Landmarks and Regions: A Robust Approach to Data Extraction

Suresh Parthasarathy
supartha@microsoft.com
Microsoft
London, UK

Lincy Pattanaik
lincy.pattanaik@microsoft.com
Microsoft Research
Bangalore, India

Anirudh Khatry
t-ankhatry@microsoft.com
Microsoft Research
Bangalore, India

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

Arjun Radhakrishna
arradha@microsoft.com
Microsoft
Redmond, United States

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

Mohammad Raza
moraza@microsoft.com
Microsoft
Redmond, United States

Abstract

We propose a new approach to extracting data items or field values from semi-structured documents. Examples of such problems include extracting passenger name, departure time and departure airport from a travel itinerary, or extracting price of an item from a purchase receipt. Traditional approaches to data extraction use machine learning or program synthesis to process the whole document to extract the desired fields. Such approaches are not robust to format changes in the document, and the extraction process typically fails even if changes are made to parts of the document that are unrelated to the desired fields of interest. We propose a new approach to data extraction based on the concepts of *landmarks* and *regions*. Humans routinely use landmarks in manual processing of documents to zoom in and focus their attention on small regions of interest in the document. Inspired by this human intuition, we use the notion of landmarks in program synthesis to automatically synthesize extraction programs that first extract a small region of interest, and then automatically extract the desired value from the region in a subsequent step. We have implemented our landmark based extraction approach in a tool LRSyn, and show extensive evaluation on documents in HTML as well as scanned images of invoices and receipts. Our results show that the our approach is robust to various types of format changes that routinely happen in real-world settings.

CCS Concepts: • **Information systems → Wrappers (data mining)**; • **Software and its engineering → Automatic programming**.

Keywords: Data extraction, Program synthesis, Landmarks and regions, Semi-structured data

ACM Reference Format:
Suresh Parthasarathy, Lincy Pattanaik, Anirudh Khatry, Arun Iyer, Arjun Radhakrishna, Sriram K. Rajamani, and Mohammad Raza. 2022. Landmarks and Regions: A Robust Approach to Data Extraction. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '22), June 13–17, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3519939.3523705

1 Introduction

Extracting data from semi-structured documents is an important and pervasive problem, which arises in many real-world situations. Our goal is to automatically synthesize extraction programs that can extract relevant fields of interest from *formed semi-structured documents*. We use the term formed document to represent a broad category of documents, ranging from invoices, receipts, business cards, booking emails, etc. While loosely defined, formed document collections have some key common characteristics:

- They are often *machine-to-human*, i.e., automatically generated by software, but are meant for consumption by humans. Usually, many documents are generated with the same format, which makes them amenable to programmatic data extraction.
- They are *heterogeneous*, i.e., each kind of document can be in many ad-hoc formats. For example, while

extracting information from flight reservation emails, each airline follows a different format, and the same airline may change formats over time.
- They are *continuously evolving* with new document formats being added often. Hence, a system for extracting documents is not a one-time task —the system needs to be robust to evolution, and needs to be routinely updated and maintained.

A *robust* extraction system should handle the heterogeneous data formats at scale, with minimal human intervention and be efficient at run-time.

Given this problem definition, our approach is inspired by how humans analyze such complex and lengthy documents when looking for specific information: we first narrow down the region that is relevant for the task at hand, and focus only on that small region to extract and understand the desired information. This inspires a key idea that we use in our approach, which is the notion of a *landmark*. In the literature, landmarks have been used to identify locations in document which are "nearby" the value we desire to extract [37, 56]. As an analogy, if we want to locate a restaurant in a map, and we know that it is near the train station, we can first locate the train station in the map, and then the restaurant by identifying its location relative to the train station. In this case, the train station is the landmark used to locate the restaurant. In the case of documents, humans tend to use keyword phrases that occur in all (or most) documents near the locations of the desired field values as landmarks. For example, the phrase "Depart:" is a potential landmark to locate the value of departure time in the travel emails shown in Figure 1(a) and (b), and the phrase "Owing" is a potential landmark to locate the value of total invoice amount in the invoice shown in the right side of Figure 1(c).

Landmarks are a form of data invariance present in all documents of a format. The key idea in this paper is to use landmarks to decompose the field extraction problem into two different sub-problems:

1. Region extraction: This step takes the document as input, and produces a small region of interest as output, guided by the landmark, such that the region of interest contains the landmark as well as the field values of interest.
2. Value extraction: This step takes the small region of interest produced by the first step as input, and extracts the desired field values from the region.

Since real-world scenarios have heterogeneous formats, we design our value extraction to be conditional on both the landmark and the layout of the identified region of interest. We formalize these notions and in a novel generic design of domain specific languages which we call *landmark-based DSLs*, which are designed for data extraction using the landmark-based approach. Such DSLs contain separate language fragments for the region and value extraction steps which can be instantiated arbitrarily for different domains, and we describe such instantiations and corresponding synthesis algorithms for the different concrete domains of HTML and scanned image documents.

While the strategy of using landmarks and region layouts nicely structures the extraction task into well-defined sub-problems, the remaining question is how we can infer the landmarks and region layouts themselves. Landmarks capture redundancy across documents with similar formats, and hence landmarks can be detected by analyzing commonalities across documents with the same format. The formats themselves can be identified using clustering techniques that group similar documents together, and the same techniques can be applied at the region level to infer different region layouts. In this work we show how the two problems of landmark detection and clustering can be solved by a general algorithm that jointly infers clusters and landmarks in a hierarchical fashion.

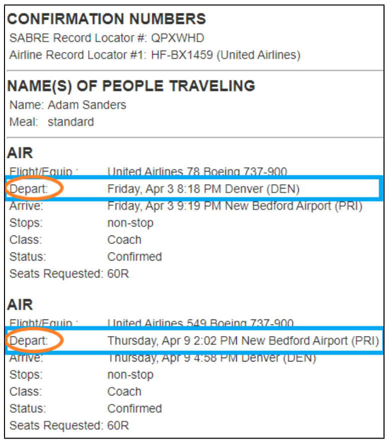
We have implemented this approach in a tool called LRSyn, which is short for **Landmark-based Robust Synthesis**. We present empirical results comparing the performance of LRSyn with current approaches on datasets containing documents from HTML and scanned image domains.

To summarize, we make the following contributions:

- We use the concept of **landmarks and regions** for extracting attributes from heterogeneous data formats and introduce the formal generic class of landmark-based DSLs in which we can express robust extraction programs that continue to work when formats change.
- We propose a joint clustering and landmark detection algorithm which automatically clusters and infers landmarks on the given input data.
- We present concrete instantiations of landmark-based DSLs and corresponding synthesis algorithms in the particular domains of HTML and scanned document images.
- Our implementation, LRSyn, is comparable to state-of-the-art when test data is from the same time period as training data, and significantly outperforms current approaches when test data is from a different time period than training data. In the HTML domain, LRSyn is able to achieve near perfect F1 score of 1.0 in most cases. In the images domain, LRSyn outperforms a released product in scenarios where test data has different formats when compared to training data. This gives evidence that LRSyn is robust to format changes that occur over time in real-world scenarios.

2 Overview

Many existing data extraction techniques [23, 28, 36], both from research literature and those used industrially, are driven by *global document structure*. For example, consider the confirmation email for a flight reservation in Figure 1(a),



CONFIRMATION NUMBERS

SABRE Record Locator #:	QPXWHD
Airline Record Locator #1:	HF-BX1459 (United Airlines)

NAME(S) OF PEOPLE TRAVELING

Name:	Adam Sanders
Meal:	standard

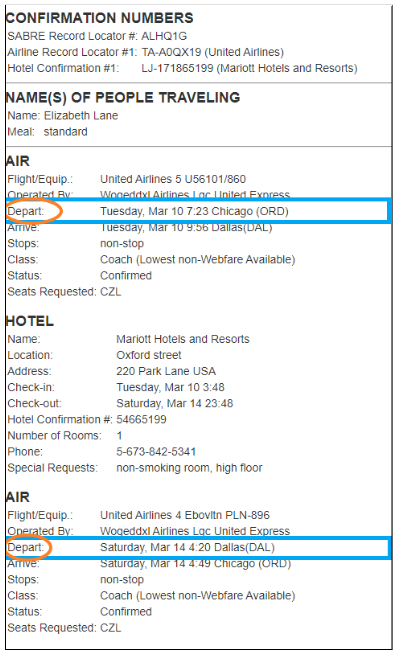
AIR

Flight/Equip.:	United Airlines 78 Boeing 737-900
Depart:	Friday, Apr 3 8:18 PM Denver (DEN)
Arrive:	Friday, Apr 3 9:19 PM New Bedford Airport (PRI)
Stops:	non-stop
Class:	Coach
Status:	Confirmed
Seats Requested:	60R

AIR

Flight/Equip.:	United Airlines 549 Boeing 737-900
Depart:	Thursday, Apr 9 2:02 PM New Bedford Airport (PRI)
Arrive:	Thursday, Apr 9 4:58 PM Denver (DEN)
Stops:	non-stop
Class:	Coach
Status:	Confirmed
Seats Requested:	60R

a. Flight booking email



CONFIRMATION NUMBERS

SABRE Record Locator #:	ALHQ1G
Airline Record Locator #1:	TA-A0QX19 (United Airlines)
Hotel Confirmation #1:	LJ-171865199 (Mariott Hotels and Resorts)

NAME(S) OF PEOPLE TRAVELING

Name:	Elizabeth Lane
Meal:	standard

AIR

Flight/Equip.:	United Airlines 5 U56101/860
Operated By:	Wogeddxl Airlines Lgc United Express
Depart:	Tuesday, Mar 10 7:23 Chicago (ORD)
Arrive:	Tuesday, Mar 10 9:56 Dallas(DAL)
Stops:	non-stop
Class:	Coach (Lowest non-Webfare Available)
Status:	Confirmed
Seats Requested:	CZL

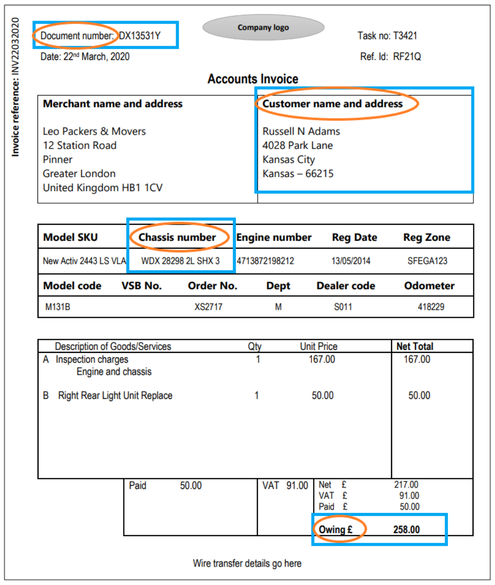
HOTEL

Name:	Mariott Hotels and Resorts
Location:	Oxford street
Address:	220 Park Lane USA
Check-in:	Tuesday, Mar 10 3:48
Check-out:	Saturday, Mar 14 23:48
Hotel Confirmation #:	54665199
Number of Rooms:	1
Phone:	5-673-842-5341
Special Requests:	non-smoking room, high floor

AIR

Flight/Equip.:	United Airlines 4 Ebovltn PLN-896
Operated By:	Wogeddxl Airlines Lgc United Express
Depart:	Saturday, Mar 14 4:20 Dallas(DAL)
Arrive:	Saturday, Mar 14 4:49 Chicago (ORD)
Stops:	non-stop
Class:	Coach (Lowest non-Webfare Available)
Status:	Confirmed
Seats Requested:	CZL

b. Flight booking email



Invoice reference: INV22032020

Document number: DX13531Y

Company logo

Task no: T3421

Date: 22nd March, 2020

Ref. Id: RF21Q

Accounts Invoice

Merchant name and address	**Customer name and address**
Leo Packers & Movers 12 Station Road Pinner Greater London United Kingdom HB1 1CV	Russell N Adams 4028 Park Lane Kansas City Kansas – 66215

Model SKU	**Chassis number**	**Engine number**	**Reg Date**	**Reg Zone**
New Activ 2443 LS VLA	WDX 28298 2L SHX 3	4713872198212	13/05/2014	SFEGA123

Model code	**VSB No.**	**Order No.**	**Dept**	**Dealer code**	**Odometer**
M131B		XS2717	M	S011	418229

	Description of Goods/Services	Qty	Unit Price	**Net Total**
A	Inspection charges Engine and chassis	1	167.00	167.00
B	Right Rear Light Unit Replace	1	50.00	50.00

Paid 50.00	VAT 91.00	Net £ 217.00 VAT £ 91.00 Paid £ 50.00 **Owing £ 258.00**

Wire transfer details go here

c. Accounts Invoice

Figure 1. Formed documents: Figure a. and b. are examples of flight reservation emails. Figure c. is an image of accounts invoice. Orange ellipses indicate landmarks and blue rectangles indicate ROIs

```
CSS selector: :nth-child(11) > TABLE >
                  TBODY:nth-child(1):nth-last-child(1)
                  > :nth-last-child(6) > :nth-child(2)
Text program: Extract TIME sub-string
```

Figure 2. NDSyn extraction program for Departure time

and the task of extracting the flight departure time from it. The NDSyn algorithm from the HDEF system [23] synthesizes the extraction program shown in Figure 2 for this task. This program starts with extracting the "TBODY" elements in the HTML document; which are the only TBODY child in a table. This results in the various blocks, "AIR", "HOTEL", etc. Within each block, the program extracts the 6th child from the end and within that node, it extracts the 2nd child. If an extra block gets added to the document between the two "AIR" blocks, like the "HOTEL" block Figure 1(b), the program extracts the "Check-in" time from the "HOTEL" block as well in addition to the departure times. This program also fails for cases where the "AIR" blocks have more elements.

In contrast, local structure based data extraction techniques focus more on the structure of the document that is close to the value to be extracted [37, 56]. Inspired by these approaches, we propose landmark-based synthesis which more closely relates to the way humans scan through documents, and functions in a local and compositional manner. Let us walk through how a human might find the departure time in document. First, the human begins by scanning the document for keyword phrases—in this case, say, the word *AIR*. Now, the human finds the first title *AIR* and then, scans the section corresponding to the keyword. Recursively, they may then search for the keyword *Depart:* and find the corresponding text that contains the actual flight departure time. Thus, humans typically navigate documents using keyword phrases that occur in all documents in the neighborhood of the field values desired.

Our landmark based synthesis algorithm LRSyn mirrors the above workflow. We introduce the notions of *landmarks*, *regions of interest* and *blueprints* to formalize this workflow. Landmarks correspond to keyword phrases used in the human workflow, regions of interest (*ROIs*, for short) correspond to the local and relevant document sections around the landmarks, and we use blueprints to compare similarity of regions. In the above example, the keywords *AIR* and *Depart* act as landmarks and their corresponding sections act as regions of interest.

LRSyn consists of two main components: (a) Identifying document landmarks from the data, which also involves clustering documents which have roughly similar local structure in the neighborhood of the desired field values together (b) Synthesizing extraction programs, which first zoom in on the region of interest given a document using the landmark as the anchor, and then extract the desired field value from the region of interest. We illustrate each of these steps below using our machine-to-human (M2H) email dataset, which consists of 3500 HTML flight reservation emails from 6 airlines.

2.1 Jointly Inferring Landmarks and Clusters

Initial Clustering. The first step in our technique is to separate the data-set into clusters that correspond to different formats of emails. For this, we use the notion of *blueprints*. For the HTML domain, the blueprint of a document (or a document fragment) contains the tag structure and values that are common across documents. We compute an initial clustering by considering of whole documents, and using the closeness of blueprints as the distance metric between documents. The initial clustering produces a large number of very fine-grained clusters. On the M2H data-set, our initial clustering produces 20 *head clusters* along with many small *tail clusters* totaling 70 clusters altogether.

Identifying Landmark Candidates. For each cluster, we identify a landmark, which is an n-gram that appears in all documents in the cluster. For each such landmark candidate (i.e., shared n-grams), we assign a score based on two metrics: (a) the distance between the landmark candidate and the field value to be extracted, and (b) the number of nodes in the document region that encloses both the landmark candidate and the field value. These two metrics capture the intuition that landmarks should be close to the values being extracted. For the email in Figure 1(a), the top-scoring landmark candidates for extracting *Departure Time* are "Depart" and "Arrive".

Regions of Interest and Re-clustering. A contiguous region in the document that encloses the landmark candidate and the field value is called a *region of interest*. We are interested in small regions of interest. In Figure 1 the landmark candidates are shown in orange ellipses, and the corresponding regions of interest are shown using blue rectangles. We now re-define a distance metric between two documents as the minimum distance (over all landmark candidates) between the blueprints of the corresponding regions of interest. Using this revised distance metric, we iteratively merge clusters if the average distance between documents in the clusters is less than a threshold, till no further merging is possible. As a result of such merging, formats with an advertisement section added, or existing sections rearranged would all be included in the same cluster, as the blueprint of the region of interest is invariant to such changes that occur outside the region of interest. In the M2H data-set, iterative merging based on closeness of regions of interest results in less than 5 clusters at a field level.

2.2 Synthesizing Extraction Programs

With the new coarse-grained clusters, we synthesize 2 sub-programs that together with the blueprint of the region of interest forms the complete extraction program. The first sub-program, called *region extraction program*, takes as input the whole document d_i and a landmark location ℓ as inputs, and produces a region of interest R as the output. In the case of HTML documents, the synthesized region extraction program starts with the landmark location and traverses the

```
Landmark: Depart
Region program: parentHops : 0,  siblingHops : 1
Blueprint: /TD
Value program:
        CSS selector: nth − child(2)
        Text program: Extract TIME sub-string
```

Figure 3. LRSyn extraction program for Depart time

tree-structure of the HTML document and grows a region starting at the location of the landmark, and ensures that all locations of field values are included. As an example,for the airline itinerary example shown in Figure 1(a), in order to extract the value of the departure time, with "Depart:" as the landmark, the synthesized region extraction program is given by *0 parent hops, 1 sibling hop*. The single sibling hop here implies that across all the training data, the extraction value lies 1 sibling away from the landmark within the same parent node in the DOM. The *parent hops* and *sibling hops* are computed based on all the annotated training documents in the given cluster, and hence produce a large enough ROI that includes the location of all the field values for extraction. We also compute the blueprint for this region, and use this blueprint during execution of the region extraction program (see below). The second sub-program, called *value extraction program*, takes the region produced by the region extraction program as input, and produces the desired field value as output. For HTML documents, we use the DSL and the corresponding synthesis algorithm from [46]. For the example in Figure 1(a), the synthesized region and value extraction programs, and the blueprint of the region of interest are shown in Figure 3.

During execution of the synthesized extraction program, we compute the blueprint of the ROI calculated by the region extraction program and compare it with the blueprint of the ROI generated during synthesis. If the distance between these two blueprints is below a threshold value t, the regions from synthesis and inference times are "roughly similar", and we use this extracted program. Otherwise, we look for other extraction programs synthesized from other clusters, which better match the current document.

3 Formed Document Extraction

In this section, we present the formed document problem definition, formalize the notions of landmarks and regions, and introduce the novel class of landmark-based DSLs.

3.1 Preliminaries and Problem Statement

Documents and Locations. We use the term *document* to represent a single record of a dataset from which we are interested in extracting data. We use the symbol doc to represent documents. A document doc has a set of *locations* Locs(doc) which can be used to index into the document and look up values. For a location $\ell \in \mathsf{Locs}(\mathsf{doc})$, the function $\mathsf{Data}[\ell]$ returns the value of the data present in the location ℓ of doc.

Example 3.1. When dealing with HTML documents, the location are XPaths that retrieve elements in the HTML DOM document tree structure. The data value of a location is the concatenation of all the text elements in the DOM element.

Data-sets and Fields. We model a heterogeneous dataset $\mathcal{D}$ as a tuple $(D, \{C_0, \ldots, C_n\})$ where: (a) D is a finite set of *input documents* (or just *documents* for short) and (b) $\{C_0, \ldots, C_n\}$ is a partition of D into *clusters*, i.e., $\bigcup C_i = D$ and $\forall i \neq j.\ C_i \cap C_j = \emptyset$. Each partition represents a *similar* set of documents in terms of format. The extraction framework has access to the inputs D, but not the partitioning. Henceforth, we write "dataset D" instead of "heterogeneous dataset $\mathcal{D}$" to denote that the exact partition of D into clusters is not provided to us as input.

For a given dataset D, a *field* F of type T is a partial function $\mathsf{F} : D \not\to \mathsf{T}$ that maps documents to values of type T. We implicitly assume that a field is either defined for all documents in a cluster or is undefined for all documents in the cluster, i.e., $\forall C_i.\forall \mathsf{doc}, \mathsf{doc}' \in C_i.\mathsf{F}(\mathsf{doc}) = \bot \Leftrightarrow \mathsf{F}(\mathsf{doc}') = \bot$. We say that $\mathsf{F}(\mathsf{doc})$ is the *value of the field* F in doc. The type of a field can either be a primitive type such as integer or string or a composite type such as a list of strings or set of integers. Though we are interested in extracting multiple fields from each document in a dataset, for simplicity of presentation, our formal treatment considers extracting the value of a single field.

Annotations. Given a field F of a data-set D, an *annotation* $\mathcal{A}(\mathsf{doc})$ of $\mathsf{doc} \in D$ is a list of locations $[\ell_1, \ldots, \ell_n]$ and an aggregation function Agg such that $\mathsf{F}(\mathsf{doc}) = \mathsf{Agg}(\mathsf{Data}[\ell_1], \ldots, \mathsf{Data}[\ell_n])$. Annotations are user provided "labels" in ML parlance, and are used as training data. For our experiments, we built a visual user interface where annotators could click on individual HTML and image documents to select annotation locations. In the background, the tool converts these clicks into locations, i.e., XPaths in HTML and x-y coordinates in PDF documents.

Example 3.2. For the departure time field in Figure 1(a), the annotation contains the two locations with text elements "Friday, Apr 3 8:18 PM" and "Thursday, Apr 9 2:02 PM", and the aggregation function collects these values into a list.

The formed document extraction problem. Fix a dataset D and a field F. The input to the formed document extraction problem is given by item a set of annotations on a *training set* $D_{\mathrm{tr}} \subseteq D$. The ideal expected output for such a problem is an *extraction function* Extract such that $\forall \mathsf{doc} \in D.\mathsf{Extract}(\mathsf{doc}) = \mathsf{F}(\mathsf{doc})$. However, it is hard to produce ideal extractions, and we instead use the standard metrics of *precision*, *recall* and *F1 score* to measure the quality of an extraction function (see, for example, [54]). In practice, we are usually interested in extracting multiple fields from a document at once, and in fact, our implementation can do so. However, for simplicity of discussion, we present our techniques and conduct our experiments for one field at a time.

3.2 Landmarks and Regions

Landmarks. A landmark is a value that we can use to identify a location in a document, such that the field value is present in a "nearby" location (or in "nearby" locations if the field value is aggregated from multiple data values). Formally, a *landmark* is given by a data value m. A given landmark m identifies a unique location ℓ in a document doc such that $\mathsf{Data}[\ell_i] \supseteq \mathsf{m}$, i.e., the landmark value is a substring of the data at ℓ_i. In order for a landmark to be useful for our purposes, we require the existence of an inexpensive "locator" function, which can locate the occurrences of a landmark in a document. More precisely, we assume a computationally inexpensive function Locate such that, $\mathsf{Locate}(\mathsf{doc}, \mathsf{m}) = \ell \implies \mathsf{Data}[\ell] = \mathsf{m}$.

Example 3.3. Consider the travel itinerary document in Figure 1(a). In order to extract departure times from this document, a possible landmark to use is the phrase "Depart:".

Remark 3.4. For ease of presentation, the definition assumes that landmarks occur in one location per document (contrast against *Depart* in Figure 1(a)). We discuss handling multiple, ambiguous, landmark locations in Section 6.

Regions. A *region* R of a document doc is a set of contiguous locations. A region can be thought of as a "sub-document". Given a set of locations L of a document doc, the *enclosing region* $\mathsf{EncRgn}(L, \mathsf{doc})$ is the smallest region that contains all locations in L. We are particularly interested in regions that enclose a landmark and the corresponding field values as our approach is based on narrowing down the document to such regions. We call such regions as *regions of interest* or *ROIs* for short.

Example 3.5. The bottom two blue rectangles in Figure 1(a) highlight the relevant ROIs that contain both the landmark "Depart:" and the associated field values.

Blueprints. Intuitively, a blueprint of a region is a "hash" of all the parts that are "common" to all such regions in the cluster. For example, the strings "Airline Record Locator", "AIR", "Meal", "Depart:" in Figure 1(a) are common values, since they will occur in all documents that follow this format. Let the *layout* of a region R in a document doc be the subset of all locations $\ell \in \mathsf{R}$ such that $\mathsf{Data}[\ell]$ is a common value. The *blueprint* $\mathsf{BP}(\mathsf{R})$ of a region R is then defined as a hash of values in the layout of R.

If two regions are similar, we want to define their blueprints such that they are close to each other. Given two blueprints b_1 and b_2, we use the notation $\delta(\mathsf{b}_1, \mathsf{b}_2)$ to denote the distance between b_1 and b_2. If R_1 and R_2 are similar in structure, we want $\delta(\mathsf{BP}(R_1), \mathsf{BP}(R_2))$ to be small value.

Example 3.6. Our notion of blueprint of an HTML region is based on the XPaths to its DOM nodes, but ignoring node order. For example, the blueprint of a region stores the path starting from a `div` node, descending through `table`, `tr`, `td`, and `span` nodes, but without storing where this path is in relationship to other paths.

3.3 Landmark-Based DSLs

To formalize the notions of extraction using landmarks, we introduce a special class of DSLs called landmark-based DSLs. This is a generic design of languages that formally captures our reasoning using landmarks and regions for extraction tasks in a domain-agnostic fashion. Figure 4 shows the structure of a landmark-based DSL. In such DSLs, the input is always assumed to be a document and a complete program returns a field value extracted from the document. Such DSLs consist of four notions: landmarks m, blueprints b, region extraction programs p_{rx} and value extraction programs p_{vx}. The region and value extraction programs can be instantiated arbitrarily for a particular domain by defining the language fragments $\mathcal{L}_{rx}$ and $\mathcal{L}_{vx}$, and we shall illustrate such instantiations of these fragments for the web and image extraction domains.

These four notions are brought together in the single top-level Extract operator, the semantics of which is defined in Algorithm 1. This operator takes a list Q of 4-tuples, where each tuple consists of a landmark, a region program, a blueprint for the region, and an extraction program. Each tuple represents an extraction strategy for a particular region format. The region program uses a landmark to identify a region of the document, and if this region matches the given blueprint, then the extraction program can be applied on the region to extract the field value. The top-level operator acts as a switch statement that applies the first tuple that successfully extracts a value from the document. Formally, for each tuple, we first use the Locate function to identify the location corresponding to the landmark m. This location is then input into the region program RProg_i to produce the region of interest R. Now, we proceed with this cluster only if the blueprint of the region is within a certain tunable threshold of similarity. If the blueprint is close enough, we return the output of the extraction program EProg_i on the region. Otherwise, we continue with the remaining tuples in Q.

4 Landmark and Region Based Synthesis

In this section we present our generic technique LRSyn for synthesizing extraction programs in landmark-based DSLs, and we present instantiations of this technique for different domains in Section 5.

```
@start   T   t   := Extract(q,..., q) where q = (m, p_rx, b, p_vx)
      R → T   p_vx := ...  L_vx  ...
(doc, str) → R   p_rx := ...  L_rx  ...
             str m        // landmark
             obj b        // blueprint
@input   doc d            // input document
```

Figure 4. Structure of a Landmark-based DSL $\mathcal{L}_{ld}$

Algorithm 1 Semantics of the Extract operator in a landmark-based DSL. The blueprint threshold t is a tunable parameter of the semantics.

```
Require: Input document $d$ of type doc
Require: List $Q = [q_1, ..., q_k]$, where each $q_i$ has the form
    $(m, p_{rx}, b, p_{vx})$ for $1 \le i \le k$
1: for $(m, p_{rx}, b, p_{vx}) \in Q$ do
2:     $\ell \leftarrow \mathsf{Locate}(d, m)$
3:     $R \leftarrow p_{rx}(d, \ell)$
4:     if $R \ne \bot \wedge \delta(\mathsf{BP}(R), b) \le t$ then
5:         return $\mathsf{Agg}(p_{vx}(R))$
6: return $\bot$
```

4.1 LRSyn: The Solution Outline

Our solution data-extraction system is parametrized by a number of components that need to be instantiated for each domain.

- *Region extraction program synthesizer.* The region extraction DSL $\mathcal{L}_{rx}$ is equipped with a synthesizer that takes as input examples of the form $(\mathsf{doc}, \ell) \mapsto R$, and produces programs from $\mathcal{L}_{rx}$. Here, the example maps a document doc and a location ℓ within doc to a region R of the document. For instance, an example in the HTML domain will have an input HTML document, an input location (DOM node), and an output region (set of contiguous DOM nodes).
- *Value extraction program synthesizer.* The value extraction DSL $\mathcal{L}_{vx}$ is equipped with a synthesizer that takes as input examples of the form $R \mapsto v$, and produces a program from $\mathcal{L}_{vx}$. Here, R is a region in a document and v is the field-value for that document. For instance, an example might have an input region given by the blue rectangle in Figure 1(a), and an output value "Friday, Apr 3 8:18 PM".
- *Blueprinting and Locating functions.* The blueprinting function BP, locating function Locate, and the blueprint distance function δ (as described in Section 3) need to be specified per domain.

Algorithm 2 presents an outline of our landmark-based robust synthesis algorithm. The high-level components are illustrated in Figure 5. Given an annotated training set D_{tr}, the first task is to infer the clustering of D_{tr} into $C_0, \ldots, C_n$. In Algorithm 2, this step is combined with the inference of landmarks. The procedure InferLandmarksAndCluster (line 1) produces a set of clusters C_i each associated with a landmark m_i. Then, for each cluster C_i, the algorithm calls

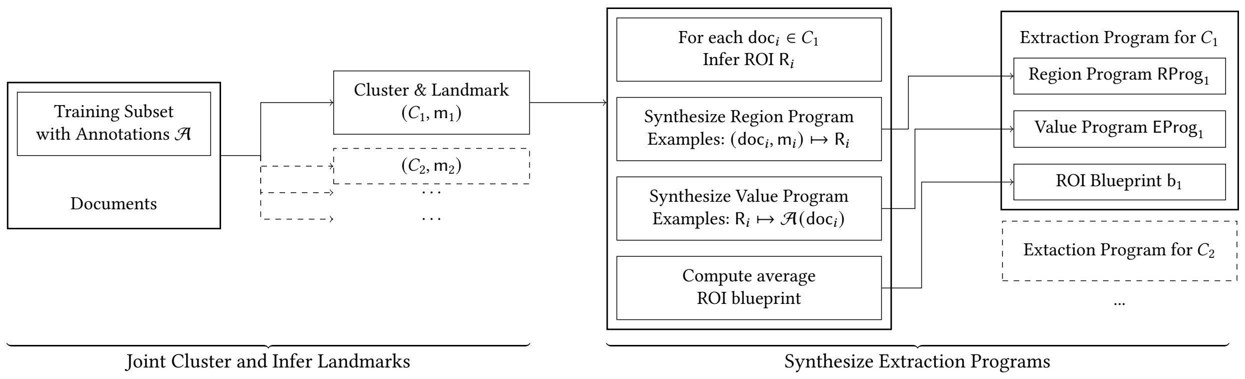

Training Subset with Annotations $\mathcal{A}$
Documents
Cluster & Landmark (C_1, m_1)
(C_2, m_2)
...
For each $doc_i \in C_1$ Infer ROI R_i
Synthesize Region Program Examples: $(doc_i, m_i) \mapsto R_i$
Synthesize Value Program Examples: $R_i \mapsto \mathcal{A}(doc_i)$
Compute average ROI blueprint
Extraction Program for C_1
Region Program $RProg_1$
Value Program $EProg_1$
ROI Blueprint b_1
Extaction Program for C_2
Joint Cluster and Infer Landmarks
Synthesize Extraction Programs

Figure 5. Outline of landmark-based robust synthesis LRSyn

Algorithm 2 Landmark-based robust synthesis LRSyn

```
Require: Training set $D_{tr} \subseteq D$
Require: Annotation $\mathcal{A}(doc)$ for each document $doc \in D_{tr}$
Require: Region extraction DSL $\mathcal{L}_{rx}$
Require: Value extraction DSL $\mathcal{L}_{vx}$
1: $[(C, m)] \leftarrow$ InferLandmarksAndCluster$(D_{tr}, \mathcal{A})$
2: for Cluster and landmark $(C_i, m_i) \in [(C, m)]$: do
3:     $(RProg_i, b_i, EProg_i) \leftarrow$
           SynthesizeExtractionProgram$(C_i, m_i, \mathcal{A}, \mathcal{L}_{rx}, \mathcal{L}_{vx})$
4: return Extract$(\{(m_i, RProg_i, b_i, EProg_i)\})$
```

the subroutine SynthesizeExtractionProgram to synthesize a region extraction program $RProg_i$, a region blueprint b_i, and a value extraction program $EProg_i$. The algorithm combines these with the landmark to output an extraction program in the Landmark-based DSL $\mathcal{L}_{ld}$, which can be executed using semantics shown in Algorithm 1.

4.2 Clustering Documents and Inferring Landmarks

The InferLandmarksAndCluster procedure (Algorithm 3) outlines how we jointly perform clustering and landmark detection, using the approach described in Section 2.1.

Initial clustering. Lines 2-3 perform the initial clustering to obtain the initial fine-grained clusters. Here, the clustering is by the blueprint of the whole document, and hence, two documents will be in the same cluster only if they have more or less exactly the same format with little or no variations.

Landmark and blueprint identification. The procedure LandmarkCandidates identifies common values in the documents of C_i as landmark candidates and orders them by a *scoring function* (line 6). The scoring function is based on two features: (a) the distance between the landmark candidate and the annotated values in the document, and (b) the size of the region that encloses the landmark candidates and annotated values. These features were determined after initial experiments on a small fraction of our evaluation datasets. The procedure LandmarkCandidates only return candidates with a score over a certain threshold. Then, for each landmark candidate and document, we compute and store the blueprint of the ROI in lines 8-9.

Algorithm 3 Joint clustering and landmark inference
Procedure InferLandmarksAndCluster$(D_{tr}, \mathcal{A})$

```
Require: Training dataset $D_{tr}$ along with annotations $\mathcal{A}$.
Require: Blueprint function BP.
Require: Blueprint distance metric dataset $\delta$.
1:     ▷ Initial clustering using whole document blueprints
2: $\Delta_{fine}(doc, doc') \leftarrow \delta(BP(doc), BP(doc')), \forall doc, doc' \in D_{tr}$
3: $[C] \leftarrow$ Cluster$(D_{tr}, \Delta_{fine})$
4:     ▷ Compute landmark and blueprint candidates
5: for $C_i \in [C]$ do
6:     $M_i \leftarrow$ LandmarkCandidates$(C_i, \mathcal{A})$
7:     for $doc \in D_{tr}$ do
8:         $R_{doc,m} \leftarrow$ EncRgn$(\mathcal{A}(doc) \cup$ Locate$(m, doc))$
                                        $, \forall m \in M_i$
9:         $roi[doc] \leftarrow \{(m, BP(R_{doc,m})) \mid m \in M_i\}$
10:                                          ▷ Merge clusters
11: Define $\Delta_c(doc_1, doc_2) \leftarrow$
        $\min(\{\delta(b_1, b_2) \mid (m_{1,2}, b_{1,2}) \in roi[doc_{1,2}] \wedge m_1 = m_2\}$
12: while No change in $[C]$ do
13:     Let $\Delta(C_1, C_2) = $ Avg$(\{\Delta_c(doc_1, doc_2) \mid doc_i \in C_i\})$
14:     if $\exists C_1, C_2 \in [C]$ such that $\Delta(C_1, C_2) \leq$ threshold then
15:         $[C] \leftarrow ([C] \setminus \{C_1, C_2\}) \cup \{C_1 \cup C_2\}$
16: return $[C,$ TopLandmarkCandidate$(C)]$
```

Coarse-grained clustering. Now, for each document, we have a number of landmark candidates along with their associated ROIs. With the ROIs, we can now define a coarse-grained distance over documents that is based only on the blueprints of the local structure of ROIs (line 11). With this coarse-grained distance, we now repeatedly merge clusters based on their average document distance (line 12-15). Since the coarse-grained distances are based on the blueprints of ROI, we now have clusters that are solely based on the local structure, which was our intention in the first place.

Finalizing landmarks. Finally, the procedure returns each coarse-grained cluster along with its top landmark candidate.

4.3 Synthesizing Extraction Programs

The SynthesizeExtractionProgram procedure (Algorithm 4) outlines how we process a cluster with a given landmark, and calculate a region extraction program, blueprint, and a value extraction program. The algorithm takes as

Algorithm 4 Synthesize Extraction Program

```
Proc. SynthesizeExtractionProgram($C$, m, $\mathcal{A}$, $\mathcal{L}_{rx}$, $\mathcal{L}_{vx}$)
Require: Cluster $C$ with annotations $\mathcal{A}$
Require: Landmark value m
Require: Region and value extraction DSLs: $\mathcal{L}_{rx}$ and $\mathcal{L}_{vx}$
 1: for all doc$_i \in C$ define:
 2:     $\ell_i \leftarrow$ Locate(m, doc$_i$)
 3:     (Locs$_i$, Agg$_i$) $\leftarrow \mathcal{A}$(doc$_i$)
 4:     R$_i \leftarrow$ EncRgn($\{\ell_i\} \cup$ Locs$_i$, doc$_i$)
 5:                                  ▷ Synthesize region program
 6: RegionSpec $\leftarrow \{($doc$_i, \ell_i) \mapsto$ R$_i \mid$ doc$_i \in C\}$
 7: RProg $\leftarrow$ Synthesize(RegionSpec, $\mathcal{L}_{rx}$)
 8:                                  ▷ Compute region blueprint
 9: b $\leftarrow$ Average($\{$BP(RegionSpec(doc))) $\mid$ doc $\in C\}$)
10:                                  ▷ Synthesize extraction program
11: ValueSpec $\leftarrow \{$R$_i \mapsto$ Agg$_i$(Locs$_i$) $\mid$ doc$_i \in C\}$
12: EProg $\leftarrow$ Synthesize(ValueSpec, $\mathcal{L}_{vx}$)
13: return (RProg, b, EProg)
```

input: (a) a cluster C and corresponding landmark m (b) the annotations $\mathcal{A}$ for the documents in C, and (c) the DSLs for region programs and extraction programs.

In the first step, the algorithm computes the ROI R$_i$ for each document doc$_i$ from the landmark and the annotations (lines 1-4). Then, we synthesize the region program RProg using a set of examples of the form $(\ell_i, \text{doc}_i) \mapsto \text{R}_i$ (lines 6 and 7). We also compute the average or typical blueprint b for all the ROIs in the cluster (line 9). The region extraction program RProg and filtering based on blueprint b (used in the execution semantics in Algorithm 1) together act as a robust system for detecting the ROIs. Next, we synthesize a value extraction program EProg using examples where the inputs are the ROIs in the document, and the outputs are the expected field values (line 11 and 12). The region and value extraction programs work not only on the documents in C, but typically also on unseen documents and formats where the global structure changes, without changes in the ROI.

The algorithm finally returns (RProg, b, EProg), which is combined with the landmark value m to produce a complete extraction program in the landmark DSL $\mathcal{L}_{ld}$ in Algorithm 2.

5 Instantiating LRSyn

We instantiate LRSyn for the domains of HTML documents and form images like the ones shown in Figure 1, describing in detail the region and value extraction DSLs

5.1 HTML Documents

Landmarks and Landmark Candidates. We use *n-grams* as landmarks ($n \leq 5$), and the Locate function lists all the document DOM nodes and finds those containing the landmark n-gram. The LandmarkCandidates procedure for identifying the top landmark candidates lists all n-grams in the document, filters out those containing stop words, retains those n-grams common to all documents in the cluster, and then scores them according to the criteria from Section 4. In particular, the score for a landmark candidate m is given by a weighted sum of: (a) the number of nodes in the path from the DOM nodes corresponding to m and field value v, (b) the number of nodes in the smallest region enclosing both m and v, and (c) the Euclidean distance between m and v in the rendered document.

Blueprints. We define the blueprint of a region to be the set of XPaths to the *common value* DOM nodes in the region, ignoring the DOM node order. For example, the XPath `body[1]/table[4]/tr[3]/td[2]` is simplified to `body/table/tr/td` before adding it to the blueprint set.

Region Extraction DSL. A program in the region extraction DSL $\mathcal{L}_{rx}$ is a pair of integers (parentHops, siblingHops) of *parent hops* and *sibling hops*. Given a landmark location ℓ, the semantics of the (parentHops, siblingHops) is as follows: (a) from ℓ go up the DOM tree parentHops steps to get node n_1, (b) from n_1 go siblingHops right to obtain node n_2, and (c) the result is the set of all descendants of all sibling nodes between n_1 and n_2 (inclusive). For synthesizing program in $\mathcal{L}_{rg}$ given the landmark location ℓ and the annotated location Locs, we first take the lowest common ancestor (LCA) n of ℓ and all nodes in Locs. The parentHops is given by the difference in depths of n and ℓ minus 1, and siblingHops is given by the difference in index of the left-most and right-most child of n that have one of ℓ or Locs as a descendant.

Value Extraction DSL. For the value extraction DSL $\mathcal{L}_{vx}$, we build upon the synthesis techniques from [44] and [21], as in [23]. We do not discuss the DSL and synthesis techniques in detail, but refer the reader to [23]. From a bird's eye view, a program in $\mathcal{L}_{vx}$ consists of two parts: a web extraction program which extracts the particular DOM node which contains the field value, and a text extraction program which extracts the field value from the text present in the extracted DOM node. Given an example R $\mapsto v$, the synthesis procedure first finds the DOM node n which contains the text v. Then, we use R $\mapsto n$ as the example to synthesize the web extraction program using techniques from [44], and text(n) $\mapsto v$ as the example to synthesize the text extraction program using techniques from [21].

Example 5.1. Consider the task of extracting departure time from the email in Figure 1 (a). The synthesized region extraction, value extraction programs and blueprint are shown in Figure 3.

5.2 Form Images

This domain concerns images that are obtained by scanning or photographing of physical paper documents. These images are first processed by an Optical Character Recognition (OCR) technique to obtain a list of text boxes along with their coordinates. The form images domain is significantly more complex than the HTML domain as: (a) The OCR output is generally very noisy, sometimes splitting up field values into

```
RProg     := Disjunct(path, path, ...)
path      := input | Expand(path, motion)
motion    := Absolute(dir, k)
           | Relative(dir, pattern, inclusive)
dir       := Top | Left | Right | Bottom
```

Figure 6. The Form Images Region extraction DSL $\mathcal{L}_{rx}$

a varying number of different text boxes. (b) These documents do not come equipped with a hierarchical structure that defines natural regions.

Landmarks and Landmark Candidates. As in the HTML case, we use *n*-grams as landmarks. The Locate and LandmarkCandidates functions work similarly to the HTML case with OCR output text boxes replacing DOM nodes. The scoring function for LandmarkCandidates computes the score for a landmark candidate m as a weighted sum of: (a) the Euclidean distance between m and field value v, and (b) the area of the smallest rectangle that encloses both m and v.

Blueprints. Rather than considering all common boxes for blueprinting as in the HTML case, we instead use only the boxes containing the top 50% most frequent *n*-grams. The blueprint of a region is defined to be the BoxSummary of each such box taken in document order. The BoxSummary of box consists of 2 parts: (a) The frequent *n*-gram that is present in the box, and (b) For each of the directions top, left, right, and bottom, the content type in the text box that immediately neighbors box in the direction. The content type of a box is either: (1) $\perp$ if the box does not exist, (2) the frequent *n*-gram in the text of the box if one exists, and (3) $\top$ if the box exists, but does not contain a frequent *n*-gram.

Example 5.2. Consider the text box enclosing *Engine number* in the Accounts Invoice image in Figure 1(c). The *n*-gram *Engine number* is frequent and hence, is included in the blueprint. The BoxSummary of the box is given by: $\langle$ngram $\mapsto$ *Engine number*, Top $\mapsto \perp$, Left $\mapsto$ *Chassis number*, Right $\mapsto$ *Reg Date*, Bot $\mapsto \top\rangle$ Here, *Engine number* and *Reg Date* are also a frequent *n*-grams, while the value of the Engine number *4713872198212* is not.

Region Extraction DSL. Figure 6 depicts a novel region extraction DSL $\mathcal{L}_{rx}$ for this domain. $\mathcal{L}_{rx}$ has the following components: The top operator is a disjunction of *path programs*: operationally, these programs are executed in sequence and the first non-null result is returned. Due to the OCR noise and variations in form images, often a single non-disjunctive path program is not sufficient. Each path program starts at the input landmark and repeatedly extends the path in steps till the path's bounding box covers all the annotated values. Each extension step is specified by a direction and a motion. The motion may be *absolute* (e.g., move right by 4 boxes) or *relative* (e.g., move down till you hit a text box that matches the regex `[0-9]{5}`). The additional `inclusive` parameter indicates whether the box that matches the pattern should be included or excluded in the path.

Example 5.3. In Figure 1(c), let us consider the landmark *Chassis number* and the annotated value *WDX 28298 2L SHX 3* . The field value here is a variable-length string and the OCR splits the value into $1-4$ separate boxes. Consider the two region programs given below:

- $\mathsf{Ext}(\mathsf{Ext}(\mathsf{input}, \mathsf{Abs}(\mathsf{down}, 1)), \mathsf{Rel}(\mathsf{Right}, [0-9]\{13\}, \mathsf{false}))$
- $\mathsf{Ext}(\mathsf{Ext}(\mathsf{input}, \mathsf{Abs}(\mathsf{down}, 1)), \mathsf{Rel}(\mathsf{Right}, \mathsf{DATE}, \mathsf{false}))$

Both programs first move one step down from the landmark *Chassis number*. However, the first moves to the right till it hits a 13 digit engine number, while the other till it hits a date. In case the engine number is present in a given form, the first program produces a path which ends with the annotated value, while the second one does so if the engine number is absent. When combined disjunctively in the right order, they together cover both cases.

The synthesis algorithm for $\mathcal{L}_{rx}$ is split into two parts: generating path programs and selecting path programs to construct a disjunction. Fix a set of input documents with annotations. We first synthesize path programs for small subsets (size ≤ 3) of input documents. For synthesizing path programs, we use *enumerative synthesis* [6, 53] to generate numerous candidate programs and then filter them by whether they cover the annotated values when starting from the landmark. We enumerate paths of up to 4 motions, bounding k to positive integers < 5. For `pattern`, we enumerate a finite set of regular expression patterns generated using a string profiling technique [11, 40] over all the common and field text values present in the cluster. For example, when given a cluster of documents of similar to Figure 1c), one of the patterns returned is $[0-9]\{13\}$ as the cluster contains many engine numbers of that form.

After the enumeration step, we have a collection $\{P_1, \ldots, P_n\}$ of path programs that are each synthesized from a small subset of input examples. Now, we use the NDSyn algorithm from [23] to select a subset of these programs to construct the disjunctive program. Now, for each program P_i, we define the set Ex_i to be the subset of Examples that P_i is correct on. The NDSyn algorithm selects a subset of $\mathcal{P}$ of programs such that $\bigcup_{P_i \in \mathcal{P}} \mathsf{Ex}_i = \mathsf{Examples}$, optimizing for F1 score and program size [23].

Example 5.4. Consider the two path programs from Example 5.3, along with the additional program $\mathsf{Ext}(\mathsf{Ext}(\mathsf{input}, \mathsf{Abs}(\mathsf{Down}, 1)), \mathsf{Abs}(\mathsf{Right}, 2))$. Given a collection of such path programs, NDSyn builds the disjunctive program using the two from Example 5.3 as they cover a large fraction of the documents in the cluster. The additional program above will be ignored as it is only correct when the chassis number field value is split into 2 boxes by the OCR.

Value Extraction DSL. For the value extraction DSL, we use FlashFill [21]. The input to the value extraction program is the concatenation of all the text values in the boxes returned by the path program.

6 Discussion

6.1 Hierarchical Landmarks

Consider again the email in Figure 1b) and a variant where the term *Pick-up* has been replaced by *Depart*. Now, using the landmark *Depart* for extraction will unintentionally also extract the car trip departure time. In Section 2, the human used a hierarchy of landmarks (i.e., *AIR* followed by *Depart*) to obtain the correct results. Algorithm 2 can similarly be extended to hierarchical extractions.

Say we first synthesized a program Prog_0 using Algorithm 2 that uses the landmark *Depart*. We run Prog_0 on the training document and realize that a spurious landmark location (car departure) is identified. In this case, we use the correct landmark locations (i.e., the first and last occurrence of *Depart*) as a new annotation. Running Algorithm 2 with this annotation will produce a program Prog_1 that uses *AIR* as a landmark and extracts precisely the relevant occurrences of *Depart*. At inference time, we run Prog_1 to identify only the correct occurrences of *Depart* and then run Prog_0 starting with only those occurrences of *Depart*. In our implementation in Section 7, we have implemented the full hierarchical extraction algorithm for the HTML domain.

6.2 Robustness of LRSyn

By design, LRSyn is robust to format variations that change (a) document structure outside the ROIs, (b) position of ROIs, and (c) the number of ROIs. However, there are 2 clear limitations to the robustness of LRSyn: If a format changes by adding a new part (e.g., car departure time) that contains the landmark the LRSyn generated program uses, the program may generate a spurious output. This is only a problem if the new part added also has similar blueprint to the existing ROIs. For example, the program will not be mislead by an new banner advertisement saying *Depart today for your dream destination!* The second case is when the format inside the ROI changes. In this case, the underlying assumption about invariant local structure is violated and LRSyn is unlikely to cope with this variation. One possible solution that we discuss in Section 9 is to use a trained ML model to automatically re-synthesize as in [23].

7 Evaluation

We evaluate LRSyn in two scenarios, each with different document types: (1) HTML documents from travel reservation emails and (2) Form image documents from invoices, receipts, and travel reservation emails. In each case, we perform experiments in 2 settings: *contemporary* and *longitudinal*. In the contemporary setting, the training data and test data consist of documents authored and collected during the same time period. On the other hand, for the longitudinal setting, the test data was collected several months after the training data, allowing for new formats to organically enter the dataset. For both domains, we are attempting to answer the following research question:

How does LRSyn *compare with previous approaches in the contemporary and longitudinal settings?*

Before describing the results, we first discuss the 3 threshold parameters that control various aspects of learning and inference in LRSyn. We pick both the cluster merging threshold in Algorithm 3 and the blueprint distance threshold in Algorithm 1 to be 0, i.e., an *exact match*. Note that the blueprint functions already lose information—hence, an exact match does not mean the ROIs need to have exactly the same format. For the score threshold for the procedure LandmarkCandidates, we pick a threshold that gives us around 10 landmark candidates in each case.

We discuss the results on the HTML and form image documents in Sections 7.1 and 7.2, and discuss some secondary results on the nature of programs learned by LRSyn in Section 7.3. In Section 7.4, we discuss how robust the experimental results are to various choices in the experimental setup.

7.1 HTML Extraction

Our HTML document dataset, called the *machine-to-human (M2H)* email dataset, consists of anonymized flight reservation emails. It consists of 3503 emails from 6 different flight providers and is divided into training and test sets of size 362 and 3141, respectively. For each provider, we synthesize and test programs using Algorithm 2, as instantiated in Section 5.1. We compare against 2 state-of-the-art techniques, namely NDSyn [23] and ForgivingXPaths [39].

Overall results. Table 1 shows the average precision, recall and F1 scores across various extraction tasks for ForgivingXPaths, NDSyn and LRSyn for both contemporary and longitudinal setting. As seen in the table, LRSyn has near perfect precision and recall, with NDSyn performing quite well with numbers > 0.9. Further, as expected, the gap between LRSyn and NDSyn is higher in the longitudinal dataset indicating that LRSyn can cope with format variations better than NDSyn.

Unlike NDSyn or LRSyn which use a combination of structure and text programs for extraction, ForgivingXPaths only outputs XPaths which correspond to the entire node, rather than the sub-text contained within that node. Consequently, it has high recall and poor precision when the field value is a substring of the entire DOM node text. We therefore omit it from the more detailed results below.

Detailed comparison. Table 2 shows a more detailed drill-down of the F1 scores for NDSyn and LRSyn, in the two

Table 1. Overall scores of LRSyn and NDSyn on the M2H Contemporary and Longitudinal datasets

Contemporary			
Metric	ForgivingXPaths	NDSyn	LRSyn
Avg. Precision	0.17	0.96	1.00
Avg. Recall	0.99	0.91	1.00
Avg. F1	0.22	0.93	1.00

Longitudinal			
Metric	ForgivingXPaths	NDSyn	LRSyn
Avg. Precision	0.15	0.99	1.00
Avg. Recall	0.98	0.89	1.00
Avg. F1	0.20	0.92	1.00

settings. In summary, in all cases, the hit to the F1-scores come from lower precision numbers rather than lower recall numbers.

LRSyn is very robust to variations in the longitudinal setting, achieving $>$ 95% F1 score in all 53 out of 53 fields, with a perfect F1 score of 1.00 in 49 cases. In comparison, NDSyn achieves $>$ 95% and perfect scores in 40 and 33 cases respectively.

In many applications, having a score of 1.00 is crucial, i.e., even a system with 0.99 precision cannot be deployed in practice. For example, even a tiny imprecision in a system that automatically adds calendar entries based on flight reservation emails is disastrous for millions of users. Comparing the numbers precisely, LRSyn outperforms NDSyn in 19 and 20 out of the 53 fields in the contemporary and longitudinal setting, respectively. In the remaining fields, the two approaches have comparable F1 scores.

We examined the domains *aeromexico* and *mytrips.amexgbt* where both LRSyn and NDSyn achieved perfect scores. In the *aeromexico* domain, each field has a unique dedicated ID attribute in the HTML domain which act as *implicit landmarks*, and both NDSyn and LRSyn are able to latch on to this unique ID. For example, the arrival time and departure city DOM nodes have id *arrival-city* and *departure-city*, and NDSyn produces a program that searches for this ID across the whole document, emulating the landmark location step of a LRSyn program.

In the *mytrips.amexgbt* domain, the NDSyn program while perfectly accurate on all the variations in our dataset, is very fragile. In the final CSS selector step of the web extraction component, it looks for the 10^{th} child of a DOM element corresponding to a flight details section. Incidentally, all new sections (car reservations, hotel reservations, etc) added in the new variations have at most 5 children, and hence, they are automatically ignored by the NDSyn program. Any new variation that would add a long enough section will break this program. In contrast, the LRSyn program narrows down on the right region with a landmark and is resistant to such variations.

7.2 Form Image Extraction

We consider two datasets of form images. For both datasets, we use a training data size of 10 for each field, and compare against Azure Form Recognizer (AFR) [36], a cloud-based form data extraction service.

- *Finance* dataset: This consists of 850 images of receipts, purchase orders, credit notes, sales invoice and similar such documents. Here, the training and test data are from the same time period and we only evaluate the contemporary setting.
- *M2H-Images* dataset: We convert M2H emails from 4 domains to images and extract the same fields as before. This represents common scenarios in practice where HTML documents such as booking mails or receipts may be printed and then scanned again, say when expense reports are filed. The OCR service we use produced extremely poor results on 2 of the 6 domains from the HTML experiments, and hence, we used only 4 domains in this dataset (The same OCR service is used by our baseline as well, see below).

Overall Results. Table 5 shows the average precision, recall, F1 and accuracy scores for AFR and NDSyn for both the Finance and M2H-image datasets. As we can see from the table, both LRSyn and AFR perform very well on the Finance dataset, with LRSyn performing marginally better. In this dataset, the image formats do not vary much, resulting in these high-quality results.

LRSyn outperforms AFR, a state-of-the-art industrial neural form extraction system with just 10 training images per field, having a precision of 0.97 vs 0.90 on the M2H-Images dataset.

Detailed comparison. Table 3 shows the results of AFR and LRSyn on the Finance dataset with respect to 34 extraction tasks. LRSyn performs better than AFR in 12 out of the 34 cases and is on par on the rest, with significant gains in some domains like "AccountsInvoice".

Though the neural model in AFR is trained with thousands of invoices and receipts, and further fine-tuned with our training data, we observe that it is sensitive to the region coordinates in a given document. If these regions are translated, or if the document scan is tilted, AFR produces erroneous results. On the other hand, LRSyn is partially robust to such changes as we use a text landmark. AFR is marginally better than LRSyn in some extraction tasks. These are cases where there is no clear bounding pattern for the field values. On the other hand, AFR's semantic understanding of the data is not affected by boundary text patterns.

Table 4 shows the results of AFR and LRSyn on the M2H dataset with respect to 45 extraction tasks. This dataset exhibits more variations at the visual level as compared to the Finance dataset, and hence, LRSyn performs better than NDSyn in 35 out of the 45 tasks and is on par on most of the remaining extraction tasks. There is 1 specific case where LRSyn fails altogether, producing no programs. These are cases where there is no local textual landmark geometrically

Table 2. F1 scores of NDSyn and LRSyn for M2H HTML dataset. The Pvdr field is not relevant for iflyalaskaair

		Contemporary		Longitudinal			Contemporary		Longitudinal			Contemporary		Longitudinal	
Fields	Domain	NDSyn	LRSyn	NDSyn	LRSyn	Domain	NDSyn	LRSyn	NDSyn	LRSyn	Domain	NDSyn	LRSyn	NDSyn	LRSyn
AIata		0.81	**1.00**	0.64	**1.00**		0.75	**1.00**	0.74	**1.00**		1.00	1.00	1.00	1.00
ATime		0.76	**1.00**	0.62	**1.00**		0.94	**1.00**	0.91	**1.00**		1.00	1.00	1.00	1.00
DIata		0.73	**1.00**	0.55	**1.00**		0.94	**1.00**	0.95	**1.00**		1.00	1.00	1.00	1.00
DDate	ifly	1.00	1.00	1.00	1.00		0.98	**1.00**	0.95	**1.00**	aero	1.00	1.00	1.00	1.00
DTime	alaska	0.73	**1.00**	0.55	**1.00**	getthere	0.76	**1.00**	0.78	**1.00**	mexico	1.00	1.00	1.00	1.00
FNum	air	1.00	1.00	1.00	1.00		0.98	**1.00**	0.98	**1.00**		1.00	1.00	1.00	1.00
Name		1.00	1.00	0.99	0.99		1.00	1.00	0.89	**1.00**		1.00	1.00	1.00	1.00
Pvdr		–	–	–	–		0.98	**1.00**	0.97	**1.00**		1.00	1.00	1.00	1.00
RId		1.00	1.00	1.00	1.00		0.93	**1.00**	0.94	**1.00**		1.00	1.00	1.00	1.00
AIata		0.67	**1.00**	0.67	**1.00**		1.00	1.00	1.00	1.00		1.00	1.00	1.00	1.00
ATime		NaN	**1.00**	NaN	**1.00**		1.00	1.00	1.00	1.00		1.00	1.00	1.00	1.00
DIata		0.67	**1.00**	0.67	**1.00**		1.00	1.00	1.00	1.00		1.00	1.00	1.00	1.00
DDate		0.67	**1.00**	0.67	**1.00**		0.94	**1.00**	0.95	**1.00**	mytrips	1.00	1.00	1.00	1.00
DTime	airasia	NaN	**1.00**	NaN	**1.00**	delta	1.00	1.00	1.00	1.00	amex	1.00	1.00	1.00	1.00
FNum		1.00	1.00	0.96	0.96		1.00	1.00	1.00	1.00	gbt	1.00	1.00	1.00	1.00
Name		1.00	1.00	1.00	1.00		0.85	**0.97**	0.91	**0.97**		1.00	1.00	1.00	1.00
Pvdr		1.00	1.00	0.96	0.96		1.00	1.00	1.00	1.00		1.00	1.00	1.00	1.00
RId		1.00	1.00	1.00	1.00		1.00	1.00	1.00	1.00		1.00	1.00	1.00	1.00

Table 3. F1 scores for Finance dataset

Domain	Fields	AFR	LRSyn
	Amount	0.99	**1.00**
	Chassis	0.82	**0.99**
	CustAddr	0.98	*0.96*
	Date	0.93	**0.98**
AccountsInvoice	Dnum	0.96	**0.97**
	Engine	0.82	**1.00**
	InvoiceAddress	0.90	**0.95**
	Model	0.75	**1.00**
	Amount	1.00	1.00
	Chassis	0.99	0.99
	CustAddr	0.99	*0.97*
CashInvoice	Date	0.99	0.99
	Dnum	0.96	0.96
	Engine	0.93	**0.95**
	InvoiceAddress	0.99	0.99
	Model	0.99	**1.00**
	Amount	1.00	1.00
	CreditNoteAddress	0.99	**1.00**
CreditNote	CreditNoteNo	0.94	*0.93*
	CustRefNo	1.00	1.00
	Date	1.00	1.00
	RefNo	1.00	1.00
	Amount	1.00	1.00
	CustomerReferenceNo	1.00	1.00
SalesInvoice	Date	1.00	1.00
	InvoiceAddress	0.94	**0.99**
	RefNo	0.99	0.99
	SalesInvoiceNo	0.99	0.99
	Amount	1.00	1.00
	CustomerAddress	1.00	*0.99*
SelfBilledCreditNote	CustomerReferenceNo	0.99	0.99
	Date	1.00	1.00
	DocumentNumber	1.00	1.00
	VatRegNo	1.00	1.00

Table 4. F1 Score for M2H-Images Dataset

Fields	Domain	AFR	LRSyn	Domain	AFR	LRSyn
AIata		0.62	**0.65**		0.94	**1.00**
ATime		0.69	**0.99**		0.87	**1.00**
DIata		0.36	**0.66**		0.93	**1.00**
DDate		0.71	**0.89**		0.96	**0.99**
DTime	aeromexico	0.65	**0.97**	getthere	0.88	**1.00**
FNum		0.66	**0.83**		0.94	**1.00**
Name		0.96	**0.98**		0.99	0.99
Pvdr		0.69	**0.78**		0.75	**1.00**
RId		1.00	1.00		0.89	**0.95**
Fields	Domain	AFR	LRSyn	Domain	AFR	LRSyn
AIata		0.99	**1.00**		0.85	**0.98**
ATime		0.95	**1.00**		0.97	**1.00**
DIata		0.98	**1.00**		0.96	**0.99**
DDate	ifly.	0.98	-	mytrips	0.93	**1.00**
DTime	alaskaair	0.95	**0.98**	amexgbt	0.99	**1.00**
FNum		0.97	**1.00**		0.98	**1.00**
Name		0.98	0.98		0.98	**1.00**
Pvdr		0.93	**0.99**		0.91	**0.99**
RId		1.00	*0.86*		0.61	**0.96**

Table 5. Average precision, recall, F1 numbers on Finance and M2H-Images dataset (Ignoring DDate field in ifly.alaskaair)

Metric	AFR	LRSyn
Avg. Pre.	0.98	0.99
Avg. Rec.	0.96	0.99
Avg. F1	0.97	0.99

Finance dataset

Metric	AFR	LRSyn
Avg. Prec.	0.90	0.97
Avg. Rec.	0.93	0.97
Avg. F1	0.91	0.97

M2H-Images dataset

near the field value. However, the region around the field value may still be similar across documents. We discuss the possibility of using visual landmarks as opposed to textual ones in Section 9.

Summary of results. In the HTML domain, the prior work NDSyn is a high-performing system with F1 scores in the range of 0.9. LRSyn is able to push the F1 scores to a perfect 1.0 in most cases. In longitudinal scenarios, LRSyn improves NDSyn in 20 out of 53 fields, with significant lift in F1 scores in many cases. In the images domain, even with very little training data, LRSyn matches AFR, which is a released product on contemporary settings, and outperforms AFR in longitudinal settings. In addition, LRSyn produces simpler interpretable programs that match human intuition for extraction, and are much easier to maintain. Hence, we see a lot of promise in this approach.

7.3 Nature of Synthesized Programs

Additionally, we performed secondary analysis to understand the features of the LRSyn programs.

Program size. In the HTML domain, we compared size of the LRSyn and NDSyn programs. Since the programs are naturally of different shapes, we only compare the web extraction part of the programs. The final text extraction program is generally the same across both algorithms. Note that these numbers need to be taken in the context that LRSyn programs additionally have a landmark and blueprint.

For the M2H dataset, the web extraction part of LRSyn programs have 2.95 CSS selector components as compared to 8.51 for NDSyn.

Quality of Inferred Landmarks. We infer landmarks automatically using the techniques and scoring functions from Sections 4 and 5. To check the quality of landmark inference, we also asked data annotators to tag landmarks manually.

In 57 out of 63 clusters across all fields, the inferred landmarks are the same as manually provided landmarks. In 5 of the remaining 6 cases, the human annotator agreed that the inferred landmark was of equal quality.

In the remaining 1 case, the algorithm chose the human annotated landmark as well, but in addition chose a disambiguating hierarchical landmark of low quality. In particular, it disambiguated the term *Name* occurred in reference to both the name of the passenger and the name in the billing address. Here, the algorithm chose to disambiguate using the term *Meal*, i.e., passengers have a meal preference while the billed person does not.

7.4 Robustness of Experimental Results

Training set choice. In our experiments, the training set is small compared to the full dataset leading to a possibility of over-fitting, with different training sets potentially producing significantly differing results. However, our techniques are robust even with small training sets: (a) Landmark identification can leverage the full dataset of both labeled and unlabeled documents. (b) LRSyn does not need to see all format variations that differ only outside the ROIs, and a small set covering only the variations within the ROIs is sufficient. To confirm this, we reran all our experiments on the M2H-HTML dataset with 4 different randomly chosen training datasets. In all runs, the F1 scores of the generated programs for each field and domain varied by no more than 0.01 from the results presented in Table 2, confirming our hypothesis that the results are robust to training set choice.

Landmark identification threshold. For the landmark candidate score threshold, we picked threshold values that resulted in ~10 candidates for each case. To study the robustness of the results to this choice, we reran all experiments with a threshold value that returned $2X$ as many candidates. The obtained results were exactly identical to the results presented in the previous sections. This is expected as "bad" landmark candidates are eliminated in subsequent steps, i.e., there is usually no program that extracts the required field value starting from the landmark. Hence, as long as the threshold is high enough to allow for some good landmark candidates, it does not matter how many bad landmark candidates are included.

8 Related Work

Program synthesis. Data extraction has been an active area of investigation in the program synthesis community. FlashExtract [28] synthesizes programs from examples for extraction from large text or web documents, using an algebra of pre-defined operators such as map, reduce and filter. FlashExtract works well when inputs are homogeneous. For heterogeneous inputs, several works have explored *disjunctive* synthesis approaches [7, 8, 45, 47, 48]. *Forgiving XPaths* [39] in particular focuses on synthesizing progressively relaxed XPaths to increase recall for web extraction. Hybrid synthesis [46] proposes a combination of deductive [41] and predictive [44] synthesis techniques to create programs that follow aligned structures inferred on webpages, but this is primarily focused on tabular data extraction. Raychev et al. [43] also learn programs from noisy data, but generate a single program from a noisy dataset rather than using clustering to learn different programs applicable to different region formats. In contrast to the above techniques, we propose the idea of landmarks and regions as a compositional approach that addresses the high noise and heterogeneity, and support evolution in the formats over time.

Wrapper induction. The goal of wrapper induction is to generate a set of extraction rules from an annotated HTML document. Wrapper induction is an active area of research with techniques based on supervised learning [50, 59, 61–63], program synthesis [44–46], unsupervised data mining [9, 15, 24, 57], and programming by demonstration [4]. While some of these works deal with HTML documents as a series of tokens [22, 25], others can leverage the DOM tree structure of HTML [38, 49]. The main distinguishing feature of our work is that LRSyn is a generic framework that can be instantiated across varied data formats as opposed to being restricted to HTML documents. Wrapper induction literature has also examined the possibility of repairing extraction rules when new data of different formats arrives [14, 27, 30, 42]. LRSyn, due to its robustness, alleviates the need to update extraction rules very frequently. However, extraction rules will still occasionally break when format changes significantly, i.e., when the landmarks change or there are format changes within the ROI. In these cases, incremental learning through repair is an interesting direction to explore in the future.

The closest wrapper induction work to ours is Muslea et al [37], where the authors use landmarks in a spirit similar to LRSyn. In [37], the wrapper induction rules go directly from the landmark to the field value using a path of parent-child relations where each step in the path specifies the type of DOM node. In contrast, our technique has an intermediate step—we go from the landmark to the ROI and from the ROI to the field value. This affords us greater flexibility in the class of synthesis techniques that can be used for extracting the field value from the ROI rather than just following a fixed sequence of parent-child hops. Another point of interest is that we use both parent hops and sibling hops in going from the landmark to the region—this simplifies the extraction task by allowing a larger class of landmark candidates.

Web testing and automation. The web testing and automation domain and HTML data extraction share a number of techniques related to robustly identifying a DOM element (field value in data extraction and element under test for web automation) In web testing, we want to robustly specify actions in the testing script—for example, rather than specifying "click the button at XPath body[2]/table[4]/tr[1]/td[2]" we want to specify "locate the text label *Complete purchase* and click the button near it". The former is not robust to changes in the global page format while the latter is. Several tools, both research and commercial, provide web developers a high level language or framework to write robust automation scripts [1–3, 31–33]. Some of these tools even allow the developer to use annotation or labeling to specify DOM elements from which scripts are automatically generated [2, 31, 33]. Most web automation techniques deal with a single annotated document for training instead of a full collection like in the data extraction setting—hence, they do not deal with the heterogeneity problem. The closest related work in this domain is Yandrapally et al [56] where landmarks are inferred as an unambiguous ancestor of the target DOM node. However, a node that is unambiguous for one document is not necessarily so across multiple heterogeneous documents. In fact, the 2 level clustering and landmark inference is crucial to the LRSyn framework, while these steps are irrelevant in the web automation scenario. Another difference between LRSyn and [56] is that we use a full-fledged DSL and synthesizer for extracting the field value from the region which facilitates more extractions on regions, as opposed to the "near" operator. Another related work is [10], where the authors use a node similarity based addressing technique to identify nodes corresponding to "labelled" DOM nodes. Here, the authors do not use any document structure at all and instead, locate the target as the node that is most similar to the labelled node according a weighted measure of the attributes common to both nodes. This manner of node identification while robust to a certain extent, fails when the identification is over a set of heterogeneous documents from different sources.

Machine learning. ML-based data extraction techniques have been explored in both the web and document image extraction domains, ranging from neural networks [13, 17, 20, 34, 55], probabilistic models [16, 60] and markov logic networks [51]. In general, ML approaches train opaque models that are neither readable nor editable by the user. One exception is the area of wrapper induction [26] which learns sets of XPath expressions to handle noise [16, 18, 29, 35]. Due to the global nature of XPaths, when applied to large heterogeneous datasets, such approaches can lead to a large number of expressions in order to cover irrelevant variations in document formats. In contrast, our technique is local and modular, and results in smaller programs in both size and number. Our technique also applies to general DSLs in different domains rather than just XPaths for web extraction. Ideas around exploiting compositionality and data invariance have also been explored in previous works: [5, 12] use commonly reoccurring phrasal patterns for web extraction given a seed set; in the vision community, modular approaches such as convolutional neural networks have been used for document image extraction [17, 34, 52, 55, 58], and notably algorithms based on R-CNN [19] use selective search to focus attention on a small number of regions from the image (*region proposals*). Our core ideas are similarly based around localised regions, but we detect them by identifying landmarks that present a common kind of invariance in formed documents.

Hybrid approaches. Recent approaches have combined program synthesis and ML techniques for data extraction. The closest related work in this area is [23], where an ML model is used to get an initial labeling of potential attribute values, and the noisy labels produced by this model are used to create interpretable programs using synthesis techniques. While this approach shows improved robustness, it still generates global programs that can fail with irrelevant changes to the document format, and we show in this work how our compositional synthesis approach performs better empirically in practice. There has been very limited work in the area of synthesis for document image extraction, but notable works in specialized areas include [52], where concepts from inductive logic programming are combined with neural approaches, and [58], which combines symbolic reasoning with CNNs, though interpretable programs are not generated.

9 Conclusion

Inspired by how humans search for data in formed documents, we designed a new approach to data extraction using the concepts of landmarks and regions. Our implementation of this approach, LRSyn, is robust to format changes, and achieves close to perfect F1 scores with HTML documents, and significantly high F1 scores in image documents when compared to existing approaches. LRSyn shines especially when test data has different formats than training data, which is a common pain-point in real-world applications.

While LRSyn is robust to format changes that occur outside the region of interest, we believe that we can further improve its robustness for format changes inside the regions of interest by combining it with ML approaches as in [23]. This is a very promising direction to explore due to the possibility of using smaller ML models trained directly on the regions of interest rather the the whole document. Additionally, in our image datasets, we encountered documents where there are no specific landmark phrases present, though the regions of interest have similar visual structure. Using R-CNN models to come up with visual rather than textual landmarks could improve the performance of extraction in such cases.

References

[1] [n.d.]. Beautiful soup: We called him tortoise because he taught us. https://www.crummy.com/software/BeautifulSoup/.

[2] [n.d.]. imacros. https://wiki.imacros.net/Main_Page.

[3] [n.d.]. Selenium-web browser automation. https://www.selenium.dev/.

[4] Brad Adelberg. 1998. NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. 283–294.

[5] Eugene Agichtein and Luis Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*. 85–94.

[6] Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2015. Syntax-Guided Synthesis. In *Dependable Software Systems Engineering*, Maximilian Irlbeck, Doron A. Peled, and Alexander Pretschner (Eds.). NATO Science for Peace and Security Series, D: Information and Communication Security, Vol. 40. IOS Press, 1–25. https://doi.org/10.3233/978-1-61499-495-4-1

[7] Rajeev Alur, Pavol Cerný, and Arjun Radhakrishna. 2015. Synthesis Through Unification.. In *CAV (2) (Lecture Notes in Computer Science, Vol. 9207)*, Daniel Kroening and Corina S. Pasareanu (Eds.). Springer, 163–179. http://dblp.uni-trier.de/db/conf/cav/cav2015.html#AlurCR15

[8] Rajeev Alur, Arjun Radhakrishna, and Abhishek Udupa. 2017. Scaling Enumerative Program Synthesis via Divide and Conquer.. In *TACAS (1) (Lecture Notes in Computer Science, Vol. 10205)*, Axel Legay and Tiziana Margaria (Eds.). 319–336. http://dblp.uni-trier.de/db/conf/tacas/tacas2017-1.html#AlurRU17

[9] Arvind Arasu and Hector Garcia-Molina. 2003. Extracting Structured Data from Web Pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, Alon Y. Halevy, Zachary G. Ives, and AnHai Doan (Eds.). ACM, 337–348. https://doi.org/10.1145/872757.872799

[10] Shaon Barman, Sarah Chasins, Rastislav Bodik, and Sumit Gulwani. 2016. Ringer: web automation by demonstration. In *Proceedings of the 2016 ACM SIGPLAN international conference on object-oriented programming, systems, languages, and applications*. 748–764.

[11] Ranjita Bhagwan, Sonu Mehta, Arjun Radhakrishna, and Sahil Garg. 2021. Learning Patterns in Configuration. In *ASE*.

[12] Sergey Brin. 1998. Extracting patterns and relations from the world wide web. In *International workshop on the world wide web and databases*. Springer, 172–183.

[13] Mengli Cheng, Minghui Qiu, Xing Shi, Jun Huang, and Wei Lin. 2020. One-shot Text Field labeling using Attention and Belief Propagation for Structure Information Extraction.. In *ACM Multimedia*, Chang Wen Chen, Rita Cucchiara, Xian-Sheng Hua, Guo-Jun Qi, Elisa Ricci, Zhengyou Zhang, and Roger Zimmermann (Eds.). ACM, 340–348. http://dblp.uni-trier.de/db/conf/mm/mm2020.html#ChengQSH020

[14] Boris Chidlovskii, Bruno Roustant, and Marc Brette. 2006. Documentum eci self-repairing wrappers: Performance analysis. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. 708–717.

[15] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. 2001. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass (Eds.). Morgan Kaufmann, 109–118. http://www.vldb.org/conf/2001/P109.pdf

[16] Nilesh N. Dalvi, Ravi Kumar, and Mohamed A. Soliman. 2011. Automatic Wrappers for Large Scale Web Extraction. *Proc. VLDB Endow.* 4, 4 (2011), 219–230. http://dblp.uni-trier.de/db/journals/pvldb/pvldb4.html#DalviKS11

[17] Han Fu, Yunyu Bai, Zhuo Li, Jun Shen, and Jianling Sun. 2020. A Machine Learning Framework for Data Ingestion in Document Images. *CoRR* abs/2003.00838 (2020). http://dblp.uni-trier.de/db/journals/corr/corr2003.html#abs-2003-00838

[18] Tim Furche, Jinsong Guo, Sebastian Maneth, and Christian Schallhart. 2016. Robust and noise resistant wrapper induction. In *Proceedings of the 2016 International Conference on Management of Data*. 773–784.

[19] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[20] Filip Gralinski, Tomasz Stanislawek, Anna Wróblewska, Dawid Lipinski, Agnieszka Kaliska, Paulina Rosalska, Bartosz Topolski, and Przemyslaw Biecek. 2020. Kleister: A novel task for Information Extraction involving Long Documents with Complex Layout. *CoRR* abs/2003.02356 (2020). http://dblp.uni-trier.de/db/journals/corr/corr2003.html#abs-2003-02356

[21] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices* 46, 1 (2011), 317–330.

[22] Chun-Nan Hsu and Ming-Tzung Dung. 1998. Generating finite-state transducers for semi-structured data extraction from the web. *Information systems* 23, 8 (1998), 521–538.

[23] Arun Shankar Iyer, Manohar Jonnalagedda, Suresh Parthasarathy, Arjun Radhakrishna, and Sriram K. Rajamani. 2019. Synthesis and machine learning for heterogeneous extraction.. In *PLDI*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 301–315. http://dblp.uni-trier.de/db/conf/pldi/pldi2019.html#IyerJPRR19

[24] Iraklis Kordomatis, Christoph Herzog, Ruslan R. Fayzrakhmanov, Bernhard Krüpl-Sypien, Wolfgang Holzinger, and Robert Baumgartner. 2013. Web object identification for web automation and meta-search. In *3rd International Conference on Web Intelligence, Mining and Semantics, WIMS '13, Madrid, Spain, June 12-14, 2013*, David Camacho, Rajendra Akerkar, and María Dolores Rodríguez-Moreno (Eds.). ACM, 13. https://doi.org/10.1145/2479787.2479798

[25] Nicholas Kushmerick. 1997. *Wrapper induction for information extraction*. University of Washington.

[26] N. Kushmerick. 2000. Wrapper induction: efficiency and expressiveness. *Artificial Intelligence* 118 (2000), 15–68.

[27] Nicholas Kushmerick et al. 1999. Regression testing for wrapper maintenance. In *Aaai/iaai*. Citeseer, 74–79.

[28] Vu Le and Sumit Gulwani. 2014. FlashExtract: a framework for data extraction by examples.. In *PLDI*, Michael F. P. O'Boyle and Keshav Pingali (Eds.). ACM, 542–553. http://dblp.uni-trier.de/db/conf/pldi/pldi2014.html#LeG14

[29] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. 2016. Robula+: an algorithm for generating robust XPath locators for web testing. *J. Softw. Evol. Process.* 28, 3 (2016), 177–204. https://doi.org/10.1002/smr.1771

[30] Kristina Lerman, Steven N Minton, and Craig A Knoblock. 2003. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research* 18 (2003), 149–181.

[31] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa A. Lau. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI 2008, 2008, Florence, Italy, April 5-10, 2008*, Mary Czerwinski, Arnold M. Lund, and Desney S. Tan (Eds.). ACM, 1719–1728. https://doi.org/10.1145/1357054.1357323

[32] Ian Li, Jeffrey Nichols, Tessa A. Lau, Clemens Drews, and Allen Cypher. 2010. Here's what i did: sharing and reusing web activity with ActionShot. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Atlanta, Georgia, USA, April*

10-15, 2010, Elizabeth D. Mynatt, Don Schoner, Geraldine Fitzpatrick, Scott E. Hudson, W. Keith Edwards, and Tom Rodden (Eds.). ACM, 723–732. https://doi.org/10.1145/1753326.1753432

[33] James Lin, Jeffrey Wong, Jeffrey Nichols, Allen Cypher, and Tessa A. Lau. 2009. End-user programming of mashups with vegemite. In *Proceedings of the 14th International Conference on Intelligent User Interfaces, IUI 2009, Sanibel Island, Florida, USA, February 8-11, 2009*, Cristina Conati, Mathias Bauer, Nuria Oliver, and Daniel S. Weld (Eds.). ACM, 97–106. https://doi.org/10.1145/1502650.1502667

[34] Weihong Lin, Qifang Gao, Lei Sun, Zhuoyao Zhong, Kai Hu, Qin Ren, and Qiang Huo. 2021. ViBERTgrid: A Jointly Trained Multi-Modal 2D Document Representation for Key Information Extraction from Documents.. In *16th International Conference on Document Analysis and Recognition.*

[35] 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 *CIKM*, Xue wen Chen, Guy Lebanon, Haixun Wang, and Mohammed J. Zaki (Eds.). ACM, 1517–1521. http://dblp.uni-trier.de/db/conf/cikm/cikm2012.html#LongGXK12

[36] Microsoft. [n.d.]. Azure Form Recognizer. https://azure.microsoft.com/en-in/services/form-recognizer/.

[37] Ion Muslea, Steve Minton, and Craig Knoblock. 1999. A hierarchical approach to wrapper induction. In *Proceedings of the third annual conference on Autonomous Agents.* 190–197.

[38] Jussi Myllymaki and Jared Jackson. 2002. Robust web data extraction with xml path expressions. *Technical reportz* (2002).

[39] Adi Omari, Sharon Shoham, and Eran Yahav. 2017. Synthesis of forgiving data extractors. In *Proceedings of the tenth ACM international conference on web search and data mining.* 385–394.

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

[41] Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: a framework for inductive program synthesis.. In *OOPSLA*, Jonathan Aldrich and Patrick Eugster (Eds.). ACM, 107–126. http://dblp.uni-trier.de/db/conf/oopsla/oopsla2015.html#PolozovG15

[42] Juan Raposo, Alberto Pan, Manuel Alvarez, and Angel Vina. 2005. Automatic wrapper maintenance for semi-structured web sources using results from previous queries. In *Proceedings of the 2005 ACM symposium on Applied computing.* 654–659.

[43] Veselin Raychev, Pavol Bielik, Martin T. Vechev, and Andreas Krause. 2016. Learning programs from noisy data.. In *POPL*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 761–774. http://dblp.uni-trier.de/db/conf/popl/popl2016.html#RaychevBVK16

[44] Mohammad Raza and Sumit Gulwani. 2017. Automated data extraction using predictive program synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[45] Mohammad Raza and Sumit Gulwani. 2018. Disjunctive Program Synthesis: A Robust Approach to Programming by Example.. In *AAAI*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 1403–1412. http://dblp.uni-trier.de/db/conf/aaai/aaai2018.html#RazaG18

[46] Mohammad Raza and Sumit Gulwani. 2020. Web data extraction using hybrid program synthesis: A combination of top-down and bottom-up inference. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* 1967–1978.

[47] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett. 2015. Counterexample-Guided Quantifier Instantiation for Synthesis in SMT.. In *CAV (2) (Lecture Notes in Computer Science, Vol. 9207)*, Daniel Kroening and Corina S. Pasareanu (Eds.). Springer, 198–216. http://dblp.uni-trier.de/db/conf/cav/cav2015.html#ReynoldsDKTB15

[48] Shambwaditya Saha, Pranav Garg, and P. Madhusudan. 2015. Alchemist: Learning Guarded Affine Functions.. In *CAV (1) (Lecture Notes in Computer Science, Vol. 9206)*, Daniel Kroening and Corina S. Pasareanu (Eds.). Springer, 440–446. http://dblp.uni-trier.de/db/conf/cav/cav2015-1.html#Saha0M15

[49] Arnaud Sahuguet and Fabien Azavant. 1999. Building light-weight wrappers for legacy web data-sources using W4F. In *Vldb*, Vol. 99. 738–741.

[50] Sandeepkumar Satpal, Sahely Bhadra, Sundararajan Sellamanickam, Rajeev Rastogi, and Prithviraj Sen. 2011. Web information extraction using Markov logic networks. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011 (Companion Volume)*, Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar (Eds.). ACM, 115–116. https://doi.org/10.1145/1963192.1963251

[51] Sandeepkumar Satpal, Sahely Bhadra, Sundararajan Sellamanickam, Rajeev Rastogi, and Prithviraj Sen. 2011. Web information extraction using markov logic networks.. In *KDD*, Chid Apté, Joydeep Ghosh, and Padhraic Smyth (Eds.). ACM, 1406–1414. http://dblp.uni-trier.de/db/conf/kdd/kdd2011.html#SatpalBSRS11

[52] Vishal Sunder, Ashwin Srinivasan, Lovekesh Vig, Gautam M. Shroff, and Rohit Rahul. 2019. One-shot Information Extraction from Document Images using Neuro-Deductive Program Synthesis.. In *14th International Workshop On Neural-symbolic Learning And Reasoning*, Vol. abs/1906.02427. http://dblp.uni-trier.de/db/journals/corr/corr1906.html#abs-1906-02427

[53] Abhishek Udupa, Arun Raghavan, Jyotirmoy V. Deshmukh, Sela Mador-Haim, Milo M. K. Martin, and Rajeev Alur. 2013. TRANSIT: specifying protocols with concolic snippets. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, Hans-Juergen Boehm and Cormac Flanagan (Eds.). ACM, 287–296. https://doi.org/10.1145/2491956.2462174

[54] Wikipedia. [n.d.]. Precision and Recall. https://en.wikipedia.org/wiki/Precision_and_recall.

[55] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. LayoutLM: Pre-training of Text and Layout for Document Image Understanding.. In *26th ACM SIGKDD Conference On Knowledge Discovery And Data Mining*, Vol. abs/1912.13318. http://dblp.uni-trier.de/db/journals/corr/corr1912.html#abs-1912-13318

[56] Rahulkrishna Yandrapally, Suresh Thummalapenta, Saurabh Sinha, and Satish Chandra. 2014. Robust test automation using contextual clues. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis.* 304–314.

[57] Yanhong Zhai and Bing Liu. 2006. Structured Data Extraction from the Web Based on Partial Tree Alignment. *IEEE Trans. Knowl. Data Eng.* 18, 12 (2006), 1614–1628. https://doi.org/10.1109/TKDE.2006.197

[58] Mengshi Zhang, Daniel Perelman, Vu Le, and Sumit Gulwani. 2020. An Integrated Approach of Deep Learning and Symbolic Analysis for Digital PDF Table Extraction.. In *25th International Conference on Pattern Recognition (ICPR).*

[59] 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

[60] 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 *KDD*, Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams (Eds.). ACM, 2257–2266. http://dblp.uni-trier.de/db/conf/kdd/kdd2015.html#ZhangAYJS15

[61] Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. 2005. 2D Conditional Random Fields for Web information extraction. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005 (ACM International Conference Proceeding Series, Vol. 119)*, Luc De Raedt and Stefan Wrobel (Eds.). ACM, 1044–1051. https://doi.org/10.1145/1102351.1102483

[62] Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. 2006. Simultaneous record detection and attribute labeling in web data extraction. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos (Eds.). ACM, 494–503. https://doi.org/10.1145/1150402.1150457

[63] Jun Zhu, Zaiqing Nie, Bo Zhang, and Ji-Rong Wen. 2008. Dynamic Hierarchical Markov Random Fields for Integrated Web Data Extraction. *J. Mach. Learn. Res.* 9 (2008), 1583–1614. https://dl.acm.org/citation.cfm?id=1442784