Pergamon

Information Systems Vol. 23, No. 8, pp. 521–538, 1998

Printed in Great Britain
0306-4379/98 $19.00 + 0.00

GENERATING FINITE-STATE TRANSDUCERS FOR SEMI-STRUCTURED DATA EXTRACTION FROM THE WEB†

Chun-Nan Hsu[1] and Ming-Tzung Dung[2]

[1]Institute of Information Science, Academia Sinica, Taipei, Taiwan
[2]Department of Computer Science and Engineering, Arizona State University, Tempe, AZ, USA

(Received 18 February 1998; in final revised form 19 October 1998)

Abstract — Integrating a large number of Web information sources may significantly increase the utility of the World-Wide Web. A promising solution to the integration is through the use of a Web Information mediator that provides seamless, transparent access for the clients. Information mediators need *wrappers* to access a Web source as a structured database, but building wrappers by hand is impractical. Previous work on wrapper induction is too restrictive to handle a large number of Web pages that contain tuples with missing attributes, multiple values, variant attribute permutations, exceptions and typos. This paper presents `SoftMealy`, a novel wrapper representation formalism. This representation is based on a finite-state transducer (FST) and contextual rules. This approach can wrap a wide range of semistructured Web pages because FSTs can encode each different attribute permutation as a path. A `SoftMealy` wrapper can be induced from a handful of labeled examples using our generalization algorithm. We have implemented this approach into a prototype system and tested it on real Web pages. The performance statistics shows that the sizes of the induced wrappers as well as the required training effort are linear with regard to the structural variance of the test pages. Our experiment also shows that the induced wrappers can generalize over unseen pages.

Key words: Semistructured Data, Wrapper Induction, Information Extraction, World Wide Web.

1. INTRODUCTION

The rise of the World-Wide Web has made a wealth of data readily available. Integrating these data may significantly increase the utility of the World-Wide Web and create innumerable new applications. For example, integrating information form a large number of Web vendors allows an on-line shopping agent to find the best bargain for the users. However, the Web's browsing paradigm does not readily support retrieving and integrating data from multiple Web sites. Today, the only way to integrate the huge amount of available data is to build specialized applications, which are time-consuming and costly to build, and difficult to maintain.

A promising approach to integrating Web information sources is through the use of *information mediators* [2, 3, 9, 12, 13, 15, 21]. An information mediator provides a query-only intermediate layer between the clients and a large number of heterogeneous information sources, including different types of structured databases and the Web sources, over computer networks. Clients of information mediators can query information sources and integrate the results without knowing their implementation details such as their addresses, access passwords, formats, languages, platforms, etc. Given a query, the information mediators will determine which information sources to use, how to obtain the desired information, how and where to temporarily store and manipulate data. Information mediators are much more extensible and flexible than traditional multidatabase approaches because they can perform dynamic integration of relevant information sources in response to a query.

Information mediators rely on *wrappers* to retrieve data on the World-Wide Web. The primary task of such a wrapper is to extract information in a given set of Web pages and return the results as structured data tuples. For example, consider the fragment of the Caltech CS department faculty page‡ in Figure 1, where we have five data tuples. Each tuple provides information about a faculty member as a vector of attributes. In this case the attributes are URL `U`, name `N`, academic title

†Recommended by Gottfried Vossen
‡`www.cs.caltech.edu/csstuff/faculty.html`

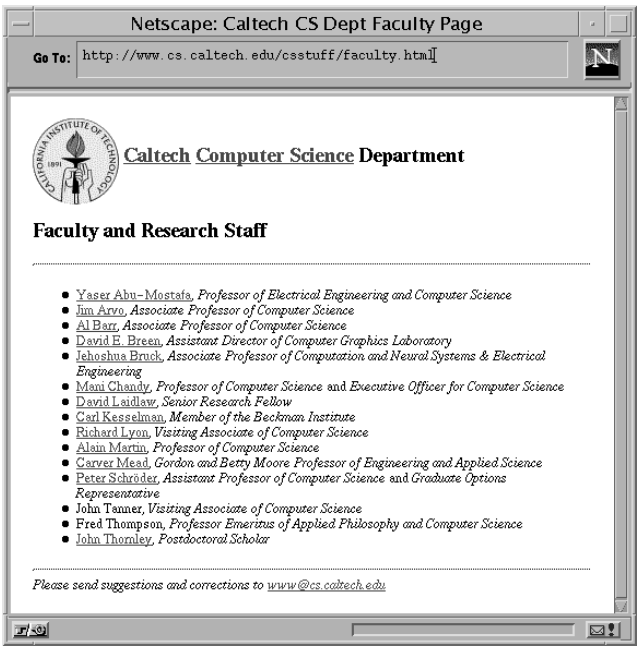

Fig. 1: Caltech CS Faculty Web Page

```
<LI> <A HREF="http://www.cs.caltech.edu/people/mani.html">
     Mani Chandy</A>, <I>Professor of Computer Science</I> and
     <I>Executive Officer for Computer Science</I>
<LI> <A HREF="http://www.cs.caltech.edu/~arvo/home.html">
     Jim Arvo</A>, <I>Associate Professor of Computer Science</I>
<LI> <A HREF="http://www.gg.caltech.edu/~david/">
     David E. Breen</A>, <I>Assistant Director of Computer
     Graphics Laboratory</I>
<LI> John Tanner,
     <I>Visiting Associate of Computer Science</I>
<LI> Fred Thompson, <I>Professor Emeritus of Applied Philosophy
     and Computer Science</I>
```

Fig. 2: A Fragment of the HTML Source of Figure 1 (as for November 1997)

`A` and administrative title `M`. A wrapper for this Web page is supposed to take its HTML source as input (see Figure 2), extract the attributes from each tuple and return a set of faculty tuples `(U,N,A,M)`.

This task is usually considered as an *information extraction* problem (see e.g., [1]). However, it is difficult to build a wrapper based on the traditional information extraction approaches because they are aimed at free text extraction in some particular domains and require substantial natural language processing. Meanwhile, a large portion of the data on the Web are rendered regularly as itemized lists of attributes, such as many searchable Web indexes (e.g., `search.com`), and the example Web page in Figure 1. For those *semistructured* Web pages, a wrapper can exploit the regularity of their *appearance* to extract data instead of using linguistic knowledge. The goal of this work is to develop an approach to rapidly wrap these semistructured Web pages.

Though it is simpler to program a wrapper by hand for semistructured Web pages than for free text, the task is not trivial and may still require substantial programming techniques. Furthermore, Web pages change frequently. Rebuilding wrappers by hand can be expensive and impractical.

According to a survey by Norvig[†], the monthly failure rate of the wrappers at Junglee is more than 8 percent. To resolve this problem, researchers are developing many approaches to automatize wrapper construction (e.g., [4, 7, 14]).

However, there are problems with the previous work. The wrappers in their work extract a tuple by scanning the input HTML string, recognizing the delimiters surrounding the first attribute, and repeating the same steps for the next attribute until all attributes are extracted. Kushmerick [14] advanced the state of the art by identifying a family of PAC-learnable wrapper classes and their induction algorithms. Wrappers of more sophisticated classes are able to locate the margins of a tuple or skip useless text at the beginning and the end of a page based on delimiters, but the attribute extraction steps remain unchanged.

For example, to extract the first tuple in Figure 1, their wrapper will scan the HTML string to locate the delimiters for attribute `U`. In this case the delimiters are "`<A HREF="`" and "`">`". After locating `U` the wrapper will proceed to extract `N`, `A` and `M` and complete the extraction of this tuple.

This approach, however, fails to extract the rest of this example page, although the page appears to be structured in a highly regular fashion. This example page is not a special case. My study shows that the Web pages that their wrappers fail to wrap can be characterized by the following features:

- **Missing attributes** (e.g, a faculty may not have an administrative title)
- **Multiple attribute values** (e.g., a faculty may have two or more administrative titles)
- **Variant attribute permutations** (e.g., the academic title that appears before the administrative title in most tuples may appear after the administrative title in others)
- **Exceptions and typos**

According to [14], 30% of the searchable Web indexes in his survey cannot be wrapped in this manner. As the number of application domains and the complexity of Web-based software are rising, the percentage may increase quickly.

Two problems make the previous work fail to wrap those pages. The first problem is **the assumption that there is exactly one attribute permutation** for a given Web site. However, Web pages that display semistructured data [6] usually need many different attribute permutations for data with missing attributes or multiple attribute values. In our example Web page, there are four different attribute permutations for just five tuples:

`(U,N,A,M) (U,N,A) (U,N,M) (N,A) (N,A)`

Their assumption makes it impossible to wrap Web pages with more than one attribute permutations and thus is invalid.

The second problem is **the use of delimiters**, because it prevents the wrapper to recognize different attribute permutations. Considering the situation when the wrapper has extracted the name `N` from a tuple in our example page and attempts to extract the next attribute. The strings that may serve as the delimiters surrounding the next attribute for the first three tuples are the same "`</A>, <I>`," and "`<I>`⇓`<LI> `"[‡], but the next attribute could be `A` or `M`. As a result, it is not sufficient to distinguish different attribute permutations based on delimiters. This situation occurs very often because minimizing the number of distinct delimiters appearing in a single document is considered as a good formatting practice. The use of delimiters has another problem. How to extract the state and zip code from "CA90210" with delimiters? There is no delimiter at all. This situation occurs very often in many Asian Web pages because no space is required between two words in their languages. Unless the wrapper can take the context of the margin between two attributes into account, it is very difficult to wrap these pages.

This paper presents a novel approach to wrapper induction. Our goal is to quickly wrap a wide range of semistructured Web pages that contain tuples with missing attributes, attributes with multiple values, variant attribute permutations, exceptions and typos. The wrapper is represented as a *finite-state transducer* (FST) where each state may have multiple outgoing edges. The wrapper uses disjunctive *contextual rules* to recognize the context of *separators* between adjacent attributes. This representation allows the wrapper to handle diverse structuring patterns while retaining

[†] Norvig is the chief scientist at Junglee, a commercial information mediator provider `www.junglee.com`.

[‡] ⇓ denotes a newline, following [14].

parsing efficiency. Moreover, theoretic properties of FSTs are well-established that allow us to predict the behavior of a wrapper. Our generalization algorithm can induce the contextual rules from training examples of the tuple extraction. The algorithm generalizes rules by climbing domain-independent taxonomy trees, which reduce the number of the examples required to induce a correct wrapper. Our induction approach is independent of any features specific to HTML and therefore can be applied to other text files.

We have implemented our approach into a prototype system and successfully wrapped many Web pages with diverse structuring patterns. The performance statistics shows that the sizes of the induced wrappers as well as the training effort are linear with regard to the structural variance of the test pages. Our experiment also shows that our system can generalize over unseen pages and structural patterns.

The remainder of the paper is organized as follows. Section 2 describes the FST representation of the wrappers. Section 3 describes the generalization algorithm that induces contextual rules. We then present the empirical evaluation of our approach in Section 4 and compare our approach with related work in Section 5. Finally, the last section concludes with a summary of the contributions and future work.

2. WRAPPER REPRESENTATION

This section describes the SoftMealy representation, which is based on a *finite-state transducer* (FST) where each distinct attribute permutation in the Web page can be encoded as a successful path and the state transitions are determined by matching *contextual rules* that characterize the context delimiting two adjacent attributes.

2.1. Data Model and Attributes

We assume that a schema that defines attributes of the tuples to be extracted is given. The order of attributes in the schema is not significant and there may be zero or more values for each attribute. Also, our approach can accommodate compound attributes of primitive attributes, but the actual model may depend on the modeling language used by client information mediators. For example, the schema of our example Web page simply states that this page contains tuples of `faculty` with four types of attributes `(U,N,A,M)`, but it is not necessary that there is exactly one value for each attribute.

2.2. Tokens

A Web page is simply an HTML source file encoded by a programmer or returned from a searchable Web index. An HTML source file is then a string of characters. The wrapper segments an input HTML string into *tokens* before it starts to extract the tuples. Each token is denoted as $t(v)$, where t is a token class and v is a string. Below are the token classes and their examples:

- All uppercase string: "ASU" → `CAlph(ASU)`
- An uppercase letter, followed by a string with at least one lowercase letter: "Professor" → `C1Alph(Professor)`
- A lowercase letter, followed by zero or more characters: "and" → `OAlph(and)`
- Numeric string: "123" → `Num(123)`
- HTML tag: "<I>" → `Html(<I>)`[†].
- Punctuation symbol: "," → `Punc(,)`
- Control characters — since they are invisible, we will use their length as the argument: a newline → `NL(1)`, four tabs → `Tab(4)`, and three blank spaces → `Spc(3)`

We also distinguish between *word* and *nonword* token classes. The first four token classes are *words* whereas the others are *nonwords*. This distinction is important for the wrapper induction.

[†]It is not necessary to define HTML tags as a class of tokens in our approach. The motivation to include HTML tags here is to improve the learning time and accuracy for Web pages.

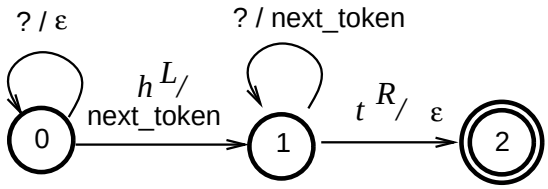

Fig. 3: Body Transducer

2.3. Separators and Contextual Rules

A *separator* is an invisible borderline between two adjacent tokens. A separator s is described by its context tokens, which consist of the left context s^L and the right context s^R. Separators are designed to replace delimiters. To extract an attribute, the wrapper recognizes the separators surrounding an attribute. Separators for an attribute may not be the same for all tuples in a semistructured Web page. *Contextual rules* are defined to characterize a set of individual separators. A contextual rule is expressed as one or more disjunctive sequences of tokens $t(v)$ and its generalized form $t(\_)$, which denotes any token of class t (e.g., Html(_) denotes any HTML tag).

Example 1 Consider the first tuple in Figure 2. The separator s between "<I>" and "Professor" is described by

s^L ::= ... `Punc(,) Spc(1) Html(<I>)`
s^R ::= `C1Alph(Professor) Spc(1) OAlph(of)` ...

This separator marks the beginning of `A`, but it is specific to this tuple. The class of separators S at the beginning of A for all tuples is characterized by the following contextual rules:

S^L ::= `Html(</A>) Punc(,) Spc(_) Html(<I>)` |
`Punc(_) NL(_) Spc(_) Html(<I>)` |
`Punc(,) Spc(_) Html(<I>)`
S^R ::= `C1Alph(_)`

where "|" means "or". S covers the separators whose left context satisfies one of the three disjuncts and the right context is any `C1Alph` token. The first disjunct states that the left context is a token string "</A>" followed by a comma, one or more spaces and "<I>", from left to right. The second and third disjuncts can be interpreted in a similar manner. □

2.4. Finite-State Transducer

The FST in SoftMealy takes **a sequence of the separators** rather than the raw HTML string as input. The FST matches the context tokens of the input separators with contextual rules to determine state transitions. This is analogous to moving a flashlight beam through a token string. When we pictorially present a finite-state transducer, we adapt the standard notation (see e.g., [18]) unless explained otherwise. A question mark "?" stands for any input separator that does not match on any other outgoing edge, and `next_token` denotes the token string right next to the input separator. For example, the `next_token` of the separator in Example 1 is "Professor". We also use "+" to denote string concatenation.

A FST is constructed for one tuple type. If there are many types of tuples in a page, we can build a FST for each type. The substring that contains a type of tuples in a Web page is called the body or the scope of that type of tuples. Figure 3 shows the part of the FST that extract a body string. The FST tries to match h^L, the left context of the separator at the head of the body, and t^R, the right context of the separator at the tail of the body. This way, the body transducer extracts the string between h^L and t^R and feed this string to the tuple transducer as the input. The tuple transducer than iteratively extracts the tuples from the body string. In each iteration, the tuple transducer accepts a tuple and returns the extracted attributes.

A tuple transducer is the core of a SoftMealy wrapper. Formally, a tuple transducer is a 5-tuple $(\Sigma_1, \Sigma_2, Q, R, E)$ such that:

- Σ_1 is the set of **separators** as the input alphabet.
- Σ_2 is the set of characters as the output alphabet.
- Q is a finite set of states containing:
  - Initial state b and final state e represent the beginning and the end of a tuple, respectively.
  - k is a state if there is an attribute k to be extracted.
  - For each attribute k there is a state $\bar{k}$ that corresponds to the *dummy attribute*, the tokens that we intend to skip between the end of attribute k and the next attribute. For example, $\bar{N}$ ="`</A>, <I>`" for the first tuple in Figure 2 because it is between N ="Mani Chandy" and the next attribute "Professor..."
- R is the set of contextual rules. We use $s\langle i, j\rangle$ to denote the set of contextual rules (a subset of R) that characterize the class of separators (a subset of Σ_1) between two attributes i, j (including dummy attributes and b, e).
- $E \subseteq Q \times R \times \Sigma_2^* \times Q$ is the set of edges. A state transition from i to j is legal if there exists an edge $\langle i, r, o, j\rangle \in E$ such that the next input separator satisfies r. The output string o is arranged as follows:
  - If $i \neq j$ and j represents an attribute to be extracted, then the output o is "j ="+`next_token`[†].
  - If $i = j$ and j represents an attribute to be extracted, then o is `next_token`.
  - Otherwise, o is ε.

An edge from state i to j encodes that attributes i and j are adjacent in the target Web pages when their separator satisfies the associated contextual rule. This way, each attribute permutation can be encoded as a successful path in the transducer. For the sake of compactness, we can combine edges that connect the same pair of states into a single edge. This is possible because the contextual rules can be disjunctive and `next_token` is used as the output string whenever applicable. Therefore, there is at most one edge between two states.

States allow the tuple transducer to decide when to extract tokens and when to skip, depending on whether it is at an attribute state or at a state for a dummy attribute. At final state e, if the FST reaches the end of the body, the extraction will terminate. Otherwise, state e will become state b for the extraction of the next tuple. Figure 4 shows the diagram of the tuple transducer for the example Web page.

The contextual rules might cover each other and make the FST nondeterministic, which is not efficient and error-prone. To deal with this problem, in the current implementation, we use a *rule priority policy* that prefers a specific rule to a general one. More precisely, the policy prefers rules that are longer or with more ground tokens. This heuristic worked well in the experiments but we already have several concrete proposals and will address the problem in the future. See the last section for more detail.

To sum up, the extraction starts by tokenizing an input Web page, and then uses a FST to recognize separators that mark the head and tail of the body and return the tuples in the body.

3. WRAPPER INDUCTION

A FST consists of input/output alphabets, states and edges. In the case of SoftMealy FSTs, the components remaining to be induced are the edges and their associated contextual rules. In this section, we first present an algorithm that generalizes contextual rules from labeled tuples in a set of sample pages, then we discuss a procedure for generating contextual rules for the body transducer. We also present an algorithm for rapid error recovery to deal with the situations when a generated contextual rule is not sufficiently expressive to correctly extract all tuples.

[†]Since the output strings are tokens, and HTML tags are a class of tokens, the FST will output an entire "`<a href=...>`" HTML tag as a person's URL. A simple post-processing step can extract the actual address from an HTML tag. This applies to other HTML anchors and images.

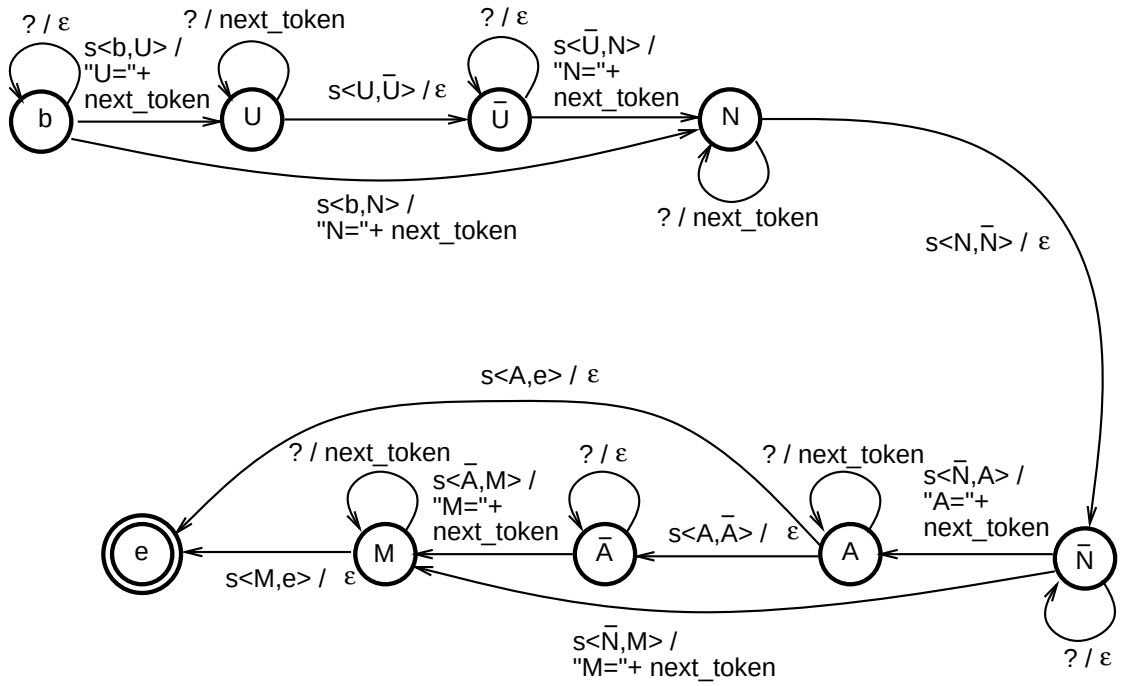

Fig. 4: Tuple Transducer

3.1. Generalization Algorithm

The input to the generalization algorithm is a set of labeled tuples, which provides the positions of the separators and the attribute permutations. A tuple is said to be *labeled* if the positions of its attributes are known. Labeling can be provided by a human or an *oracle*. The labeled tuples provides the positions of the separators and the attribute permutations (including dummy attributes) of the tuples. The algorithm takes these positions as training examples and generalizes their context into contextual rules as the output.

The first step of learning is to construct corresponding states and edges according to the attribute permutations appearing in the training tuples. The next step is to learn the contextual rules for the edges. The learner will collect the separators that delimit the same pairs of attributes together. This produces a set of training instances of $s\langle i, j\rangle$ for each pair of adjacent attributes i and j. Each instance is expressed as ground context tokens of the separator. Our goal is to induce contextual rules that cover all separator instances in the sample pages.

To constrain the search space of a large number of the contextual rules that can be induced, we incorporate an inductive bias that constrains the length of separator instances. This bias requires that the context of a separator instance includes the adjacent zero or more *nonword* tokens plus at most one *word* token (i.e., `CAlph`, `C1Alph`, `OAlph` and `Num`; others are nonword tokens), and that the context should not span across the separator of another pair of attributes. This bias is based on our heuristic that a short context string including a word token is sufficient to characterize a separator class.

Example 2 Consider the first tuple in Figure 2. Labeling this tuple yields an attribute permutation $(b, U, \bar{U}, N, \bar{N}, A, \bar{A}, M, e)$ and the positions of their separators. After applying the inductive bias for the length of training instances, the instances for separators $s\langle \bar{N}, A\rangle$ and $s\langle A, \bar{A}\rangle$ and $s\langle \bar{A}, M\rangle$ are as follows:

$s\langle \bar{N}, A\rangle^L$ ::= `Html(</A>) Punc(,) Spc(1) Html(<I>)`
$s\langle \bar{N}, A\rangle^R$ ::= `C1Alph(Professor)`
$s\langle A, \bar{A}\rangle^L$ ::= `C1Alph(Science)`

$s\langle A, \bar{A}\rangle^R$ ::= `Html(</I>) Spc(1) OAlph(and)`
$s\langle \bar{A}, M\rangle^L$ ::= `OAlph(and) NL(1) Spc(5) Html(<I>)`
$s\langle \bar{A}, M\rangle^R$ ::= `C1Alph(Executive)`

We note that from our inductive bias, $s\langle \bar{N}, A\rangle^L$ does not include `C1Alpha(Chandy)` because it will make the context string span across the preceding separator between N and $\bar{N}$. Also, the bias excludes the control characters after the token "and" for $s\langle A, \bar{A}\rangle^R$. □

With the training set for each separator class, we can apply an induction algorithm (e.g., [16]) to produce contextual rules. However, due to the bias, the length of each training instance may be different. We cannot directly apply existing induction algorithms because the training instances of those algorithms must be feature vectors with fixed length. Therefore, we develop an algorithm that aligns training instances into columns, and then generalize each columns to generate new rules.

The alignment of the tokens in the training instances is as follows. Formally, a training instance for the left context is of the form

$$[word]\ \ nonword^*,$$

that is, an optional word token followed by zero or more nonword tokens. Symmetrically, the right context is of the form

$$nonword^*\ \ [word].$$

Our algorithm will align word tokens to the same column if they exist, and align nonword tokens to the right for left context and to the left for right context. The motivation of this alignment is that nonword tokens should be aligned according to their relative positions with the separator. This way, the tokens next to the separator will be aligned together. Also, the closest word token to the separator is usually an important signal of attribute transition and thus should be aligned together.

Example 3 Suppose we label tuples number 1, 2, 4 and 5 in Figure 2 and collect a training set for $s\langle \bar{N}, A\rangle$. After the alignment, we have four columns for $s\langle \bar{N}, A\rangle^L$ and one column for $s\langle \bar{N}, A\rangle^R$:

```
1.s⟨N̄,A⟩^L ::= Html(</A>) Punc(,) Spc(1) Html(<I>)
2.s⟨N̄,A⟩^L ::= Html(</A>) Punc(,) Spc(1) Html(<I>)
4.s⟨N̄,A⟩^L ::=     Punc(,)   NL(1) Spc(5) Html(<I>)
5.s⟨N̄,A⟩^L ::=               Punc(,) Spc(1) Html(<I>)
```

```
1.s⟨N̄,A⟩^R ::= C1Alph(Professor)
2.s⟨N̄,A⟩^R ::= C1Alph(Associate)
4.s⟨N̄,A⟩^R ::= C1Alph(Visiting)
5.s⟨N̄,A⟩^R ::= C1Alph(Gordon)
```

For an example where both word tokens and nonword tokens present, consider the following two training instances for $s\langle \bar{A}, M\rangle^L$ and the resulting alignment:

```
s⟨Ā,M⟩^L ::= OAlph(and) NL(1) Spc(5) Html(<I>)
s⟨Ā,M⟩^L ::= OAlph(and)       Spc(1) Html(<I>)
```

In this example, the leftmost word tokens (i.e., `OAlph(and)`) are aligned together and the remaining nonword tokens are aligned to the right. □

The generalization algorithm induces contextual rules by taxonomy tree climbing [16]. The algorithm generalizes each column by replacing each token with their least common ancestor with other tokens in the same taxonomy tree. Figure 5 shows fragments of the taxonomy trees. We introduce a new token class `CnAlph` in the taxonomy tree to cover all tokens led by an uppercase letter. Our current implementation applies a heuristic that always generalizes a control character token $t(v)$ to $t(\_)$ (i.e., when t is one of `NL`, `Spc` or `Tab`). Another heuristic generalizes a token $t(v)$ to $t(\_)$ when there is no other token in the column that shares the same taxonomy tree with t.

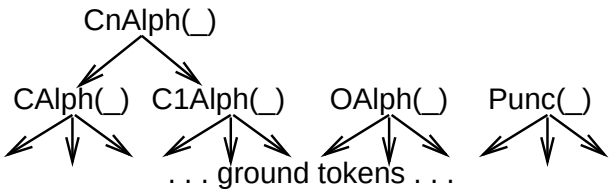

Fig. 5: Token Taxonomy Trees

This heuristic allows us to generate contextual rules for well structured Web pages using a single training tuple.

After the generalization, duplicate instances will be removed and the remaining instances constitute the output contextual rules. The complete algorithm is given as follows:

Algorithm 1 (Generalization)

```
1 INPUT D = training set that has been aligned;
2  FOR each column c in D
3   FOR each token t(v) in c
4    IF ∃ t'(v') ∈ c such that t and t' in the same taxonomy tree THEN
5     replace t(v) with their least common ancestor
6  remove duplicate instances in D;
7  RETURN D;
```

Example 4 The generalization of the instances in Example 3 yields the following contextual rules:

$1.s\langle\bar{N},A\rangle^L$ ::= Html(</A>) Punc(,) Spc(_) Html(<I>)
$2.s\langle\bar{N},A\rangle^L$ ::= Html(</A>) Punc(,) Spc(_) Html(<I>)
$3.s\langle\bar{N},A\rangle^L$ ::= Punc(_) NL(_) Spc(_) Html(<I>)
$4.s\langle\bar{N},A\rangle^L$ ::= Punc(,) Spc(_) Html(<I>)

$1.s\langle\bar{N},A\rangle^R$ ::= C1Alph(_)
$2.s\langle\bar{N},A\rangle^R$ ::= C1Alph(_)
$3.s\langle\bar{N},A\rangle^R$ ::= C1Alph(_)
$4.s\langle\bar{N},A\rangle^R$ ::= C1Alph(_)

Removing the duplicates, we obtain the contextual rules

$s\langle\bar{N},A\rangle^L$ ::= Html(</A>) Punc(,) Spc(_) Html(<I>) |
Punc(_) NL(_) Spc(_) Html(<I>) |
Punc(,) Spc(_) Html(<I>)
$s\langle\bar{N},A\rangle^R$ ::= C1Alph(_)

They are identical with S^L and S^R in Example 1. □

3.2. Generating Contextual Rules to Extract the Body

The body transducer in Figure 3 is designed to skip distracting text around the body that contains the list of the interested tuples in the input Web page. Unlike the separators between attributes, the head and tail separators of the body usually appear in unstructured free text where our length bias may not be adequate. Therefore, we developed a slightly different algorithm to generate contextual rules for the separators of the body.

Given the positions of the head separator h and the tail separator t, our algorithm starts by extracting their context strings under the length constraint that follows the bias described in Section 3.1 (i.e., one word token, followed by a sequence of nonword tokens). The context strings serve as the initial contextual rules. The system then tests the initial rules by actually applying

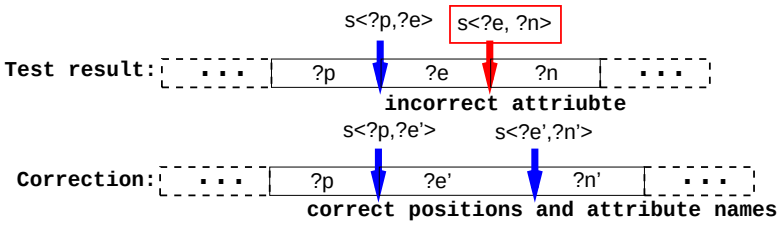

Fig. 6: Incorrect Extraction

them to extract the body. If the result is incorrect, then the rules are too general and need to be specialized. The system specializes an incorrect rule by including the next token in the left context for h^L or in the right context for t^R. These two steps are repeated until the rules can correctly extract the body. We note that all tokens in the resulting rules are ground because the system does not generalize them.

Example 5 The contextual rules for the head and tail separators of the Caltech faculty page are as follows:

h^L ::= `C1alpha(Staff) Html(</H2>) NL(1) Html(<HR>) NL(1) Html(<UL>)`
t^R ::= `Html(</UL>) NL(1) Html(<HR>) NL(1) Html(<ADDRESS>) NL(1) Html(<I>) C1alpha(Please)`

In this case, the initial rules selected based on the length bias yield correct extraction and thus no more tokens are included. □

We can extend this algorithm to generalize the contextual rules of the head and tail separators for the output pages of searchable Web indexes. After generating the rules for the body of each example page, the system takes these ground rules as the training instances and applies Algorithm 1 to induce new rules that cover all example pages. If the resulting rules are too general, we can repeatedly apply the same specialization steps to include new tokens to the training instances until the rules can recognize the body of each example page correctly.

3.3. Rapid Error Recovery

The generalized contextual rules may yield incorrect extraction in several situations. The first situation is when the wrapper encounters an unseen attribute permutation. Since our algorithm cannot induce rules for an unseen pair of adjacent attributes, we need to include at least one training instance for each possible pair. The second situation is when the data type (e.g., numeric or string) of the attribute to be extracted is different from any training example, and the wrapper can not identify the separators around this attribute. As in the first situation, we need to include this new instance as a training example. The third situation is when the generalized rules cover each other and make the transducer nondeterministic. To deal with this problem, we use a *rule priority policy* that prefers a specific rule to a general one. However, this heuristic policy may still make a wrong choice. Exceptions and typos in semistructured Web pages also contribute to errors. Nevertheless, the wrapper must return all tuples correctly.

We have an approach to rapid error recovery. Suppose $s\langle ?p, ?e\rangle$ and $s\langle ?e, ?n\rangle$ are the separators around the first incorrectly recognized attribute $?e$ during the extraction (see Figure 6)†. To recover from this error an oracle is required to provide labeling for the attribute $?e$. In effect the labeling should involve providing the correct position of $s\langle ?e, ?n\rangle$ and the correct names of the target attribute $?e'$ and its next attribute $?n'$. Our error recovery system takes the labeling as the input and infers the cause of the error and the recovery steps to correct the error. The inference is based on the decision tree shown in Figure 7.

†$?p$, $?e$, $?n$ and other symbols started by a "?" mark denote variables that can be instantiated by attribute names.

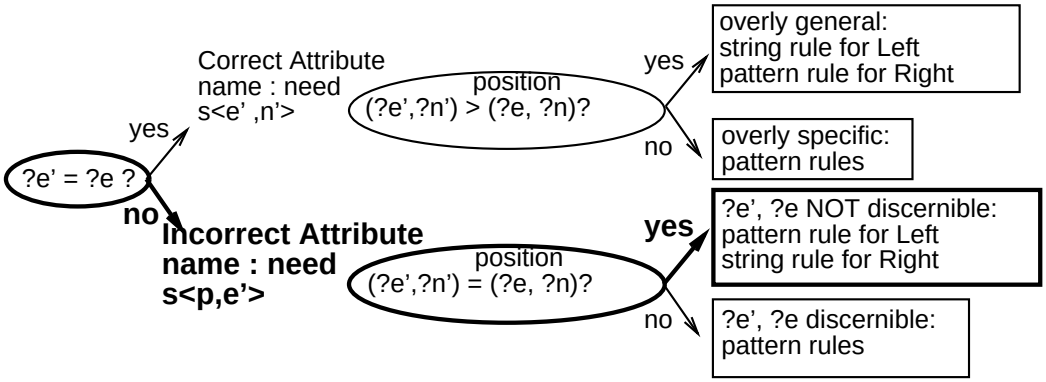

Fig. 7: Decision Tree of Error Recovery

The recovery involves generating new rules to compensate existing rules. This is achieved by creating new training instances and applying Algorithm 1 to generate correct new rules. This approach recovers most of the error cases. However, there are two cases that cannot be recovered in this way. The first case is when the existing rules are overly general and need to be specialized. The second case is when the names of the surrounding attributes are incorrect but it is not discernible by any contextual rules due to the lack of their expressive power. We deal with these cases by memorizing the entire string of the left attribute for the first case, and the right attribute for the second case. We refer to the type of rules obtained in this manner as *string rules*, and to make distinction, the generalized contextual rules are referred to as *pattern rules*. The priority of string rules is higher than pattern rules. This approach is simple yet effective in our tests.

Example 6 Consider the third tuple shown in Figure 2. Using the learned contextual rules for $s\langle \bar{N}, A\rangle$ in Example 4, the transducer incorrectly regards "Assistant Director ... Laboratory" as A (academic title) rather than M (administrative title). That is, we have $?p = \bar{N}$, $?e = A$, $?n = e$ from the test result, but the correct attribute names are $?e' = M$ and $?n' = e$. According to Figure 7, the root of the decision tree is to check whether $?e' = ?e$. Since $?e' = A \neq M = ?e$, we need to generate new rules for $s\langle ?p, ?e'\rangle$, that is, $s\langle \bar{N}, M\rangle$. Now, we need to check the next condition specified in the lower node of the second layer of the decision tree. Since the position of the separator between A and e is equal to the position of the separator between M and e, moving to the "yes" branch in the decision tree, we see that A and M are not discernible using contextual rules and we need to generate a pattern rule for the left context and a string rule for the right context to cover this tuple. The resulting new rules are as below:

$s\langle \bar{N}, M\rangle^R$ ::= `''Assistant Director of Computer Graphics Laboratory''`

$s\langle \bar{N}, M\rangle^L$ ::= `C1Alph(Bruck) Punc(,) Spc(1)`

We note that it includes all control characters. With this new rule, the transducer can correctly extract this tuple and the others. In our experiment, a correct wrapper for the example Web page is completed using a total of four training tuples and two recovered error attributes. □

Figure 7 only shows the inference for the cases where the contextual rules of the affected state are all pattern rules. If the rules include string rules generated from previous error recovery steps, and the resulting FST still make mistakes, the system will need to generate a string rule to overrule the existing string rules. This occurs when the existing string rule matches a substring of a wrong attribute or appears as a partial combination of two adjacent attributes. These situations, however, never happened in our tests so far.

There is a case beyond the capability of our error recovery approach. The case occurs when two identical strings, with the same previous attribute, represent different attributes. Consider the situation when we want to extract the first name and the last name of a person but we also want to skip the middle names. Since the number of middle names varies and they may appear exactly the

same as a last name, unless our extractor has the expressive power of the context-free languages, there is no way to extract the data correctly in general. However, if all the middle names appear differently from the last names, our wrapper can *memorize* the confusing cases as string rules and cover this case.

4. EXPERIMENTAL RESULTS

We have implemented our learning algorithm into a prototype system in JAVA. The system has a GUI that allows a user to open a Web site, define the attributes, and label the tuples in the Web page with a mouse. The system learns a wrapper from labeled tuples. If an error appears in the output, the user can provide correct labeling and invoke the error recovery program to correct the error until the output is completely correct.

We conducted two experiments to evaluate SoftMealy. The first experiment is to test its expressiveness. The second experiment is to test whether the learning system can generalize over unseen pages. The remainder of this section presents the results of these two experiments.

4.1. Experiment for Expressiveness

We have successfully applied SoftMealy to wrap a set of Web pages that itemize Computer Science faculty members at universities in North America. The test pages are selected randomly from the index provided by the Computing Research Association (`cra.org`). For each page, we constructed a wrapper to extract a subset of predefined 14 attributes (e.g., name, URL, picture, *et cetra.*), depending on their availability in the page. We chose this domain because it is a rich source of itemized Web pages with diverse structural variations. It is our intention not to select test pages with any presumed format in mind.

Table 1[†] shows the profile of the test pages and the performance statistics. The average number of different attribute permutations of the test pages is 2.63, which shows that these pages are not strictly structured. The number of training instances `TI` for each page is the maximal number of labeled tuples used to learn a correct wrapper in our random trials.

To evaluate the generalization of the learned contextual rules, we compared the number of learned disjuncts `R` with the number of separator classes `SP` (i.e., the number of edges). We also compared `R` with the total number of tuples `NT` in a page. Ideally, `R` should be independent of `NT` but correlated with `SP`. Figure 8(a) and (b) show the correlations. The results favor our approach because we obtained a strong correlation coefficient 0.980 (≈ 1) between `R` and `SP` and almost no correlation -0.031 (≈ 0) between `R` and `NT`.

The number of training instances `TI` is also correlated with `SP`, with correlation coefficient equal to 0.773 (see Figure 8(c)). This shows that our learning algorithm can learn correct contextual rules for each class of separators with a handful of training tuples. The outlier at the upper right corner is obtained from the faculty page of the Drexel University[‡]. This page is nested and particularly difficult to wrap because, in addition to its many exceptions, it is a "nested" structured page [14] with distracting free text in some (*not all*) inter-tuple regions. However, SoftMealy still managed to wrap it.

4.2. Generalizing over Unseen Pages

Wrappers are particularly useful when applied to wrap searchable Web information sources. These Web sources output a semistructured Web page that contains a list of tuples in response to a user query. It is important for a wrapper to generalize over unseen output pages for these information sources. To evaluate this capability, we apply our system to wrap the ASU Web directory[§]. This Web source provides the information of the faculty, staff and students at ASU with a total of 12 possible attributes. This source is particularly interesting for our evaluation because

[†] This experiment was conducted in November 1997.

[‡] `cbis.ece.drexel.edu/ECE/ece_bios.html`

[§] `www.asu.edu/asuweb/directory/`

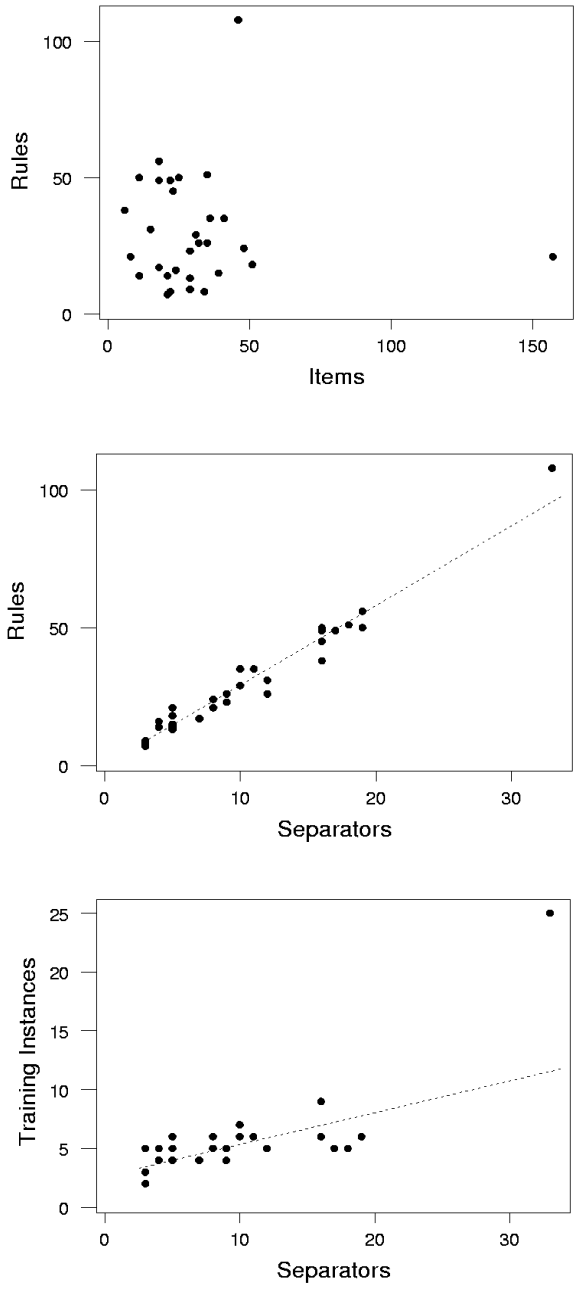

Fig. 8: Scatterplots of (a) Rules and Tuples, (b) Rules and Separators, and (c) Training Instances and Separators

URL	NT	A	AP	S	SP	R	TI
www.cs.nmt.edu/Faculty.html	6	6	3	14	16	38	6
www.cs.olemiss.edu/faculty/	8	3	2	8	8	21	5
www.cs.fit.edu/people/faculty.html	11	1	2	4	4	14	5
www.cs.brandeis.edu/faculty-descriptions/	11	7	2	20	19	50	6
www.cs.caltech.edu/csstuff/faculty.html	15	4	4	10	12	31	5
www.cs.colostate.edu/people.html	18	2	2	6	7	17	4
www.cs.dartmouth.edu/faculty/	18	7	5	16	19	56	6
www.cs.msstate.edu/FACULTY_AND_STAFF/	18	6	3	14	16	49	9
www.cse.fau.edu/faculty.html	21	2	1	6	5	14	5
www.cs.gmu.edu/faculty/	21	1	1	4	3	7	2
www.cs.columbia.edu/home/people/people.html#faculty	22	1	1	4	3	8	3
cyclone.cs.clemson.edu/html/whoswho/facultyindex.shtml	22	7	4	15	17	49	5
www.cs.gmu.edu/ofchours.html	23	5	4	15	16	45	9
www.cs.jhu.edu/faculty.html	24	2	1	5	4	16	4
www.cs.wm.edu/cspages/people/faculty.html	25	6	5	20	16	50	9
www.cs.msu.edu/fac/index.html	29	2	2	5	5	13	4
www.cs.iastate.edu/faculty/index.html	29	3	3	7	9	23	4
www.cs.nps.navy.mil/people/faculty/	29	1	1	4	3	9	5
www.cs.concordia.ca/People/Faculty.html	31	4	2	10	10	29	7
www.cs.duke.edu/cgi-bin/factable?text	32	5	2	12	12	26	5
www.cs.washington.edu/people/faculty/	34	1	1	4	3	8	2
www.eas.asu.edu/~csedept/people/faculty.html	35	7	3	17	18	51	5
gauss.nmsu.edu:8000/faculty/faculty.html	35	4	2	9	9	26	5
www.cs.nyu.edu/cs/new_faculty.html?	36	3	2	8	10	35	6
simon.cs.cornell.edu/Info/Faculty/faculty-list.html	39	2	1	6	5	15	5
www.seas.gwu.edu/seas/eecs/faculty.html	41	4	3	8	11	35	6
cbis.ece.drexel.edu/ECE/ece_bios.html	46	10	13	28	33	108	25
class.ee.iastate.edu/fac_staff/index.html	48	3	2	8	8	24	6
www.ri.cmu.edu/ri-home/people.html	51	2	1	6	5	18	5
www-eecs.mit.edu/faculty/index.html	157	2	1	6	5	21	6

Key: NT= # of tuples, A= # of attributes, AP = # of attribute permutations, S= # of states, SP = # of separator classes,R = # of disjuncts, TI = # of training instances.

Table 1: Performance Statistics on Wrapping CS Faculty Web Pages

its output pages contain a large number of different attribute permutations due to the diversity of the information available for different personnel categories at a university (e.g., students, staffs, and faculty).

The experiment was conducted as follows. We sent 11 random queries to the source and obtained 11 output pages. We sorted the pages on their size (i.e., the number of tuples) and selected the largest page as the test page and the other 10 pages as the training pages. The test page contains 69 tuples and 17 different attribute permutations while the training pages contain a total of 85 tuples and 18 different attribute permutations. Among these attribute permutations, only seven appearing in the test page also appear in the training pages.

To plot the learning curve, we sorted the training pages in the ascending order of their size, and applied the learning system to generate a wrapper for each observed training page so far, and used the generated wrapper to extract the test page. We labeled a total of 15 tuples in order to successfully wrap the 10 training pages and obtained 92 contextual rule disjuncts. The result is shown in Figure 9, where all the data points are cumulative. Interestingly, even though there are 10 unseen attribute permutations in the test page, SoftMealy still covered 60 out of 69 tuples ($\approx 87\%$) in the test page. We note that because an error may affect the extraction of the rest of the page, four tuples not covered in the test page would have been covered if their previous tuples had been extracted correctly.

To see why SoftMealy can generalize over unseen attribute permutations, consider the example FST in Figure 4, which covers not only the four attribute permutations appearing in the example Web page, but also the other two attribute permutations because the learned contextual rules constitute a total of six paths in this FST.

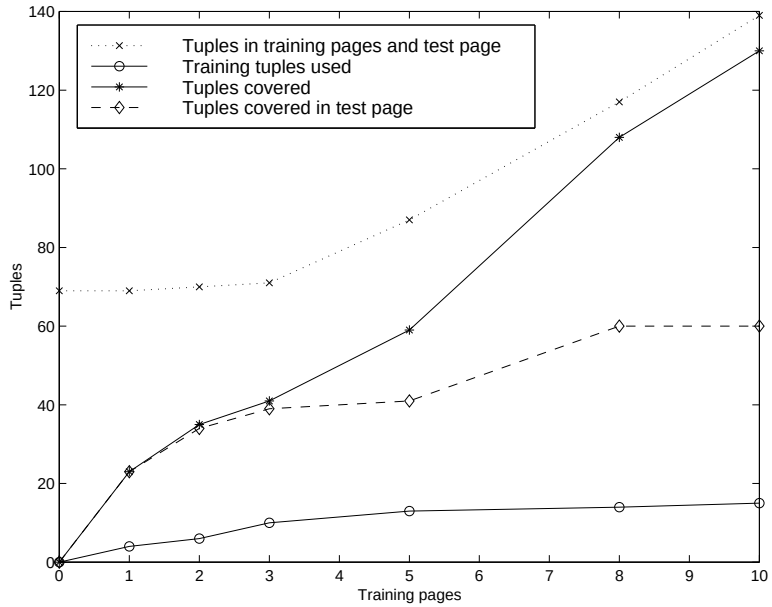

Fig. 9: Learning Curve of SoftMealy on Wrapping ASU Directory

Except for a small portion of the tuples that serve as the training instances, in the training page, the total number of unseen tuples in both training and test page is 139. The generalized wrapper covered 130 tuples among these unseen tuples. This amounts to a 94% overall tuple extraction accuracy.

To further verify the utility of SoftMealy, we surveyed a list of searchable Web indexes given in [14]. Since these Web sites create pages by a program, they have a uniform format. But still, [14] concluded that his approach cannot wrap 30% of them. We surveyed the same list and have yet to encounter any semistructured resource that cannot be wrapped by SoftMealy.

5. RELATED WORK

This section first discusses the notion of semistructuredness of data and documents in the literature. Next, we survey approaches to wrapper construction for semistructured Web pages and compare our work with these approaches. We discuss briefly the impact of XML on the wrappers.

5.1. Semistructuredness of Data and Documents

The term "semistructured" has been used in several different ways in previous work related to the World-Wide Web. In AI community, including this work, the term relates to the manner that a Web page is presented, while in the databases community, the term concerns the manner that the content information of a Web page is organized (see e.g., [6]).

We consider a Web page that provides itemized information as *structured* if each attribute in a tuple can be correctly extracted based on some uniform syntactic clues, such as delimiters or the orders of the attributes. On the other hand, a Web page is *unstructured* if linguistic knowledge is required to extract the attributes correctly. For example, the job page of Computing Research Association[†] is structured because every attribute is contained in a cell of an HTML table, while the SIGMOD job page[‡] is unstructured, because it lists each job offer as a free sentence. It is important to emphasize that the *structuredness* of a Web page must depend on the attributes that the users are interested to extract. In the case of the SIGMOD job page, it is considered

[†] `cra.org/jobs`
[‡] `www.acm.org/sigmod/jobs`

unstructured because we intend to extract job titles, employers, application deadlines, *et cetera*, but if each job offer description is regarded as a single attribute then the page is structured. Usually, machine generated Web pages are very structured, while hand generated Web pages are less structured but there are plenty of exceptions. It is easy to construct a wrapper for a structured Web page but difficult for an unstructured one because it requires lexical knowledge.

Semistructured Web pages are those that are not unstructured. Semistructured Web pages may contain tuples with missing attributes, attributes with multiple values, variant attribute permutations, exceptions and typos. Therefore, it is likely that the contents of a semistructured Web page does not fit in a structured data model. In other words, the contents of a semistructured Web page are usually semistructured data. However, it is also possible that structured data are presented in a semistructured or even unstructured fashion. The approach described in this paper focuses on generating wrappers for semistructured Web pages that contain semistructured or structured data.

5.2. Wrapper Construction

Wrapper construction is essential to allow machines to extract data from semistructured Web pages. The most primitive approach to this problem is to program a wrapper for each Web site by hand in some programming language such as Perl. This approach was found impractical because it is too difficult to update and maintain the wrappers. To solve this problem, many approaches are proposed to rapid wrapper construction. One class of the approaches is to use an easy-to-program language to create wrappers. Some researchers use existing grammar definition tools such as *yacc*, LL(k) grammars, etc. Others define their own specialized languages for wrapper construction, such as the wrapper construction language in the TSIMMIS project [8], the concept description frame of InfoExtractor [19], and the procedural language defined in [5]. These languages are sufficiently expressive (usually at least as expressive as regular expressions, i.e., finite state automata), but mastering those languages still requires substantial computer background. These approaches become unacceptable when skillful programmers are not available.

Another class of the approaches to rapid wrapper construction relies on heuristics to automatize the construction. Doorenbos et al. [7] proposed an approach to generating wrappers automatically in their ShopBots project. Their approach applies the knowledge about the possible attribute values and "environmental assumptions" to wrap machine-generated pages of on-line venders. Ashish and Knoblock [4] proposed another approach by locating HTML tags that presumably surround an interesting attribute. Hsu et al. [11] proposed a template-based approach that also takes advantage of HTML tags. These approaches, however, only cover a small proportion of the Web pages. It is difficult to extend their heuristic to other Web pages.

Kushmerick [14] advanced the state of the art with a wrapper induction approach. Like in our work, their wrappers are induced from labeled training tuples. He defined a family of wrapper classes, which essentially consist of linear FSTs that extract data by recognizing delimiters between attributes. He was able to establish a PAC bound on the number of training examples. However, his representation is too restrictive to wrap semistructured pages with variant structuring patterns. Our approach is more flexible because the states in our FST may have multiple outgoing edges. Another advantage of our approach is the use of separators. Recognizing separators is more flexible than recognizing delimiters because separators are described by their contexts which allow a wrapper to distinguish different attribute transitions, but in many cases it is impossible by recognizing delimiters.

More recently, Muslea et al. proposed Stalker [17] that can learn wrappers more expressive than Kushmerick's work. Their wrappers are based on a set of "disjunctive landmark automata". Each landmark automaton is specialized in extracting an attribute. As a result, to complete the extraction, their wrapper needs to apply landmark automata several times, each time the wrapper needs to scan the Web page once. In contrast, our wrapper can read in a Web page as a stream and output extracted tuples as a stream from the partially read Web page in real time. This feature

is important to allow a client information mediator to efficiently answer a query on line. It is not clear, however, how landmark automata compare with SoftMealy in terms of expressiveness and learnability. We plan to perform an empirical comparative study soon.

Doorenbos et al. [7] and Kushmerick [14] also describe approaches to automatically labeling a Web page using the knowledge about possible attribute values. Though we think it is more difficult to obtain the required knowledge than labeling tuples by hand, if the knowledge is available, we can apply their approaches to produce training examples for our generalization algorithm.

5.3. XML and the Wrappers

A wrapped World-Wide Web implies a wide-open machine-comprehensible environment available to all kinds of intelligent agents. The explosive growth of the Web sites on the Internet gives us an example that it is possible to wrap the World-Wide Web if we provide practical tools for ordinary users. The emergence of XML [20] appears to be "*the*" tool that will solve *all* the problems for the wrappers. We think XML will make wrapper induction simpler but will not eliminate the need of wrappers. After all, XML markup only provides one of many possible semantic interpretations of a document. When an application needs to extract data at a different granularity, or to integrate data in different domains, we still need wrappers to provide other interpretations. Moreover, wrappers can help mark up huge amount of legacy data in XML.

6. CONCLUSIONS AND FUTURE WORK

This paper has presented an approach to wrapping semistructured Web pages. The key features of our approach are as follows. First, the FST representation allows a wrapper to encode different attribute permutations. Second, the disjunctive contextual rules allow a wrapper to characterize the context of separators. Third, the learning algorithm can bring to bear token taxonomy trees and induce accurate contextual rules with a handful of examples. Finally, exceptions can be taken care of by memorizing string rules. The experimental results show that the approach performs well on a wide range of semistructured Web pages.

A limitation of our approach is that it is not able to generalize over unseen separators. As a result, the number of training instances required to generate a correct wrapper may depend on how we select the instances. The number of training instances can be minimized by selecting representative instances for each new pair of adjacent attributes. On the other hand, if all of our selected instances contain the same set of separators, we will need many error recovery steps for unseen separators. To make the performance more reliable, we can improve the quality of training instances by using an interactive GUI to guide the labeling or using the knowledge of the ranges of attribute values to analyze the page and recognize possible new separators in advance.

We plan to perform theoretical analysis to reveal more insights on the behavior of our wrapper. The future work also includes applying a number of algorithms for removing ambiguity, determinization, and minimization [18] to optimize the learned wrapper. We are currently working on including negative examples in the induction of contextual rules to improve the accuracy. It is also important to make the wrappers robust against changes so that they can skip errors and return the most reasonable tuples.

Acknowledgements — This work was partly performed while the first author worked as an assistant professor at Arizona State University. This paper was extended from [10]. We wish to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] Defense advanced research projects agency. *Proceedings of the Sixth Message Understanding Conference (MUC-6)*. Morgan Kaufmann Publisher Inc. (1995).

[2] Y. Arens, C.Y. Chee, C.-N. Hsu, and C.A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, **2**(2):127–159 (1993).

[3] Y. Arens, C.A. Knoblock, and C.-N. Hsu. Query processing in the SIMS information mediator. In Austin Tate, editor, *Advanced Planning Technology*. The AAAI Press, Menlo Park, CA (1996).

[4] N. Ashish and C.A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *Proceedings of the International Conference on Cooperative Information Systems (Coopis-97)*, Charleston, South Carolina (1997).

[5] P. Atzeni and G. Mecca. Cut and paste. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems (PODS-97)*, pp. 114–153, Tucson, Arizona (1997).

[6] P. Buneman. Semistructured data. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems (PODS-97)*, pp. 117–121, Tucson, Arizona (1997).

[7] R.B. Doorenbos, O. Etzioni, and D.S. Weld. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the first international conference on Autonomous Agents*, pp. 39–48, ACM Press, New York, NY (1997).

[8] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the Web. In *Proceedings of the Workshop on Management of Semi-Structured Data*, Tucson, Arizona (1997).

[9] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Information translation, mediation, and mosaic-based browsing in the TSIMMIS system. In *Proceedings of the ACM SIGMOD Internation Conference on Management of Data*, page 483, San Jose, CA (1995).

[10] C.-N. Hsu and M.-T. Dung. Wrapping semistructured web pages with finite-state transducers. In *Working Note of the Conference on Automated Learning and Discovery, Workshop on Learning from Text and the Web*, Center for Automated Learning and Discovery, Carnegie Mellon University, Pittsburg, PA (1998).

[11] J.Y.-J. Hsu and W.-T. Yih. Template-based information mining from html documents. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 256–262, AAAI Press, Menlo Park, CA (1997).

[12] T. Kirk, A.Y. Levy, Y. Sagiv, and D. Srivastava. The information manifold. In *Working Notes of the AAAI Spring Symposium on Information Gathering in Heterogeneous, Distributed Environments,Technical Report SS-95-08*, AAAI Press, Menlo Park, CA (1995).

[13] C.A. Knoblock, Y. Arens, and C.-N. Hsu. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Intelligent and Cooperative Information Systems (Coopis-94)*, pp. 122–133, Toronto, Ontario, Canada (1994).

[14] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, Department of Computer Science and Engineering, University of Washington, Seatle, WA (1997).

[15] C.T. Kwok and D.S. Weld. Planning to gather information. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, AAAI Press, Menlo Park, CA (1996).

[16] R.S. Michalski. A theory and methodology of inductive learning. In *Machine Learning: An Artificial Intelligence Approach*, volume I, pp. 83–134 Morgan Kaufmann Publishers Inc., Los Altos, CA (1983).

[17] I. Muslea, S. Minton, and C.A. Knoblock. STALKER: Learning extraction rules for semistructured, Web-based information sources. In *Proceedings of AAAI-98 Workshop on AI and Information Integration, Technical Report WS-98-01*, AAAI Press, Menlo Park, CA (1998).

[18] E. Roche and Y. Schabes, editors. *Finite-State Language Processing*. MIT Press, Cambridge, MA (1997).

[19] D. Smith and M. Lopez. Information extracting for semistructured documents. In *Proceedings of the Workshop on Management of Semi-Structured Data*, Tucson, AZ (1997).

[20] The World-Wide Web Consortium (W3C). Extensible markup language (XML). `http://www.w3.org/XML/`, 1997.

[21] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, **25**(3):38–49 (1992).