Meta-interpretive learning of data transformation programs

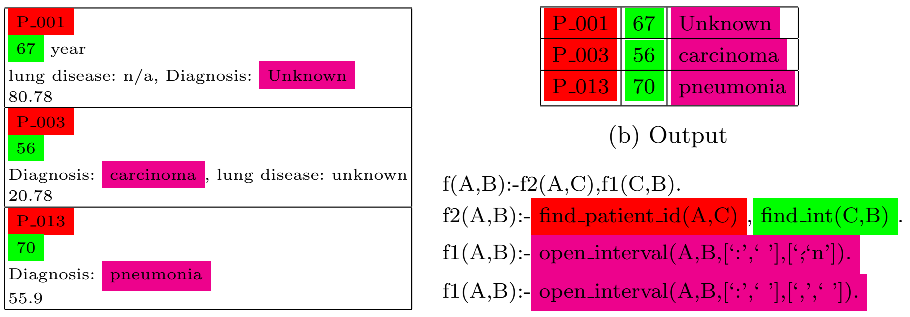
Andrew Cropper, Alireza Tamaddoni-Nezhad, and Stephen H. Muggleton

Department of Computing, Imperial College London

Abstract. Data transformation involves the manual construction of large numbers of special-purpose programs. Although typically small, such programs can be complex, involving problem decomposition, recursion, and recognition of context. Building such programs is common in commercial and academic data analytic projects and can be labour intensive and expensive, making it a suitable candidate for machine learning. In this paper, we use the meta-interpretive learning framework (MIL) to learn recursive data transformation programs from small numbers of examples. MIL is well suited to this task because it supports problem decomposition through predicate invention, learning recursive programs, learning from few examples, and learning from only positive examples. We apply Metagol, a MIL implementation, to both semi-structured and unstructured data. We conduct experiments on three real-world datasets: medical patient records, XML mondial records, and natural language taken from ecological papers. The experimental results suggest that high levels of predictive accuracy can be achieved in these tasks from small numbers of training examples, especially when learning with recursion.

1 Introduction

Suppose you are given a large number of patient records in a semi-structured format and are required to transform them to a given structured format. Fig. 1 shows such a scenario relating to medical patient records, where (a) is the input and (b) is the desired output. To avoid manually transforming the records, you might decide to write a small program to perform this task. Fig. 1c shows a Prolog program for this task which transforms the input to the output. However, manually writing this relatively simple program is somewhat laborious. In this paper, we show how such data transformation programs can be machine learned from a small number of input/output examples. Indeed, the program shown in Fig. 1c was learned from the given input/output examples by our system, described in Section 4. In this program, predicate invention is used to introduce the predicates $f1$ and $f2$ and the primitive background predicates *find_patient_id/2*, *find_int/2*, and *open_interval/4* are used to identify the various fields to be transposed from the input to the output.



P_001
67 year
lung disease: n/a, Diagnosis: Unknown
80.78

P_003
56
Diagnosis: carcinoma, lung disease: unknown
20.78

P_013
70
Diagnosis: pneumonia
55.9

(a) Input

P_001	67	Unknown
P_003	56	carcinoma
P_013	70	pneumonia

(b) Output

```
f(A,B):-f2(A,C),f1(C,B).
f2(A,B):- find_patient_id(A,C) , find_int(C,B) .
f1(A,B):- open_interval(A,B,[‘:’,‘ ’],[‘;‘n’]).
f1(A,B):- open_interval(A,B,[‘:’,‘ ’],[‘,’,‘ ’]).
```

(c) Learned Prolog program

Fig. 1: Transformation of medical records from semi-structured format

In this paper, we investigate the problem of learning data transformation programs as an ILP application. In general, this is a challenging problem for ILP which for example requires learning recursive rules from a small number of training examples. In order to address this problem, we use the recently developed meta-interpretive learning (MIL) framework [12, 13, 5] to learn data transformation programs. MIL differs from most state-of-the-art ILP approaches by supporting predicate invention for problem decomposition and the learning of recursive programs.

Using MIL to learn recursive programs has been demonstrated [4] to be particularly powerful in learning robot strategies, which are applicable to a potentially infinite set of initial/final state pairs, in contrast to learning non-recursive robot plans, applicable to only a specific initial/final state pair. We investigate learning recursive data transformation programs, applicable to a potentially infinite set of input/output examples.

The paper is arranged as follows. Section 2 describes related work. Section 3 describes the theoretical framework. Section 4 describes the transformation language used in the experiments. Section 5 describes experiments in learning recursive data transformation programs in three domains: medical patient records, XML mondial documents, and ecological scholarly papers. Finally, Section 6 concludes the paper and details future work.

2 Related work

In [3] the authors compared statistical and relational methods to extract facts from a MEDLINE corpus. The primary limitation of the statistical approach, they state, is its inability to express the linguistic structure of the text; by contrast, the relational approach allows these features to be encoded as background knowledge, as parse trees, specifically. The relational approach used an ILP system similar to FOIL [14] to learn extraction rules. Similar works include [6], who used the ILP system Aleph [16] to learn rules to extract relations from a MEDLINE corpus; and [1], who used the ILP system FOIL to learn rules to extract relations from Nature and New Scientist articles. These works focused on constructing the appropriate problem representation, including determining the necessary linguistic features to be included in the background knowledge. In contrast to these approaches, we use the state-of-the-art ILP system Metagol, which supports predicate invention, the learning of recursive theories, and positive-only learning, none of which is supported by FOIL nor Aleph.

FlashExtract [8] is a framework to extract fields from documents, such as text files and web pages. In this framework, the user highlights one or two examples of each field in a document and FlashExtract attempts to extract all other instances of such fields, arranging them in a structured format, such as a table. FlashExtract uses an inductive synthesis algorithm to synthesise the extraction program using a domain-specific language built upon a pre-specified algebra of a few core operators (map, filter, merge, and pair). In contrast to FlashExtact, our approach allows for the inclusion of background knowledge.

Wu et. al. [19] presented preliminary results on learning data transformation rules from examples. They demonstrated that specific string transformation rules can be learned from examples, given a grammar describing common user editing behaviors (i.e. insert, move, and delete). Their approach then uses a search algorithm to reduce the larger grammar space to a disjunction of subgrammar spaces (i.e. transformation rules) which are consistent with the examples. Depending on the grammar, the search could still generate many consistent transformations and they use a ranking algorithm to order transformation rules, e.g. based on the homogeneity of the transformed data. In their approach the set of transformation rules that the system can generate are pre-defined and not universal. By contrast, in our work, the transformation programs are not pre-defined and can be learned using predicate invention. Wu and Knoblock [18] recently extended their approach into a Programming-by-Example technique which iteratively learns data transformation programs by example. Their technique works by identifying previous incorrect subprograms and replacing them with correct subprograms. They demonstrated their technique on a set of string transformation problems and compared the results with the Flashfill approach [7] and Metagol$_{DF}$ [9]. While the overall aims of their approach are similar to ours, their approach does not support automatic problem decomposition using predicate invention, nor learning recursive programs. In this paper we also demonstrated our approach on a wider range of applications.

In [9], MIL was used to learn string transformation programs. In this approach, the authors perform transformations at the character level. However, this approach significantly increases the search space, and is unsuitable for learning from large input/output examples. By contrast, in this work, we look at the more general problem of learning data transformation programs, which are applicable to larger inputs/outputs and also a wider range of inputs/outputs. For instance, in Section 5, we apply our technique to relatively larger (55kb) XML files.

ILP has been used in the past for the task of learning recursive rules from biological text. For example in [2], recursive patterns are discovered from biomedical text by inducing mutually dependent definitions of concepts using the ILP system ATRE. However, this is restricted in that they have a fixed number of slots in the output which need to be filled. By comparison, both the ecological and XML experiments in this paper show that we are not limited to this in our approach.

Similarly, ATRE has been used in [10] to learn a recursive logical theory of the ontology from a biological text corpus. However these ILP approaches cannot be easily extended for the general task of learning data transformation programs from examples. Both [2] and [10] have only been demonstrated on extraction of specific types of information from biological text, and did not attempt more general text transformation tasks of the kinds demonstrated in our paper. Moreover, these approaches were not shown to learn recursive rules from a small number of training examples and also do not support predicate invention.

3 Framework

MIL [12, 13] is a form of ILP based on an adapted Prolog meta-interpreter. Whereas a standard Prolog meta-interpreter attempts to prove a goal by repeatedly fetching first-order clauses whose heads unify with a given goal, a MIL learner attempts to prove a set of goals by repeatedly fetching higher-order metarules (Fig. 3) whose heads unify with a given goal. The resulting meta-substitutions are saved in an abduction store, and can be re-used in later proofs. Following the proof of a set of goals, a hypothesis is formed by applying the meta-substitutions onto their corresponding metarules, allowing for a form of ILP which supports predicate invention and the learning of recursive theories.

General formal framework In the general framework for data transformation we assume that the user provides examples E of how data should be transformed. Each example $e \in E$ consists of a pair $\langle d1, d2 \rangle$ where $d1 \in D1$ and $d2 \in D2$ are input and output data records respectively. Given background knowledge B, in the form of existing transformations and the user-provided examples E the aim of the learning is to generate a transformational function $\tau : D1 \rightarrow D2$ such that $B, \tau \models E$.

4 Implementation

Fig. 2 shows the implementation of Metagol[1] as a generalised meta-interpreter [13], similar in form to a standard Prolog meta-interpreter.

```
prove([],H,H).
prove([Atom|Atoms],H1,H2):-
    prove_aux(Atom,H1,H3),
    prove(Atoms,H3,H2).
prove_aux(Atom,H,H):-
    call(Atom).
prove_aux(Atom,H1,H2):-
    member(sub(Name,Subs),H1),
    metarule(Name,Subs,(Atom :- Body)),
    prove(Body,H1,H2),
prove_aux(Atom,H1,H2):-
    metarule(Name,Subs,(Atom :- Body)),
    new_metasub(H1,sub(Name,Subs)),
    abduce(H1,H3,sub(Name,Subs)),
    prove(Body,H3,H2).
```

Fig. 2: Prolog code for Metagol, a generalised meta-interpreter

Metagol works as follows. Metagol first tries to prove a goal deductively delegating the proof to Prolog (*call(Atom)*). Failing this, Metagol tries to unify the goal with the head of a metarule (*metarule(Name,Subs,(Atom :- Body))*) and to bind the existentially quantified variables in a metarule to symbols in the signature. Metagol saves the resulting *meta-substitutions* (*Subs*) in an abduction store and tries to prove the body goals of the metarule. After proving all goals, a Prolog program is formed by projecting the meta-substitutions onto their corresponding metarules.

[1] `https://github.com/metagol/metagol`

Name	**Metarule**	**Order**
Base	$P(x,y) \leftarrow Q(x,y)$	$P \succ Q$
Chain	$P(x,y) \leftarrow Q(x,z), R(z,y)$	$P \succ Q, P \succ R$
Curry	$P(x,y) \leftarrow Q(x,y,c_1,c_2)$	$P \succ Q$
TailRec	$P(x,y) \leftarrow Q(x,z), P(z,y)$	$P \succ Q, x \succ z \succ y$

Fig. 3: Metarules with associated ordering constraints, where $\succ$ is a pre-defined ordering over symbols in the signature. The letters P, Q, and R denote existentially quantified higher-order variables; x, y, and z denote universally quantified first-order variables; and c_1 and c_2 denote existentially quantified first-order variables.

4.1 Transformation language

In the transformation language, we represent the state as an $Input/Output$ pair, where $Input$ and $Output$ are both character lists. The transformation language consists of two predicates $skip\_to/3$ and $open\_interval/4$. The $skip\_to/3$ predicate is of the form $skip\_to(A, B, Delim)$, where A and B are states, and $Delim$ is a character list. This predicate takes a state $A = InputA/OutputA$ and skips to the delimiter $Delim$ in $InputA$ to form an output B = $InputB/OutputA$. For example, let $A = [i, n, d, u, c, t, i, o, n]/[]$ and $Delim = [u, c]$, then $skip\_to(A, B, Delim)$ is true where $B = [u, c, t, i, o, n]/[]$. The $open\_interval/4$ predicate is of the form $open\_interval(A, B, Start, End)$, where A and B are states, and $Start$ and End are character lists. This predicate takes a state $A = InputA/OutputA$, finds a sublist in $InputA$ denoted by $Start$ and End delimiters, appends that sublist to $OutputA$ to form $OutputB$, and skips all elements up to End delimiter in $InputA$ to form $InputB$. For example, let $A = [i, n, d, u, c, t, i, o, n]/[]$, $Start = [n, d]$, and $End = [t, i]$, then open_interval(A,B,Start,End) is true where $B = [t, i, o, n]/[u, c]$.

5 Experiments

We now detail experiments[2] in learning data transformation programs. We test the following null hypotheses:

Null hypothesis 1 Metagol cannot learn data transformation programs with higher than default predictive accuracies.

Null hypothesis 2 Metagol$_R$ (with recursion) cannot learn data transformation programs with higher predictive accuracies than Metagol$_{NR}$ (without recursion).

[2] Experiments available at `https://github.com/andrewcropper/ilp15-datacurate`

To test these hypotheses, we apply our framework to three real-world datasets: XML mondial files, patient medical records, and ecological scholarly papers. To test null hypothesis 2, we learn using two versions of Metagol: Metagol$_R$ and Metagol$_{NR}$. Both versions use the *chain* and *curry* metarules, but Metagol$_R$ also uses the *tailrec* metarule.

We do not compare our results with other state-of-the-art ILP systems because they cannot support predicate invention, nor, crucially, the learning of recursive programs, and so such a comparison would be unfair.

5.1 XML data transformations

In this experiment, the aim is to learn programs to extract values from semi-structured data. We work with XML files, but the methods can be applied to other semi-structured mark-up languages, such as JSON.

Materials The dataset[3] is a 1mb worldwide geographic database XML file which contains information about 231 countries, such as population, provinces, cities, etc. We split the file so that each country is a separate XML file. We consider the task of extracting all the city names for each country. Fig. 4 shows three simplified positive examples, where the left column (input) is the XML file and the right column (output) is the city names to be extracted. Appendix A shows a full input example used in the experiments. Note the variations in the dataset, such as the varying number of cities (from 1 to over 30) and the differing XML structures. To generate training and testing examples, we wrote a Python script (included as Appendix B) to extract all the city names for each country. This was a non-trivial task for us, and it would be even more difficult for non-programmers, which supports the claim that this should be automated). The 231 pairings of a country XML file and the cities in that country form the positive examples. We do not, however, use all the positive examples as training examples because the amount of information varies greatly depending on the country. For instance, the file on the British crown dependency of Jersey is 406 bytes, whereas the file on the USA is 55kb. We postulate that in a real-world setting a user is unlikely to manually annotate (i.e. label) a 55kb file. Therefore, we only use country files less than 2kb as positive training examples, of which there are 182. We do, however, use the larger country files for testing. To generate negative examples, for each XML file we extracted all k text entries and randomly selected j values to form the negative output, ensuring that the random

[3] http://www.cs.washington.edu/research/xmldatasets/www/repository.html#mondial

sample did correspond to the expected output. Fig. 5 displays an example negative instance.

Input	**Output**
`<country id='f0_136' name='Albania' capital='f0_1461'> <name>Albania</name> <city><name>Tirane</name></city> <city><name>Shkoder</name></city> <city><name>Durres</name></city> </country>`	`Tirane, Shkoder, Durres`
`<country id='f0_144' name='Andorra' capital='f0_1464'> <name>Andorra</name> <city><name>Andorra la Vella</name></city> </country>`	`Andorra la Vella`
`<country id='f0_149' name='Austria' capital='f0_1467'> <name>Austria</name> <province name="Burgenland"> <city><name>Eisenstadt</name></city> </province> <province name="Vienna"> <city><name>Vienna</name></city> </province> </country>`	`Eisenstadt, Vienna`

Fig. 4: Three simplified XML transformation positive examples where all the city names have been extracted. Most of the XML has been removed for brevity, but is included in the experiments. The actual examples contain a lot more information and a full example is included as Appendix A.

Methods For each m in the set $\{1, \ldots, 10\}$ we randomly select without replacement m positive and m negative training examples. The default predictive accuracy is therefore 50%. We average predictive accuracies and learning times over 10 trials. We set a maximum solution length to 5.

Results Fig. 6 shows that Metagol learns solutions with higher than default predictive accuracies, refuting null hypothesis 1. Fig. 6 also shows that learning with recursion results in higher predictive accuracies compared to learning without recursion, refuting null hypothesis 2. The difference in performance is because the non-recursive learner cannot handle the varying number of cities, whereas the recursive solution (Fig. 7) can handle any number of cities.

Input

```
<country capital="f0_1557" name="Liechtenstein">
  <name>Liechtenstein</name>
  <city>
    <name>Vaduz</name>
    <population>27714</population>
  </city>
  <ethnicgroups>Italian</ethnicgroups>
  <ethnicgroups>Alemannic</ethnicgroups>
  <religions>Roman Catholic</religions>
  <religions>Protestant</religions>
</country>
```

Output

```
Italian, 27714, Vaduz, Liechtenstein, Alemannic, Protestant, Roman Catholic
```

Fig. 5: Simplified XML transformation negative example

5.2 Ecological scholarly papers

In this experiment, the aim is to learn programs to extract relations from natural language taken from ecological scholarly papers.

Materials The dataset contains 25 positive real-world examples of a natural language sentence paired with a list of values to be extracted. These examples are taken from ecological scholarly papers adopted from [17]. Fig. 8 shows two example pairings where we want to extract the predator name, the predation term, and a list of all prey. We provide all species and predication terms in the background knowledge. No negative examples are provided, so we learn using positive examples only.

Methods For each m in the set $\{1, \ldots, 10\}$, we randomly select without replacement m positive training examples and 10 positive testing examples. We average predictive accuracies and learning times over 20 trials. We set a maximum solution length to 5.

Results Fig. 9 shows that learning with recursion significantly improves predicate accuracies compared to learning without recursion, again refuting null hypothesis 2. Fig. 10 shows an example learned recursive solution.

5.3 Patient medical records

In this experiment, the aim is to learn programs to extract values from patient medical records.

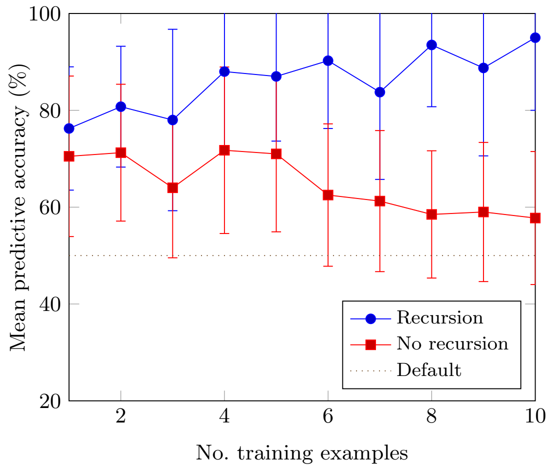


Fig. 6: XML learning performance when varying number of training examples

```
f(A,B):- f1(A,C), f(C,B).
f1(A,B):- open_interval(A,B,['a','m','e','>'],['<o','/','n','a']).
f(A,B):- skip_to(A,B,['<','/','n','a']).
```

Fig. 7: Example recursive solution learned by Metagol on the XML dataset

Materials The dataset contains 16 positive patient medical records, modelled on real-world examples[4], paired with a list of values to be extracted. Figs. 1a and 1b show simplified input/output example pairings. The experimental dataset, however, contains one additional input and output value, which is a floating integer value. We provide the following background predicates: find_int/2, find_float/2, and find_patient_id/2. We do not, however, provide a predicate to identify the diagnosis field, so Metagol must use the general purpose background predicates, described in Section 4, to learn a solution. To generate negative testing examples, we create a frequency distribution over input lengths from the training examples and a frequency distribution over words and punctuation in the training examples. To create a negative testing example input, we randomly select a length n from the length distribution and then randomly select with replacement n words or punctuation characters.

Methods For each m in the set $\{1, \ldots, 5\}$, we randomly select without replacement m positive training examples. We test using 20 positive and

[4] http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE8581

input 1: This species also has a wide food range, but whereas Feronia melanaria took Coleoptera adults as the main item of the diet, Nebria brevicollis took spiders, Collembola, Coleoptera adults and larvae in equal number in the present study.
output 1: [‘Nebria brevicollis’, ‘took’, ‘spiders’, ‘Collembola’, ‘Coleoptera adults and larvae’]

input 2: Bembidion lampros. This species is an important predator of cabbage root fly eggs (Hughes 1959; Coaker & Williams 1963) and it also feeds on Collembola, mites, pseudo-scorpions, earthworms and small bettles (Mitchell 1963a).
output 2: [‘Bembidion lampros’, ‘predator’, ‘cabbage root fly eggs’, ‘Collembola’, ‘mites’, ’pseudo-scorpions’, ‘earthworms’, ‘small bettles’]

Fig. 8: Two input/output examples from the ecological experiment

20 negative testing examples. The default predictive accuracy is therefore 50%. We average predictive accuracies and learning times over 20 trials. We set a maximum solution length to 5.

Results Fig. 11 shows that predictive accuracies improve with an increasing number of training examples, with over 80% predictive accuracy from a single example. In all cases, the predictive accuracies of learned solutions are greater than the default accuracy, and thus null hypothesis 1 is refuted. Fig. 12 shows an example solution.

6 Conclusion and further work

We have investigated learning programs which transform data from one format to another, and we have introduced a general framework for the problem. Our experiments on medical patient records, XML mondial files, and ecological natural language texts, indicate that MIL is capable of generating accurate recursive data transformation programs from small numbers of user-provided examples.

This paper provides an initial investigation into an important new group of ILP applications relevant to information extraction by example problems. Further work remains to achieve this potential.

Several issues need to be studied further in scaling-up the work reported. To begin with, although training data required will be small, owing to the requirements of user-provided annotation, test data will typically consist of large numbers of instances. Running Prolog hypotheses on such data will be time-consuming, and we would like to investigate generating hypotheses in a scripting language, such as Python.

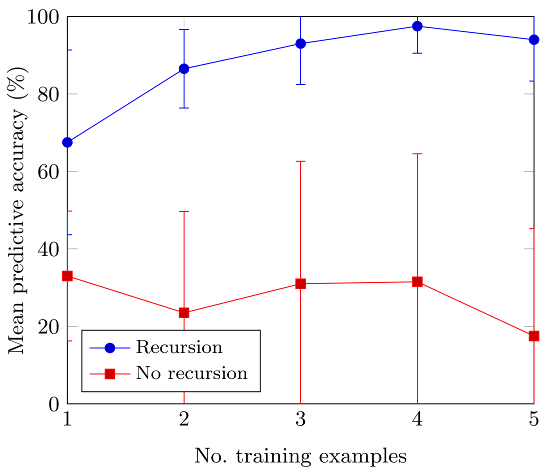


Fig. 9: Ecological learning performance when varying number of training examples

```
f(A,B):- f3(A,C), f2(C,B).
f3(A,B):- f2(A,C), find_predation(C,B).
f2(A,B):- find_species(A,B).
f2(A,B):- find_species(A,C), f2(C,B).
```

Fig. 10: Example recursive solution learned by Metagol on the ecological dataset

For data transformation problems such as the ecological dataset we would also like to investigate the value of large-scale background knowledge, which might provide deeper natural language analysis based on dictionaries, tokenisation, part-of-speech tagging, and specialised ontologies.

We would also like to investigate Probabilistic ILP approaches [15], such as [11], which have the potential to not only provide probabilistic preferences over hypothesised programs, but also the potential of dealing with issues such as noise which are ubiquitous within real-world data. In the context of free-text data these approaches might also be integrated with finding highest probability parses.

Acknowledgements

The first author acknowledges the support of the BBSRC and Syngenta in funding his PhD Case studentship. The second author acknowledges the support from the IMI eTRIKS project. The third author would like

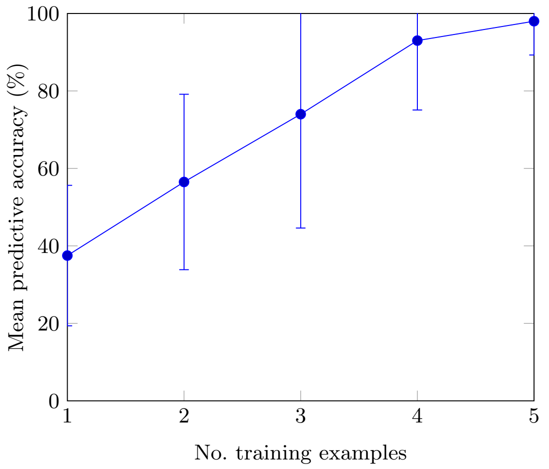


Fig. 11: Medical record learning performance when varying number of training examples

```
f(A,B):- f2(A,C), f2(C,B).
f2(A,B):- find_patient_id(A,C), find_int(C,B).
f2(A,B):- f1(A,C), find_float(C,B).
f1(A,B):- open_interval(A,B,[‘:’,‘ ’],[‘;‘n’]).
f1(A,B):- open_interval(A,B,[‘:’,‘ ’],[‘,’,‘ ’]).
```

Fig. 12: Learned program

to thank the Royal Academy of Engineering and Syngenta for funding his present 5 year Research Chair.

References

1. James S Aitken. Learning information extraction rules: An inductive logic programming approach. In *ECAI*, pages 355–359, 2002.
2. Margherita Berardi and Donato Malerba. Learning recursive patterns for biomedical information extraction. In *Proceedings of the 16th International Conference on Inductive Logic Programming (ILP 2006)*, pages 79–93, 2007.
3. Mark Craven, Johan Kumlien, et al. Constructing biological knowledge bases by extracting information from text sources. In *ISMB*, volume 1999, pages 77–86, 1999.
4. A. Cropper and S.H. Muggleton. Learning efficient logical robot strategies involving composable objects. In *Proceedings of the 24th International Joint Conference Artificial Intelligence (IJCAI 2015)*, pages 3423–3429. IJCAI, 2015.
5. A. Cropper and S.H. Muggleton. Logical minimisation of meta-rules within meta-interpretive learning. In *Proceedings of the 24th International Conference on Inductive Logic Programming*, pages 65–78. Springer-Verlag, 2015. LNAI 9046.

6. Mark Goadrich, Louis Oliphant, and Jude Shavlik. Learning ensembles of first-order clauses for recall-precision curves: A case study in biomedical information extraction. In *Inductive logic programming*, pages 98–115. Springer, 2004.
7. Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 317–330, 2011.
8. Vu Le and Sumit Gulwani. Flashextract: A framework for data extraction by examples. In *ACM SIGPLAN Notices*, volume 49, pages 542–553. ACM, 2014.
9. D. Lin, E. Dechter, K. Ellis, J.B. Tenenbaum, and S.H. Muggleton. Bias reformulation for one-shot function induction. In *Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI 2014)*, pages 525–530, Amsterdam, 2014. IOS Press.
10. Alain-Pierre Manine, Erick Alphonse, and Philippe Bessires. Extraction of genic interactions with the recursive logical theory of an ontology. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing (CICLing 2010)*, volume 6008 of *Lecture Notes in Computer Science*, pages 549–563. Springer Berlin Heidelberg, 2010.
11. S.H. Muggleton, D. Lin, J. Chen, and A. Tamaddoni-Nezhad. Metabayes: Bayesian meta-interpretative learning using higher-order stochastic refinement. In Gerson Zaverucha, Vitor Santos Costa, and Aline Marins Paes, editors, *Proceedings of the 23rd International Conference on Inductive Logic Programming (ILP 2013)*, pages 1–17, Berlin, 2014. Springer-Verlag. LNAI 8812.
12. S.H. Muggleton, D. Lin, N. Pahlavi, and A. Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94:25–49, 2014.
13. S.H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 2015. Published online: DOI 10.1007/s10994-014-5471-y.
14. J.R. Quinlan and R.M Cameron-Jones. FOIL: a midterm report. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 3–20. Springer-Verlag, 1993.
15. L. De Raedt, P. Frasconi, K. Kersting, and S.H. Muggleton, editors. *Probabilistic Inductive Logic Programming*. Springer-Verlag, Berlin, 2008. LNAI 4911.
16. Ashwin Srinivasan. *The Aleph Manual*. University of Oxford, 2007.
17. KD Sunderland. The diet of some predatory arthropods in cereal crops. *Journal of Applied Ecology*, 12(2):507–515, 1975.
18. Bo Wu and Craig A. Knoblock. An iterative approach to synthesize data transformation programs. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.
19. Bo Wu, Pedro Szekely, and Craig A. Knoblock. Learning data transformation rules through examples: Preliminary results. In *Proceedings of the Ninth International Workshop on Information Integration on the Web*, IIWeb '12, pages 8:1–8:6, New York, NY, USA, 2012. ACM.

A Appendix 1

```
<?xml version="1.0" encoding="UTF-8"?>
<country capital="f0_2148" car_code="GH" datacode="GH"
gdp_agri="47" gdp_ind="16" gdp_serv="37" gdp_total="25100"
government="constitutional democracy" id="f0_1269"
indep_date="06 03 1957" infant_mortality="80.3"
inflation="69" name="Ghana" population="17698272"
population_growth="2.29" total_area="238540">
  <name>Ghana</name>
  <city country="f0_1269" id="f0_2148" latitude="5.55" longitude="-0.2">
    <name>Accra</name>
    <population year="84">867459</population>
    <located_at type="sea" water="f0_38068" />
  </city>
  <city country="f0_1269" id="f0_16328">
    <name>Kumasi</name>
    <population year="84">376249</population>
  </city>
  <city country="f0_1269" id="f0_16333">
    <name>Cape Coast</name>
    <population year="84">57224</population>
  </city>
  <city country="f0_1269" id="f0_16338">
    <name>Tamale</name>
    <population year="84">135952</population>
  </city>
  <city country="f0_1269" id="f0_16343">
    <name>Tema</name>
    <population year="84">131528</population>
  </city>
  <city country="f0_1269" id="f0_16348">
    <name>Takoradi</name>
    <population year="84">61484</population>
  </city>
  <city country="f0_1269" id="f0_16353">
    <name>Sekondi</name>
    <population year="84">31916</population>
  </city>
  <ethnicgroups percentage="0.2">European</ethnicgroups>
  <ethnicgroups percentage="99.8">African</ethnicgroups>
  <religions percentage="30">Muslim</religions>
  <religions percentage="24">Christian</religions>
  <encompassed continent="f0_132" percentage="100" />
  <border country="f0_1189" length="548" />
  <border country="f0_1231" length="668" />
  <border country="f0_1422" length="877" />
</country>
```

B Appendix 2

```
from xml.dom import minidom
doc = minidom.parse('mondial.xml')
countries = doc.getElementsByTagName('country')
i=1
for country in countries:
  cities = country.getElementsByTagName('city')
  names = [city.getElementsByTagName('name')[0].childNodes[0].data.strip()
          for city in cities]
  with open('parsed/output-{0}.txt'.format(i),'w') as f:
    f.write(','.join(names))
  i+=1
```