OPAL: Automated Form Understanding for the Deep Web[*]

Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart

Department of Computer Science, Oxford University, Wolfson Building, Parks Road, Oxford OX1 3QD
firstname.lastname@cs.ox.ac.uk

ABSTRACT

Forms are our gates to the web. They enable us to access the deep content of web sites. Automatic form understanding unlocks this content for applications ranging from crawlers to meta-search engines and is essential for improving usability and accessibility of the web. Form understanding has received surprisingly little attention other than as component in specific applications such as crawlers. No comprehensive approach to form understanding exists and previous works disagree even in the definition of the problem.

In this paper, we present OPAL, the first comprehensive approach to form understanding. We identify form labeling and form interpretation as the two main tasks involved in form understanding. On both problems OPAL pushes the state of the art: For form labeling, it combines signals from the text, structure, and visual rendering of a web page, yielding robust characterisations of common design patterns. In extensive experiments on the ICQ and TEL-8 benchmarks and a set of 200 modern web forms OPAL outperforms previous approaches by a significant margin. For form interpretation, we introduce a template language to describe frequent form patterns. These two parts of OPAL combined yield form understanding with near perfect accuracy ($> 98\%$).

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: On-line Information Services—*Web-based services*

General Terms

Languages, Experimentation

Keywords

form understanding, web interfaces, deep web

1. INTRODUCTION

Are you looking for a house? Are you tired of filling registration forms with your search criteria on the websites of hundreds of local agencies? You fear to miss the site with the very best offer? Wouldn't you wish to automatize these tiresome tasks? Web forms

are the gates of all these websites. Gates designed for human admission, leaving programs in the conundrum of web design ambiguity: Even web forms within a single domain denote search criteria differently, e.g., "address", "city", "town", and "neighborhood" all refer to locations, while other terms denote different criteria ambiguously, e.g., "tenure" might refer to the choice either between "freehold" vs. "leasehold" or between "buy" vs. "rent". Moreover, web forms present their criteria in different manners, e.g., for a choice among several options, a form may contain either a drop-down lists or a set of radiobuttons. Automatically understanding these variants is key for programs to pass these gates as needed by a broad range of applications: crawling and surfacing the deep web [14, 10, 5], classifying the domain of web databases [2] for web site classification, sampling the contents of web databases [11, 1], and matching interfaces across domains [4, 15]. In web engineering, automated form understanding contributes e.g. to web accessibility and usability [7], web source integration [6], automated testing on form-related web applications.

Form understanding has attracted a number of approaches [16, 15, 6, 13, 8], for a recent survey see [9]. These approaches turn observations on common features of web forms (in general, across domains) into specifically tailored algorithms and heuristics, but generally suffer from three major limitations:

(1) Most approaches are *domain independent* and thus limited to observations that hold for forms across all domains. This limitation is acknowledged in [16, 13, 8], but addressed only through domain specific training data, if at all. Our evaluation supports [8] in that a set of generic design rules underlies all domains, but that specific domains parameterise or adapt these design patterns in ways uncommon to other domains.

(2) Most approaches are limited in the *classes of features* they use in their heuristics and often based on a single sophisticated heuristics based on one class of features, e.g., only visual features [6] or textual and field type features in [8].

(3) Heuristics are translated into monolithic algorithms limiting maintainability and adaptability. For example, [15] and [13] encode specific assumptions on the spatial distance and alignment of fields and labels, [8] employs hard-coded token classes for certain concepts such as "min", "from" vs. "max", "to".

To overcome these limitations, we present OPAL (*ontology based web pattern analysis with logic*), a domain-aware form understanding system that combines visual, textual, and structural features with a thin layer of domain knowledge. The visual, textual, and structural features are used in a domain-independent analysis to produce a highly accurate form labeling. However, for most applications what is actually needed is a form model consistent with a given domain schema, where all the fields are associated with given types. In OPAL, the domain schema is not only used to classify the

| | (a) web page | (b) form | (c) field labeling | (d) segment labeling | (e) classification |
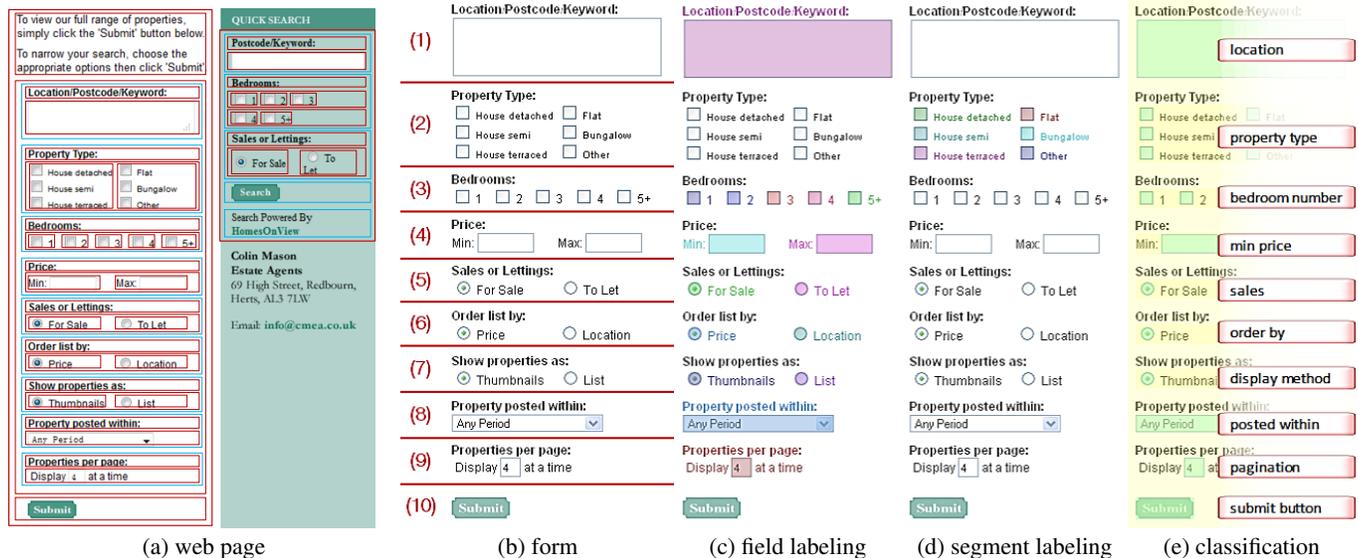
Figure 1: OPAL on *Colin Mason*

fields and segments of the form model, but also to improve the form model based on a set of structural constraints that describe typical fields and their arrangement in forms of the domain, e.g., how price ranges are presented in forms. To ease the development of these domain schemata, OPAL extends Datalog, a staple for declarative ontology and schema specification, with templates to enable reuse of generic form, e.g., how ranges (of any type) are presented in forms. With this approach, OPAL achieves nearly perfect analysis results ($> 98\%$ accuracy).

Contributions. OPAL's main contributions are:

(1) *Multi-scope domain-independent analysis* (Section 3) that combines structural, textual, and visual features to associate labels with fields into a form labeling using three, sequential "scopes" increasing the size of the neighbourhood from a subtree to every-thing visually to the left and top of a field. (i) At field scope, we exploit the structure of the page between fields and labels; (ii) at segment scope, observations on fields in groups of similar fields, and (iii) at layout scope, the relative position of fields and texts in the visual rendering of the page. We impose a strict preference on these scopes to disambiguate competing labelings and to reduce the number of fields considered in later scopes, as the complexity of the analysis increases from earlier to latter scopes (though overall bounded by $O(n^2)$ where n is the page size).

(2) *Domain awareness.* (Section 4) OPAL is domain-aware while being as domain-independent as possible without sacrificing ac-curacy. This is based on the observation that generic rules con-tribute significantly to form understanding, but nearly perfect accu-racy is only achievable through a thin layer of domain knowledge. To this end, we add an optional, domain-dependent classification and form model repair stage after the domain-independent analy-sis. Driven by a domain schema OPAL classifies form fields based on textual annotations of their labels and values assigned in the domain-independent form labeling, as well as the structure of that form labeling. This classification is often imperfect due to miss-ing or misunderstood labels. OPAL addresses this in a repair step, where structural constraints on the domain types, such as price, are used to disambiguate and complete the classification and reshape the form segmentation.

(3) *Template Language* OPAL-TL. (Section 4.1) To specify a do-main schema, we introduce OPAL-TL. It extends Datalog to express common patterns as parameterizable templates, e.g., describing a group consisting of a minimum and maximum field for some do-main type. Together with some convenience features for annota-tion queries and access to the field labeling, OPAL-TL allows for very compact, declarative specification of domain schemata. We also provide a template library of common phenomena, such that the adaption to new domains often requires only instantiating these templates with domain specific types. OPAL-TL preserves the poly-nomial data complexity of Datalog.

(4) *Extensive Evaluation.* (Section 5) In an evaluation on over 700 forms of four different datasets, we show that OPAL achieves highly accurate (>93%) form labelings for any domain and, with a suitable domain schema, near perfect accuracy in form classifica-tion ($> 98\%$). To compare with existing approaches (which only perform form labeling), we show that OPAL's domain-independent analysis achieves $94-100\%$ accuracy on the ICQ benchmark and $92-97\%$ on TEL-8. Thus, even without domain knowledge OPAL outperforms existing approaches by at least 5%.

At [17], we presented a preliminary version of OPAL, only demon-strating the relevance of the domain awareness, for a single domain with a limited ontology, not only lacking OPAL-TL but also missing the visual analysis, and the extensive evaluation we provide now.

1.1   Motivating Example

We present the OPAL approach to form understanding using the form from the UK real estate agency Colin Mason (`cmea.co.uk/properties.asp`). Figure 1a presents the web page with its sim-plified CSS box model. The page contains two forms: one for de-tailed search and the other for quick search. OPAL is able to iden-tify, separate, label, and classify both forms correctly yielding two real-estate form models. The following discussion focuses on the detailed search form (Figure 1b), in which each of the components (1)-(10), each of the fields (3)-(7) and the two groups of checkboxes in (2) are enclosed in a `table`, `tr`, or `td` element. Labels for each of the components such as "Bedrooms:" appear in separate `tr`'s.

Field scope. (Section 3.1) OPAL starts by analysing individual fields assigning labels that explicit reference the field (using the `for` attribute) or have a common ancestor that has no other fields as descendant. In our example, no explicit references occur, but the second approach correctly labels all fields except the checkboxes in (2). In Figure 1c we show this initial form labeling using same color for fields and their labels.

Segment scope. (Section 3.2) We increase the scope of the anal-

ysis from the elements to groups of similar elements, called *segments*. OPAL constructs these segments from the HTML structure, but eliminates segments that likely have no semantic relevance and are only introduced, e.g., for formatting reasons. This elimination is primarily based on similarity between elements approximated via semantic attributes such as `class` and visual similarity. In our example, components (2)-(7) become segments, with (2) further divided into two segments for each of the vertical checkbox groups.

In each of the segments, OPAL identifies repeated patterns of interleaving fields and texts. Here, each check box in (2) is labeled with the text appearing after it as shown in Figure 1d. OPAL also associates text nodes to segments to create segment labels. Segment labels can be useful to verify the form model and to classify fields that have no labels otherwise. In this example, OPAL assigns the text in bold face appearing atop each segment as the label, e.g., "Price:" becomes the label for (4).

Layout scope. (Section 3.3) In the layout scope, OPAL further enlarges the scope of the analysis to all fields visually to the left and above a field. The primary challenge in this scope is "overshadowing", i.e., if other fields appear to in the quadrants to the left and above a field. In this example the layout scope is not needed.

The result of the layout scope is the form labeling derived without using domain knowledge.

Domain scope. If a classification and semantic grouping of the form fields is desired, the final step in OPAL produces a *form model* that is consistent with a given domain schema. It uses domain knowledge to classify and verify the labeling and segmentation from the form labeling. In the classification step, OPAL annotates fields and segments with types based on annotation on the text labels. The verification step repairs and verifies the domain model if needed. For both steps, OPAL uses constraints specified in OPAL-TL. These constraints model typical patterns of forms in the domain. E.g., the first field in (4) is classified as MIN_PRICE as we recognise this segment as an instance of a price range pattern. These constraints also disambiguate between multiple annotations, e.g., fields in (6) are annotated with *order_by* and *price*, but the *price* annotation is disregarded due to the group label. Even without the group label, *price* would be disregarded as the domain schema give precedence to *order_by* over *price* due to the observation that if both occur as labels of a field, that field will likely be about the order of the returned results rather than about the actual price. In this case, only one repair is performed: We collapse the two checkbox segments in (2) as they are the only children of their parent segment and both of the same type. Figure 1e shows the final field classification as produced by OPAL.

2. THE OPAL APPROACH

OPAL constructs a conceptual model of a form consistent with a *domain schema*. A domain schema describes the form patterns occurring in a given domain, such as the UK real estate domain. OPAL divides the general form understanding problem into *form labeling* and *form interpretation*. The form labeling identifies forms and their fields, arranges the fields into a tree, and labels the found fields, segments, and forms with text nodes from the page. The form interpretation aligns a form labeling with the given domain schema and thereby classifies the form fields based on their labels.

2.1 Problem Definition

Form Labeling. A **web page** is a DOM tree $P = ((U)_{U \in \text{Unary}}, R_{\text{child}}, R_{\text{next-sibl}}, R_{\text{attribute}})$ where $(U)_{U \in \text{Unary}}$ are unary type and label relations, R_{child} is the parent-child, $R_{\text{next-sibl}}$ the direct next sibling, and $R_{\text{attribute}}$ the attribute relation. Further XPath relations (such as

descendant) are derived from these basic relations as usual [3]. U contains relations for types as in XPath (element, text, attribute, etc.) and two kinds of label relations, namely label^l for text nodes containing string l, and box^b for elements with bounding box b in the canonical rendering of the page. To normalize the representation of textual content, we represent the value of an attribute as text child node of the attribute (thus, label^l also applies to attributes).

DEFINITION 1. *A **form labeling** of a web page P is a tree F with mappings ϕ and ψ, such that ϕ maps the nodes of F into P. Leafs in F are mapped to form fields and inner nodes to* form segments*, that is an element grouping a set of fields. Each node n in F is also mapped to a set $\psi(n)$ of text nodes, the* labels *of n.*

A node can be labeled with no, one, or many labels. The form labeling contains a representative for each form. A representative contains all fields (and segments) of that form. This allows us to distinguish multiple forms on a single page, even if no form element is present or multiple forms occur in a single form element.

DEFINITION 2. *Given a DOM tree P, the **form labeling problem** (or schema-less form understanding problem) asks for a form labeling F where for each form f in P (i) there is a node $r \in F$ such that $\phi(r)$ is a suitable representative of f and (ii) for each field e in f, there exists a leaf node $n_e \in F$ such that n_e is a descendant of r and $\phi(n_e) = e$ where $\psi(n_e)$ is a suitable label set for e.*

We call a form labeling *complete* for a web page, if, for all e, $\psi(n_e)$ contains *all* text nodes suitable as labels for e and define the corresponding *complete form labeling* problem.

The suitability of a form representative $\phi(r)$ and a label set $\psi(n_e)$ cannot be defined formally, but needs to be evaluated by human annotators. Our evaluation (Section 5) shows that OPAL produces form labelings F_f that match the gold standard in nearly all cases ($> 95\%$ without using any domain knowledge).

Form Interpretation. To define the form interpretation problem, we formalize the notion of schema and introduce a form model as a form labeling extended with type information consistent with a given domain schema. First, we define an annotation schema that provides the necessary knowledge to interpret text nodes.

DEFINITION 3. *An **annotation schema** $\Lambda = (\mathcal{A}, \sqsubset, \prec, (\text{isLabel}_a, \text{isValue}_a : a \in \mathcal{A}))$ defines a set $\mathcal{A}$ of annotation types, a transitive, reflexive subclass relation $\sqsubset$, a transitive, irreflexive, antisymmetric precedence relation $\prec$, and two characteristic functions isLabel_a and isValue_a on text nodes for each $a \in \mathcal{A}$.*

For each annotation type $a \in \mathcal{A}$, we distinguish proper labels and values, with isLabel_a and isValue_a as corresponding characteristic functions. Proper labels are text nodes, such as *"Price:"*, describing the field type, values, such as *"more than £500"*, contain possible values of the field. Hence $\text{isLabel}_{price}(\text{"Price:"})$ and $\text{isValue}_{price}(\text{"more than £500"})$ hold.

The $\sqsubset$ relation holds for subtypes, e.g., *postcode $\sqsubset$ location*, and the $\prec$ relation defines precedence on annotation types used to disambiguate competing annotations. For example, an unlabeled select box with options *"Choose sorting order"*, *"By price"*, and *"By postcode"* is annotated with *order-by*, *price*, and *postcode*. If *order-by $\prec$ price* and *order-by $\prec$ postcode*, we pick *order-by*.

DEFINITION 4. *A **domain schema** $\Sigma = (\Lambda, \mathcal{T}, \mathcal{C}_{\mathcal{T}}, \mathcal{C}_{\Lambda})$ defines an annotation schema Λ, a set of domain types $\mathcal{T}$, and $\mathcal{C}_{\mathcal{T}}$ and $\mathcal{C}_{\Lambda}$ that map domain types to classification and structural constraints.*

For example, $\mathcal{C}_{\Lambda}(\text{PRICE})$ requires an annotation *price* and prohibits *order-by* annotations for a field to be typed as PRICE. The structural constraint set $\mathcal{C}_{\mathcal{T}}(\text{PRICE-RANGE})$ for a PRICE-RANGE segment requires a MIN-PRICE and MAX-PRICE field or a PRICE-RANGE field. We
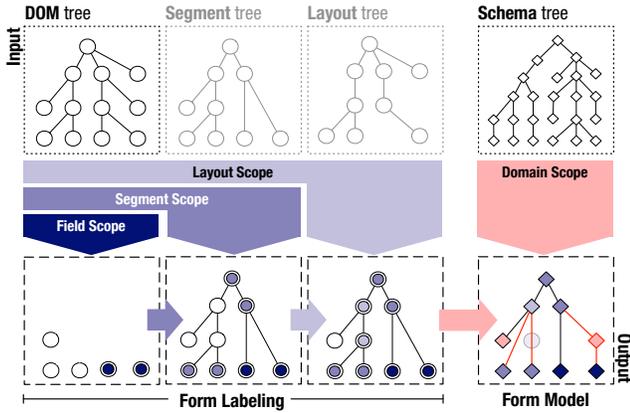
Figure 2: OPAL Overview

write $S \models C$, if a constraint set C is satisfied by a set S of annotation or domain types. The empty constraint set is always satisfied.

Formally, a *form interpretation* (F, τ) is a form labeling F with a partial type-of relation τ, relating nodes in F with the types $\mathcal{T}$ of Σ. Given a node n in F, we denote with $\mathcal{A}(n) = \{a \in \mathcal{A}_\Lambda : \exists l \in \psi(n)$ with $\mathsf{isValue}_a(l)$ or $\mathsf{isLabel}_a(l)\}$ the set of annotation types associated with n via its labels, and with $child\text{-}\mathcal{T}(n) = \bigcup_{(n,n') \in F} \tau(n')$ the set of domain types of the children of n.

DEFINITION 5. *A form interpretation* (F, τ) *is a* **form model** *for* Σ, *iff* $\mathcal{A}(n) \models \mathcal{C}_\Lambda(t)$ *and* $child\text{-}\mathcal{T}(n) \models \mathcal{C}_\mathcal{T}(t)$ *for all* $n \in F$, $t \in \tau(n)$.

DEFINITION 6. *Given a domain schema* Σ *and a form labeling* F, *the* **form interpretation problem** *asks for a form model* (F', τ) *for* Σ *such that* F' *differs from* F *only in inner nodes.*

Thus, form representatives, fields, and labels are shared between F and F', but the form segments may be rearranged to conform with form patterns prescribed by the structural constraints of Σ.

Form Understanding

DEFINITION 7. *Given a domain schema* Σ *and a DOM tree* P, *the* **form understanding (or schema-based form understanding) problem** *asks for a form model* (F, τ) *of* P *under* Σ, *such that* F *is a solution of the complete form labeling problem for* P *and for each form field* e *in* P, *there is a leaf node* n_e *in* F *with* $\phi(n_e) = e$ *and* $\tau(n_e)$ *is a suitable concept from* Σ *for* e.

2.2 System Overview

OPAL is divided in two parts, a domain-independent part to address the form labeling problem and a domain-dependent part for form interpretation according to a domain schema.

OPAL produces form labelings in a novel multi-scope approach that incrementally constructs a form labeling combining textual, structural, and visual signals (Figure 2). Each of the three labeling scopes considers signals not considered in prior scopes:

(1) In *field scope*, we consider only fields and their immediate neighbourhood and thus use only the DOM tree as input.

(2) In *segment scope*, we detect and arrange form segments into a segment tree to interleave the contained text nodes and fields.

(3) In *layout scope*, we broaden the potential labels of a field by searching in the layout tree, i.e., the visual rendering of the page, and assign text nodes to fields, given a strong visual relation.

Each scope builds on the partial form labeling of the previous scope and uses the information from the additional input to find labels for previously unlabeled fields (or segments). Only the segment scope adds nodes, namely form segments, whereas field and layout scope only add labels.

Algorithm 1: FieldScopeLabelling(*DOM P*)

1   **foreach** *field f in P* **do**
2    $n \leftarrow f$;
3   **while** *n has a parent* **do**
4   **if** *n is already coloured* **then** colour *n* red; **break**;
5   colour *n* orange;
6     $n \leftarrow$ parent of *n*;

7   $F \leftarrow$ empty form labeling ;
8   **foreach** *field f in P* **do**
9    $n \leftarrow$ new leaf node in F;
10   $\phi(n) \leftarrow f$;
11   **if** $\exists l \in P$ *with* for *attribute referencing f* **then**
12   assign all text node descendants of *l* as labels to *n* ;
13   $p \leftarrow$ parent of *f*;
14   **while** *p not coloured* red **do**
15    $f \leftarrow p$; $p \leftarrow$ parent of *f*;
16   assign all text node descendants of *f* as labels to *n* ;

Finally, in the **(4)** *domain scope* (Section 4) we turn the form labeling produced by the first three scopes into a form model consistent with a given domain schema. (i) The labeling model is extended with (domain-specific) annotations on the textual content of proper labels and values. (ii) Fields and segments of the form labeling are classified according to classification constraints in the domain schema. (iii) Finally, violations of structural schema constraints are repaired in a top-down fashion.

Types and constraints of the domain schema are specified using OPAL-TL, an extension of Datalog that combines easy querying of the form labeling and of annotations with a rich template system. Datalog rules already ease the reuse of common types and their constraints, but the template extension enables the formulation of generic templates for such types and constraints that are instantiated for concrete types of a domain. An example of a type template is the range template, that describes typical patterns for specifying range values in forms. In the real estate domain it is instantiated, e.g., for price and various room ranges. In the used car domain, we also find ranges for engine size, mileage, tax band, etc. Thus, creating a domain schema is in many cases as easy as importing common types and instantiating templates.

3. FORM LABELING

In OPAL, form labeling is split into three scopes. Each scope is focused on a particular class of input features (e.g., visual, structural, textual). By combining form labeling approaches for these different features, OPAL captures the diverse range of form design patterns and eases extensions, such as the introduction of new scopes that future web design trends might require. This contrasts with previous approaches that rely on one or two such feature classes.

The form labeling scopes, *field*, *segment*, and *layout* scope, use **domain-independent** labeling techniques to associate form fields or segments with textual labels, building a form labeling F. If a domain schema is available, the form labeling is extended to a form model in the domain-dependent analysis (Section 4).

The form labeling F is constructed bottom-up, applying each scope's technique in sequence to yet unlabelled fields. Whenever a field is labelled at a certain scope level, further scopes do not consider this field again. This application order reflects higher confidence in earlier scopes and addresses competing label assignments.

3.1 Field Scope

Based on the DOM tree of the input page, the **field scope** assigns text nodes in unique structural relation to individual fields as labels to these fields (see Algorithm 1). To that end, OPAL **(1)** colours

---

Algorithm 2: SegmentTree(*DOM P*), $\perp \nsim n$ for any *n*

1 $P' \leftarrow P$;
2 **while** $\exists n \in P' : n$ not a field $\wedge \left(\nexists d : R_{descendant}(d,n) \in P' \wedge d \text{ a field} \right)$ **do**
3 　 delete *n* and all incident edges from P';
4 **while** $\exists n \in P' : |\{c \in P' : R_{child}(c,n) \in P'\}| = 1$ **do**
5 　 delete *n* from P' and move its child to the parent of *n*;
6 **foreach** *inner node n in P' in bottom-up order* **do**
7 　 $C \leftarrow \{f : R_{child}(f,n) \in P' \wedge f \text{ is a field}\}$;
8 　 $C \leftarrow C \cup \{\text{Representative}(n') : R_{child}(n',n) \in P'\}\}$;
9 　 choose $r \in C$ arbitrarily ;
10 　 **if** $\forall r' \in C : r \sim r'$ **then**
11 　　 Representative$(n) \leftarrow r$;
12 　　 delete all non-field children of *n* and move their children to *n*;
13 　 **else** Representative$(n) \leftarrow \perp$;
14 **return** P';

(lines 1–6) all nodes in *P* that are ancestors of a field and do not have other form fields as descendants in orange. The least ancestor that violates that condition is coloured red. **(2)** It identifies (line 7–10) all form fields and initialises the form labeling *F* with one leaf node for each such field. **(3)** It considers (lines 11–12) explicit HTML label elements with *direct reference* to a form field. **(4)** It labels (lines 13–16) each field *f* with all text nodes *t* whose *least common ancestor* with *f* has no other form field as descendant. This includes all text nodes in the content of *f*. We find these text nodes in linear time with the tree colouring. Each value *v* of a field *f* (in select, input, or textarea element) becomes a label for *f*, as the least common ancestor of *f* and *v* is *f*.

3.2 Segment Scope

At **segment scope**, the labeling analysis expands from individual fields to form segments, i.e., groups of consecutive fields with a common parent. These segments are then used to distribute text nodes to unlabeled fields in that segment.

Segmentation tree. We observe that the DOM is often a fair approximation of the semantic form structure, as it reflects the way the form author grouped fields into segments. Therefore, we start from the DOM structure to find the form segments, but we eliminate all nodes that can be safely identified as superflous: nodes without field descendants, nodes with only one child, and nodes *n* where all fields in *n* are style-equivalent to the fields in the siblings of *n*. Two fields are **style-equivalent** ($\sim$) if they carry the same class attribute (used to indicate a formatting or semantic class) or the same type attribute and CSS style information.

If all field descendants of the parent of an inner node *n* are style-equivalent, then *n* should be eliminated from the segment tree, as it artificially breaks up the sequence of style-equivalent fields and is thus referred to as *equivalence breaking*.

DEFINITION 8. *The **segment tree** P' of a form page P is the maximal DOM tree included in P (i.e., obtained by collapsing nodes) such that the leafs of P' are all fields and for all its inner nodes n*
(1) $\left| \{c \in P' : R_{child}(c,n)\} \right| > 1$,
(2) $\exists d \in P' : R_{descendant}(d,n) \wedge d \text{ is a field, and}$
(3) n is not equivalence breaking.

As an example, consider the DOM tree on the left of Figure 3, where diamonds represent fields and style-equivalent fields carry the same colour. On the right hand side, we show OPAL's segment tree for that DOM. Nodes 1 and 3 from the original DOM are eliminated as they have only one child, and node 2 as it is equivalence breaking. Nodes 4 and 5 are retained due to the red field.

THEOREM 1. *The **segment tree** P' of a DOM tree P can be computed in $O(n \times d)$ where n is the size and d the depth of P.*
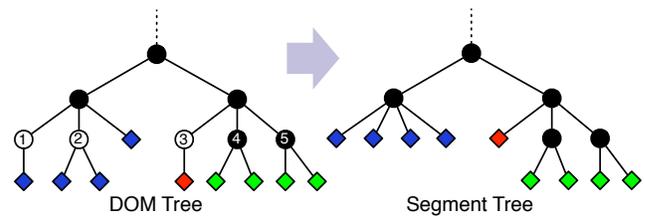
Figure 3: Example DOM and Segment Tree

Algorithm 3: SegmentScopeLabeling(*DOM P*, *Form Labeling F*)

1 $S \leftarrow$ SegmentTree(P) ;
2 **foreach** *inner node s in S in bottom-up order* **do**
3 　 create a new segment n_s in *F*;
4 　 $\phi(n_s) \leftarrow s$;
5 　 create an edge (n_s, c_s) in *F* for every $\phi(c_s)$ child of *s*;
6 **foreach** *segment n in F* **do**
7 　 Nodes, Labels $\leftarrow$ **new** *List*();
8 　 textGrp $\leftarrow \emptyset$;
9 　 **foreach** $c : R_{descendant}(c, \phi(n)) \in P$ in document order **do**
10 　　 **if** $\exists f \in F : \phi(f) = c$ **then**
11 　　　 **if** textGrp $\neq \emptyset$ **then** Labels.add(textGrp); textGrp $\leftarrow \emptyset$;
12 　　　 Nodes.add(c);
13 　　　 skip all descendants of *c* in the iteration ;
14 　　 **else if** *c is a text node* $\wedge \nexists d \in F : c \in \psi(d)$ **then**
15 　　　 textGrp $\leftarrow$ textGrp $\cup \{c\}$;
16 　 **if** textGrp $\neq \emptyset$ **then** Labels.add(textGrp); textGrp $\leftarrow \emptyset$;
17 　 **if** Labels.size() $=$ Nodes.size() $+ 1$ **then**
18 　　 add Labels[0] to $\psi(n)$;
19 　　 delete Labels[0] from Labels;
20 　 **if** Labels.size() $=$ Nodes.size() **then**
21 　　 **foreach** *i* **do** add Labels[i] to ψ(Nodes[i]);

PROOF. Algorithm 2 computes the segment tree P' for any DOM tree *P*. Its leafs are fields (as any non field leafs are eliminated in line 2–3) and any inner node must have more than 1 child (due to line 4–5), a field descendant (due to line 2–3), and not be equivalence breaking (due to lines 6–13). In lines 6–13, we compute a Representative for each inner node in a bottom-up fashion: If all field children (line 7) and the representatives of all inner children (line 8) are style-equivalent (line 9–10; since $\sim$ is an equivalence relation it suffices to compare a representative to each of the elements in *C*), we choose an arbitrary representative and collapse all inner children of that node. Otherwise, we assign $\perp$ as representative, which is not style-equivalent to any node or to itself. Thus it prevents this node (and its ancestors) from ever being collapsed. This can not introduce new violations to condition (1) and (2), as we never decrease the number of children, turn a leaf into a inner node, or remove fields. P' is maximal: Any tree P'' that includes P' but is included in *P* must contain at least one node from *P* that has been deleted by one of the above conditions. Such a node, however, violates at least one of the conditions for a segment tree and thus P'' is not a segment tree. This holds because the order of the node deletions does not affect the nodes deleted. Algorithm 2 runs in $O(n \times d)$: Lines 2–3 are in $O(n)$. Lines 4–5 and lines 6–13 are both in $O(n \times d)$ as they are dominated by the collapsing of the nodes. In the worst case, we collapse $d - 2$ inner nodes and thereby move $O(n)$ leafs $d - 2$ times. □

Segment Labeling. We extend the existing form labeling *F* of the field scope with form segments according to the structure of the segment tree and distribute labels in regular groups, see Algorithm 3. First (lines 2–5), we create a form segment node *s* in the form labeling for each inner node n_s in the segment tree and choose
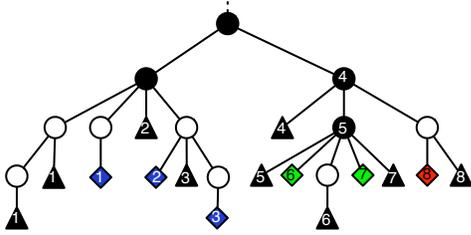
Figure 4: Example for Segment Scope Labeling

n_s as representative for s ($\phi(s) = n_s$). For each segment with regular interleaving of text nodes and field or segment nodes, we use those text nodes as labels for these nodes, preserving any already assigned labels and fields (from field scope). In detail, we iterate over all descendants c of each segment in document order, skipping any nodes that are descendants of another segment or field itself contained in n (line 13). In the iteration, we collect all field or segment nodes in Nodes, and all sets of text nodes between field or segment nodes in Labels, except those text nodes already assigned as labels in field scope (line 14), as we assume that these are outliers in the regular structure of the segment. We assign the i-th text node group to the i-th field, if the two lists have the same size (possibly using the first text node as labels of the segment, line 17–19).

Figure 4 illustrates the segment scope labeling with triangles denoting text nodes, diamonds fields, black circles segments, and white circles DOM nodes not in the segment tree. The numbers indicate which text nodes are assigned as labels to which segments or fields. E.g., for the left hand segment, we observe a regular structure of (text node+, field)+ and thus we assign the i-th group of text nodes to the i-th field. For the right hand segment (4), we find a subsegment (5) and field 8 that is already labeled with text node 8 in the field scope. Thus 8 is ignored and only one text node remains directly in 4, which becomes the segment label. In 5, we find one more text node group than fields and thus consider the first text node group as a segment label. The remaining nodes have a regular structure (field, text node+)+ and get assigned accordingly.

3.3 Layout Scope

At **layout scope**, we further refine the form labeling for each form field not yet labelled in field or segment scope, by exploring the visible text nodes in the west, north-west, or north quadrant, if they are not overshadowed by any other field. To this end, OPAL constructs a layout tree from the CSS box labels of the DOM nodes:

DEFINITION 9. *The **layout tree** of a given DOM P is a tuple $(N_P, \lhd, w, nw, n, ne, e, se, s, sw, aligned)$ where N_P is the set of DOM nodes from P, $\lhd, w, nw, n, \ldots$ the "belongs to" (containment), west, north-west, north, $\ldots$ relations from RCR [12], and aligned(x, y) holds if x and y have the same height and are horizontally aligned.*

We call $w, nw, \ldots$ the neighbour relations. The layout tree is at most quadratic in size of a given DOM P and can be computed in $O(|P|^2)$. For convenience, we write, e.g., w-nw-n to denote the union of the relations w, nw, and n.

In cultures with left-to-right reading direction, we observe a strong preference for placing labels in the w-nw-n region from a field. However, forms often have many fields interspersed with field labels and segment labels. Thus we have to carefully consider overshadowing. Intuitively, for a field f, a visible text node t is overshadowed by another field f' if t is above f' or also visible from, but closer to f'. In the particular case of aligned fields, the former would prevent any labeling for these fields and thus we relax the condition.

DEFINITION 10. *Given a text node t, a field f' **overshadows** another field f if **(1)** f and f' are unaligned, w-nw-n(f', f), and*

w-nw-n-ne-e(t, f') *or* **(2)** f *and* f' *are aligned and (i)* w(t, f') *or (ii)* nw-n(t, f') *and there is a text node* t' *not overshadowed by another field with* ne-e(t', f') *and* w-nw-n(t', f).
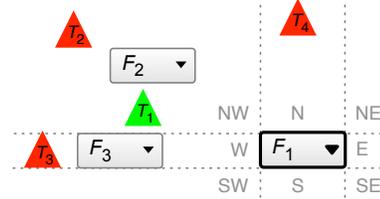
Figure 5: Layout Scope Labeling

To illustrate this overshadowing, consider the example in Figure 5. For field F_1, T_2 and T_4 are overshadowed by F_2 and T_3 by F_3, only T_1 is not overshadowed, as there is no other text node that is south-east or south from T_3 not overshadowed by another field.

The layout scope labeling is then produced as follows: For each field f, we collect all text nodes t with w-nw-n(t, f) and add them as labels to f if they are not overshadowed by another field and not contained in a segment that is no ancestor of f. The latter prevents assignment of labels from unrelated form segments.

4. FORM INTERPRETATION

There is no straightforward relationship between form fields for domain concepts, such as location or price, and their structure within a form. Even seemingly domain-independent concepts, such as price, often exhibit domain specific peculiarities, such as "guide price", "current offers in excess", or payment periods in real estate. OPAL's domain schemata allow us to cover these specifics. We recall from Section 2 that a form model (F', τ) for a schema Σ is derived from a form labeling F by extending F with types and restructuring its inner nodes to fit the structural constraints of Σ.

OPAL performs form interpretation of a form labeling F in two steps: **(1)** the *classification* of nodes in F according to the domain types $\mathcal{T}$ to obtain a partial typing τ_P. This step relies on the annotation schema Λ and its typing of labels in F; **(2)** the *model repair* where the segmentation structure derived in the segmentation scope (Section 3.2) is aligned with the structure constraints of Σ.

4.1 Schema Design: OPAL-TL

OPAL provides a **t**emplate **l**anguage, OPAL-TL, for easily specifying domain schemata reusing common concepts and their constraints as well as concept templates. To implement a new domain, we only need to provide **(1)** a set of annotators implementing isLabel$_a$ and isValue$_a$ and **(2)** an OPAL-TL specification of the domain types and their classification and structural constraints.

OPAL-TL extends Datalog with templates and predefined predicates for convenient querying of annotations and DOM nodes. An OPAL-TL program is executed against a form labeling F and a DOM P. Relations from F and P are mapped in the obvious way to OPAL-TL. We only use child (descendant, resp.) for the child (descendant, resp.) relation in F. We extend document and sibling order from P to F: follows(X, Y) for $X, Y \in F$, if $R_{\text{following}}(\phi(X), \phi(Y)) \in P$ and no other node in F occurs between X and Y in document order; adjacent(X, Y), if $R_{\text{next-sibling}}(\phi(X), \phi(Y)) \in P$ or vice versa. Finally, we abbreviate label$^l(\phi(X))$ as $l(X)$.

Annotation types and their queries. Annotations (instances of annotation types) are characterised by an external specification of the characteristic functions isLabel$_a$ and isValue$_a$ for each $a \in \mathcal{A}$. In the current version of OPAL, these functions are implemented with simple GATE (gate.ac.uk) gazetteers and transducers, that

Figure 6: Example Form Labeling

are either provided by human domain experts or derived from external sources such as DBPedia and Freebase. The current OPAL version contains a large set of such artefacts for common domain types such as price, location, or date.

DEFINITION 11. *Given a form labeling F on a DOM P and an annotation schema* Λ, *an* **OPAL-TL** *annotation query is an expression of the form:* $X@A\{d,p,e\}$ *where X is a first-order variable,* $A \in \mathcal{A}$, *and d, p, and e are annotation* modifiers. *An annotation query* $X@A\mu$ *with* $\mu \subseteq \{d,p,e\}$ *holds for all* $X \in [\![A\mu]\!]$ *with*

$$[\![@A\mu]\!] = \{n \in P : Allow_\mu(n) \cap Match_\mu(A) \neq \emptyset\} \setminus Block_\mu(A)$$

with $Allow_\mu(n)$ *set to* $\psi(n)$ *for* $d \in \mu$, *and* $\psi(n) \cup \psi(parent\ of\ n)$ *otherwise.* $Match_\mu(A)$ *is to* $\{l : \bigcup_{A' \sqsubseteq^* A} isLabel_{A'}(l)\}$ *for* $p \in \mu$, *and* $\{l : \bigcup_{A' \sqsubseteq^* A}(isLabel_{A'}(l) \vee isValue_{A'}(l))\}$ *otherwise.* $Block_\mu(A)$ *equals* $\{n : \exists A' \prec A, |Match_\mu(A)| < |Match_\mu(A')|\}$ *if* $e \in \mu$, *and* $\emptyset$ *otherwise.*

Intuitively, an annotation query $X@A$ returns all nodes labeled with a label that is annotated with A. If the modifier d (direct) is *not* present, we also consider the (direct) segment parents, otherwise only *direct* labels are considered. If the modifier p (proper) is present, only $isLabel_A$ is used, otherwise also $isValue_A$. If the modifier e (exclusive) is present, a node that fullfils all other conditions is still not returned, if there are more labels with annotations of a type that has precedence over A.

Consider the form labeling of Figure 6 under a schema with $C \sqsubseteq B$ and $B \prec A$. Labels are denoted with triangles, fields with diamonds, segments with circles. Labels are further annotated with matching annotation types (here always only one). If value labels are drawn as outlines. Then, $X@A\{\}$ matches $2,3,4$; $X@A\{e,d\}$ matches $2,4$, but not 3 as 3 has more labels of B (or one of its subclasses) than of A and the exclusive modifier e is present; $X@A\{e,p\}$ matches $2,3$, but not 4 as the proper modifier p prevents the value labels in white to be considered. The latter matches 3 despite the presence of e, as we consider also the labels of the parent of 3 (since the direct modifier d is absent) and thus there are two A labels.

OPAL-TL templates. OPAL-TL extends Datalog¬ (Datalog with stratified negation) by templates to define reusable patterns for domain concepts. Examples of such patterns are basic classification patterns that derive a domain type from a conjunction of annotation types or min-max range patterns where we look for multiple fields with related annotations in a group and some clue that they represent a range. In general, there are two types of template patterns, one for classification constraints, one for structural constraints. The former specify patterns for relationships between domain and annotation types, the latter the abstract structure of domain concepts.

DEFINITION 12. *An* **OPAL-TL** *template is an expression* **TEMPLATE** $N<D_1,\ldots,D_k> \{\ p \Leftarrow expr\ \}$ *where N names the template,* $D_1,\ldots,D_k$ *are template parameters, p is a template atom, expr a conjunction of template atoms and annotation queries. A template atom* $p<C_1,\ldots,C_k>(X_1,\ldots,X_n)$ *consists of first-order predicate name p, template variables* $C_1,\ldots,C_k$, *and first-order variables* $X_1,\ldots,X_n$.

Multiple rules with the same head express union as usual. For convenience, we use $\vee$ and $\neg$ over conjunctions, which are translated to pure Datalog¬ rules as usual (not effecting data complexity).

```
TEMPLATE basic_concept<C,A> { concept<C>(N)⇐N@A{d,e,p} }

TEMPLATE concept_by_segment<C,A> { concept<C>(N)⇐N@A{e,p} }

TEMPLATE concept_minmax<C,CM,A> {
  concept<CM>(N1)⇐child(N1,G),child(N2,G),adjacent(N1,N2),
    N1@A{e,d},(concept<C>(N2) ∨ N2@A{e,d})
  concept<CM>(N2)⇐child(N1,G),child(N2,G),follows(N2,N1),
    concept<C>(N1),N2@range_connector{e,d},¬(A1 ≺ A, N2@A1{d})
  concept<CM>(N1)⇐child(N1,G),child(N2,G),adjacent(N1,N2),
    N1@A{e,p},N2@A{e,p},((N1@min{e,p},N2@max{e,p})
      ∨ (N1@max{e,p},N2@min{e,p}))
```

Figure 7: OPAL-TL classification templates

As an example, the following template defines a family of constraints that associate the domain type D to a node N whenever N is labeled by an exclusive direct and proper annotation of type A.

```
TEMPLATE basic_concept<D,A> { concept<D>(N) ⇐ N@A{e,d,l}  }
```

A template *tpl* is *instantiated* to produce a family of rules where the formal template variables $D_1,\ldots,D_k$ are instantiated using values $v_1^i,\ldots,v_k^i$ from a *template instantiation* expression of the form

```
INSTANTIATE tpl<D1,…,Dk> using { <v11,…,vk1> … <v1n,…,vkn> }
```

For example, the following expression instantiates `basic_concept` replacing D with type RADIUS and A with annotation type *radius*

```
INSTANTIATE basic_concept<D,A> using {<RADIUS, radius>}
```

and produces the following instantiated rule:

```
concept<RADIUS>(N)⇐N@radius{e,d,l}
```

PROP. 1. OPAL-TL *has the same data complexity as Datalog¬.*

4.2 Classification

Classification is based on the classification constraints of the domain schema. In OPAL these constraints are specified using OPAL-TL to enable reuse of domain concepts and concept patterns. In the real estate and used car domains, we identify three patterns that suffice to describe nearly all classification constraints. These patterns effectively capture very common semantic entities in forms and are parametrized using domain knowledge. The building blocks are a domain type (or concept) C and an annotation type A that is used to define a classification constraint for C. None of these patterns uses more than one annotation type as template parameter, though many query additional (but fixed) annotation types in their bodies.

Figure 7 shows the classification templates for real-estate and used car: (**1**) *Basic concept.* The first template captures direct classification of a node N with type C, if N matches X@A{d,e,p}, i.e., has more proper labels of type A than of any other type A' with $A' \prec A$. This template is used by far most frequently, primarily for concepts with unambiguous proper labels. (**2**) *Concept by segment.* The second template relaxes the requirement by considering also indirect labels (i.e., labels of the parent segment). In the real estate and used car domains, this template is instantiated primarily for control fields such as ORDER_BY or DISPLAY_METHOD (grid, list, map) where the possible values of the field are often misleading (e.g., an ORDER_BY field may contain *"price"*, *"location"*, etc. as values). (**3**) *Min-max concept.* Web forms often show pairs of fields representing min-max values for a feature (e.g., the number of bedrooms of a property). We specify this pattern with three simple rules (line 5–12), that describe three configurations of segments with fields associated with value labels only (proper labels are captured by the

```
   TEMPLATE segment<C>{
2    segment<C>(G)⇐outlier<C>(G),child(N₁,G),¬(child(N₂,G),
       ¬(concept<C>(N₂) ∨ segment<C>(N₂))) }
4
   TEMPLATE segment_range<C,C_M> {
6    segment<C>(G)⇐outlier<C>(G),concept<C_M>(N₁),concept<C_M>(N₂),
       N₁ ≠ N₂,child(N₁,G),child(N₂,G) }
8
   TEMPLATE segment_with_unique<C,U> {
10   segment<C>(G)⇐outlier<C>(G),child(N₁,G), concept<U>(N₁,G),
       ¬(child(N₂,G),N₁ ≠ N₂,¬(concept<C>(N₂)∨segment<C>(N₂))). }
12
   TEMPLATE outlier<C>{
14   outlier<C>(G)⇐root(G)∨child(G,P),child(G′,P),¬(segment<C>(G′)) }
```

Figure 8: OPAL-TL structural constraints

first two templates). It is the only template with two concept template parameters, C and C_M where $C_M \sqsubseteq C$ is the "minmax" variant of C. The first locates, adjacent pairs of such nodes or a single such node and one that is already classified as C. The second rule locates nodes where the second follows directly the first (already classified with C), has a *range_connector* (e.g., "from" or "to"), and is not annotated with an annotation type with precedence over A. The last rule also locates adjacent pairs of such nodes and classifies them with C_M if they carry a combination of *min* and *max* annotations.

In addition to these templates, there is also a small number of specific patterns. In the real estate domain, e.g., we use the following rule to describe forms that use a links for submission (rather than submit buttons). Identifying such a link (without probing and analysis of Javascript event handlers) is performed based on an annotation type for typical content, `title` (i.e., tooltip), or `alt` attribute of contained images. This is mostly, but not entirely domain independent (e.g., in real estate a "rent" link).

```
concept<LINK_BUTTON>(N₁)⇐form(F),descendant(N₁,F),link(N₁),
  N₁@LINK_BUTTON{d},¬(descendant(N₂,F),
    (concept<BUTTON>(N₂) ∨ follows(N₁,N₂)))
```

4.3 Model Repair

With fields and segments classified, OPAL verifies and repairs the structure of the form according to structural constraints on the segments, such that it fits to the patterns prescribed by the domain schema. As for classification constraints, we use OPAL-TL to specify the structural constraints. The actual verification and repair is also implemented in OPAL-TL, but since it is not domain independent, it is not exposed to the user for modification. Here, we first introduce typical structural constraints and their templates and then outline the model repair algorithm, but omit the OPAL-TL rules.

Structural constraints. The structural constraints and templates in the real estate and used car domains are shown in Figure 8 (omitting only the instantiation as in the classification case). All segment templates require that there is an outlier among the siblings of the segment: `outlier<C>(G)` holds if at least one of G's siblings is not a C segment. **(1)** *Basic segment.* A segment is a C segment, if its children are only other segments or concepts typed with C. This is the dominant segmentation rules, used, e.g., for ROOM, PRICE, or PROPERTY_TYPE in the real estate domain. **(2)** *Minmax segment.* A segment is a C segment, if it has at least two field children typed with C_M where $C_M \sqsubseteq C$ is the minmax type for C. This is used, e.g., for PRICE and BEDROOM range segments. **(3)** *Segment with mandatory unique.* A segment is a C segment, if its children are only segments or concepts typed with C except for one (mandatory) field child typed with U where $U \not\sqsubseteq C$. This is used for GEOGRAPHY segments where only one RADIUS may occur.

Repairing form interpretations. The classification yields a form interpretation F, that is, however, not necessarily a model under Σ, and may contain violations of structural constraints. We adapt the types of fields and segments and the segment hierarchy of F with the rewriting rules described below to construct a form model compliant with Σ. OPAL performs the rewriting in a stratified manner to guarantee termination and introduces at most n new segments where n is the number of fields in the form.

(1) *Under Segmentation:* If there is a segment n with type t such that $\mathcal{C}_{\mathcal{T}}(t)$ requires additional child segments of type $t_1,\ldots,t_k \notin child\text{-}\mathcal{T}(n)$, we try to partition the children of n into $k+1$ partitions $P_1,\ldots,P_k,P_n$ such that $P_i \models \mathcal{C}_{\mathcal{T}}(t_i)$ and $P_n \cup \{t_1,\ldots,t_k\} \models \mathcal{C}_{\mathcal{T}}(t)$. For each P_i we add a new segment node as child of n, classify it with t_i, and move all nodes assigned to P_i from n to that segment. In practice, few cases of multiple under segmentations occur at the same node and we can limit the search space using a total order on $\mathcal{T}$. Though in general this would require value invention, the number of segments is actually bounded by the number of fields in the form, which is typically between 2–10. Therefore, we provide a pool of unused segments in the segmentation.

(2) *Over Segmentation:* If there is a segment n of type t with children $c_1,\ldots,c_k$ such that $\bigcup child\text{-}\mathcal{T}(c_i) \cup \bigcup_{n'\in C} \tau(n') \models \mathcal{C}_{\mathcal{T}}(t)$ where C is the set of children of n without $c_1 \ldots c_k$, then we move the children of each c_i to n and delete all c_i.

(3) *Under Classification:* If there is a segment n of type t with untyped children $c_1,\ldots,c_k$ and corresponding types $t_1,\ldots,t_k$ such that $child\text{-}\mathcal{T}(n) \cup \{t_1,\ldots,t_k\} \models \mathcal{C}_{\mathcal{T}}(t)$ and, for each c_i, $child\text{-}\mathcal{T}(c_i) \models \mathcal{C}_{\mathcal{T}}(t_i)$ holds, then we type c_i with t_i.

(4) *Over Classification:* If there is a segment node n of type t with child c typed t_1 and t_2 such that $\{t_1\} \cup \bigcup_{c'\in C} \tau(c') \models \mathcal{C}_{\mathcal{T}}(t)$ where C is the set of children of n without c, we drop t_2 from $\tau(c)$.

(5) *Miss Classification:* If there is a node n of type t where $child\text{-}\mathcal{T}(n) \not\models \mathcal{C}_{\mathcal{T}}(t)$, then we delete the classification of n as t.

5. EVALUATION

We perform experiments on several domains across four different datasets. Two datasets are randomly sampled from the UK real estate and UK used-car domains, respectively. We compare with existing approaches via ICQ and TEL-8, two public benchmark sets, on which we only evaluate OPAL's form labeling for fair comparison to existing approaches, as they only label forms and do not use domain knowledge. Even with these limitations, OPAL outperforms these approaches in most domains by at least 5%. We also perform an introspective analysis of OPAL to show **(1)** the impact of field, segment, layout, and domain scope and **(2)** OPAL's performance and scalability with increasing page size.

For the evaluation, we evaluate the proper assignment of text nodes to form fields using precision, recall and F_1-score (harmonic mean $F_1 = 2PR/(P + R)$ of precision and recall). Precision P is defined as the proportion of correctly labeled fields over total labeled fields, while recall R is the fraction of correctly labeled fields over total number of fields. For all considered datasets, we compare the extracted result to a manually constructed gold standard. We evaluate segmentation through their impact on classification, see Figure 10b and the improved performance on the two datasets where we perform form interpretation (UK real estate and used car) versus the ICQ and TEL-8 datasets.

Datasets. For UK real estate domain, we build a dataset randomly selecting 100 real estate agents from the UK yellow pages (`yell.com`). Similarly, we randomly pick 100 used-car dealers from the UK largest aggregator website `autotrader.co.uk`. The forms in these two domains have significantly different characteris-

(a) OPAL on 734 forms

(b) ICQ results

(c) TEL-8 results
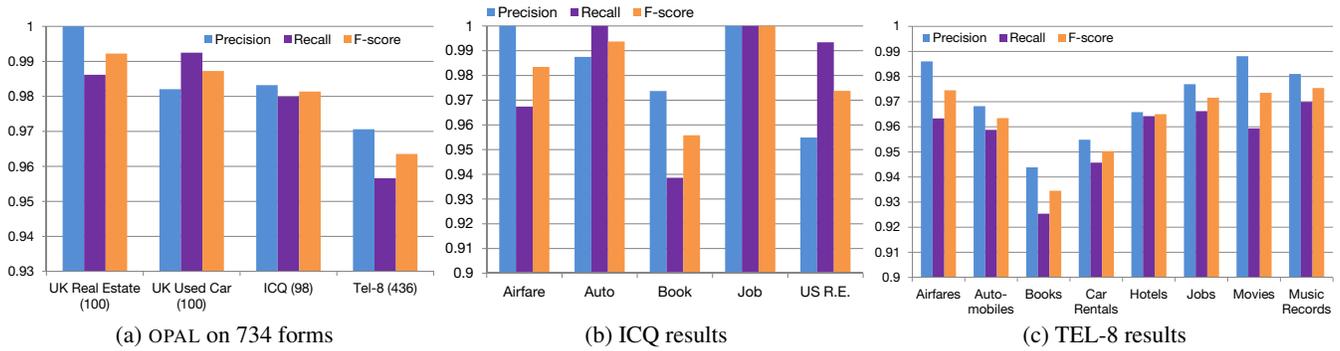
Figure 9: OPAL comparison

(a) Time

(b) Scopes

Figure 10: OPAL evaluation

tics than the ones in ICQ and TEL-8, mainly due to changes in web technology and web design practices. The usage of CSS stylesheets for layout and AJAX features are among the most relevant.

The ICQ and TEL-8 datasets cover several domains. ICQ presents forms from five domains: air traveling, car dealer, book, job, real estate. There are 20 web pages for each of the domains, but two of them are no longer accessible and thus excluded from this evaluation. TEL-8, on the other hand, contains forms of eight domains: books, car rental, jobs, hotels, airlines, auto, movies and music records. The dataset amounts to 477 forms, but only 436 of them are accessible (even in the cached version).

Field Labeling Accuracy. In our first experiment we evaluate the accuracy of OPAL's field labeling on all four datasets, but only in the UK real estate and used car domain we employ the form interpretation to further improve the field labeling. Figure 9a shows the results. The first two bars are for the random sample datasets. For the real estate domain, OPAL classifies fields with perfect precision and 98.6% recall. Overall we obtain a remarkable 99.2% F-score. The result is similar for the used car domain, where OPAL obtain 98.2% precision and 99.2% recall, that amount to 98.7% F-score. OPAL achieves lower precision than recall in the used car domain due to the fact that web forms in this domain are more interactive: certain fields are enabled only when some other field is filled properly. However, instead of the HTML attribute disabled, a placeholder is used with text displaying the original field's value label. This introduces noise to field labeling and thus classification.

For the real estate domain, our domain schema consists of a few dozen element and segment types and about 40 annotation types. Similarly, in the used car domain, there are about 30 annotation types. In our experience, creating an initial domain schema (including gazetteers and testing) for a domain takes a single person familiar with the domain and OPAL-TL roughly 1 week.

The other two bars in Figure 9a regard field labeling on ICQ and TEL-8 datasets. On these, OPAL applies only its domain-independent scopes (field, segment, scope) as no domain schema is available for these domains. Nonetheless, OPAL reports very high accuracy also on these forms, confirming the effectiveness of our domain-independent analysis. However, not unexpected, OPAL performs significantly better in presence of domain knowledge.

Cross Domain Comparison. We use ICQ and TEL-8 to compare field labeling in OPAL against existing approaches, on a wide set of domains. Figure 9b details the result of OPAL on each domain of the ICQ dataset. It shows perfect F-score values for the jobs domain (100%) as well as auto and air travelling (99.3% and 98.3%). For comparison, [6] reports 92% F-score for labeling on ICQ on average, which we outperform even in the most difficult domain (books). [15] reports slightly better precision and recall than [6], but OPAL still outperforms it by several percents.
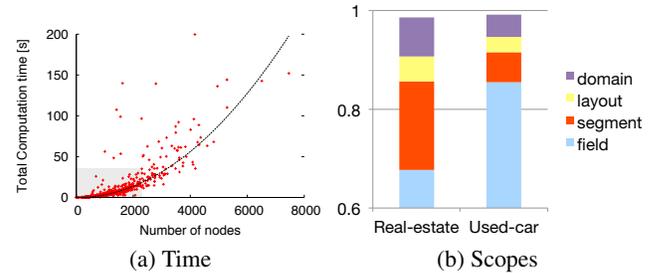
The results for the TEL-8 dataset are depicted in Figure 9c. Here, the overall F-score is 96.3%, again mostly affected by the performance in the books domain. Note that, especially on TEL-8, OPAL obtains very high precision compared to recall. Indeed, lower recall means OPAL is not able to assign labels to all fields, missing some of them. For comparison, [6] reports $88 - 90\%$ overall F-score, which we outperform by a wide margin. [13] reports F-scores between 89% and 95% for four domains in the TEL-8 dataset. Though they perform slightly better on books, we significantly outperform them on the three other domains included in their results, as well as on average.

Contributions of Scopes. We demonstrate the effectiveness of combining different types of analysis by measuring to what extent each of our four scopes contributes to the overall quality of form understanding. We use again the two domain datasets from the previous experiment. For both we show the results for recall (though the picture is similar for precision and F-score). As illustrated in Figure 10b, for the field labeling in the real-estate dataset, the field scope already contributes significantly (67%). The Segment scope increases recall by 18%, page and domain scope add together another 13%. Note that, the contribution of the domain scope is more significant than that of the layout scope, indicating the importance of domain knowledge to achieve very high accuracy form understanding. In the used car domain, field scope alone is even more significant 85% (as many of the websites use modern web technologies and frameworks with reasonable structure).

Scalability. As discussed in Sections 3 and 4, overall the analysis of OPAL is bounded by $O(n^2)$ due to the layout scope. As expected actual performance follows a quadratic curve, but with very low constants. There is a significant amount of outliers, partially due to long page rendering time and partially due to variance in the depth and sophistication of the HTML structure. Figure 10a reports OPAL performance on all 534 forms in the combined TEL-8 and ICQ datasets. The highlight area covers 80% of the forms with 2200 nodes. OPAL requires at most 30s for the analysis (including page rendering) of these forms. Further analysis on the effect of in-

creasing field or form numbers confirms that these have little effect and page size is the dominant factor.

6. RELATED WORK

Current approaches to form understanding roughly fall into two categories: rule and heuristic approaches, such as MetaQuerier [16], ExQ [15], and SchemaTree [6], and machine learning approaches, such as LabelEx [13] and HMM [8], cf. [9] for a general survey.

Aside of system design, OPAL primarily differs from these approaches in two aspects: **(1)** They mostly incorporate only one or two of OPAL's scopes (and feature classes): MetaQuerier,ExQ, and SchemaTree mostly ignore the HTML structure (and thus field and segmentation scope) and rely on visual heuristics only; LabelEx ignores field grouping; HMM visual information. **(2)** None of the approaches provides a proper form model classifying the form fields according to a given schema. Furthermore, no domain knowledge is used to improve the labeling, though LabelEx analyses domain specific term frequencies of label texts and HMM checks for generic terms, such as "min". As evident in our evaluation, each of the scopes in OPAL considerably affects the quality of the form labeling and classification. The fact, that each of these approaches omits at least one of the domain-independent scopes, explains the significant advantage in accuracy OPAL exhibits on Tel-8 and ICQ. Notice also that not using domain knowledge keeps these approaches out of reach of the nearly perfect field classification achieved by OPAL.

Rule and heuristic approaches. Most closely related in spirit to OPAL, though very different in realisation and accuracy, is MetaQuerier [16]. It is built upon the assumption that web forms follow a "hidden syntax" which is implicitly codified in common web design rules. To uncover this hidden syntax, MetaQuerier treats form understanding as a parsing problem, interpreting the page a sequence of "atomic visual elements", each coming with a number of attributes, in particular with its bounding box. In a study covering 150 forms, the authors of MetaQuerier identified 21 common design patterns. These patterns are captured by production rules in a 2P grammar. In contrast, the domain independent part of OPAL achieves nearly perfect accuracy with only 6 generic patterns by combining visual, structural, and textual features. Metaquerier is not parameterisable for a specific domain.

ExQ [15] is similarly based primarily on visual features such as a bias for the top-left located labels comparable to OPAL, but disregards most structural clues, such as explicit `for` attributes of `label` tags and does not allow for any domain specific patterns.

Also [6] uses only visual features (and the `tabindex` and `for` attributes for fields and labels). It follows nine observations on form design, e.g., that query interfaces are organized top-down and left-to-right or that fields form semantic groups. It uses a hierarchical alignment between fields and text nodes similar to OPAL's segment scope and a "schema tree" where the nine observations are observed. Again no adaptation to a specific domain is possible.

Machine Learning Approaches. In contrast to the above approaches, the following machine learning approaches can be trivially adapted to a specific domain using domain-specific training data. The evaluation in [8], however, shows little effect of domain-specific training data: a training set from the biological domain outperforms domain-specific training set in four out of five domains.

LabelEx [13] uses limited domain knowledge when considering the occurrence frequencies of label terms. Domain relevance of the terms occurring in a label, measured as the occurrence frequency in previous forms, is one signal used to score field-label candidates. Field-label candidates are otherwise created primarily using neighbourhood and other visual features, as well as their HTML markup. However, LabelEx does not consider field groups and thus is unable

to describe segments of semantically related fields or to align fields and labels based on the group structure and does not use any domain knowledge aside of term frequency.

HMM [8] uses predefined knowledge on typical terms in forms, such as "between", "min", or "max", but does not adapt these for a specific domain. HMM employs two hidden Markov models to model an "artificial web designer". During form analysis, the HMMs are used to explain the phenomena observed on the page: The state sequences, that are most likely to produce the given web form, are considered explanations of the form. Compared to OPAL, HMM uses no visual features and no domain knowledge.

7. CONCLUSION

As premise of this paper, we assume that form understanding has been limited by overly generic, domain independent, monolithic algorithms relying on narrow feature sets. With OPAL we present a system that addresses these limitations without requiring an inordinate effort in domain engineering. This is achieved through a very accurate domain independent form labeling, exploiting visual, textual, and structural features, by itself already outperforming existing approaches. We complement the domain independent part with a domain dependent form field classification that significantly improves the overall quality of the form understanding in verifying the labeling, classification, and segmentation with domain constraints. To minimize the efforts in domain engineering, we provide a template language that facilitates compact domain knowledge specifications and enables significant cross domain reuse.

8. REFERENCES

[1] Z. Bar-Yossef and M. Gurevich. Random Sampling from a Search Engine's Index. *J. ACM*, 55(5), 2008.

[2] L. Barbosa and J. Freire. Combining Classifiers to identify Online Databases. In *WWW*, 2007.

[3] M. Benedikt and C. Koch. XPath leashed. *CSUR*, 2007.

[4] A. Bilke and F. Naumann. Schema Matching using Duplicates. In *ICDE*, 2005.

[5] M. J. Cafarella, E. Y. Chang, A. Fikes, A. Y. Halevy, W. C. Hsieh, A. Lerner, J. Madhavan, and S. Muthukrishnan. Data Management Projects at Google. *SIGMOD Rec.*, 37(1), 2008.

[6] E. C. Dragut, T. Kabisch, C. Yu, and U. Leser. A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration. In *VLDB*, 2009.

[7] O. Kalijuvee, O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Efficient Web Form Entry on PDAs. In *WWW*, 2001.

[8] R. Khare and Y. An. An Empirical Study on using Hidden Markov Model for Search Interface Segmentation. In *CIKM*, 2009.

[9] R. Khare, Y. An, and I.-Y. Song. Understanding Deep Web Search Interfaces: A Survey. *SIGMOD Rec.*, 39(1), 2010.

[10] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's Deep Web Crawl. In *VLDB*, 2008.

[11] A. Maiti, A. Dasgupta, N. Zhang, and G. Das. HDSampler: Revealing Data behind Web Form Interfaces. In *SIGMOD*, 2009.

[12] I. Navarrete and G. Sciavicco. Spatial Reasoning with Rectangular Cardinal Direction Relations. In *ECAI*, 2006.

[13] H. Nguyen, T. Nguyen, and J. Freire. Learning to Extract From Labels. In *VLDB*, 2008.

[14] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In *VLDB*, 2001.

[15] W. Wu, A. Doan, C. Yu, and W. Meng. Modeling and Extracting Deep-Web Query Interfaces. In *Adv. in Inf. & Intelligent S.*, 2009.

[16] K. Chang, Z. Zhang, B. He. Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. In *SIGMOD*, 2004.

[17] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart. Real understanding of real estate forms. In *WIMS '11*, 2011