SynthLog: A Language for Synthesising Inductive Data Models (Extended Abstract)

Yann Dauxais⋆, Clément Gautrais⋆, Anton Dries⋆, Arcchit Jain, Samuel Kolb, Mohit Kumar, Stefano Teso, Elia Van Wolputte, Gust Verbruggen, and Luc De Raedt

KU Leuven, Department of Computer Science, Celestijnenlaan 200A, Leuven, Belgium
First.Last@cs.kuleuven.be

Abstract. We introduce SynthLog, an extension of the probabilistic logic programming language ProbLog, for synthesising inductive data models. Inductive data models integrate data with predictive and descriptive models, in a way that is reminiscent of inductive databases. SynthLog provides primitives for learning and manipulating inductive data models, it supports data wrangling, learning predictive models and constraints, and probabilistic and constraint reasoning. It is used as the back-end of the automated data scientist approach that is being developed in the SYNTH project. An overview of the SynthLog philosophy and language as well as a non trivial example of its use, is given in this paper.

Keywords: Automated data science, Inductive Databases, Probabilistic programming

1 Introduction

Automated data science has received a lot of attention in the last decade [2], and has been recognized as an important challenge and solutions promise to democratize data science and make it available to non-expert end-users. Most current approaches tackle the problem of automatically constructing the best prediction pipeline [6,7]. These approaches typically target expert end-users, that can understand most of the steps in the pipeline. In contrast, the SYNTH framework wants to democratize data science and make it available to the naive end-user. The central setting in SYNTH is that of autocompletion in spreadsheets [4]. Spreadsheets are used ubiquitously and the autocompletion task consists of predicting the next cell and value that the user wants to fill out, of course, under the assumption that there are sufficient regularities in the data to enable meaningful predictions.

The autocompletion task constitutes the front-end of the SYNTH framework, and it is easy to see how this can be included in spreadsheet software such as Excel. The back-end, however, consists of the SynthLog language that should support the underlying data science processes and components. This includes tools to automate various steps in data science, from data wrangling to predictive modeling and constraint learning. But rather than viewing this as a data science workflow or pipeline, SYNTH has the SynthLog language that allows the knowledgeable user to define and steer the data

⋆ The three authors contributed equally

science process in a declarative manner. It is this language that we briefly introduce and illustrate in the present note. SynthLog builds on the inductive database idea [8] in that we are looking for a small and non-trivial set of primitives that supports data science processes. Rather than building on top of databases [9], however, SynthLog extends the probabilistic programming language ProbLog that already supports deductive and probabilistic inference, learning and (a limited form of) constraint processing, which are all important for data science.

The idea that SynthLog borrows from inductive databases is that it should treat models (such as predictors or constraints) as first class citizens, that is, SynthLog should support manipulating, constructing, using, and learning such models. Indeed, SynthLog should not only allow to handle the inputs and outputs of the data science components, but also to reason about which models should be learned, used or combined for a particular dataset or task. The models will be represented as SynthLog theories, which are essentially ProbLog programs, consisting of a set of probabilistic facts and clauses. Combining data science components then corresponds to performing operations on theories: adding/deleting facts, adding/deleting clauses, and combining theories.

In Section 2, we introduce the main contribution of this paper: the SynthLog language. Then, in Section 3, we present a case-study illustrating how SynthLog can be used to bridge many components of data science: from data wrangling to constraints.

2 Introduction to SynthLog

SynthLog is a language for supporting automated data science processes. It allows to construct and manipulate inductive data models. An *Inductive Data Model (IDM)* consists of 1) a set of data models (DM) that specifies an adequate data structure for the dataset (like a database), and 2) a set of inductive models (IMs), that is, a set of patterns and machine learning models (like classifiers) that have been discovered in the data. While the DM can be used to retrieve information about the dataset and to answer questions about specific data points, the IMs can be used to make predictions, find inconsistencies and redundancies, etc. IDMs integrate data and inductive models in a SynthLog *theory*.

SynthLog is built on top of the ProbLog probabilistic programming language. It essentially assumes that both the data models and the inductive models are ProbLog programs, and allows to refer to and manipulate such models by means of a new ProbLog operator. As SynthLog manipulates both data and inductive models, it borrows ideas from inductive databases that also consider both data and inductive models as first class citizens. For example, SynthLog follows the mantra of inductive databases that requires the closure property to be satisfied [8,3]. In the SynthLog context this means that the result of any operation must be a theory, and thus must be a ProbLog program. At the same time, as each theory is a ProbLog program, SynthLog supports deductive and probabilistic reasoning, a form of answer set programming (through DTProbLog [1]) and machine learning. We first introduce ProbLog on a simple example, and then introduce the notion of a theory.

2.1 ProbLog by example

ProbLog [5] is a probabilistic logic programming language, that extends Prolog by adding probabilistic primitives and inference. Let us take the example (from [5]) of a small social network, where smoking behavior depends on friendship among people.

```
0.4::asthma(X) :- smokes(X).
0.3::smokes(X).
0.2::smokes(X) :- friend(X,Y), smokes(Y).
friend(1,2). friend(2,1). friend(2,4). friend(3,2). friend(4,2).
query(asthma(2)).
```

For example, the first rule states that somebody that smokes has a probability of 40% to have asthma. Likewise, the second rule states that any person has a 30% chance of smoking. The query corresponds to the answer we want to get: we want to know the probability that person 2 has asthma. In this case, the result is 0.15.

As SynthLog extends ProbLog, which extends Prolog, a basic knowledge of Prolog and ProbLog is assumed in the remainder of this paper. For the interested reader, a more detailed presentation of ProbLog is available[1].

2.2 SynthLog Theories

We now extend ProbLog with the notion of a theory. Each theory will consist of a ProbLog program and it will be possible to define theories through the scope operator `':'/2`. For example, the fact `theory(a):knowledge(1)` states that the theory or ProbLog program identified by `theory(a)` contains the fact `knowledge(1)`.

The following SynthLog listing defines various theories:

```
    constraints:(a:-b).
    data:b.
    global:X :- constraints:X; data:X.
    query(global:_).
```

In this case, the clause `a:-b` is defined in the theory `constraints` and is interpreted as `constraints:a :- constraints:b`. Beyond the syntactic sugar allowing to factorize the theory name in each terms in a clause, such representation allows to share constraints between theories and automatically interpret them. In this example, the `global` theory is the union of the `constraints` and `data` theories. `global` contains the fact `b` and the clause `a:-b`. Thus, `global:a` can be inferred.To support the inductive database aspect of SynthLog, and to allow for further manipulating inductive models, theories can be loaded from or stored in a database or file.

2.3 A language for data science

To facilitate the use of SynthLog as a language dedicated to data science, several predicates are introduced to infer properties of relational datasets, build classifiers and learn or apply constraints on theories. SynthLog supports the definition of custom

[1] https://dtai.cs.kuleuven.be/problog/index.html

predicates, that take a theory (i.e. an inductive data model) as input and returns a theory as output. Many tasks fit within that framework: learners typically take data as input to output a model, data wrangling takes data as input and outputs data, applying a predictor requires data and model as input and outputs data. Some of these custom predicates are detailed in the next section.

3 Case study: Auto-completion

Table 1. Data representing the historical sales of an ice-cream factory.

Type	Country	June	July	August	Total	Profit
Vanilla	BE	610	190	670	1470	1
Banana	BE	170	690	520	1380	1
Chocolate	BE	560	320	140	1020	1
Banana	DE	610	640	320	1570	0
Speculaas	BE	300	270	290	860	0
Chocolate	FR	430	350	300	1080	1

Table 2. Data representing the sales of an ice-cream factory, with missing profit.

Type	Country	June	July	August	Total	Profit
Banana	DE	250	650	630	1530	
Chocolate	NL	210	280	270	760	

In this Section, we show how SynthLog can tackle a classic challenge in data science: automatically filling missing values in a spreadsheet. More precisely, these missing values are predicted with inductive models. The auto-completion task has been identified as a simple, yet challenging task, that illustrates the core of the SYNTH framework [4].

This case study shows that SynthLog successfully use both predictors, such as logistic regression; and probabilistic rules to infer the most likely missing values. We can therefore build on the large literature of the automation of predictor learning [7], while also providing an easy way to add user knowledge in the inference process. We also illustrate how inductive database ideas are used to store and query models depending on the task at hand. We use a toy dataset emulating sales of an ice-cream factory. The data is shown in Tables 1 and 2, with missing profit for the two rows in Table 2. It will be inferred using logistic regression combined with user defined constraints. The code performing the auto-completion is presented below:

```
magic_cells:X :- load_csv('magic_ice_cream.csv', X).
missing_data_cells:X :- load_csv('magic_test1.csv', X).

magic_tables:X :- detect_tables(magic_cells, X).
missing_data:X :- detect_tables(missing_data_cells, X).

magic_models:X :-sklearn_predictor(magic_tables,
    'linear_model.LogisticRegression',
 9      [column('T1',3), column('T1',4)], [column('T1',6)], X).
10
11  magic_predict:X :- magic_models:predictor(Y),
12      magic_models:source(Y, column('T1', 3)),
13      magic_models:source(Y, column('T1', 4)),
14      predict(missing_data,Y,[column('T1',3),column('T1',4)],X).
15
16  final_pred:table_cell('T1', X, 7, V) :-
17      magic_predict:cell_pred(X, Y, V, _).
18
19  magic_constraints:
20      (0.7::table_cell(T,X,7,0):- table_cell(T,X,5,V), V<300).
21  magic_constraints:table_cell('T1', X, Y, V) :-
22      missing_data:table_cell('T1', X, Y, V).
23
24  combined_pred:table_cell(T,X,Y,V) :-
25      magic_constraints:table_cell(T,X,Y,V);
26      final_pred:table_cell(T,X,Y,V).
27
28  query(combined_pred:_).
```

In **Line 1**, we create the theory *magic_cells* from a csv file containing the data in Table 1, by using the custom predicate `load_csv/2`. Details about the custom predicates and their exact behavior are presented in Appendix A. Likewise, **Line 2** creates the theory *missing_data_cells* by loading the data represented in Table 2.

The rest of the program manipulates these 2 theories using SynthLog primitives and custom predicates to perform wrangling, prediction and inference. For example, **Lines 4 and 5** perform wrangling, by using the custom predicate `detect_tables/2`. More precisely, in **Line 4**, `detect_tables/2` transforms the theory *magic_cells* to output the theory *magic_tables*. The new theory *magic_tables* contains the same data as the theory *magic_cells* (i.e. the data from Table 1), but uses a different data model. Indeed, `detect_tables/2` takes a cell based data model and transforms it into a table based data model. Details of this transformation are given in Appendix A. In this simple example, wrangling is straightforward, as the data is already nicely formatted. However, `detect_tables/2` still provides information about cell types and detects headers.

From the theory *magic_tables*, the custom predicate `sklearn_predictor/5` learns an inductive model (**Lines 7 to 9**). More precisely, it learns a logistic regression model [2] that predicts column 6 of Table T1 (the Table depicted in Table 1) from columns 3 and 4 of Table T1. The theory *magic_predict* contains this newly learned inductive model. The theory *magic_predict* also contains additional information about the learned inductive model: on which theory was it learned, using which columns and what type of inductive model it is. Keeping track of all these information allows us to easily query any model, hence treating them as first class citizens.

Lines 11 to 13 query an inductive model by manipulating the theory *magic_predict*. To retrieve an inductive model, we simply specify its properties: it is a predictor and was

[2] We use the scikit-learn library: https://scikit-learn.org/stable/index.html

trained on columns 3 and 4 from Table T1. If several inductive models in *magic_predict* satisfy these requirements, they are all used. SynthLog therefore handles models following the inductive database idea of treating them as first class citizens. Then, **Line 14** applies the queried inductive model on the theory *missing_data* to create the new theory *magic_predict*, using the custom predicate `predict/4`. The theory *magic_predict* contains probabilistic facts representing the predictions of the logistic regression.

Lines 16 and 17 create the theory *final_pred* by selecting a sub-part of the theory *magic_predict*, using a simple ProbLog rule. **Lines 19 and 20** create a new inductive model, by storing a user-defined rule in the theory *magic_constraints*. This rule states that if column 5 of Table T in row X has a value below 300, then column 7 (profit) of Table T in row X has a value of 0 with probability 0.7. In this simple case, this rule could be specified by a user. However, SynthLog supports learning such rules through the use of custom predicates. **Lines 21 and 22** add a sub-part of theory *missing_data* to the theory *magic_constraints*. Since the theory *magic_constraints* now contains `table_cell` predicates, the rule defined in Line 20 will automatically trigger, hence creating the probabilistic fact `0.7::table_cell(T,X,7,0)` when applicable.

Finally, **Lines 24 to 26** create the theory *combined_pred* by performing the union of sub-parts from the theories *magic_constraints* and *final_pred* through the `';'/2` operator of ProbLog. As SynthLog combines probabilistic facts from *final_pred* with the probabilistic rule from *magic_constraints* to create *final_pred*, probabilistic inference has to be performed. Because SynthLog extends ProbLog, it relies on its probabilistic inference mechanism to soundly combine both theories. As in ProbLog, the query of **Line 28** determines what probabilistic facts the program should infer. In this case, we query for the theory *final_pred* to infer the cell values of Table 2, by combining the logistic regression predictions with the user defined rule. The result is shown in Table 3.

Overall we have seen that SynthLog manipulates theories by using either custom predicates or native ProbLog operators. This simple way of manipulating theories is nonetheless powerful, as the resulting program is performing complex inference, taking into account predictive models and rules, while remaining simple to read.

Table 3. Data (from Table 2) with filled profit values and probability on predictions

Type	Country	June	July	August	Total	Profit	Probability
Banana	DE	250	650	630	1530	0	0.04
Banana	DE	250	650	630	1530	1	0.96
Chocolate	NL	210	280	270	760	0	0.46
Chocolate	NL	210	280	270	760	1	0.54

4 Conclusion

We have introduced SynthLog, a declarative language for synthesising Inductive Data Models (IDM). IDMs integrate data and inductive models in a SynthLog theory. Theories can also be seen as ProbLog programs, consisting of probabilistic facts and clauses. Assembling data science components corresponds to manipulating theories, hence making

SynthLog a language suitable for automating data science. As SynthLog is an extension of ProbLog, it natively supports probabilistic reasoning and we have illustrated through a use case how SynthLog can use probabilistic inference to effortlessly combine results from different type of models (predictors and constraints).

Having a language to assemble data science components, based on probabilistic logic, opens new possibilities. First, the inherent uncertainty of data and inductive models can be leveraged to perform probabilistic inference and provide predictions that reflect our confidence in our data and inductive models. Second, SynthLog handles different types of inductive models. More specifically, it handles rules or constraints along with other machine learning models. Hence, SynthLog provides a great opportunity to bridge user interaction and model learning through a unique language.

In the SYNTH framework, SynthLog is also first step towards the automation of data science. Indeed, with a single language combining all data science components, we can tackle the more challenging task of learning to learn, that is learning which SynthLog programs are suitable to automatically solve the data science task at hand.

Finally, the further development of SynthLog will likely require the development of new implementation techniques to support fast inference and learning. This will allow smoother user interaction and the analysis of larger datasets.

5 Acknowledgements

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No [694980] SYNTH: Synthesising Inductive Data Models)

References

1. Van den Broeck, G., Thon, I., Van Otterlo, M., De Raedt, L.: Dtproblog: A decision-theoretic probabilistic prolog. In: Twenty-Fourth AAAI Conference on Artificial Intelligence (2010)
2. De Bie, T., De Raedt, L., Hoos, H.H., Smyth, P.: Automating data science (dagstuhl seminar 18401). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2019)
3. De Raedt, L.: A perspective on inductive databases. ACM SIGKDD Explorations Newsletter **4**(2), 69–77 (2002)
4. De Raedt, L., Blockeel, H., Kolb, S., Teso, S., Verbruggen, G.: Elements of an automatic data scientist. In: International Symposium on Intelligent Data Analysis. pp. 3–14. Springer (2018)
5. Dries, A., Kimmig, A., Meert, W., Renkens, J., Van den Broeck, G., Vlasselaer, J., De Raedt, L.: Problog2: Probabilistic logic programming. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 312–315. Springer (2015)
6. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in neural information processing systems. pp. 2962–2970 (2015)
7. Hutter, F., Kotthoff, L., Vanschoren, J. (eds.): Automated Machine Learning: Methods, Systems, Challenges. Springer (2018), in press, available at http://automl.org/book.
8. Imielinski, T., Mannila, H.: A database perspective on knowledge discovery. Communications of the ACM **39**(11), 58–64 (1996)

9. Malec, M., Khot, T., Nagy, J., Blasch, E., Natarajan, S.: Inductive logic programming meets relational databases: An application to statistical relational learning. In: Inductive Logic Programming (ILP) (2016)
10. Verbruggen, G., De Raedt, L.: Automatically wrangling spreadsheets into machine learning data formats. In: International Symposium on Intelligent Data Analysis. pp. 367–379. Springer (2018)

Appendix A SynthLog custom predicates documentation

- `load_csv/2`: loads the content of a csv file in a theory
  - Input
    - csv file
  - Output: Theory with predicates:
    - `cell/3`: row id, column id and value of each cell
- `detect_tables/2`: calls a data wrangler[10] to detect tables in the spreadsheet
  - Input
    - Theory with `cell/3` predicates
  - Output: Theory with predicates:
    - `table/5`: table id, top left row, top left column, height, width
    - `table_cell/4`: table id, row id, column id and value of each cell
    - `table_cell_type/4`: table id, row id, column id and type of each cell
    - `table_header/5`: table id, column id, name, type, list of unique values
- `sklearn_predictor/5` learns a scikit-learn predictor
  - Input
    - Theory with `table_cell/4` predicates
    - Inductive model type (from scikit-learn models)
    - List of columns to use as features
    - List of columns to predict
  - Output: Theory with predicates:
    - `sklearn_predictor/1`: inductive model
    - `target/2`: inductive model, predicted column
    - `source/2`: inductive model, feature column
- `predict/5` makes prediction using a previously trained model
  - Input
    - Theory with `table_cell/4` predicates
    - Inductive model
    - List of columns to use as features
    - List of columns to predict
  - Output: Theory with predicates:
    - `cell_pred/4`: table id, row id, column id and value of each cell
    - `predictor/1`: inductive model
    - `source/2`: inductive model, feature column
    - `confidence/2`: inductive model, confidence score