

One-shot Information Extraction from Document Images using Neuro-Deductive Program Synthesis

Vishal Sunder¹, Ashwin Srinivasan², Lovekesh Vig¹, Gautam Shroff¹, Rohit Rahul¹

¹TCS Research, ²BITS Pilani, Goa

s.vishal3@tcs.com, ashwin@goa.bits-pilani.ac.in, lovekesh.vig@tcs.com, gautam.shroff@tcs.com, rohitrahul@tcs.com

ABSTRACT

Our interest in this paper is in meeting a rapidly growing industrial demand for information extraction from images of documents such as invoices, bills, receipts etc. In practice users are able to provide a very small number of example images labeled with the information that needs to be extracted. We adopt a novel ‘two-level’ neuro-deductive’ approach where (a) we use pre-trained deep neural networks to populate a relational database with facts about each document-image; and (b) we use a form of deductive reasoning, related to meta-interpretive learning of transition systems to learn extraction programs: Given task-specific transitions defined using the entities and relations identified by the neural detectors and a small number of instances (usually 1, sometimes 2) of images and the desired outputs, a resource-bounded meta-interpreter constructs proofs for the instance(s) via logical deduction; a set of logic programs that extract each desired entity is easily synthesized from such proofs. In most cases a single training example together with a noisy-clone of itself suffices to learn a program-set that generalizes well on test documents, at which time the value of each entity is determined by a majority vote across its program-set. We demonstrate our two-level neuro-deductive approach on publicly available datasets (“Patent” and “Doctor’s Bills”) and also describe its use in a real-life industrial problem.

KEYWORDS

Information Extraction, Document Images, Inductive Logic Programming, Meta-Interpretive Learning, Program Synthesis

1 INTRODUCTION

Extraction of information from structured documents has long been an important problem in the research and application of Information Retrieval (IR) techniques. A challenging version of this task arises when the contents of the documents are not already in a structured database, but are captured as images. In industrial settings, this is especially common: images of invoices, bills, forms, etc., are readily available, and information needs to be extracted from them (“What is the address to which this invoice was sent?”, etc.). The images can be noisy (captured using a low-quality camera, at a sub-optimal

angle, for example), and can even be ambiguous and the information extraction task is often manually done.

With the rapid advancement of Deep Learning (DL) for computer vision problems, many DL architectures are available today for document image understanding ([11], [18], [22], [28]). But like most DL-based techniques, training these models from scratch is resource and data intensive. This is a major stumbling block for industrial problems for which collecting and annotating data incur significant costs in time and money. In this paper, we use two complementary forms learning to address this problem:

- (1) *Neural-learning*: Using pre-trained DL models for reading document images and converting them into a structured form by populating a predefined database schema.
- (2) *Deductive-learning*: Using the entities and primitive relations identified by neural-learning, synthesize re-usable logic programs for extracting entities from a document image, using proofs constructed by a meta-interpreter in a manner similar to explanation-based generalization (EBG: [10]), and generalizing the proofs using techniques developed in Inductive Logic Programming (ILP: [15]).

A schematic overview our approach is given in Fig. 1. The choice of logic-based EBG for symbolic learning has two attributes that are of interest to us: (1) EBG methods generalize from a single data instance (or sometimes just a few) by exploiting strong domain constraints (in effect, the constraints act as a prior over possible models for the data); and (2) The logical models can often be converted into an (human-)interpretable form. This makes it possible to allow human intervention, which is important in practice.

The neuro-symbolic learners are used to deploy a one-shot learning strategy for information extraction from images of documents of a particular kind (invoices, for example). Given one training instance, the neural-learning results in a database containing the objects and relations recognised in the corresponding image. The symbolic learner then synthesizes a (re-usable) program that can extract entities from any document of the same kind. Some additional machinery is needed for tackling outlier cases. We allow human intervention to make corrections by providing few additional annotations (usually just one or two). This is made possible by the interpretable nature of the output produced by symbolic learning. Although such cases are few in number, they is an important step towards building robust “human-in-the-loop” systems ([29]) that use human expertise to enhance their performance.

This paper makes the following contributions:

- (1) It combines deep-learning based image processing and deductive program synthesis to address industrial problems involving information extraction from document images.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

A short version of this paper appears in *NeSy’19 @ IJCAI 2019, August 2019, Macau, China*

© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00
<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

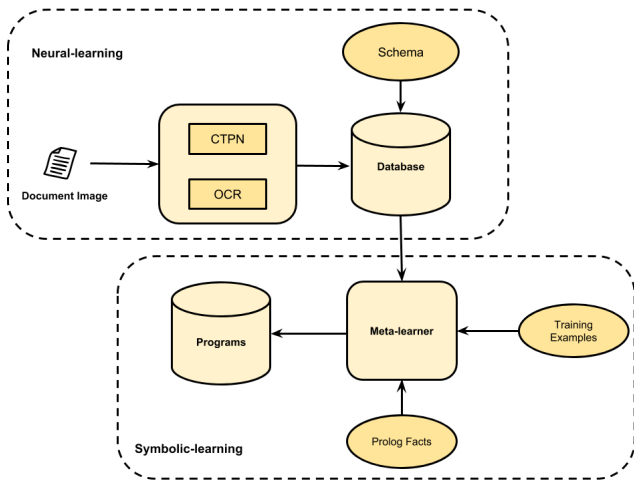


Figure 1: Overview of the system

- (2) It reports on the implementation of an end-to-end one-shot learning strategy for synthesizing programs that extract entities from a document image.
- (3) It presents a human-in-the-loop based N -shot learning algorithm for extracting entities from document images in when required due to outlier cases.

These contributions are supported by results on some publicly available datasets. We also describe, within the constraints allowed, results on a small proprietary dataset which is nevertheless indicative of the industrial applications of the approach.

The rest of this paper is organized as follows: Section 2 gives an overview of the DL-based vision stage. Section 3 formalizes the approach used for program synthesis. Section 4 describes our one-shot learning technique. It also proposes a human-in-the-loop based few-shot learning algorithm. Results are then presented and discussed in Section 5. Section 6 provides a performance analysis of the results and a few ideas for enhancement and debugging. We conclude in Section 8.

2 NEURAL LEARNING: IMAGE \rightarrow DATABASE

Given a document image, the task of extracting spatial relationships between different entities in the document and subsequently populating a database schema is handled by a suite of deep vision APIs which we shall refer to as the *VisionAPI* in the rest of the paper.

The VisionAPI comprises of two modules: 1) A *Visual detection and recognition module* 2) *Spatial-relationship generation module*.

2.1 Visual detection and recognition

The job of this module is to locate the *bounding boxes* around horizontally aligned text in an image. Post detection, recognition amounts to inferring the text present in this bounding box via OCR.

2.1.1 Text Detection. For this task, we use state-of-the-art Connectionist Text Proposal Network (CTPN) [26] which is commonly used for text detection in scene text images. This is a Convolutional Neural Network (CNN) which takes an image as input and generates text for the given image. The sequence of proposals are

then passed to a Recurrent Neural Network (RNN). This allows the network to exploit the contextual visual features of continuous text. The output is in the form of bounding box coordinates around the text. We use a pretrained version of this network (trained on the ICDAR 2013 dataset, [9]).

2.1.2 Recognition of Entities. The bounding boxes returned by the text detection module are then cropped from the input images and each one of the box is then fed to an OCR module (Optical Character Recognition). We have used the Google Vision API for this purpose but in principle any other OCR like Tesseract can also be used. As a result, we get a string corresponding to the text that is inferred by the OCR module in the unicode format. For each word of the string, we apply a data-type detection module which identifies the abstract data-type of the word. (For example, $\langle name \rangle$, $\langle city \rangle$, $\langle date \rangle$, $\langle word \rangle$, $\langle alphanumeric \rangle$ etc)

2.2 Identification of Primitive Relations

To generate basic spatial relationships, we exploit the bounding box coordinates and the corresponding text that we get from the previous module. In this manner, we generate 17 different relationships each of which come under one of the following categories. (In principle, one can come up with any number of such relationships given the bounding box coordinates and the corresponding text.)

- (1) *Text blocks*: We define a textblock as a set of lines which begin at approximately the same x-coordinate and the vertical distance between them is not more than twice the height of the text line (calculated through the coordinates). This yields relationships which give us the words and lines that are part of the same text block.
- (2) *Page lines*: A line of text which is horizontally aligned is defined as a page line. We build a relationship which is a mapping between a word and the page line in which it occurs.
- (3) *Above-below*: This captures the relationships between lines and blocks in the vertical direction. We have 4 relationships in this category where for every line and every block, we have an above and below relationship indicating which lines/blocks are above or below other lines/blocks.
- (4) *Left-right*: This category is the same as above-below except that it is in the horizontal direction and has additional relationships between words.
- (5) *Substring*: For a line/block, we have a relationship which maps every pair of words in that line/block to the substring between the pair. To account for multiple occurrences of word pairs, we assign a unique index to it.
- (6) *Datatype*: Similar to substring except that here, instead of word pairs we use datatype pairs.

A tabulation of the relations found by the neural-learner is in Table 1, which define the domain theory for program synthesis. (A more detailed explanation of the deep-learning techniques used in the VisionAPI can be found in [20].)

3 DEDUCTION: DATABASE \rightarrow PROGRAMS

The task of the deductive stage is to automatically construct a programmatic mechanism to extract entities from the database

```

text_blocks_master, page_lines_master
lines_below_block_word, word_in_line
  above_block, below_block
  above_line, below_line
word_right_left, right_block
  left_block, right_line
left_line, block_to_substring
block_to_substring_dtype, line_to_substring
line_to_substring_dtype

```

Table 1: Primitive relations obtained from the VisionAPI.

populated by the neural learner, for all documents in the same template-class of ‘similar’ documents.

Deductive program synthesis can be seen as a form of Explanation-Based Generalization (EBG) [10]. Although EBG was originally formulated for concept learning, here the goal is to identify programs that implement functions. Given a training example e that identifies the output(s) O from some input image I , the task is to identify a program that is sufficient to compute O from I , given B . Conceptually, this is done in two steps: (a) A *proof* is constructed for the computation of O from I , given B ; and (b) The proof is *generalised* to obtain a (re-usable) program.

In this paper, we adopt the representation of logic programs for B : In our case B consists of (Prolog) rules defined using the primitive relations in the database populated by the neural learning stage (1).

EXAMPLE 3.1. *Here are two background definitions (in Prolog syntax) for an “invoice” template, that use the primitive relations identified by the VisionAPI.*

```

has_keyword(Word, [In], [In, LineId, WordId]) :-
  word_in_line(In, _, _, LineId, Word, WordId).

```

```

left_of(RWord, [In, LineId, RWordId], [In, LineId, LWordId]) :-
  word_right_left(In, _, LWord, LType, LWordId, RWord, RType, RID)
  word_in_line(In, _, LType, _, LineId, LWord, LWordId),
  word_in_line(In, _, RType, _, LineId, RWord, RWordId).

```

The first rule definition looks for a word $Word$ in the `word_in_line` relation and returns its `WordId` and the `LineId` of the line its contained in. The second rule definition returns the `LWordId` of the word to the left of $RWord$.

For the computational system we use the operational semantics of a transition system (in the sense identified by Plotkin in [19]).

DEFINITION 3.2. (**Transition system**) *A logic program defining a simple transition system T is:*

```

ts((C, C)) ←
ts((Ci, Cf)) ←
  trans(T, Ci, C),
ts((C, Cf))

```

As defined above, the transition system can compute indefinitely, and in practice, we impose a bound on the computation, by including a depth-limit. The C ’s are used to denote *configurations* that can be more general than states. With this definition a training example v is then a simply a specification of a specific input-output

configuration pair $ts((i, o))$, where i is a document, in which the value of the desired entity e is o .

REMARK 3.3. (**Transition Systems and Meta-interpretation**) *A depth-bounded version of the transition system, T_d can be readily implemented as a logic program (shown here as a Prolog program):*

```

ts((Ci, Cf), D) :-
  D = 1,
  trans(T, Ci, Cf).
ts((Ci, Cf), D) :-
  D > 1,
  trans(T, Ci, C),
  D1 is D - 1,
  ts((C, Cf), D1).

```

Given this definition, a depth-bound d and a domain-theory B consisting of definitions of *trans*/3 predicates, We will say a depth-bounded explanation exists for $v = ts((i, o))$ if a Prolog interpreter is able to prove e using SLD-resolution and denote this by:

$$B \wedge T_d \vdash v$$

In [10], a simple modification of the usual Prolog interpreter is described that allows a retention of the proof-tree. The modification employs additional clauses B_M for proving Prolog clauses, and:

$$(B_M \wedge B \wedge T_d \vdash prove(v, P)) \equiv (B \wedge T_d \vdash v)$$

Here, $prove(e, P)$ denotes “ e can be proved using P ”. We will call B_M a meta-interpreter for clauses in $B \wedge T_d$, and P as the set of literals that are *TRUE* in a meta-interpretive proof for v (the elements of P are obtained from the goals, or negated literals, that resolve in a refutation-proof for v).

Programs are constructed by generalising the literals obtained in a meta-interpretive proof.

EXAMPLE 3.4. *A fragment of a document image is shown below:*

<pre> TO DNB NOR BANK ASA TRADE FINANCE/GUARANTEE DEPT. NO.0021 OSLO NORWAY NO </pre>	<pre> Please Quote in all correspondence 186FDBC1802472 </pre>
---	--

From such images we want to extract the reference number for correspondence. The neural-learner extracts primitive entities and relations as Prolog facts. The training instance provided to the program synthesis stage is the input-output pair $([d1], [186FDBC1802472])$, where $d1$ is the identifier of the image above. Program synthesis then is the result of the following steps:

- (1) A meta-interpretive proof P for $ts(([d1], [186FDBC1802472]))$ consists of the set of ground literals:

```

{trans(has_keyword('Please'), [d1], [d1, loc1]),
 has_line_below([d1, loc1], [186FDBC1802472])}

```
- (2) The literals in P are used construct a ground clause G (this is the “explanation” step):

```

ts(([d1], [186FDBC1802472])) ←
  has_keyword('Please'), [d1], [d1, loc1]
  has_line_below_word([d1, loc1], [186FDBC1802472])

```
- (3) The ground clause is generalised and with some trivial re-naming, resulting in the final program definition (this is the “generalization” step):

```

corr(A,B) :-
  has_keyword('Please', A,C),
  has_line_below(C,B).

```

Given domain constraints B , there can be many proofs and corresponding programs, for each training example v for a desired entity e . Meta-interpretive program synthesis, referred to from here on as $MIP(D, B, v)$, returns a set of programs to extract the value of the entity e from the database D populated by the neural learner. As we shall argue and demonstrate below, the same set of programs can also be used to extract the value of e from other similar documents.

In general $MIP(D, B, v)$ produces a number of programs given a single training instance, thus raising the difficulty of choice. One way to address that may reduce the number of possible explanations is to provide more examples, that add more constraints (we seek an explanation now for all the examples). Traditionally, EBG has preferred the specification of some extra-logical criterion for selection amongst multiple explanations. In an industrial setting, both of these options translate to requiring high-cost expertise. We describe next a one-shot learning with a form of “re-sampling” that is surprisingly effective.

4 ONE-SHOT LEARNING: NOISY CLONING

In general, providing more than one training instance should result in programs that is in some sense “more general”. Since our task is to extract entities in any document in a template-class (and not just one document in the class), it is important that the program synthesized applies to as many documents as possible in the class (we will discuss outliers later).

EXAMPLE 4.1. *An unsatisfactory program for extracting the number for correspondence is the one below:*

```

corr(A,B) :-
  has_keyword('Please', A,C),
  left_of('Please', C,D),
  right_of('ASA', D,C),
  has_line_below_word(C,B).

```

The program has unnecessary conditions left_of and right_of. It also uses the keyword 'ASA' which is part of the address to the left of the correspondence number. The address will change in other documents rendering the program incorrect for these.

One way of correcting the result of program synthesis is to keep adding example-pairs until the programs become correct. This method is data intensive and one would need to annotate multiple documents of a given template. Instead, we create additional examples automatically by “noisy cloning”, that is shown later to be surprisingly effective in practice.

A document d' is a noisy clone of a document d , if:

- (1) d and d' have the same template; and
- (2) d and d' have the same entities, but each entity in d has a different value to the entity in d'

Given a document d , we can obtain a noisy clone d' by simply altering the values of all its entities. This can be seen as a form of re-sampling with noise added to all entity values in d , resulting in two documents. Provided the the entity to be extracted from d

Algorithm 1 $TrainOS(A, I_{train}, B)$

Given: (1) A training example e consisting of a document image, I_{train} and the corresponding annotation, A for m entities of that document; (2) A database D ; and (3) Domain rules and facts B defined based on the database schema.

Find: A set of programs P for all the entities.

- 1: Populate the database $D = VisionAPI(I_{train})$
 - 2: **for** all $(f_i, v_i^{train}) \in A$ **do**
 - 3: Create a noisy clone, \widetilde{v}_i^{train}
 - 4: **end for**
 - 5: Create a noisy copy, \widetilde{D} for D by replacing v_i^{train} with \widetilde{v}_i^{train} in F .
 - 6: **for** $i := 1$ to m **do**
 - 7: Get a set of n_i programs,

$$\bigcup_{j=1}^{n_i} p_i^j := MIP(D, B, v_i^{train}) \cap MIP(\widetilde{D}, B, \widetilde{v}_i^{train})$$
 - 8: **end for**
 - 9: Return $P := \bigcup_{i=1}^m \bigcup_{j=1}^{n_i} p_i^j$
-

occurs only once in d , this form of re-sampling can assist in generalising programs identified using a single example. We propose two different procedures for program synthesis using *few-shot* learning:

4.0.1 *TrainOS.* Algorithm 1 corresponds to the *TrainOS* algorithm (corresponds to Train One-Shot) which requires just one document image, I_{train} and an annotation for all entities in that document, $\{(f_i, v_i^{train})\}_{i=1}^m$ to find programs for all entities (m of them) in the corresponding document template. Here, (f_i, v_i^{train}) corresponds to the i th entity-value pair. Given this annotation, we add some noise to each of the entity values and create a noisy counterpart which is then used as a second “training” instance.

Thus, we get two training examples corresponding to the training document and its noisy counterpart. Completing the proofs for these two examples and then taking the intersection for the two sets of logical programs that follow is equivalent to finding programs which can extract entities from both the documents.

4.0.2 *TrainNS.* One of the instances where the *TrainOS* fails to produce generalized programs are for entity values that occur at multiple locations in the document. In such scenarios, it becomes impossible for the system to disambiguate between these locations to get the actual position of the entity. The actual position of the entity may include all the positions where it occurs in the training document or just a subset of these positions. Figure 2 shows an example of such a situation. Notice that in figure 2a the date appears in three different locations. The actual position of the entity “date” is not evident from only this document. All three or less may be the actual location of this entity. But running the *TrainOS* algorithm would be assuming that the same entity always occurs at three different locations for any document of this template.

The only way to clear this ambiguity is to provide another document with an annotation for the corresponding entity. Figure 2b shows another example of the same template. This example clearly disambiguates between the three locations as now we are sure that the first location corresponds to a different entity. We are still not

Frankfurt, den 15.05.2005

LIQUIDATION

Für meine ärztliche Leistungen erlaube ich mir zu berechnen:

Patient(in): Alfred Stöger, geb. 10.11.1978
VNR: 753883420

EUR 37,52

Diagnose(n): 15.05.2005 Reizdarm, Rektum-Prolaps, Raynaud-Syndrom;
25.05.2005 Quinke-Ödem

Datum	Ziffer	(laut GOÄ - Vertrag)	Einfach	Faktor	Betrag
15.05.2005	3583 . B1	Harnsäure	2, 33	1, 15	2, 68
	3584 . B1	Harnstoff (Harnstoff -N, Bun)	2, 33	1, 15	2, 68
	3585 . B1	Kreatinin	2, 33	1, 15	2, 68

(a) The 3 dates marked in oval are same for this document.

Frankfurt, den 17.06.2005

LIQUIDATION

Für meine ärztliche Leistungen erlaube ich mir zu berechnen:

Patient(in): Paula Bergmann, geb. 29.11.1981
VNR: 5356753

EUR 50,20

Diagnose(n): 17.05.2005 Hämiplegie, Reiter-Syndrom, Rippenbruch

Ziffer	(laut GOÄ - Vertrag)	Einfach	Faktor	Betrag
3511	Untersuchung v. Körpermaterialie	2, 23	1, 15	3, 35
3532	Phasenkontrastmikroskop, Urinsediment	3, 24	1, 15	6, 03
298	Abstrichentnahme z. mikrobiolog. Unters.	1, 03	2, 30	5, 36
505	Atmungsunterweisung	4, 04	1, 50	6, 06

(b) The 2 dates marked in oval are the same but the one marked in the rectangle is different.

Figure 2: Location ambiguity

sure about the other two locations but giving more documents to the meta-interpreter will solve this issue.

We facilitate this by modifying the TrainOS algorithm to incorporate a “human-in-the-loop” approach, where a human is asked to annotate another document from a set, l of supplementary documents ($I_{supp} = \bigcup_{i=1}^l I_i$) whenever an entity occurs more than once. Therefore, until there is complete disambiguity in the entity locations, a human will be asked to feed in more annotated documents. Note that this annotation is done only for the one, ambiguous entity. If the human annotates k different documents ($0 \leq k \leq l$) and these documents are also used for training (i.e. each document corresponds to an additional training instance), line 7 of algorithm 1 becomes:

$$\bigcup_{j=1}^{n_i} p_i^j := \left(\bigcap_{o=1}^k MIP(D_o, B, v_i^o) \right) \cap MIP(D, B, v_i^{train}) \cap MIP(\tilde{D}, B, \widetilde{v_i^{train}}) \quad (1)$$

Here, each annotated document, o will correspond to an additional document D_o .

We call this extension of TrainOS algorithm the *TrainNS* algorithm (corresponds to Train N-Shot).

4.1 Entity extraction

Once we get the programs for all entities in a document template, we extract an entity from a new document (not used for training) using algorithm 2.

For the i th entity f_i in a document template, we get n_i different programs. We run each of these programs and store their outputs. After this, a *majority voting* technique is used wherein we take the most frequently produced output as the correct output. If the most frequent output is “NULL” (which may happen in case a program does not return anything), then we consider the second most frequent output as correct.

5 RESULTS

We use the following publicly available datasets for testing our system and benchmarking results:

Algorithm 2 *Extraction*(I_{test}, P, f_i)

Input: A document image, I_{test} , the set of all programs P obtained for the corresponding template and the entity to be extracted f_i .

Output: The entity value v_i^{test} corresponding to f_i .

- 1: Populate the database $D = VisionAPI(I_{test})$.
 - 2: **for** all $p_i^j \in P$ **do**
 - 3: $o_j := p_i^j(f_i)$
 - 4: **end for**
 - 5: Get the set of distinct outputs, $\bigcup_{j=1}^K o_k$ from $\bigcup_{j=1}^{n_i} o_j$ sorted by their frequency of occurrence (decreasing order). Here, $K \leq n_i$.
 - 6: **if** $o_1 = NULL$ **then**
 - 7: $v_i^{test} := o_2$
 - 8: **else**
 - 9: $v_i^{test} := o_1$
 - 10: **end if**
 - 11: Return v_i^{test}
-

- (1) *Doctor’s Bills dataset*: This dataset is a collection of invoices for medical bills ([27]). It comprises of two different templates which we call Doctor-1 and Doctor-2 that have a total of 50 and 40 documents respectively. For each template, we keep aside 5 documents in the *training pool* and the rest for testing. We do manual annotation for each of the templates and mark 8 entities for Doctor-1 and 9 entities for Doctor-2 which are to be extracted.
- (2) *Patent dataset*: This dataset is a part of the *Ghega dataset* [13]. It comprises of 136 patent forms from 10 different templates with annotations for different entities. For 8 of the templates, we keep 3 documents in the training pool and the rest for testing. We use 2 and 1 document(s) in the training pool respectively for two other templates as they contained a very few documents.

Figure 4 shows a document sample for each type.

We select a document from the training pool for one-shot learning and the rest for use as supplementary documents for running TrainNS algorithm. This is done multiple times such that each document in the training pool is used in one-shot learning once. The average extraction accuracy for every entity is reported in table 2.

Doctor-1			Doctor-2			Patent		
Entity name	TrainOS	TrainNS	Entity name	TrainOS	TrainNS	Entity name	TrainOS	TrainNS
Date-1	91.00	91.00 (0)	Date-1	81.71	100.00 (1)	Classification No.	77.67	77.67 (0)
Date-2	91.11	91.11 (0)	Date-2	58.28	90.00 (2)	Abstract	68.50	68.50 (0)
Amount-1	85.33	85.33 (0)	Amount	80.00	100.00 (1)	Applicants name	86.67	86.67 (0)
Amount-2	60.40	100.00 (2)	Invoice no.	100.00	100.00 (0)	Application No.	77.50	77.50 (0)
Patient name	92.00	92.00 (0)	Patient name	100.00	100.00 (0)	Representative	74.33	74.33 (0)
Patient address	97.78	97.78 (0)	Patient address	100.00	100.00 (0)	Title	78.25	78.25 (0)
Diagnosis	100.00	100.00 (0)	Diagnosis	79.43	79.43 (0)	Publication date	81.30	81.30 (0)
Ref. no.	100.00	100.00 (0)	VNR no.	100	100 (0)	Inventors name	83.33	83.33 (0)
NA	NA	NA	DOB	100	100 (0)	Filing date	70.00	70.00 (0)

Table 2: Entity extraction accuracy on 3 different public datasets (in %) for TrainOS and TrainNS approaches for program generation. The number in brackets gives the number of supplementary documents (on average) required.

Entity name	TrainOS
Account no.	100.00
Addressee	100.00
Amount	100.00
Contract no.	100.00
Correspondence no.	100.00
Drawee	100.00
Drawer	100.00
Tenor	67.67

Table 3: Extraction accuracy (in %) on proprietary data.

From the results, it is clear that for most entities (entities for which the TrainNS algorithm requires 0 supplementary documents to be annotated), TrainOS algorithm is sufficient to obtain good extraction accuracies. For the cases where TrainNS algorithm gives better results, notice that the difference between extraction accuracies of TrainNS and TrainOS quite high (as high as 40%). This is evidencial of the fact that location ambiguity is indeed a significant problem and the TrainNS algorithm alleviates this to a large extent. Also, note that even for such cases, the human had to annotate no more that 2 documents from a supplementary document pool size of 4 which cannot be considered a big overhead.

Although a good amount of work has been done to solve the problem of information extraction from document images, results by most of them are on proprietary documents which makes it impossible to test our method against a baseline. To the best of our knowledge, there is only one benchmarking results available on the Patent dataset by Ref. [13] who use a probabilistic approach to solve the problem. Their task is slightly different from ours in that they detect the bounding box sequence for a given entity whereas we extract the exact value of the entity. Using one document, they report the average success-rate to be 50% whereas our one-shot method obtains an average accuracy of 77%. (However, with 14 documents though, they achieve a success-rate of 90% in their task of bounding-box detection).

Table 4 gives three examples of programs generated for Doctor-2 template and their interpretations. Each step of a program is completely interpretable unlike the intermediate steps (layers) in a

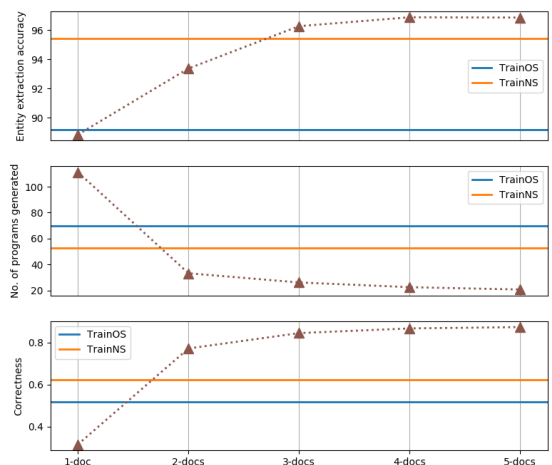


Figure 3: Effect of increasing training size on performance.

deep neural network. This makes debugging of the system fairly simple which is of prime importance in any real-world deployment.

As an evidence of real-world application of our system, we also give results on one proprietary document template in table 3. For this, we were given just one document for training and testing was done on three other documents of the same template. Note that we do not give results for the TrainNS approach as we had just one document for training.

6 PERFORMANCE ANALYSIS

We do a two-part analysis of our system performance. The first part pertains to the effect of the number of training documents on three different performance metrics. In the second part, an error analysis is done.

Program	Interpretation
date1(A,B):- get_blockid('Diagnosen',A,C), get_block_above(0,C,D), get_substring('Rechnungsdatum','Bitte',1,D,B).	C = The block with the word "Diagnosen" D = The block above C with index 0 B = Output (String in D between the two words in args)
diagnosis(A,B):- get_line('Diagnosen',A,C), get_keyword('<medical_term>',C,B).	C = The line with the word "Diagnosen" B = Output (String in C which is a medical term)
ref(A,B):- word_to_left('Rechnungsdatum',A,B).	B = Output (Word to the left of 'Rechnungsdatum')

Table 4: Examples of programs generated by the meta-interpreter.

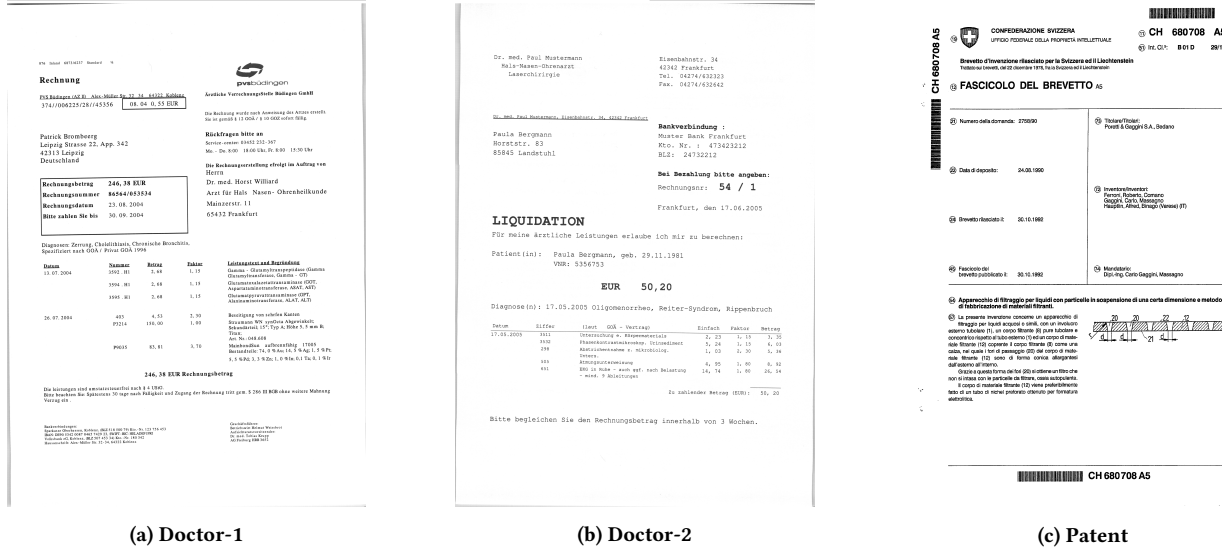


Figure 4: Sample document images

6.1 Effect of training size

As explained in section 4, the meta-interpreter has a potential to produce more generalised programs if more training examples are used to generate the programs. In other words, if the number of training documents are increased, we can expect the performance to get better. Thus, we experimented with this idea by providing the meta-interpreter with as high as 5 documents for training on the doctor's bills dataset (as we found that this has more documents per template compared to the patent dataset) and evaluated its performance on three metrics. We used all combinations of n documents ($1 \leq n \leq 5$) and the average of their performance on the test set is shown in figure 3.

- (1) *Extraction accuracy*: This is the percentage of times the correct output is extracted. The extraction follows algorithm 2. The first plot of figure 3 shows this performance and it is clear from this plot that providing more training examples indeed makes the performance better. Also note that the extraction accuracy of the TrainNS algorithm is very close in performance to the best possible performance.
- (2) *Number of programs generated and correctness*: Correctness is the fraction of programs that give the correct output and its variation with training size is given in the third plot of figure 3. As we increase the training size, the number of

programs generated by the meta-interpreter tends to fall rapidly as shown by the second plot of figure 3. This is a consequence of the fact that as the training size increases, the meta-interpreter produces only the most general programs and hence the correctness score also increases.

Note from the plot that both the TrainOS and TrainNS algorithms produce some non-general programs (their correctness values being close to 0.5 and 0.6 respectively). This we observed is because there are almost always some differences in the documents even within the same template. These differences may either be due to some noisy images or due to some minor formatting variations in the documents which effect the output of the VisionAPI. Introducing noise in such cases is not a complete solution for non-generality as the observation mentioned in section 4 holds true for documents of the same template.

We observed that the extraction accuracy and the correctness curves of figure 3 reach a plateau as the number of training documents increase and in fact for some combination of the 5 documents, there is a dip in these scores. This, we observe is because as the number of documents increase, there is a high probability that at least one of the documents is noisy. When this happens, it becomes impossible for the meta-interpreter to come up with programs that work for both the noisy and the non-noisy training instances.

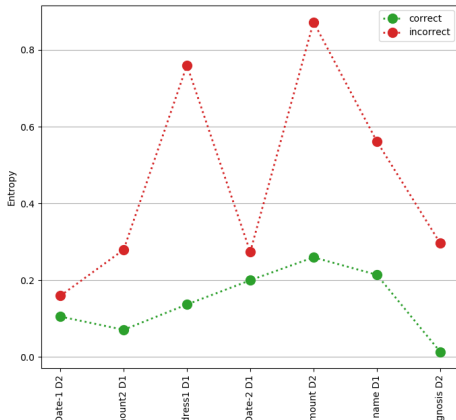


Figure 5: Variation of entropy of different output distributions for correct and incorrect extractions.

Also note that there is a vast improvement in the correctness score when using just one document for training versus using TrainOS algorithm. This suggests that one-shot learning by using a noisy clone does improve performance (an absolute improvement as high as 0.5 on the correctness score) by narrowing down the output to the most general programs.

6.2 Error analysis

6.2.1 Source of errors. We observe that most of the errors in the extraction of an entity by one-shot learning is due to one of the following reasons:

- (1) *Ambiguity in entity location:* These errors are mostly tackled when we use supplementary documents and run the TrainNS algorithm. This is evidenced in table 2 for cases where TrainNS accuracy is higher than TrainOS.
- (2) *Inconsistencies in the output of the VisionAPI:* As discussed before, these errors occur when a test image is either noisy or there are some formatting differences in the training and testing images. In such cases, we essentially have different templates during train and test time. Such errors have to be fixed by making the VisionAPI more robust which can be an interesting direction for future work.

6.2.2 Erroneous outputs are identifiable. Any deployable system needs to be such that its debugging does not produce overheads. Hence, it is of prime importance that a failure instance is identifiable. In our case, these instances fall into one of the two categories: (1) *No output extracted* (2) *Wrong output extracted*. From a debugging point of view, errors that fall into category (1) are more desirable than (2). But we find the even the instances that produce a wrong output are identifiable to some extent.

For this, we calculate the entropy of the distribution over all distinct outputs produced by the programs for the cases when the extraction is either correct or incorrect but not for the cases when there is no extraction. This is done for every entity in the doctor’s dataset and we show the plot for this in figure 5. From this, we

can clearly see that for correct output cases, the entropy values are always lower suggesting a more sparse distribution over the distinct outputs. Also, for every entity there is a visible difference between entropies of correct and incorrect predictions which suggests that using a suitable threshold for the entropy, incorrect extractions can be identified.

7 RELATED WORK

Information Extraction from documents is a well established and explored field and a large body of work is available for this domain. But most of these works have focussed on structured or semi-structured documents rather than on document images. One such system is the RAPIER system [14] which induces a pattern-match for entities using relational learning. This method is similar to ILP but is more constrained in terms of the rules it generates as it uses a slot-filling framework. On the other hand, our method works on document images and is diverse as it generates programs which, in principle, may use any number of spatial relationships. Another similar system is WHISK [25] which learns regular expression like rules to extract patterns and like RAPIER, it works on semi-structured text rather than on document images. Effective extraction techniques have been proposed using shallow domain knowledge like in Ref. [4]. They introduce the $(LP)^2$ system which uses some domain knowledge to insert SGML tags on relevant entities. It then uses a correction mechanism to fine-tune the results. In contrast, our proposed method does not require any domain knowledge except the information in the documents themselves.

Efficiently extracting information from document images has been of some interest mainly for industrial applications. Ref. [7] have developed a system by graphically modeling relationships between words in the form of document models. They also use a predefined set of keywords and document models. Another approach particularly for templated documents has been of learning a spatial structure from one part of a document and using this to extract entities from some other part [3]. A method similar to ours, that uses a single document example to learn about spatial relations is proposed in Ref. [23]. This method represents word relations using a continuous polar coordinate system and rely on these for indexing different entities. This makes it sensitive to minor spatial variations. Another such real-world system is *Intellix* [24]. This system uses categorization of entities based on position and context to come up with rules for extraction. Most of these works, to the best of our knowledge, do not benchmark results on public datasets which makes it difficult to validate results.

A reader familiar with Inductive Logic Programming (ILP) will find a relationship of our program synthesis approach to the techniques used there. This is not surprising, and EBG was shown to be a special case of ILP in [15] given a single example. The relationship to ILP extends, albeit more tenuously, to recent work on meta-interpretive learning (MIL) in that field. In MIL, proofs constructed by a meta-interpreter using meta-rules in a higher-order logic are used to instantiate first-order logic programs. Computation of outputs from inputs can be seen as repeated applications of one or more meta-rules, instantiated appropriately with predicates and terms from B . Both standard ILP and MIL systems are more powerful (and complex to use) than what is needed for our purpose.

Logic programming is used by Ref. [1] to characterize the geometric properties of text units to reason over relationships between parts of a document. This approach is used in document understanding context. MIL ([16],[17]) has regained popularity in the recent past primarily because of its high interpretability and data efficiency. MIL has even proved to be much more robust as compared to many deep learning methods for certain applications in vision domain ([6],[5]). A particularly popular recent successful application of MIL in the one-shot learning context has been reported in Ref. [12] where the authors replicate results of Microsoft’s Flashfill by using one-shot MIL.

ILP for Information Extraction has also been explored in the past by Ref. [21] who have use ILP for extracting useful features. These features are then used to train machine learning models like SVM. It has been shown that ILP can come up with features that have exploitable signals to be used by the SVM. Another work which uses ILP for IE ([8]) defines only three primitive predicates and uses a separate and conquer strategy to come up with rules for extraction. There is an underlying assumption in this approach that a document is a list of words. This makes it unsuitable in domains where two dimensional relationships between words are important as in templated documents like invoices and forms. Our system incorporates relations like “above”, “below” etc. and uses meta-interpretive program synthesis which makes it novel.

Our work extends the work in Ref. [2] in the following ways. First, it shows that meta-interpretive learning of transition systems can be used beyond the biological domain used in that paper. Further, unlike in [2], we do not allow invention of new transitions, and our transition system is not probabilistic.

8 CONCLUSION

In this paper, we presented a new approach for synthesizing programs that extract information from document images. To the best of our knowledge, this is the first attempt in combining Deep Learning based computer vision APIs with a Meta-Interpretive Learning based framework. Our approach is highly data efficient and in most cases requires just one document image to generalize well while requiring no more than three for a few cases. This is evidenced by results given on publically available datasets which can serve as a benchmark for future work. One-shot learning is highly indispensable in industrial scenarios where there is a limit on the proprietary documents that one can use for training modern deep learning based models. We have also shown that with the introduction of a few documents, our system can reach near perfect performance which is a direct consequence of using an Inductive Logic Programming (ILP) based approach for program synthesis.

Furthermore, after thorough analysis of the results, we conclude that there is still a lot of scope for improvement particularly in making the vision API more tolerant to noise and formatting variations in documents. Another direction worth pursuing in this context is to synthesize programs that are template invariant such that a part of the program is a template identifier. This would of course require incorporation of certain template specific rules in our current framework. Even with this minor limitation, our system is highly robust and scalable to large-scale industrial applications. Finally,

we have also begun applying the same approach for very different document types, such as engineering drawings.

REFERENCES

- [1] Weronika T Adrian, Nicola Leone, Marco Manna, and Cinzia Marte. 2017. Document Layout Analysis for Semantic Information Extraction. In *Conference of the Italian Association for Artificial Intelligence*. Springer, 269–281.
- [2] Michael Bain and Ashwin Srinivasan. 2018. Identification of biological transition systems using meta-interpreted logic programs. *Machine Learning* 107, 7 (2018), 1171–1206.
- [3] Evgeniy Bart and Prateek Sarkar. 2010. Information extraction by finding repeated structure. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*. ACM, 175–182.
- [4] Dr Ciravegna et al. 2001. Adaptive information extraction from text by rule induction and generalisation. (2001).
- [5] Wang-Zhou Dai, Stephen Muggleton, Jing Wen, Alireza Tamaddoni-Nezhad, and Zhi-Hua Zhou. 2017. Logical vision: One-shot meta-interpretive learning from real images. In *International Conference on Inductive Logic Programming*. Springer, 46–62.
- [6] Wang-Zhou Dai, Stephen H Muggleton, and Zhi-Hua Zhou. 2015. Logical Vision: Meta-Interpretive Learning for Simple Geometrical Concepts. In *ILP (Late Breaking Papers)*. 1–16.
- [7] Yasuto Ishitani. 2001. Model based information extraction and its application to document images. In *Workshop on document layout interpretation and its applications, DLIA*.
- [8] Markus Junker, Michael Sintek, and Matthias Rinck. 1999. Learning for text categorization and information extraction with ILP. In *International Conference on Learning Language in Logic*. Springer, 247–258.
- [9] Dimosthenis Karatzas, Faisal Shafait, Seiichi Uchida, Masakazu Iwamura, Lluís Gomez i Bigorda, Sergi Robles Mestre, Joan Mas, David Fernandez Mota, Jon Almazan Almazan, and Lluís Pere De Las Heras. 2013. ICDAR 2013 robust reading competition. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE, 1484–1493.
- [10] Smadar T Kedar-Cabelli and L Thorne McCarty. 1987. Explanation-based generalization as resolution theorem proving. In *Proceedings of the fourth international workshop on machine learning*. Elsevier, 383–389.
- [11] Iuliu Konya. 2012. Adaptive methods for robust document image understanding. (2012).
- [12] Dianhuan Lin, Eyal Dechter, Kevin Ellis, Joshua B Tenenbaum, and Stephen H Muggleton. 2014. Bias reformulation for one-shot function induction. (2014).
- [13] Eric Medvet, Alberto Bartoli, and Giorgio Davanzo. 2011. A probabilistic approach to printed document understanding. *International Journal on Document Analysis and Recognition (IJ DAR)* 14, 4 (2011), 335–347.
- [14] R Mooney. 1999. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Vol. 334.
- [15] Stephen Muggleton and Luc De Raedt. 1994. Inductive logic programming: Theory and methods. *The Journal of Logic Programming* 19 (1994), 629–679.
- [16] Stephen H Muggleton, Dianhuan Lin, Niels Pahlavi, and Alireza Tamaddoni-Nezhad. 2014. Meta-interpretive learning: application to grammatical inference. *Machine learning* 94, 1 (2014), 25–49.
- [17] Stephen H Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. 2015. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning* 100, 1 (2015), 49–73.
- [18] Dário Augusto Borges Oliveira and PM Viana. [n. d.]. Fast CNN-based document layout analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*. 1173–1180.
- [19] Gordon D Plotkin. 2004. The origins of structural operational semantics. *The Journal of Logic and Algebraic Programming* 60 (2004), 3–15.
- [20] Rohit Rahul, Gunjan Sehgal, Arindam Chowdhury, Monika Sharma, Lovekesh Vig, Gautam Shroff, Ashwin Srinivasan, et al. 2018. Deep Reader: Information extraction from Document images via relation extraction and Natural Language. *arXiv preprint arXiv:1812.04377* (2018).
- [21] Ganesh Ramakrishnan, Sachindra Joshi, Sreeram Balakrishnan, and Ashwin Srinivasan. 2007. Using ilp to construct features for information extraction from semi-structured text. In *International Conference on Inductive Logic Programming*. Springer, 211–224.
- [22] Rizlene Raoui-Outach, Cecile Million-Rousseau, Alexandre Benoit, and Patrick Lambert. 2017. Deep Learning for automatic sale receipt understanding. In *Image Processing Theory, Tools and Applications (IPTA), 2017 Seventh International Conference on*. IEEE, 1–6.
- [23] Marçal Rusinol, Tayeb Benkhelfallah, and Vincent Poulain dAndecy. 2013. Field extraction from administrative documents by incremental structural templates. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE, 1100–1104.

- [24] Daniel Schuster, Klemens Muthmann, Daniel Esser, Alexander Schill, Michael Berger, Christoph Weidling, Kamil Aliyev, and Andreas Hofmeier. 2013. Intellix-End-User Trained Information Extraction for Document Archiving. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE, 101–105.
- [25] Stephen Soderland. 1999. Learning information extraction rules for semi-structured and free text. *Machine learning* 34, 1-3 (1999), 233–272.
- [26] Zhi Tian, Weilin Huang, Tong He, Pan He, and Yu Qiao. 2016. Detecting text in natural image with connectionist text proposal network. In *European conference on computer vision*. Springer, 56–72.
- [27] Joost Van Beusekom, Faisal Shafait, and Thomas M Breuel. 2008. Document signature using intrinsic features for counterfeit detection. In *International Workshop on Computational Forensics*. Springer, 47–57.
- [28] Yufei Wang. 2017. *Deep Learning for Image Understanding*. University of California, San Diego.
- [29] Doris Xin, Litian Ma, Jialin Liu, Stephen Macke, Shuchen Song, and Aditya Parameswaran. 2018. Accelerating Human-in-the-loop Machine Learning: Challenges and Opportunities. *arXiv preprint arXiv:1804.05892* (2018).

A SUPPLEMENTARY MATERIAL

A.1 Computer Vision Components of the VisionAPI

Besides the identification of the textual entities mentioned in section 3, we also perform the following additional visual pre-processing.

A.1.1 Image Alignment. To correct for documents that were imperfectly aligned, we detect the bounding box around all of the high intensity pixels in the image and correct for any angular shift

A.1.2 Image De-Noising. We also address the issue of degradation in quality of images due to camera shake, improper focus, imaging noise, coffee stains, wrinkles, low resolution, poor lighting, or reflections. These kind of problems drastically affect the performance of many computer vision algorithms like text detection, OCR and localization. The objective here is to reconstruct high-quality images directly from noisy inputs and also to preserve the highly structured data in the images. We do this via an impementation of cyclic GANs, details of which may be found in [20].

A.1.3 VisionAPISchema. Once all the entities are identified as mentioned in section 3, relations between the entities need to be populated and stored in the database. So a schema should be designed to facilitate information extraction. All the entities are associated with their spatial coordinates and this information conveys the whereabouts of the neighbouring text entities. This information is then used to infer different logical and spatial relationships.

Figure 6 shows the representation of this schema populated in the database after the relevant relationships have been extracted from the raw image. The main entities of the schema includes words, lines, text blocks, boxes and tables. The inter and intra entity relationships have been illustrated by the directions of the arrow. The schema may get richer over time, we have only highlighted the entities that are useful for scanned document images at the moment.

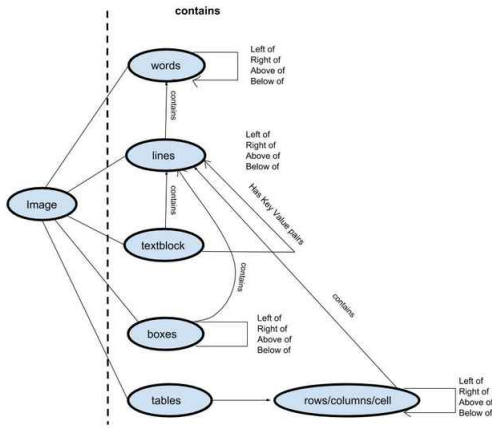


Figure 6: VisionAPI Schema

A.2 Correctness of Program Synthesis

Given the definition T_d of a transition system as in Remark 3.3; a set of clauses B ; and ground terms i, o . Let t_1, t_2, \dots, t_k denote *trans*/3

literals in a meta-interpretive proof for $ts((i, o))$. For simplicity, we will assume that the t_i are ground (in general, they may contain existentially-quantified variables, and a Skolemisation step will be needed for what follows). Then, from the properties of the meta-interpreter and SLD-resolution we know $B \wedge T_d \models (t_1 \wedge t_2 \wedge \dots \wedge t_k)$. Let $G : ts((i, o)) \leftarrow t_1, t_2, \dots, t_k$ be a ground clause. Then from the semantics of modus ponens, it follows that $B \wedge G \models ts((i, o))$. G is called a ground explanation for $ts((i, o))$. Let H be a clause that θ -subsumes G . It follows from the properties of θ -subsumption that $B \wedge H \models ts((i, o))$. H is called a generalized explanation for $ts((i, o))$.

A.3 Extract of Prolog code for the deductive reasoning module

```

MIP(D):- % finds all possible programs with depth D
clean_up,
set(cwa,true), %flag - ignore
example(pos,Name,_,_[Si,Sf]), % training example
MIP_do((Si,Sf),Name,D),
fail.
MIP(_,_).

MIP_do(S,Name,DepthBound):-
S = (Input,Output),
cts(S,DepthBound,Trace),
% prove transition for example
trace_to_func(S,Name,Trace,Func), % generalization step
check_soundness(Name,Input,Func),
% check that program derives other examples
check_completeness(Name,_,Func),
% check that program derives nothing else
update_cache(Func). % store the program in a cache

% depth-bounded transition system
cts((S,S),D,[]):- D >= 0.
cts((Si,Sf),D,[trans(T,Si,S)|Rest]):-
D >= 1,
D1 is D - 1,
transition(T), % domain transition
trans(T,Si,S),
cts((S,Sf),D1,Rest).

```
