Synthesis and Machine Learning for Heterogeneous Extraction

Arun Iyer
Microsoft Research, Bangalore, India
ariy@microsoft.com

Manohar Jonnalagedda*
Inpher, Lausanne, Switzerland
manohar.jonnalagedda@gmail.com

Suresh Parthasarathy
Microsoft Research, Bangalore, India
supartha@microsoft.com

Arjun Radhakrishna
Microsoft, Bellevue
arradha@microsoft.com

Sriram Rajamani
Microsoft Research, Bangalore, India
sriram@microsoft.com

Abstract

We present a way to combine techniques from the program synthesis and machine learning communities to extract structured information from heterogeneous data. Such problems arise in several situations such as extracting attributes from web pages, machine-generated emails, or from data obtained from multiple sources. Our goal is to extract a set of structured attributes from such data.

We use machine learning models ("ML models") such as conditional random fields to get an initial labeling of potential attribute values. However, such models are typically not interpretable, and the noise produced by such models is hard to manage or debug. We use (noisy) labels produced by such ML models as inputs to program synthesis, and generate interpretable programs that cover the input space. We also employ type specifications (called "field constraints") to certify well-formedness of extracted values. Using synthesized programs and field constraints, we re-train the ML models with improved confidence on the labels. We then use these improved labels to re-synthesize a better set of programs. We iterate the process of re-synthesizing the programs and re-training the ML models, and find that such an iterative process improves the quality of the extraction process. This iterative approach, called HDEF, is novel, not only the in way it combines the ML models with program synthesis, but also in the way it adapts program synthesis to deal with noise and heterogeneity.

More broadly, our approach points to ways by which machine learning and programming language techniques can be combined to get the best of both worlds — handling noise, transferring signals from one context to another using ML, producing interpretable programs using PL, and minimizing user intervention.

1 Introduction

Extracting structured attributes from heterogeneous unstructured or semi-structured data is an important problem, which arises in many situations. One example is processing websites in domains such as travel, shopping, and news and extracting specific attributes from them. Another example is in processing machine generated emails in such domains, and extracting specific attributes. A third example is data wrangling where the goal is to transform and map raw data to a more structured format, with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.

In the ML community, these problems have been handled by training ML models. While impressive progress has been made in making use of signals from noisy and large scale data [12, 23, 24, 26], the models produced are not interpretable and hence hard to maintain, debug and evolve. In the PL community, program synthesis has been used to generate programs, such as Excel macros, from a small number of training examples [7, 13, 18]. If the data-sets are large and heterogeneous, and training data is small and noisy, neither ML models nor program synthesis work well in isolation. In this paper, we show an approach to combine these ideas from the ML and PL communities to perform extraction from heterogeneous data.

Our approach, called HDEF, *Heterogeneous Data Extraction Framework*, works by first training an ML model (such as a conditional random field) using some training data. We use this base ML model to produce candidate output labels for the entire heterogeneous data-set, including on input formats for which there is no training data. Due to the generalization, the base ML model produces output labels even for the formats with no training data. However, the labels so generated are typically noisy. We then use these noisy labels as input specifications to a modified program synthesis algorithm. Since the input data is heterogeneous, a single program cannot handle all the inputs. Our synthesis algorithm, called NoisyDisjSyn, *Noisy Disjunctive Program Synthesis*, produces a set of programs that *cover* the entire input data-set and maximizes the number of inputs for which correct outputs are produced, where correctness of an output is defined using a type specification called *field constraint*. These base programs partition the data-set into clusters, where each program works for inputs from one cluster. Our approach

*This work was done when the author was at Microsoft Research, Bangalore, India.

PLDI 2019, June 22–26, 2018, Phoenix, AZ
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

works by iteratively refining both the ML model and synthesized programs using each other. We use the synthesized programs to generate output labels for all the inputs. Next, we compare the outputs from ML models and synthesized programs to vote on the inferred labels –increasing confidence on the labels in cases where there is agreement and picking a winner in a semi-automated manner when there is disagreement. We then re-train the ML model with the revised confidence on the labels, and use the ML model's output to re-synthesize a better set of programs. We iterate the process of re-synthesizing the programs and re-training the ML models, and find that such an iterative process improves the quality of the extraction process, while minimizing the user intervention needed.

We use state-of-the art ML models such as Conditional Random Fields (CRFs) and Bidirectional Long Short-Term Memory augmented with CRFs (LSTM-CRFs) as ML models. Our program synthesis needs to handle both noisy labels, as well as heterogeneous inputs. We use a novel disjunctive program synthesis algorithm to generate a set of programs (rather than a single program) to deal with heterogeneity. In cases where the ML model and synthesized programs disagree in their output, we use ranking techniques or human annotation to pick a winner. For example, when a highly ranked program (which works over many inputs in a format) produces an output which is different from the ML model, we pick the program output as the winner. In cases where neither the program nor the ML model's output is highly ranked, we seek user intervention.

This paper makes the following contributions:

- We present a novel iterative algorithm HDEF which interleaves ML model training (Section 4) and program synthesis to reduce noise from ML models and produce interpretable programs (Sections 3, 6).
- We also present a novel disjunctive synthesis algorithm NoisyDisjSynth (used by the HDEF algorithm as a subroutine), which takes noisy input-output examples as inputs, and produces a set of programs that handle heterogeneous inputs from different formats (Section 5).
- We show how type specifications and appropriately designed Domain Specific Languages can greatly improve the performance of the HDEF approach (Sections 5.3, 5.4, and 7.3).
- We present a thorough evaluation of the approach on two data-sets (Section 7). We show that our approach is able to improve the quality of the extraction process (as measured by F-1 score) in 24 of the 30 extraction tasks, as compared to a plain ML model based extraction. In several cases, precision and recall improves dramatically from below $0.3-0.4$ to close to the perfect 1.0 score. Further, in the cases where the quality did not improve, we are able to qualitatively characterize the extraction task that leads to failure. Additionally, for the political data-set, we perform ablation studies to demonstrate the significance of the choice of DSL and the importance of field constraints.

2 Overview

We first present two exemplar data-sets that motivate the design of our data-extraction framework. Then we present an overview of our extraction approach.

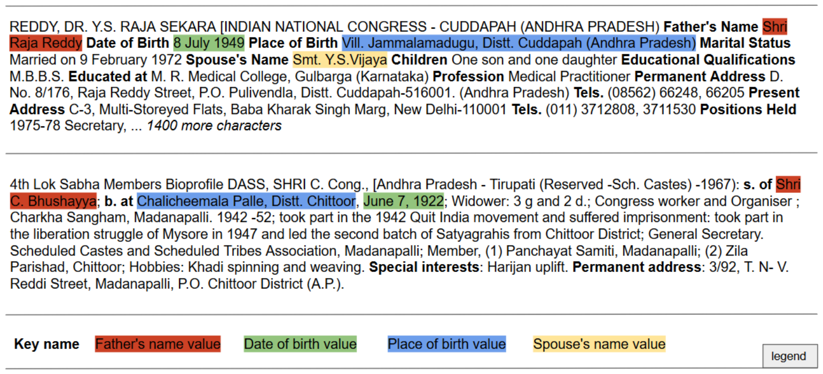
2.1 Exemplar Data-sets

Political Data-set. This data-set consists of 605 individual string inputs, each containing the personal details of candidates in the Indian parliamentary elections. Two example inputs are shown in Figure 1. Each string (formatted in an ad-hoc manner) contains information such as the candidate's name and date of birth, as well as other unstructured textual descriptions of "Other information", "Special Interests" and "Positions held". Further, the data has undergone significant *bit rot*: a fraction of the strings are missing the new line characters, etc. The data-set has the following characteristics:

- Fields are commonly specified using an indicator prefix (that we call *key*). However, the same field can have different keys in different inputs. In the figure, the father's name field has two separate keys: "Father's name: " and "s of.".
- Some fields have no clear keys. In the second input, the date of birth field has no specific key, but instead the place of birth field and the date of birth field are together prefixed by the key "b. at".
- There can be fields missing in each input. In the second input, the field spouse's name has no value.
- There are no fixed field separators and field values can be large amounts of "unstructured text" (e.g., the value of "Social activities" field in the second input).

M2H Flight Email Data-set. This data-set consists of automated *machine-to-human* emails related to flight reservations, cancellations, etc. These emails are sent automatically by airlines and travel agencies (e.g. Expedia, Orbitz, Oman Air, etc), each of which we refer to as a *domain*. Two examples of such emails are shown in Figure 2. Here, the task is to automatically extract details such as passenger names, flight numbers, origin, destination, etc. The following aspects of this data-set makes extraction difficult:

- *Long tail of formats.* Each domain corresponds to a few formats – for example, one for reservation confirmation, cancellation, etc. However, the number of these formats is quite large with a large fraction of the emails coming from a small number of *leading domains*, and a large number of *tail domains* having a small number of emails from them.
- *Variation within formats.* While each format follows a high-level template, there is significant variation even within formats. For example, even for flight reservation confirmation emails from a single domain, there are variations based on the number of passengers, the

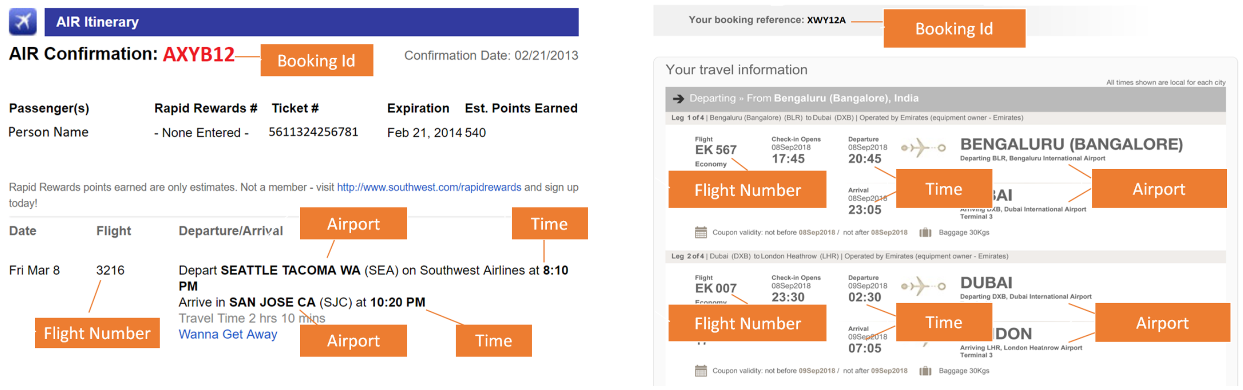


REDDY, DR. Y.S. RAJA SEKARA [INDIAN NATIONAL CONGRESS - CUDDAPAH (ANDHRA PRADESH) **Father's Name** Shri Raja Reddy **Date of Birth** 8 July 1949 **Place of Birth** Vill. Jammalamadugu, Distt. Cuddapah (Andhra Pradesh) **Marital Status** Married on 9 February 1972 **Spouse's Name** Smt. Y.S.Vijaya **Children** One son and one daughter **Educational Qualifications** M.B.B.S. **Educated at** M. R. Medical College, Gulbarga (Karnataka) **Profession** Medical Practitioner **Permanent Address** D. No. 8/176, Raja Reddy Street, P.O. Pulivendla, Distt. Cuddapah-516001. (Andhra Pradesh) **Tels.** (08562) 66248, 66205 **Present Address** C-3, Multi-Storeyed Flats, Baba Kharak Singh Marg, New Delhi-110001 **Tels.** (011) 3712808, 3711530 **Positions Held** 1975-78 Secretary, ... *1400 more characters*

4th Lok Sabha Members Bioprofile DASS, SHRI C. Cong., [Andhra Pradesh - Tirupati (Reserved -Sch. Castes) -1967): **s. of** Shri C. Bhushayya; **b. at** Chalicheemala Palle, Distt. Chittor, June 7, 1922; Widower: 3 g and 2 d.; Congress worker and Organiser ; Charkha Sangham, Madanapalli. 1942 -52; took part in the 1942 Quit India movement and suffered imprisonment: took part in the liberation struggle of Mysore in 1947 and led the second batch of Satyagrahis from Chittoor District; General Secretary. Scheduled Castes and Scheduled Tribes Association, Madanapalli; Member, (1) Panchayat Samiti, Madanapalli; (2) Zila Parishad, Chittoor; Hobbies: Khadi spinning and weaving. **Special interests**: Harijan uplift. **Permanent address**: 3/92, T. N- V. Reddi Street, Madanapalli, P.O. Chittoor District (A.P.).

Key name 🟥 Father's name value 🟩 Date of birth value 🟦 Place of birth value 🟨 Spouse's name value legend

Figure 1. Two inputs from the political data-set. Keys and values are highlighted as per the legend in the third row



AIR Itinerary

AIR Confirmation: AXYB12 Confirmation Date: 02/21/2013

Passenger(s)	Rapid Rewards #	Ticket #	Expiration	Est. Points Earned
Person Name	- None Entered -	5611324256781	Feb 21, 2014	540

Rapid Rewards points earned are only estimates. Not a member - visit http://www.southwest.com/rapidrewards and sign up today!

Date	Flight	Departure/Arrival
Fri Mar 8	3216	Depart **SEATTLE TACOMA WA** (SEA) on Southwest Airlines at **8:10 PM** Arrive in **SAN JOSE CA** (SJC) at **10:20 PM** Travel Time 2 hrs 10 mins Wanna Get Away

Your booking reference: **XWY12A**

Your travel information

All times shown are local for each city

Departing » From **Bengaluru (Bangalore), India**

Leg 1 of 4 | Bengaluru (Bangalore) (BLR) to Dubai (DXB) | Operated by Emirates (equipment owner - Emirates)

Flight EK 567 Economy — Check-in Opens 08Sep2018 17:45 — Departure 08Sep2018 20:45 — **BENGALURU (BANGALORE)** Departing BLR, Bengaluru International Airport
Arrival 08Sep2018 23:05 — AI Arriving DXB, Dubai International Airport Terminal 3

Coupon validity: not before 08Sep2018 / not after 08Sep2018 Baggage 30Kgs

Leg 2 of 4 | Dubai (DXB) to London Heathrow (LHR) | Operated by Emirates (equipment owner - Emirates)

Flight EK 007 Economy — Check-in Opens 08Sep2018 23:30 — Departure 09Sep2018 02:30 — **DUBAI** Departing DXB, Dubai International Airport
Arrival 09Sep2018 07:05 — DON Arriving LHR, London Heathrow Airport Terminal 3

Coupon validity: not before 09Sep2018 / not after 09Sep2018 Baggage 30Kgs

Figure 2. Two emails from the M2H email data-set

number of flight legs, presence of an associated hotel booking or car rental, presence of an advertisement for upgrades, etc.

2.2 Data Extraction Framework

There are several commonalities in the above two data-sets that motivate the design of our data extraction framework. Due to the large number of different formats it is infeasible to provide training data for each format in the data-set. We need to generalize across formats, and produce reasonable extraction output for a format even if no training data of that format is provided. We use ML models for this purpose. Even though the number of formats is large, inputs within a single format are similar, and the logic of extracting fields from inputs of a single format can be expressed precisely. Hence we design domain-specific languages and use program synthesis to produce synthesized programs to perform the extraction for each format. Using programs instead of ML models enables interpretability and debuggability of the extraction, as programs (as compared to ML models) are human readable. We employ a feedback loop to iteratively refine the extraction labels. Figure 3 shows a block diagram of our extraction framework. The framework consists of an ML model, a program synthesizer, and a semi-automated annotator, which completes the feedback loop. We explain each of these components next.

ML models. We assume that the user gives us annotations on a small number of inputs. Using these annotations, we train an ML model so that the model can produce (noisy) labels on all inputs. For instance, a conditional random field (CRF) is an ML model which learns dependencies between adjacent tokens and generalizes across formats. Figure 4 illustrates a CRF model on an example from the political data set. The goal of this CRF model is to learn dependencies between the tokens and labels. Specifically, this CRF can learn that the token next to a "father name key or K-fname" is a "father name or V-fname" with high probability, from the few training examples given by the user. For the political data-set, we use 25 annotated inputs to train a CRF model for extraction. For the M2H data-set we use a more complex model called LSTM-CRF (explained in Section 4).

The CRF model for the political data-set predicts the value of the father's name field when the key is "daughter of", even though there is no training input with that format. The LSTM-CRF model for M2H data-set predicts the "Airport" field values with precision and recall 0.78 and 0.34 on the emails from the *Expedia* domain, even though the training data does not include any emails from that domain.

Program synthesizer. We take the noisy outputs produced by the ML model and use it as a *soft specification* for the

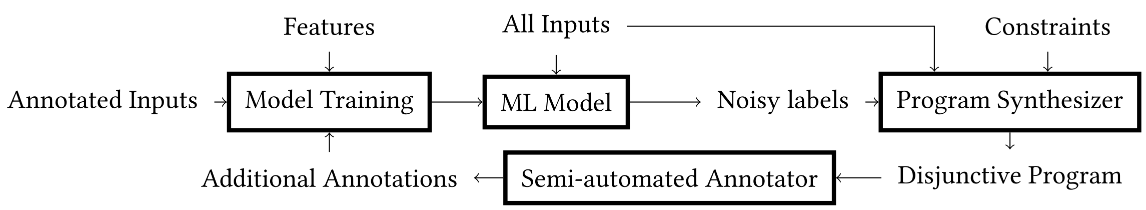
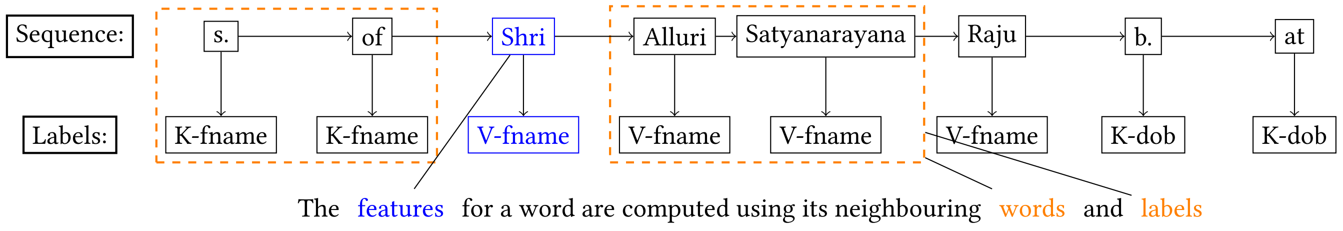


Figure 3. Data Extraction Framework



"K-" stands for key-based labels, and "V-" for value-based labels

Figure 4. Conditional Random Fields (CRFs)

program synthesizer. The synthesis algorithm needs to deal with two major issues: noise in the specification and the multiplicity of formats. To overcome both these issues, the algorithm uses sampling to generate *candidate programs* that *cover* subsets of the data-set. Intuitively, the algorithm repeatedly samples small sets (of size $1-4$) of inputs with the goal of sampling a subset of inputs that are all of the same format, such that the ML model produces correct outputs on all the sampled inputs. Given these inputs and the corresponding ML model outputs as the specification, a standard inductive synthesizer is capable of producing a program that is not only correct for those specific inputs, but also other inputs of the same format. Optionally, the user may provide a type specification, which we call *field constraint*, which identifies if a given extracted output (for a field) is correct for a given input. For example, for the “Father’s name” field, the constraint we use checks if each token of the produced output is present in a dictionary of common Indian names.

Given 2 inputs where the value we want to extract lies between "Father’s Name:" and "Date of Birth" (see Figure 1), the candidate synthesized in the DSL is ValueBetween("Father’s Name", "Date of Birth", inputTokens), i.e., extract between the two relevant keys. On the other hand, given inputs of different formats, say with the keys “Father’s Name” and “s. of”, along with the correct outputs, there is no candidate synthesized.

From the generated candidate programs we compute a subset of programs that cover all the inputs (across various formats), and maximizes agreement with the ML model outputs, as well as satisfaction of any given constraints.

The synthesis algorithm is parameterized by a Domain Specific Language (DSL) $\mathcal{L}$. Intuitively, we tune the expressivity of the DSL such that, for a fixed format of inputs, the DSL only contains programs that can extract *equivalent* parts of the outputs. As a result, the synthesized disjunctive program produces correct output on almost all inputs of a format if the ML model is correct on a sizable fraction of inputs in the format. This leads to significant accuracy improvements in the extraction algorithm.

Semi-automated annotator: Completing the loop. We employ a feedback loop to iteratively improve the ML model and the synthesis output. In cases where the ML model and programs agree on the labels, we increase the confidence value of these labels. In cases where there are disagreement, we use a semi-automatic procedure for resolution: We rank programs by the number of inputs that are correctly processed by the program. If an ML model output disagrees with output produced by a highly ranked program, we declare the program’s output as the winner. We use the modified labels to re-train the ML models and find that the re-trained models have improved accuracy.

For the “Airport” field of the Expedia domain of the M2H data-set, we automatically produce additional training data by using the outputs of the disjunctive program on these outputs (since these programs are highly ranked). Using this additional training data, the re-trained model has precision and recall of 0.99 and 0.99, as compared to 0.78 and 0.34 of the initial model.

In cases where there is a disagreement, but the program is not highly ranked, we resort to user intervention to resolve the disagreement. In this case, the synthesized programs are shown to the user, who may either individually accept or reject some of the programs, or provide annotations to fix the errors. The readability of programs greatly helps users make such debugging decisions.

In the "Father's name" field, one program in the disjunctive program extracted the value following "Spouse's name:"—the ML model was consistently mislabelling spouses as fathers for 0.3-0.4% of inputs. A cursory examination of the program was sufficient to spot and rectify this error. Debugging just the model would involve thorough examination of all the labels produced.

3 Problem and HDEF Solution Framework

We now formalize the heterogeneous data-extraction problem and the present the outline of the HDEF framework.

3.1 Setting and Problem Statement

Data-sets and Fields. We model a heterogeneous data-set $\mathcal{D}$ as a tuple $(D, \{F_0, \ldots, F_n\})$ where: (a) D is a finite set of *inputs*, and (b) $\{F_0, \ldots, F_n\}$ is a partition of D into *formats*, i.e., $\bigcup F_i = D$ and $\forall i \neq j.\ F_i \cap F_j = \emptyset$. The extraction framework has access to the inputs D, but not the partitioning. Henceforth, we write "data-set D" instead of "heterogeneous data-set $\mathcal{D}$" to denote that the exact partition of D into formats $\{F_0, \ldots, F_n\}$ is not explicitly available to the algorithm.

For a given data-set $(D, \{F_0, \ldots, F_n\})$, a *field* $f : D \not\to T$ is a partial function that maps inputs to values of type T and further, conforms to the partition $\{F_0, \ldots, F_n\}$. Formally, for each F_i, either f is defined for all inputs in F_i or is undefined for all inputs in F_i. In practice, we are interested in multiple fields for a single data-set. However, for simplicity of presentation, our formal treatment considers extracting a single field value.

Field Constraints. Given a field f on a data-set D, a *field constraint* $C_f : D \times T \longrightarrow \{\mathsf{true}, \mathsf{false}\}$ is predicate on input and candidate field value pairs. A ideal constraint would have $C_f(i, o) = \mathsf{true}$ if and only if $o = f(i)$. However, such constraints are practically in-feasible to write, and we allow for *soft constraints*, which are noisy approximations of the actual constraints. Note that our framework is able to handle noisy constraints that are neither strict over- nor under-approximations.

Example 1. In the political data-set, the constraint we use for the date of birth field checks that: (1) the output is of the form *"<date> <month> <year>"* (or a permutation thereof), where *"<date>"* and *"<year>"* are integers within valid ranges, and *"<month>"* is the name of a month (partial dates are also accepted), and (2) no extension of the output is a valid date.

For the father's name field, the constraint we use checks that: (a) the output is a sub-string of the input, (b) each word that occurs in the output is present in a dictionary of common Indian names, and (c) no extension of the output satisfies the above two conditions. For the first input from Figure 1, the output "Shri Raja Reddy" satisfies the above constraint, but the output "Shri Raja" does not. While both satisfy the first two properties, "Shri Raja" can be extended (to "Shri Raja Reddy") while maintaining the two properties.

Annotations. Given a set of fields $\{f_0, \ldots, f_n\}$ of a data-set D, an *annotation* annotation(i) of an input $i \in D$ is a tuple (o, e) where: (a) $o = (f_0(i), \ldots, f_n(i))$ are the *ground-truth outputs*, and (b) e is *auxiliary information*. Informally, an annotated input contains not only the ground-truth field values for a given input, but also some additional information that *explains* the output value. For different data-sets, the auxiliary information may take different forms, including being absent in some cases.

Example 2. In the political data-set, the auxiliary information takes the form of a labelling of keys for each field in the input (as shown in Figure 4). While the keys are not a part of the ground truth, they are important for extracting the field values from the input.

The Heterogeneous Extraction Problem. We now define the heterogeneous extraction problem. As input, an extraction framework is provided: (a) inputs D from a data-set, (b) an initial set of *training inputs* $D_{tr} \subseteq D$, which are annotated by human annotators, (c) a field constraint $C_f : D \times T \longrightarrow \{\mathsf{true}, \mathsf{false}\}$, (d) access to an annotation oracle AOracle, which can provide the annotation for any input $i \in D$. The annotation oracle is semi-automatic. In some cases, it can guess the correct annotation with high confidence, using ML models and synthesized programs. In cases it cannot guess the annotations, it queries a human annotator.

For a field f of type T, the extraction framework is expected to output a partial function $f^* : D \not\to T$. The quality of the output f^* is measured using the following metrics:

- *Precision p*. This is the standard measure of false positives used in information retrieval, and is defined as:
$$p = \frac{\mathbf{card}\left(\{i \in D \mid f(i) = f^*(i) \wedge f^*(i) \neq \bot\}\right)}{\mathbf{card}\left(\{i \in D \mid f^*(i) \neq \bot\}\right)}.$$
- *Recall r*. This is the standard measure of false negatives used in information retrieval, and is defined as:
$$r = \frac{\mathbf{card}\left(\{i \in D \mid f(i) = f^*(i) \wedge f^*(i) \neq \bot\}\right)}{\mathbf{card}\left(\{i \in D \mid f(i) \neq \bot\}\right)}.$$
- *F1 score*. This measure combines precision and recall into a single number, and is computed as $2pr/(p + r)$.

3.2 The Heterogeneous Data-Extraction Framework

Our heterogeneous data-extraction framework HDEF is shown in Algorithm 1. In addition to the inputs D, D_{tr}, C_f, AOracle described above, it takes a Domain Specific Language $\mathcal{L}$, a threshold t, and a weight w as input parameters. These parameters are described in later sections. We briefly explain each of the components of the framework here; full explanations follow in the subsequent sections.

The TrainML procedure takes a set of annotated inputs and produces an ML model $\mathcal{M}$. We describe the ML models we use in Section 4. Given any input $i \in D$, the ML model $\mathcal{M}$ produces an output $\mathcal{M}(i)$ which we call the *model output*. Intuitively, the primary advantage of using ML models is that they can *generalize across formats* — even if $D_{tr} \cap F_j = \emptyset$ for a format $F_j \subseteq D$, with significant probability, there may exist $i \in F_j$. $\mathcal{M}(i) = f(i)$. The model training procedure we use is detailed in Section 4.

The NoisyDisjSynth procedure is given all model outputs for the whole data-set, along with any constraints the user has provided. The procedure returns a *disjunctive program* $\mathcal{P} = (P_0, \ldots, P_n)$ consisting of many programs. Intuitively, we want each P_i to always produce the correct output on a single format. As we show in subsequent sections, the synthesized disjunctive program $\mathcal{P}$ can boost the precision and recall significantly as compared to the $\mathcal{M}$. Further, it provides interpretability and debuggability. The disjunctive synthesis procedure is detailed in Section 5.

The ChooseInputs procedure suggests additional inputs to add to the training data. These additional inputs will be annotated in a semi-automated manner and passed as additional training data to the next iteration.

For the sake of formalism, the above 3 steps are repeated until the ML model $\mathcal{M}$ and the disjunctive program $\mathcal{P}$ agree. In practice, however, we either fix the number of iterations (usually a small number 2-3) of the loop, or instead run till there is sufficient agreement between $\mathcal{M}$ and $\mathcal{P}$.

Algorithm 1 Heterogeneous Data-Extraction Framework

```
HDEF($D$, $D_{tr}$, $C_f$, AOracle, $\mathcal{L}$, $t$,$w$)
Require: Set of inputs $D$
Require: Initial training data $D_{tr}$
Require: Field constraint $C_f : D \times T \to \{\text{true}, \text{false}\}$
Require: Annotation oracle AOracle
Require: DSL $\mathcal{L}$
Require: Threshold $t \in [0, 1]$ and Constraint weight $w \in [0, 1]$
1: Annots ← $\{(i, \text{AOracle}(i)) \mid i \in D_{tr}\}$
2: do
3:     $\mathcal{M}$ ← TrainML($D_{tr}$, Annots)
4:     SoftSpec ← $\{(i, \mathcal{M}(i)) \mid i \in D\}$
5:     $\mathcal{P}$ ← NoisyDisjSynth($D$, SoftSpec, $\mathcal{L}$, $C_f$, $t$, $w$)
6:     $D'_{tr}$ ← ChooseInputs($D$, $\mathcal{P}$, $\mathcal{M}$, $t$, $w$)
7:     $D_{tr}$ ← $D_{tr} \cup D'_{tr}$
8:     Annots ← Annots $\cup \{(i, \text{AOracle}(i)) \mid i \in D'_{tr}\}$
9: while $D'_{tr} \neq \emptyset$
```

4 A Recap of ML Models

We briefly explain the ML models we use in HDEF. Our ML models assign a categorical label to each token of a sequence of observed values. The output of a model for a field f is the set of tokens to which the label corresponding to the field is assigned. Below, we discuss two ML models we use, namely, Conditional Random Fields (CRFs) and Bidirectional Long Short-Term Memory augmented with CRFs (LSTM-CRFs).

Conditional Random Fields (CRFs) is a state-of-the-art method for assigning labels to token sequences. CRFs provide a flexible mechanism for exploiting arbitrary feature sets along with dependency in the labels of neighboring words [11], as shown in Fig 4. Each feature set f_j in a CRF takes as inputs a sentence s, the position i of a word in the sentence, the label l_i of the current word, the label l_{i-1} of the previous word (or more adjacent words if required). Suppose we are given a sentence s and an associated weight λ_j with each feature set f_j. Further suppose that the sentence s has n words, and we have m features. Then, we can score a labelling l of s as follows:

$$score\{l|s\} = \sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s, i, l_i, l_{i-1}) \qquad (1)$$

During the training process, we learn the weights λ_j associated with the feature set, given annotated training data. That is, with annotated training data, the score is known, and we use an optimization algorithm (such as gradient descent) to learn the weights that best fit the training data. CRFs model the conditional distribution $P(l|s)$, without having to model $P(s)$ explicitly. While CRFs perform exceedingly well in extraction tasks, one of the requisites to train such a model is a good set of feature functions. Deep neural nets obviate this need by providing mechanisms to learn feature functions automatically.

The other class of ML models we use is Long Short-Term Memory (LSTM) networks [8]. Specifically, we use a *Bidirectional LSTM-CRF* as the ML model. This model also utilizes CRFs to learn labeling over a sequence of tokens. However, the key difference between CRFs and LSTM-CRFs is that the features are manually defined in CRFs, and automatically learned from the data in LSTM-CRF. LSTM-CRFs have been shown to work well for tasks such as Named Entity Recognition and other sequence labeling tasks [12].

5 Program Synthesis with Noisy Inputs

The program synthesis procedure receives input-output examples, where the outputs are generated by an ML model, and hence are noisy. Its goal is to produce a set of programs that cover the different formats in a heterogeneous data-set.

As shown in Algorithm 1, the noisy disjunctive program synthesis step is given the set of inputs D, a soft specification SoftSpec : $D \not\to T$ composed of the model outputs on D, a domain specific language $\mathcal{L}$, a field constraint $C_f : D \times T \to \{\text{true}, \text{false}\}$, a threshold parameter t, and a weight parameter w, which parameterizes the scoring function used in the algorithm. The aim of the disjunctive synthesis step is to produce a disjunctive program that extracts "good" output values from as many inputs as possible. The definition of "good" includes agreement with ground truth (whenever provided), satisfaction of field constraints, and agreement between ML models and extracted programs in $\mathcal{L}$ (see below for a precise definition of agreement).

Algorithm 2 Noisy Disjunctive Synthesis

```
NoisyDisjSynth($D$, SoftSpec, $\mathcal{L}$, $C_f$, $t$, $w$)
Require: Inputs from data-set $D$
Require: Soft specification SoftSpec $\subseteq D \times T$
Require: DSL $\mathcal{L}$
Require: Field constraint $C_f : D \times T \to \{\text{true}, \text{false}\}$
Require: Threshold $t \in [0,1]$ and Constraint weight $w \in [0,1]$
 1: $\mathcal{P} \leftarrow \langle\rangle$ ; candidates $\leftarrow \emptyset$
 2: while Score($\mathcal{P}$, $D$, SoftSpec, $C_f$, $w$) $< t \cdot$ card($D$) do
 3:     candidates $\leftarrow$ candidates $\cup$
 4:       GenerateCandidate($D$, $\mathcal{L}$, SoftSpec)
 5:     $\mathcal{P} \leftarrow$ ApproxMaxCover($D$, candidates, SoftSpec, $C_f$, $w$)
 6: return $\mathcal{P}$
 7:
 8: fun GenerateCandidate($D$, $\mathcal{L}$, SoftSpec)
 9:     SubSpec $\leftarrow$ Sample($D$, SoftSpec)
10:     return Synth($\mathcal{L}$, SubSpec)
11:
12: fun ApproxMaxCover($D$, candidates, SoftSpec, $C_f$, $w$, $t$)
13:     $\mathcal{P} \leftarrow \langle\rangle$ ; uncovered $\leftarrow D$
14:     while Score($\mathcal{P}$, $D$, SoftSpec, $C_f$, $w$) $< t \cdot$ card($D$) do
15:         $P_b \leftarrow \{P \in$ candidates such that
16:            Score($P$, uncovered, SoftSpec, $C_f$, $w$) is max $\}$
17:         if Score($P_b$, uncovered, SoftSpec, $C_f$, $w$) $= 0$
18:             break
19:         $\mathcal{P} \leftarrow \mathcal{P} + P$
20:         uncovered $\leftarrow \{i \in D \mid \mathcal{P}(i)$ is undefined$\}$
21:     return cover
22:
23: fun Score($P$, $S$, SoftSpec, $C_f$, $w$)
24:     conformance $\leftarrow$ card($\{i \in$S$\mid P(i) =$ SoftSpec$(i)\}$)
25:     constraint $\leftarrow$ card($\{i \in$S$\mid C_f(i, P(i)) =$ true$\}$)
26:     return $w \cdot$ conformance $+ (1 - w) \cdot$ constraint
```

5.1 The Noisy Disjunctive Synthesis Problem

Example-based Synthesis. Our disjunctive program synthesis algorithm uses example-based synthesis as a sub-procedure (denoted by Synth). The example-based synthesis is parameterized by a *domain specific language* (DSL) $\mathcal{L}$. Given a set of examples $\{(i_0, o_0), \ldots, (i_n, o_n)\}$ as input, the procedure Synth returns a program $P \in \mathcal{L}$ from the DSL such that $\forall j : P(i_j) = o_j$. The example-based synthesis procedure we use is based on the FlashMeta synthesis framework (See Section 5.4 for a full explanation), initialized with specific DSLs for the political and M2H email data-sets.

Disjunctive Programs. A *disjunctive program* $\mathcal{P} : D \not\rightarrow T$ is a sequence of programs $\langle P_0, \ldots, P_n\rangle$ where each program $P_j : D \not\rightarrow T$ is a partial function. We define constrained semantics of disjunctive programs: Given an input i and constraint C_f, $\mathcal{P}$ returns $P_j(i)$ for the least j where $P_j(i)$ is defined, and satisfies the constraint C_f.

Synthesis Objectives. Informally, the objective of the program synthesizer is to maximize the number of inputs $i \in D$ where the output $\mathcal{P}(i)$ satisfies the field constraint C_f, and the number of inputs $i \in D$ where the ML model $\mathcal{M}$ and $\mathcal{P}$ are in agreement. However, these objectives can often conflict with each other: for example, if most ML model outputs $\mathcal{M}(i)$ do not satisfy the field constraint C_f, agreement with the model hurts the satisfaction of constraints. We define the following scores for a disjunctive program $\mathcal{P}$:

- *Constraint satisfaction score* as the number of inputs on which the $\mathcal{P}$ outputs satisfy the constraint, i.e., constraintScore$(\mathcal{P}) = |\{i \in D \mid C(i, \mathcal{P}(i)) = \text{true}\}$
- *Conformance score* as the number of inputs on which $\mathcal{P}$ conforms to the the ML model $\mathcal{M}$, i.e., conformanceScore$(\mathcal{P}) = |\{i \in D \mid \mathcal{M}(i) = \mathcal{P}(i)\}|$

We combine the above scores with a weight factor $0 \le w \le 1$, i.e., Score $= w \cdot$constraintScore$+ (1-w)\cdot$conformanceScore.

Given a threshold $0 \le t \le 1$, the aim of the disjunctive synthesis procedure is to produce a disjunctive program $\mathcal{P}$ such that Score$(\mathcal{P}) > t \cdot \mathbf{card}(D)$. Of all such possible disjunctive programs, we want to produce the one with the smallest cardinality, i.e., number of component programs.

5.2 The Noisy Disjunctive Synthesis Algorithm

Our algorithm for noisy disjunctive synthesis (depicted in Algorithm 2) has two major steps. First, it generates a number of *candidate programs* P_i, each of which can be a part of the final disjunctive program. Second, it picks a *cover* of candidate programs that minimize the given synthesis objective – informally, this cover approximates the best disjunctive program can be produced with the generated candidates. While the produced disjunctive program does not cover a given fraction of the inputs (as measured by a score of the objective), we generate more candidate programs and repeat.

Candidate Generation. In the candidate generation step, the algorithm repeatedly samples small subsets of examples from the soft specification SoftSpec, and synthesizes programs using the domain-specific example-based synthesis engine Synth. From the soft specification SoftSpec : $D \not\rightarrow T$, the Sample procedure randomly picks small subset SubSpec $= \{(i_0, o_0), \ldots, (i_n, o_n)\}$ such that for each i_k, SoftSpec(i_k) is defined and is equal to o_k. SubSpec becomes a *hard specification* for the example-based synthesis engine to produce a candidate program P.

Guided Sampling. In abstract, the algorithm produces the sampled sub-specification SubSpec through uniform random sampling over subsets of SoftSpec. However, in practice, we sample with some specific constraints:

- First, we usually restrict SubSpec to have very few examples ($1 - 4$ in practice) as program synthesis techniques can generalize from very few examples.
- Second, we guide the sampling to pick inputs for which none of the already generated candidates produce a valid output that satisfies C_f.

Informally, the aim of the candidate generation is to produce *candidate programs that correspond to formats in the data-set*, i.e., for each format $F_j \subseteq D$, we want a program

P_{F_j} such that $\forall i \in F_j.\ P_{F_j}(i) = f(i)$. In practice, for such a candidate to be produced, we need: (a) Sample to pick a SubSpec containing only inputs from F_j, (b) that $\mathcal{M}$ is correct on all the inputs in SubSpec, i.e., $\mathcal{M}(i) = f(i)$ for the sampled inputs, and (c) that the DSL $\mathcal{L}$ and the synthesis procedure Synth are "sufficiently inductive" to generalize SubSpec to program that is correct on all of F_j.

Approximate Cover. Given the set of program candidates, we now need to construct the minimal size disjunctive program $\mathcal{P}$ that satisfies the given threshold t. We can show that even a simplified version of this problem (without field constraints) and $t = 1$ is NP-hard.

Theorem 3. *Given a set of inputs D, a set of candidate programs $\{P_0, \dots, P_n\}$, and a soft specification* SoftSpec, *the task of deciding if there exists a disjunctive program $\mathcal{P}$ of size at most k such that $\forall i \in D.\ \mathcal{P}(i) = \mathrm{SoftSpec}(i)$ is NP-complete.*

The proof is straightforward through a reduction to the set cover problem. Hence, we use an approximation algorithm ApproxSetCover. Informally, the ApproxSetCover algorithm treats the problem of constructing a disjunctive program as a variant of the generalized set maximum coverage problem, and uses the classical greedy heuristic of picking the candidate with the maximum score on the uncovered inputs (line 16). Each candidate program P represents a set consisting of all the inputs on which it produces outputs that conform to the ML model $\mathcal{M}$ and satisfies the field constraint C_f. It can be shown using standard techniques that the approximation factor is at most $\log(\mathbf{card}(D))$.

Theorem 4. *Given inputs D and candidate programs* candidates, *suppose there exists a disjunctive program $\mathcal{P}_{\mathrm{opt}}$ consisting only of programs from* candidates *of size k such that* $\textsc{Score}(\mathcal{P}_{\mathrm{opt}}, D, \mathrm{SoftSpec}, C_f, w) > t \cdot \mathbf{card}(D)$. *Then, the procedure* $\textsc{ApproxMaxCover}(D, \mathrm{candidates}, \mathrm{SoftSpec}, C_f, w, t)$ *returns a disjunctive program $\mathcal{P}$ such that:*
(a) $\textsc{Score}(\mathcal{P}, D, \mathrm{SoftSpec}, C_f, w) > t \cdot \mathbf{card}(D)$, *and*
(b) $\mathbf{card}(\mathcal{P}) \leq \mathbf{card}(\mathcal{P}_{\mathrm{opt}}) \cdot \log \mathbf{card}(D)$

5.3 Parameter Spaces: A Discussion

We discuss the high level parameters to the algorithm (the values w and t, and the DSL $\mathcal{L}$), their choice and their impact on the algorithm.

Tuning parameters w and t. Informally, w controls which of the two noisy artifacts ($\mathcal{M}$ and C_f) we "trust" more? The exact choice depends on how tight the constraint C_f is, and on how good the ML models are likely to be? We can use one of two techniques to choose w:

- *Human intervention.* Given the programs produced by the disjunctive synthesis algorithm are human-interpretable, a user may manually examine the produced programs and their relation to the constraints and the model $\mathcal{M}$ to fix w.
- *Validation data.* Here, the quality of $\mathcal{P}$ is compared to that of $\mathcal{M}$ on a validation set that is apart from the test and training data-sets. If $\mathcal{M}$ is better, then we increase w, thereby driving $\mathcal{P}$ towards $\mathcal{M}$.

For our experiments, we found $w = 0.5$ to be a good trade-off between the ML models and constraints in both data-sets. This value was found using the first technique, i.e., the programs were examined manually and w was changed based on the tightness of the constraints C_f.

On the other hand, we need not fix a threshold t in advance: we can set $t = 1$ and lower t if no disjunctive program is returned after a sufficiently many iterations. In our experiments, t was tuned to 0.95 for the M2H data-set, and to $0.6 - 1.0$ for the political data-set depending on the field.

Bias-Variance trade-off and DSL design. Intuitively, we are relying on the program synthesizer's ability to *inductively generalize* from a small number of *consistent* examples. Modern program synthesis engines are designed to achieve good inductive generalization from a small number of examples [7, 13, 18]. We additionally have the following *informal* restrictions on the power of the DSL $\mathcal{L}$: (a) lack of cross-format generalization, i.e., $\mathcal{L}$ programs should not be able to extract outputs out of different formats, as they are likely to be incorrect; and (b) no generalization over inconsistent outputs, i.e., when we sample incorrect SubSpec even from a single format, an $\mathcal{L}$ program should not be able to extract these inconsistent outputs. These informal requirements are a reflection of the bias-variance trade-off expressed in terms of the expressivity of the DSL. If the DSL $\mathcal{L}$ is very expressive, the synthesizer is able to more closely approximate $\mathcal{M}$—however, this will not "filter the noise" in SoftSpec as desired. On the other hand, if $\mathcal{L}$ is not very expressive, it might not be able to approximate $\mathcal{M}$ at all.

5.4 Domain-specific Example-based Synthesis

To complete the discussion of the disjunctive synthesis algorithm, we provide the details of the synthesis engine Synth we use and the individual DSLs we use. As stated before, our synthesis engine is based on the FlashMeta framework [18] for synthesizing from examples, customized with appropriate domain specific languages as detailed below. A synthesis task in the FlashMeta framework is given by a pair (L, Ex) where L is a fragment of the DSL and Ex is the example specification, i.e., input-output pairs. The FlashMeta framework is based on the idea of writing *back-propagation transformers* for each operator in the DSL, transforming (L, Ex) to a series of sub-tasks (L_1, Ex_1), (L_2, Ex_2), etc that correspond to sub-fragments of L. The full FlashMeta framework is too complex to reproduce here; the reader is referred to [18].

Our intended target users are not end-users of extractions, but experts who design extractors. These experts need not specify one DSL per dataset, but one per input-domain. Indeed, a DSL for extracting flight information from emails can easily be re-purposed for parcel emails or web-form sites, as they share the common domain of semi-structured HTML.

```
      Prog   :=  let iTokens = Tokenize(input) in tokenProg
 tokenProg   :=    iTokens
                 | ValueAt(key, keys, tokenProg)
                 | ValueBetween(key, key', tokenProg)
                 | Slice(tokenProg, pos, pos)
                 | SubSequence(tokenProg, separator, pos)
       pos   :=  FromStart(i) | FromEnd(i)
     integer i; string key, key', separator; Set[string] keys;
```

Figure 5. Syntax of the token-based key-value language $\mathcal{L}_T$

```
      Prog   :=  map(λ node . FFProg(node), Nodes)
     Nodes   :=  AllNodes(input) | Descendants(Nodes)
                 | filter(Selector, Nodes) | Children(Nodes)
  Selector   :=  tag = c | class = c | id = c | nth-child(n) | ...
    FFProg   :=  Substring | Concat(SubString, FFProg)
 SubString   :=  node.TextValue
                 | Extract(RegexPos, RegexPos, SubString)
  RegexPos   :=  RegexSearch(regex, k)
```

Figure 6. Syntax of the M2H extraction language $\mathcal{L}_W$

Token-based Key-Value Language. The syntax of the token-based key-value language $\mathcal{L}_T$ is shown in Figure 5. The DSL $\mathcal{L}_T$ is a variant of the FlashFill DSL that operates on tokens and uses keys as anchor points, rather than strings and using regular expressions as anchor points. The symbols i , *key*, *key′*, and *keys* can be replaced by any constant value of the given types. In practice, the strings *key*, *key′*, and the set *keys* are drawn from the most frequent *n*-grams in the data-set, and the strings *separator* are drawn from common separator tokens such as comma, semi-colons, periods, etc.

Each program in $\mathcal{L}_T$ first tokenizes the input string (using Tokenize) to obtain a token sequence. The tokenization splits the input string at each white-space and special character (e.g., semi-colons, commas, etc) boundaries. Now, the program can perform the following operations: (a) Extract the tokens between the constant strings *key* and *key′* (ValueBetween), or between a constant string *key* and any string from *keys* (ValueAt), (b) Take the sub-sequence between two indices (Slice), (c) Pick a separator *separator* to partition the token sequence into sub-sequences, and pick the sub-sequence at the given index (SubSequence).

Example 5. Consider the following subsequence of an input from the political data-set: *"... s. of Shri Allabux Jafri; b. at Burhanpur, Madhya Pradesh; on March 1, 1929; ed. at H. L. College ... "*. We seek to extract the date “March 1, 1929”. A valid program in $\mathcal{L}_T$ would be the following: **let** iTokens = Tokenize(input) **in** Slice(SubSequence(ValueBetween("b. at", "ed. at", iTokens), ";", FromStart(1)), FromStart(1), FromEnd(0)). A short explanation of the program:

- Tokenize returns [..., *s.*, *.of*, *Shri*, *Allabux*, ...].
- ValueBetween extracts between *b.at* and *ed. at*, yielding the sequence [*Burhanpur*, *Madhya*, *Pradesh*, *“;”*,*on*, *March*, *1*, *“,”*, *1929*, *“;”*].
- SubSequence splits using “;” as a separator and picks the second sub-sequence, i.e., [*on*, *March*, *1*, *“,”*, *1929*]
- The Slice takes each token except the first, yielding [*March*, *1*, *“,”*, *1929*], which is desired result.

The Web Extraction DSL. The DSL $\mathcal{L}_W$ we use for the M2H email data-set is a combination of the HTML extraction DSL from [20] and the FlashFill text-transformation DSL [7]. We do not explain the semantics of the DSL in detail, but instead refer to the corresponding papers. Informally, we use the HTML extraction DSL to select a set of DOM nodes from each email, and then use the FlashFill DSL to transform the text inside the DOM nodes to the field values. The HTML extraction DSL is inspired by CSS selectors and each program is a composition of atomic selectors that act as filters on the set of DOM nodes in the email. The FlashFill DSL extracts sub-strings based on regular expression matches within the text content of each DOM node.

6 Generating Annotations

We now discuss the semi-automated annotator to produce additional annotated training inputs. This step is provided as input the data-set D, the ML model $\mathcal{M}$, the synthesized disjunctive program $\mathcal{P}$, and the field constraint C_f. We provide two strategies for producing the additional training data.

Distinguishing Inputs. This is a fully automated strategy for producing additional training data in the scenario where annotations only consist of the output values of the fields, and no auxiliary information. Here, we use distinguishing inputs, i.e., inputs $D_{\text{diff}} = \{i \in D \mid \mathcal{M}(i) \neq \mathcal{P}(i)\}$ as additional inputs. The corresponding outputs are given by the $\mathcal{P}$ outputs, i.e., the additional training data will be $D'_{tr} = \{(i, \mathcal{P}(i) \mid \mathcal{M}(i) \neq \mathcal{P}(i)\}$. Informally, we use $\mathcal{P}$, which contains highly ranked programs produced by the covering algorithm, as a (possibly noisy) annotation oracle to fully automatically produce training data. We use this strategy to generate additional annotations for the M2H email data-set.

Sampling from $\mathcal{P}$-defined Clusters. In the scenario where a human annotator is required to produce the auxiliary information for the annotations, the goal is to minimize the number of annotations while maximizing the variety of inputs annotated. To this end, we use a clustering approach: the disjunctive program $\mathcal{P} = \langle P_0, \ldots, P_n \rangle$ induces a set of clusters $\{D_0, \ldots, D_n, D_\perp\}$ on the data-set D with each $D_j = \{i \in D \mid \mathcal{P}(i) = P_j(i)\}$ and $D_\perp = \{i \in D \mid \mathcal{P}(i) = \perp\}$. Informally, these clusters are the “closest approximations” of the formats available to the extraction framework. Now, we measure the conformance score and the constraint score on each cluster D_j, and sample over the $D_j \cap D_{\text{diff}}$ weighted by the constraint score and inversely by the conformance score. Effectively, we want to provide a sufficient variety of additional training data over the clusters on which $\mathcal{M}$ performs worse on. For the political data-set, we use this strategy to pick a small number of additional inputs to annotate for the next iteration.

7 Evaluation

We evaluate our extraction framework HDEF with two data-sets introduced in Section 2, namely the political data-set and M2H flight emails data-set. The political data-set consists of 25 annotated training inputs, and 605 randomly selected test inputs, with an average of 423 tokens per input. There are about 4 high-level shapes to the data, each shape having around 4 – 5 variations. The M2H data-set consists of 570 emails from 19 different domains as training data, and 400 emails distributed across 4 separate domains as test data. In our experiments, we perform the evaluation on the test data, one domain at a time, for greater clarity. Within each domain, there can be 5 or more structural variations. Our goal is to extract various attributes from these data-sets with an intention to answer the following questions:

- *Q1: Does combining program synthesis with ML models improve the model performance?*
- *Q2: Does the feedback loop between synthesis and ML models improve the performance?*
- *Q3: What is the impact of field constraints and DSL choice on the quality of data-extraction?*

An important point to note is that the framework itself does not provide any formal guarantees due to the stochastic nature of ML model training and disjunctive synthesis, and the unknown inductive power of the underlying synthesizer.

Experimental setup for political data-set. We train a CRF model (Section 4) on 25 annotated inputs. We use a standard implementation of a CRF from [17]. The features we use are bigram and trigram frequencies, token frequencies, dictionary look-ups for punctuation marks, months, names, etc. We use Prose [1], a freely available implementation of the FlashMeta framework, instantiated with the key-value DSL $\mathcal{L}_T$ described in Section 5.4 as the Synth procedure. The noisy disjunctive synthesis procedure produces 5 – 13 programs in each disjunctive program in this data-set. For the next iteration, we use 5 additional training inputs (human annotated) that were picked by the strategy described in Section 6. For the fields “Father’s Name”, “Spouse’s name”, and “Date of Birth”, we use the field constraints mentioned in Example 1. For the “Place of Birth” field we do not use any constraint.

Experimental setup for M2H data-set. We use 570 annotated emails to train a LSTM-CRF model using the code available at [15]. We use an embedding size of 100, 16 LSTM units and run the algorithm for 30 epochs. We utilize only the HTML text, after removing all the tags, for both training and prediction. We use Prose, instantiated with the web extraction DSL $\mathcal{L}_W$ as the Synth procedure. For each additional iteration, we use the $\mathcal{P}$ outputs on the whole test set as additional training inputs (with no human intervention). For the M2H data-set, we use a field constraint only for the “AirportIata” field: the constraint specified that the output should have 3 upper case characters. In addition, we impose an implicit constraint that the output be non-empty. The noisy disjunctive synthesis procedure produces 2 – 6 programs in each disjunctive program in this data-set.

Transductive learning We do not provide ground-truth labels for test data to the HDEF framework in any manner — their only use is to compute precision and recall numbers. However, labels predicted by one HDEF component (ML model or PROSE) on the entire data set (which includes both training and test) are used for training the other component. This is similar to semi-supervised or transductive settings in ML [6], where one has some labelled training data and lots of unlabelled data. Here, the model trained on labelled data is used to label unlabelled data, and then newly labelled points are augmented to the training data.

7.1 Combining ML models and Program Synthesis

Table 1 presents precision/recall numbers for various extractions. Each row in the table corresponds to a single iteration of the HDEF. The iteration number listed under column “Iter”, for a particular field in a certain domain (column “Domain/-Field”). The first set of numbers labeled "ML model" represent the performance of the ML model $\mathcal{M}$ in the beginning of the iteration, while the second set of numbers labeled "Synthesis" represent the performance of the disjunctive program $\mathcal{P}$ synthesized using the outputs of $\mathcal{M}$ in the end of that iteration. We **highlight** the cases where the performance improves at the end of each iteration.

We notice that in most cases (24 out of 30 extraction tasks, as seen from the **highlighted** last column), we see performance improvement (as evidenced by the F1 score increasing). In some cases, the improvement is spectacular, such as, for example with the “Expedia Aiport” and the “Political Date of Birth” fields. In case of “Expedia Aiport”, the precision-recall numbers improve from 0.78/0.34 to 0.99/0.99. This is because the NoisyDisjSynth algorithm is able to synthesize programs that cover most of the inputs, in spite of the noisy specifications produced by the ML model. Also, we note performance improvements of two kinds with synthesis: low precision-high recall ML model as in “Expedia Date” and high precision-low recall ML model as in “Orbitz Airport”. This demonstrates the broad applicability of our framework. We analyze the fields that perform worse with HDEF as compared to the base ML model, and found the following two underlying reasons to explain these cases (6 of 30):

• *Semantic Conflation.* In some fields of M2H data-set, the values are from 2 or more semantically distinct categories, even within the same format. For example, the “Orbitz/Reservation Id” field conflates entities such as Aggregator Id (6 alpha-numeric characters), Booking Id (6 alpha-numeric characters), and ticketing Id (13 digits). These fields occur in different parts of the input email and cannot be extracted with a single $\mathcal{L}_W$ program. We need to increase the expressiveness of $\mathcal{L}_w$ to handle these fields better.

• *Noise Amplification.* In the place of birth field of political data-set, the ML model consistently produces only a prefix of the true output. The HDEF procedure consistently amplifies this error, leading to worse results.

7.2 Effect of Feedback Loop

We categorize the effect of the feedback loop in four different buckets: 1. Results saturate after first iteration itself, 2. Results saturate after second iteration, 3. Results increase across 3 iterations, and 4. Results deteriorate across iterations. The saturation of results imply that the ML model and synthesis have reached an optimum for the given inputs. As seen from Tables 1 and 2, the results for 13 out of 30 extraction tasks saturate after one iteration of HDEF. Some of these fields are "Political Father's Name", "Expedia AirportIata" and "FlyFrontier Time". We consider "Orbitz FlightNumber" and "FlyFrontier Date" in this bucket, as their results are within the noise levels. Furthermore, we can see that the results for 9 labelling tasks saturate after two iterations of HDEF. Some example tasks in this bucket are "Expedia Date" and "OmanAir Name". Results for 5 tasks increase across the three iterations of HDEF. Some of these include "Orbitz AirportIata" and "FlyFrontier Name". These three buckets comprise 27 out of the total 30 tasks; this shows the positive impact of the Feedback loop in 90% of the extraction tasks.

There are 3 cases where the results reduce across iterations, namely "Political Place of Birth", "Expedia Name" and "Orbitz ReservationId". The reduction in precision-recall numbers for these labels are due to the problems of Semantic conflation and Noise amplification as described above. As an example, the field "Expedia Name" occurs in multiple contexts such as "Passenger Name", "Hotel Booking Name", or "Car Rental Name" within an email. We do not handle polysemous labels in our framework currently.

7.3 Ablation study on the political data-set

We perform an ablation study on the political data-set to understand the impact of DSLs and field constraints on the results. Table 3 presents precision/recall numbers for variants of the synthesis algorithm, done on fields from the political data-set. We vary two axes: each row in the table corresponds to an instance of synthesis for a different DSL, listed under column "DSL". $\mathcal{L}_{FE}$ is the FlashExtract language from [13], and $\mathcal{L}_t$ is for the token-based key-value language introduced in Section 5.4. The first set of numbers represent the performance without constraints and the second set of numbers shows the impact of field constraints.

The numbers clearly demonstrate the impact of choosing the right DSL and the right field constraints. For the FlashExtract DSL, the poor precision and recall reflect the bias-variance trade-off discussed in Section 5.3. For instance, the F1 score on "Father's name" field is 0.23 with FlashExtract DSL as compared to 0.98 with language $\mathcal{L}_t$. FlashExtract can express "very general" programs: For example, one of the programs for the "Father's Name" field find the first occurrence of 3 consecutive alpha-numeric tokens followed by a semi-colon and an alpha-numeric token, and outputs the 2^{nd} and 3^{rd} alpha-numeric tokens. While this program can produce outputs on inputs of many formats, the outputs are unlikely to be precise. Using a better adapted language, we explore a smaller, more consistent space of programs. Moreover, we increase interpretability in the domain, thus *qualitatively* impacting the feedback loop experience.

Similarly, a good field constraint rejects general programs very efficiently. The FlashExtract program above may produce outputs for many inputs, but the "Father's Name" field constraint rightly rejects almost all of these, resulting ultimately in a low score for the program (and in its subsequent rejection from the cover). For the "Place of Birth" field, we do not have a field constraint and hence, cannot overcome the bad precision/recall for the FlashExtract DSL.

7.4 Execution times

Our focus in the evaluation has been to measure precision and recall of the HDEF system. While we did not focus on execution times, we report on them briefly here. For the political data-set, CRF training took 20 minutes on CPU, and synthesis took less than 30 seconds. For M2H, LSTM-CRF training took around 90 seconds/epoch on a Tesla-P100 GPU (we used 30 epochs) and synthesis took less than 2 minutes. These execution times are representative of training and synthesis times for all our experiments.

8 Related Work

Extracting attributes from heterogeneous data has been well studied in the machine learning and data mining communities. The prevalent approach is to train an ML model, such as a CRF, using training data, and make the model work on heterogeneous inputs. Recent approaches include neural models such as LSTMs and LSTM-CRFs which work across multiple natural languages in the context of named entity recognition [12]. Common ML models rely heavily on structural features. In order to build machine learning models that take into account content features in addition to structural features, Satpal et al [24] have explored using Markov Logic Networks, which allow constraints about content to be written using the full expressive power of first order logic.

A recurrent problem which arises in heterogeneous extraction is to manage the noise that arises from such ML models. Zhang et al [26] use a two-layered approach, where they first train a CRF model using weak binary learners, and then use an Expectation Maximization (EM) algorithm on the CRF to reduce the noise and improve the accuracy. Another approach, which has been proposed to handle noise, is to use the labels generated by ML models (such as CRFs) to generate simple wrapper programs. Generating wrappers from labeled examples has been well studied [9, 10]. However, such wrappers are not robust to noisy labels that are typically produced by ML models. Dalvi et al [4] propose an interesting approach to deal with noisy labels, which they

Domain Field	Iter	ML model			Synthesis		
		Pre.	Rec.	F1	Pre.	Rec.	F1
Political Father's Name	1	0.99	0.96	0.98	1.00	0.99	**0.99**
	2	0.99	0.96	0.98	1.00	0.99	**0.99**
Political Spouse's Name	1	1.00	0.91	0.95	0.99	0.95	**0.97**
	2	1.00	0.91	0.95	0.99	0.95	**0.97**
Political Date of Birth	1	0.99	0.81	0.89	1.00	1.00	**1.00**
	2	0.99	0.80	0.88	1.00	1.00	**1.00**
Political Place of Birth	1	0.99	0.77	0.87	0.99	0.71	0.83
	2	0.99	0.76	0.86	0.99	0.71	0.83
Expedia AirportIata	1	0.98	0.93	0.95	1.00	0.97	**0.98**
	2	0.96	1.00	0.98	1.00	0.97	**0.98**
	3	0.97	1.00	0.98	1.00	0.97	**0.98**
Expedia Airport	1	0.78	0.34	0.47	0.99	0.99	**0.99**
	2	0.96	0.97	0.96	0.99	0.99	**0.99**
	3	0.93	0.99	0.96	0.99	0.99	**0.99**
Expedia FlightNumber	1	0.91	0.99	0.95	1.00	1.00	**1.00**
	2	0.97	0.97	0.97	1.00	1.00	**1.00**
	3	0.96	1.00	0.98	1.00	1.00	**1.00**
Expedia Name	1	0.45	0.99	0.62	0.59	0.99	**0.74**
	2	0.48	1.00	0.65	0.50	0.99	**0.66**
	3	0.39	1.00	0.56	0.50	0.99	**0.66**
Expedia ReservationId	1	0.85	0.83	0.84	0.98	1.00	**0.99**
	2	0.91	1.00	0.95	0.98	1.00	**0.99**
	3	0.79	1.00	0.88	0.98	1.00	**0.99**
Expedia Time	1	0.80	0.99	0.88	0.85	0.96	**0.90**
	2	0.77	1.00	0.87	0.85	0.96	**0.90**
	3	0.78	1.00	0.88	0.85	0.96	**0.90**
Expedia Date	1	0.45	1.00	0.62	0.98	1.00	**0.99**
	2	0.93	1.00	0.96	1.00	1.00	**1.00**
	3	0.91	1.00	0.95	1.00	1.00	**1.00**
Orbitz AirportIata	1	0.79	0.99	0.88	0.79	1.00	**0.88**
	2	0.93	1.00	0.96	0.93	0.99	**0.96**
	3	0.96	1.00	0.98	0.98	0.99	**0.98**
Orbitz Airport	1	0.82	0.61	0.70	0.91	1.00	**0.95**
	2	0.89	0.90	0.89	1.00	1.00	**1.00**
	3	0.86	0.94	0.90	1.00	1.00	**1.00**
Orbitz FlightNumber	1	0.94	0.94	0.94	0.77	0.92	0.84
	2	0.97	0.87	0.92	0.76	0.88	0.82
	3	0.96	0.87	0.91	0.76	0.87	0.81
Orbitz Name	1	0.51	0.93	0.66	0.70	0.94	**0.80**
	2	0.61	0.99	0.75	0.80	0.94	**0.86**
	3	0.55	0.97	0.70	0.84	0.87	**0.85**
Orbitz ReservationId	1	0.80	0.98	0.88	0.69	0.68	0.68
	2	0.79	0.97	0.87	0.78	0.91	0.84
	3	0.83	0.96	0.89	0.75	0.74	0.75
Orbitz Time	1	0.86	0.89	0.87	1.00	0.92	**0.96**
	2	0.94	0.99	0.96	1.00	0.99	**0.99**
	3	0.94	1.00	0.97	1.00	0.99	**0.99**
Orbitz Date	1	0.75	0.90	0.82	1.00	0.88	**0.94**
	2	0.88	0.97	0.92	1.00	0.97	**0.98**
	3	0.87	0.99	0.93	1.00	0.97	**0.98**

Table 1. Precision, Recall and F1 numbers on coupling ML models and synthesis

Domain Field	Iter	ML model			Synthesis		
		Pre.	Rec.	F1	Pre.	Rec.	F1
FlyFrontier AirportIata	1	0.06	0.04	0.05	0.55	0.39	**0.46**
	2	0.36	0.01	0.01	0.55	0.39	**0.46**
	3	0.30	0.36	0.33	0.37	0.99	**0.54**
FlyFrontier Airport	1	0.26	0.28	0.27	0.54	0.84	**0.66**
	2	0.37	0.69	0.48	0.43	0.95	**0.59**
	3	0.62	0.72	0.67	0.59	0.98	**0.73**
FlyFrontier FlightNumber	1	0.01	0.01	0.01	0.00	0.00	nan
	2	0.37	0.66	0.47	0.47	1.00	**0.64**
	3	0.47	1.00	0.64	0.47	1.00	**0.64**
FlyFrontier Name	1	0.46	0.61	0.52	0.44	1.00	**0.61**
	2	0.52	0.97	0.68	0.51	1.00	0.66
	3	0.54	0.89	0.67	0.51	1.00	0.67
FlyFrontier ReservationId	1	0.23	0.69	0.35	0.21	1.00	0.35
	2	0.81	0.87	0.84	1.00	1.00	**1.00**
	3	0.92	0.79	0.85	1.00	1.00	**1.00**
FlyFrontier Time	1	0.96	1.00	0.98	1.00	1.00	**1.00**
	2	0.96	1.00	0.98	1.00	1.00	**1.00**
	3	0.96	1.00	0.98	1.00	1.00	**1.00**
FlyFrontier Date	1	0.81	0.95	0.87	0.78	0.93	0.85
	2	0.52	0.96	0.67	0.70	0.93	**0.80**
	3	0.55	0.96	0.70	0.75	0.96	**0.84**
OmanAir FlightNumber	1	0.96	1.00	0.98	0.99	0.92	0.95
	2	0.99	0.98	0.98	0.99	0.92	0.95
	3	1.00	0.98	0.99	0.99	0.92	0.95
OmanAir Name	1	0.66	0.68	0.67	0.80	0.57	0.67
	2	0.56	0.41	0.47	0.86	0.64	**0.73**
	3	0.52	0.32	0.40	0.86	0.64	**0.73**
OmanAir ReservationId	1	0.69	0.91	0.78	0.62	1.00	0.77
	2	0.76	0.99	0.86	1.00	1.00	**1.00**
	3	0.93	0.99	0.96	1.00	1.00	**1.00**
OmanAir Time	1	1.00	1.00	1.00	1.00	0.82	0.90
	2	1.00	0.69	0.82	1.00	0.82	**0.90**
	3	1.00	0.66	0.80	1.00	0.82	**0.90**
OmanAir Date	1	0.60	0.63	0.61	0.47	0.48	0.47
	2	0.69	0.60	0.64	0.91	0.48	0.63
	3	0.73	0.60	0.66	1.00	0.48	0.65

Table 2. Table 1 continued.

Domain Field	DSL	No Constraint			With Constraint		
		Pre.	Rec.	F1	Pre.	Rec.	F1
Political Father's Name	$\mathcal{L}_{FE}$	0.13	0.9	0.23	1.00	0.99	**0.99**
	$\mathcal{L}_{T}$	0.97	0.99	0.98	1.00	0.99	**0.99**
Political Spouse's Name	$\mathcal{L}_{FE}$	0.01	0.16	0.02	0.82	0.79	**0.81**
	$\mathcal{L}_{T}$	0.90	0.96	0.93	0.99	0.95	**0.97**
Political Date of Birth	$\mathcal{L}_{FE}$	0.32	0.43	0.37	0.97	0.96	**0.97**
	$\mathcal{L}_{T}$	0.88	0.90	0.89	1.00	1.00	**1.00**
Political Place of Birth	$\mathcal{L}_{FE}$	0.33	0.12	0.17	–	–	–
	$\mathcal{L}_{T}$	0.99	0.71	0.83	–	–	–

Table 3. Precision, Recall and F1 numbers, ablation study.

call *generate and test.* They generate all possible wrappers, each of which satisfies different subsets of labels, using an efficient enumeration algorithm. Long et al [14] build on this approach by using an extractor scoring model that uses various forms of domain knowledge and features to choose the extractors. Our approximate cover algorithm is inspired by Dalvi et al's generate and test approach. However, Dalvi et al. use the simple language of XPaths to generate wrappers. Our approximate cover algorithm is more general and can work for synthesizing programs in any Domain Specific Language (DSL). Also, our interleaving of ML models and synthesized programs iteratively to improve the performance of extraction, and use of type specifications to reduce noise in the iteration process are novel.

Extraction using program synthesis is an active area of work in the programming languages community. Flashfill [7] is a feature released in Microsoft Excel, which uses program synthesis technology to automatically generate Excel macros from a few examples. FlashExtract [13] is an extraction system based on program synthesis, where the user gives a few training examples, and the system synthesizes an extraction program using an algebra of pre-defined operators such as map, reduce and filter. The above approaches work well when inputs are homogeneous. When inputs are heterogeneous, it is challenging to generate programs that cover all inputs using only a few training examples. A recent work, called FlashProfile [16], attempts to characterize clusters in the input using a given language of patterns. Several works explore disjunctive program synthesis to handle varying requirements for different subsets of inputs [2, 3, 21, 22]. However, none of these works can handle noise in the specifications. Raza and Gulwani have explored another strategy to deal with multiple and unknown input formats by postponing the decision of which program to use to the runtime [20].

Raychev et al. [19] also learn programs from noisy data. The main difference between their work and ours is that our synthesis task is intuitively a combination of clustering and synthesis. Raychev et al. generate a single program from a noisy dataset that minimizes empirical risk, while our approach produces a minimal disjunctive program (i.e., a set of programs) that cover the entire noisy dataset, where each disjunct intuitively identifies a cluster. In other words, Raychev et al. still work with a homogeneous (though noisy) dataset, where one program can cover the whole dataset, whereas our setting has heterogeneous data with many clusters with each cluster requiring a different program. In addition, our work iteratively improves both ML model and synthesized programs using a feedback loop which Raychev et al. do not do. Devlin et al. [5] use RNNs to synthesize programs directly — the model is not used to generate a specification for synthesis. The domain considered is still homogeneous and there is no iterative interaction between models and programs.

Intrepretabilty is an important advantage of our approach and we believe that there is a rich space to explore here. As another point in this design space, Verma et al [25] propose an approach to generate interpretable policies in a high level programming language, from a policy generated using a deep neural network, obtained by reinforcement learning.

9 Conclusion

ML models and program synthesis have complementary strengths. Our extraction framework HDEF combines the two techniques and produces good results for extracting fields from heterogeneous data. We have shared these results with a team that builds an enterprise-scale M2H email extractor, and they are working with us to incorporate these ideas in their product.

While the HDEF algorithm is able to handle random noise in ML models using sampling, it does not work in cases with systematic noise, and ends up boosting it. We have seen cases where ML models consistently identify spouse's names as father's names, or hotel booking ids as flight reservation ids. The HDEF algorithm does not work well in such cases. One possible solution to this issue is to write field constraints that eliminate the systematic noise. For example, if we provide a field constraint that flight reservation ids always match a given regular expression (6 alpha-numeric characters), which distinguishes flight reservation ids from hotel booking ids, the systematic noise can be eliminated easily. However, it might not be possible to write such field constraints, for all the cases. The HDEF algorithm also has difficulty handing inputs with multiple semantic contexts. We plan to pursue these problems in future work.

In this work, we focused on extracting fields from heterogeneous data. In future work, we also plan to explore other problems, where ML models and PL techniques can be productively combined to produce useful solutions.

References

[1] 2015. Microsoft Program Synthesis using Examples SDK. https://microsoft.github.io/prose/.

[2] Rajeev Alur, Pavol Cerný, and Arjun Radhakrishna. 2015. Synthesis Through Unification. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II.* 163–179. https://doi.org/10.1007/978-3-319-21668-3_10

[3] Rajeev Alur, Arjun Radhakrishna, and Abhishek Udupa. 2017. Scaling Enumerative Program Synthesis via Divide and Conquer. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I.* 319–336. https://doi.org/10.1007/978-3-662-54577-5_18

[4] Nilesh N. Dalvi, Ravi Kumar, and Mohamed A. Soliman. 2011. Automatic Wrappers for Large Scale Web Extraction. *PVLDB* 4, 4 (2011), 219–230. https://doi.org/10.14778/1938545.1938547

[5] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. 2017. RobustFill: Neural Program Learning under Noisy I/O. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, 990–998. http://proceedings.mlr.press/v70/devlin17a.html

[6] Alexander Gammerman, Volodya Vovk, and Vladimir Vapnik. 1998. Learning by Transduction. In *UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998*, Gregory F. Cooper and Serafín Moral (Eds.). Morgan Kaufmann, 148–155. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=243&proceeding_id=14

[7] Sumit Gulwani. 2011. 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*, Thomas Ball and Mooly Sagiv (Eds.). ACM, 317–330. https://doi.org/10.1145/1926385.1926423

[8] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[9] Chun-Nan Hsu and Ming-Tzung Dung. 1998. Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. *Inf. Syst.* 23, 8 (1998), 521–538. https://doi.org/10.1016/S0306-4379(98)00027-1

[10] Nicholas Kushmerick. 2000. Wrapper induction: Efficiency and expressiveness. *Artif. Intell.* 118, 1-2 (2000), 15–68. https://doi.org/10.1016/S0004-3702(99)00100-9

[11] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001.* 282–289.

[12] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. *CoRR* abs/1603.01360 (2016). arXiv:1603.01360 http://arxiv.org/abs/1603.01360

[13] Vu Le and Sumit Gulwani. 2014. FlashExtract: a framework for data extraction by examples. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, Michael F. P. O'Boyle and Keshav Pingali (Eds.). ACM, 542–553. https://doi.org/10.1145/2594291.2594333

[14] Chong Long, Xiubo Geng, Chang Xu, and Sathiya Keerthi. 2012. A simple approach to the design of site-level extractors using domain-centric principles. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, Xue-wen Chen, Guy Lebanon, Haixun Wang, and Mohammed J. Zaki (Eds.). ACM, 1517–1521. https://doi.org/10.1145/2396761.2398464

[15] Hiroki Nakayama. 2018. Bidirectional LSTM-CRF and ELMo for Named-Entity Recognition. https://github.com/Hironsan/anago.

[16] Saswat Padhi, Prateek Jain, Daniel Perelman, Oleksandr Polozov, Sumit Gulwani, and Todd D. Millstein. 2018. FlashProfile: a framework for synthesizing data profiles. *PACMPL* 2, OOPSLA (2018), 150:1–150:28. https://doi.org/10.1145/3276520

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[18] Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: a framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, Jonathan Aldrich and Patrick Eugster (Eds.). ACM, 107–126. https://doi.org/10.1145/2814270.2814310

[19] Veselin Raychev, Pavol Bielik, Martin T. Vechev, and Andreas Krause. 2016. Learning programs from noisy data. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 761–774. https://doi.org/10.1145/2837614.2837671

[20] Mohammad Raza and Sumit Gulwani. 2018. Disjunctive Program Synthesis: A Robust Approach to Programming by Example. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 1403–1412. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17055

[21] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett. 2015. Counterexample-Guided Quantifier Instantiation for Synthesis in SMT. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II.* 198–216. https://doi.org/10.1007/978-3-319-21668-3_12

[22] Shambwaditya Saha, Pranav Garg, and P. Madhusudan. 2015. Alchemist: Learning Guarded Affine Functions. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I.* 440–446. https://doi.org/10.1007/978-3-319-21690-4_26

[23] Sunita Sarawagi. 2008. Information Extraction. *Found. Trends databases* 1, 3 (March 2008), 261–377. https://doi.org/10.1561/1900000003

[24] Sandeepkumar Satpal, Sahely Bhadra, Sundararajan Sellamanickam, Rajeev Rastogi, and Prithviraj Sen. 2011. Web information extraction using markov logic networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, Chid Apté, Joydeep Ghosh, and Padhraic Smyth (Eds.). ACM, 1406–1414. https://doi.org/10.1145/2020408.2020615

[25] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. 2018. Programmatically Interpretable Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (JMLR Workshop and Conference Proceedings)*, Jennifer G. Dy and Andreas Krause (Eds.), Vol. 80. JMLR.org, 5052–5061. http://proceedings.mlr.press/v80/verma18a.html

[26] Weinan Zhang, Amr Ahmed, Jie Yang, Vanja Josifovski, and Alexander J. Smola. 2015. Annotating Needles in the Haystack without Looking: Product Information Extraction from Emails. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams (Eds.). ACM, 2257–2266. https://doi.org/10.1145/2783258.2788580