wei Chang, Lev Ratinov, and Dan Roth. Structured learning with constrained conditional models. *Machine Learning*, 88(3):399–431, 6 2012.

[Cohen *et al.*, 2017] William W. Cohen, Fan Yang, and Kathryn Mazaitis. Tensorlog: Deep learning meets probabilistic dbs. *CoRR*, abs/1707.05390, 2017.

[Cumby and Roth, 2003] C. Cumby and D. Roth. On kernel methods for relational learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

[Daumé III *et al.*, 2014] Hal Daumé III, John Langford, and Stéphane Ross. Efficient programmable learning to search. *CoRR*, abs/1406.1837, 2014.

[De Raedt *et al.*, 2007] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: a probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2468–2473. AAAI Press, 2007.

[De Raedt *et al.*, 2016] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Morgan & Claypool Publishers, 2016.

[Domingos and Richardson, 2004] Perdo Domingos and Matthew Richardson. Markov logic: A unifying framework for statistical relational learning. In *ICML'04 Workshop on Statistical Relational Learning and its Connections to Other Fields*, pages 49–54, 2004.

[Eisner, 2008] Jason Eisner. Dyna: A *non*-probabilistic programming language for probabilistic AI. Extended abstract for talk at the NIPS*2008 Workshop on Probabilistic Programming, 2008.

[Fischer and Schumann, 2003] Bernd Fischer and Johann Schumann. Autobayes: a system for generating data analysis programs from statistical models. *Journal of Functional Programming*, 13(3):483–508, 2003.

[Frasconi *et al.*, 2014] Paolo Frasconi, Fabrizio Costa, Luc De Raedt, and Kurt De Grave. klog: A language for logical and relational learning with kernels. *Artificial Intelligence*, 217:117–143, 2014.

[Gilks *et al.*, 1994] Wally R. Gilks, Ashita Thomas, and David J. Spiegelhalter. A language and program for complex bayesian modelling. *The Statistician*, 43(1):169–177, 1994.

[Goodman *et al.*, 2008] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. In *In UAI*, pages 220–229, 2008.

[Gottlob *et al.*, 1989] G. Gottlob, S. Ceri, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge Data Engineering*, 1:146–166, 03 1989.

[Graves *et al.*, 2014] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[Kersting *et al.*, 2017] Kristian Kersting, Martin Mladenov, and Pavel Tokmakov. Relational linear programming. *Artif. Intell.*, 244, 2017.

[Kordjamshidi *et al.*, 2015] Parisa Kordjamshidi, Hao Wu, and Dan Roth. Saul: Towards declarative learning based programming. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 7 2015.

[Kordjamshidi *et al.*, 2016] Parisa Kordjamshidi, Daniel Khashabi, Christos Christodoulopoulos, Bhargav Mangipudi, Sameer Singh, and Dan Roth. Better call saul: Flexible programming for learning and inference in nlp. In *Proc. of the International Conference on Computational Linguistics (COLING)*, 2016.

[Kordjamshidi *et al.*, 2017] Parisa Kordjamshidi, Sameer Singh, Daniel Khashabi, Christos Christodoulopoulos, Mark Summons, Saurabh Sinha, and Dan Roth. Relational learning and feature extraction by querying over heterogeneous information networks. In *Seventh International Workshop on Statistical Relational AI (StarAI)*, 2017.

[Krishnamurthy *et al.*, 2009] Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. Systemt: A system for declarative information extraction. *SIGMOD Rec.*, 37(4):7–13, March 2009.

[Mansinghka *et al.*, 2014] Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, abs/1404.0099, 2014.

[Milch *et al.*, 2005] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey

Kolobov. BLOG: Probabilistic models with unknown objects. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[Minka *et al.*, 2014] T. Minka, J. M. Winn, J.P. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill. Infer.NET 2.6, 2014. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

[Mitchell *et al.*, 2015] Tom M. Mitchell *et al.* Never-ending learning. In *Proc. of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI)*, pages 2302–2310, 2015.

[Muggleton and De Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.

[Pfeffer *et al.*, 2016] Avi Pfeffer, Brian E. Ruttenberg, and William Kretschmer. Structured factored inference: A framework for automated reasoning in probabilistic programming languages. *CoRR*, abs/1606.03298, 2016.

[Pfeffer, 2016] Avi Pfeffer. *Practical Probabilistic Programming*. Manning Publications, 2016.

[Riedel *et al.*, 2014] S. Riedel, S. Singh, V. Srikumar, T. Rocktäschel, L. Visengeriyeva, and J. Noessner. WOLFE: strength reduction and approximate programming for probabilistic programming. *Statistical Relational Artificial Intelligence*, 2014.

[Rizzolo and Roth, 2010] Nicholas Rizzolo and Dan Roth. Learning based java for rapid development of NLP systems. In *Proc. of the International Conference on Language Resources and Evaluation (LREC)*, Valletta, Malta, 5 2010.

[Rizzolo, 2011] Nicholas Rizzolo. Learning based programming, 2011.

[Roth, 2005] Dan Roth. Learning based programming. *Innovations in Machine Learning: Theory and Applications*, 2005.

[Roth, 2017] Dan Roth. Incidental supervision: Moving beyond supervised learning. In *Proc. of the Conference on Artificial Intelligence (AAAI)*, 2 2017.

[Sabour *et al.*, 2017] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. In *Proc. of the Annual Conference on Neural Information Processing Systems (NIPS)*, pages 3859–3869, 2017.

[Sammons *et al.*, 2016] Mark Sammons, Christos Christodoulopoulos, Parisa Kordjamshidi, Daniel Khashabi, Vivek Srikumar, and Dan Roth. Edison: Feature extraction for nlp, simplified. In *LREC*, 2016.

[Sato and Kameya, 1997] Taisuke Sato and Yoshitaka Kameya. PRISM: a language for symbolic-statistical modeling. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1330–1335, 1997.

[Stahl, 1995] Irene Stahl. The appropriateness of predicate invention as bias shift operation in ILP. *Machine Learning*, 20(1/2):95–117, 1995.

[Suciu *et al.*, 2011] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 2011.

[Thornton *et al.*, 2013] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *Proc. of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 847–855, 2013.

[Wang *et al.*, 2014] William Yang Wang, Kathryn Mazaitis, and William W. Cohen. Proppr: Efficient first-order probabilistic logic programming for structure discovery, parameter learning, and scalable inference. In *Proc. of the 13th AAAI Conference on Statistical Relational AI*, AAAIWS'14-13, pages 133–134. AAAI Press, 2014.

[Wood *et al.*, 2014] Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Proc. of the 17th International conference on Artificial Intelligence and Statistics*, pages 1024–1032, 2014.

[Zhang *et al.*, 2017] Ce Zhang, Christopher Ré, Michael J. Cafarella, Jaeho Shin, Feiran Wang, and Sen Wu. Deepdive: declarative knowledge base construction. *Commun. ACM*, 60(5):93–102, 2017.