Initial Results on Wrapping Semistructured Web Pages with Finite-State Transducers and Contextual Rules

Chun-Nan Hsu*
Department of Computer Science and Engineering
Arizona State University, P.O. Box 875406
Tempe, AZ 85287-5406, USA
chunnan@isi.edu, http://www.isi.edu/sims/chunnan

Abstract

This paper presents SoftMealy, a novel Web wrapper representation formalism. This representation is based on a finite-state transducer (FST) and contextual rules, which allow a wrapper to wrap semistructured Web pages containing missing attributes, multiple attribute values, variant attribute permutations, exceptions and typos, the features that no previous work can handle. A SoftMealy wrapper can be learned from labeled example items using a simple induction algorithm. Learnability analysis shows that SoftMealy scales well with the number of attributes and the number of different attribute permutations. Experimental results show that the learning algorithm can learn correct wrappers for a wide range of Web pages with a handful of examples and generalize well over unseen pages and structural patterns.

Introduction

Information integration systems (see e.g., (Kirk *et al.* 1995; Arens, Knoblock, & Hsu 1996)) rely on *wrappers* to retrieve data on the World-Wide Web. The primary task of such a wrapper is to extract the data items listed in a given set of Web pages and return the results as data tuples. For example, consider the fragment of the Caltech CS department faculty page[1] in Figure 1(a), where we have five data items. Each item provides information about a faculty member as a sequence of attributes. In this case the attributes are URL `U`, name `N`, academic title `A` and administrative title `M`. A wrapper for this Web page is supposed to take its HTML source as input (see Figure 1(b)), extract the attributes from each item and return a set of faculty tuples `(U,N,A,M)`.

Since constructing a wrapper by hand is impractical, researchers have developed many approaches to

- Mani Chandy, *Professor of Computer Science* and *Executive Officer for Computer Science*
- Jim Avro, *Associate Professor of Computer Science*
- David E. Breen, *Assistant Director of Computer Graphics Laboratory*
- John Tanner, *Visiting Associate of Computer Science*
- Fred Thompson, *Professor Emeritus of Applied Philosophy and Computer Science*

(a)

```
<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>
```

(b)

Figure 1: (a) fragment of Caltech CS faculty Web page and (b) its HTML source (as for November, 1997)

rapid wrapper construction (e.g., (Doorenbos, Etzioni, & Weld 1997; Ashish & Knoblock 1997; Kushmerick 1997)). Essentially, these wrappers 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 1997) 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 an item 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, the 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.

*New address: chunnan@iis.sinica.edu.tw, Institute of Information Sciences, Academia Sinica, Taipei, Taiwan. Ph: +886-2-27.88.37.99. The author wishes to thank Ming-Tzung Dung for his help on coding the prototype system.

[1] `www.cs.caltech.edu/csstuff/faculty.html`

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 items may appear after the administrative title in others)
- **Exceptions and typos**

According to (Kushmerick 1997), 30% of the searchable Web sites 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 (Buneman 1997) 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 items:
`(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 an item 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 items are the same "`</A>, <I>`," and "`<I>`⇓`<LI>` "[2], 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. Unless the wrapper takes the context of the margin between two attributes into account, it is very difficult to wrap these pages.

This paper presents a novel wrapper representation called SoftMealy[3]. The reminder of the paper shows how SoftMealy solves all the problems.

Wrapper Representation

The SoftMealy representation 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 describes the context delimiting two adjacent attributes.

Tokens, Separators and Contextual Rules

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" → `0Alph(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)`

A *separator* is an invisible borderline between two adjacent tokens. A separator s is described by its context tokens, which consists 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 items 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 item in Figure 1(b). The separator s between "`<I>`" and "Professor" is described by

[2]⇓ denotes a newline, following (Kushmerick 1997)

[3]It is named after G. H. Mealy who originated finite-state transducers in 1955.

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 item. The class of separators S at the beginning of A for all items is characterized as 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. □

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., (Roche & Schabes 1997)) 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 item type. If there are many types of items in a page, as many FSTs can be constructed. The substring that contains a type of items in a Web page is called the body or the scope of that type of items. Figure 2 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. The tuple transducer in the middle iteratively extracts the tuples from the string between h^L and t^R. In each iteration, the tuple transducer accepts an item and returns the extracted tuple of attributes.

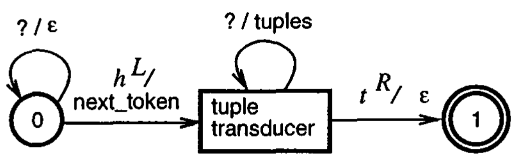


Figure 2: Body transducer

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 an item, 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 item in Figure 1(b) because it is between N ="Mani Chandy" and the next attribute "Professor..."
- R is the set of contextual rules. Each contextual rule, denoted as $s\langle i, j\rangle$, describes 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. An edge from 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 recognizes t^R then there is no more items and the extraction will terminate, otherwise, state e will become state b for the next tuple. Figure 3 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 rule. More precisely, the policy prefers rules that are longer or with

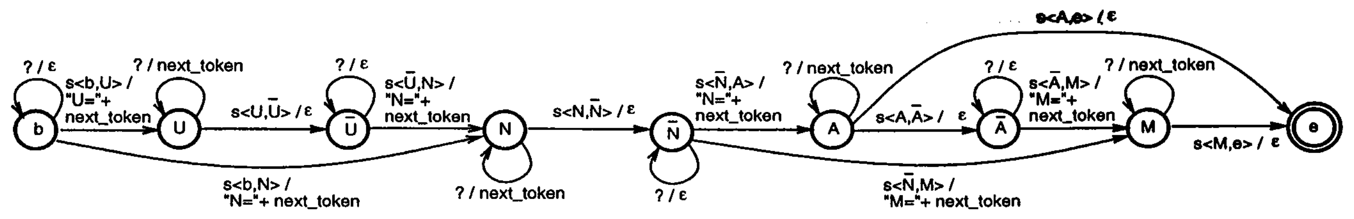

? / ε
s<b,U> / "U="+ next_token
? / next_token
s<U,Ū> / ε
? / ε
s<Ū,N> / "N="+ next_token
s<N,N̄> / ε
s<N̄,A> / "N="+ next_token
? / next_token
s<A,e> / ε
? / ε
s<A,Ā> / ε
s<Ā,M> / "M="+ next_token
? / next_token
s<M,e> / ε
s<b,N> / "N="+ next_token
? / next_token
? / ε
s<N̄,M> / "M="+ next_token
b
U
Ū
N
N̄
A
Ā
M
e

Figure 3: Tuple transducer

more ground tokens. This heuristic worked well in the experiments but I 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.

Expressiveness Analysis

The SoftMealy representation provides sufficient flexibility to wrap semistructured Web pages because it allows multiple outgoing inter-state edges for each state and the use of contextual rules. As a result, we can conclude the expressiveness of SoftMealy as follows:

1. *SoftMealy can deal with missing attributes, multiple attribute values, and variant attribute permutations*: For missing attributes, an edge bypassing the state corresponding to a missing attribute solves the problem. For example, the edge from b to N in Figure 3 allows the wrapper to extract the last two items in Figure 1 where U is missing. Similarly, by properly arranging the edges, the wrapper can deal with any attribute permutations.

2. *SoftMealy can deal with exceptions and typos* with its disjunctive contextual rules.

3. *SoftMealy subsumes the wrapper classes in (Kushmerick 1997)*: His wrappers are special cases of SoftMealy FSTs that are linear, that is, each state has at most one incoming inter-state edge and one outgoing inter-state edge. Delimiters are also special cases of contextual rules where one side of the context is empty. This follows that SoftMealy can wrap any pages his wrappers can wrap.

4. *SoftMealy can wrap nested sources*: (Kushmerick 1997) discusses a class of nested sources that generate Web pages appearing like a "table of contents." Figure 4 shows that SoftMealy can wrap that class of Web pages.

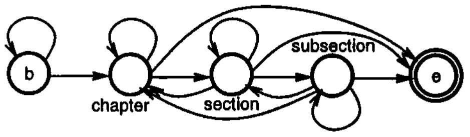


Figure 4: FST for nested Web sources

Induction Algorithm

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 present an algorithm that generalizes contextual rules from labeled items in a set of sample pages.

The labeled items provides the positions of the separators and the attribute permutations (including dummy attributes) of the items. The first step of learning is to construct corresponding states and edges according to the attribute permutations appearing in the training items. 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`, `0Alph` 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 item in Figure 1. Labeling this item 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 $s\langle A, \bar{A}\rangle$ 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>) OAlph(and)` □

With the training set for each separator class, we can apply an induction algorithm (e.g., (Michalski 1983)) to produce contextual rules. Due to the bias, the length of each training instance may be different and we need to decide how to align the tokens into columns for the generalization. Our solution is to align word tokens together and align nonword tokens to the right for left context and to the left for right context.

Example 3 Suppose we label items number 1, 2, 4 and 5 in Figure 1 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.$s\langle \bar{N}, A\rangle^L$ ::=	`Html(</A>)`	`Punc(,)`	`Spc(1)`	`Html(<I>)`
2.$s\langle \bar{N}, A\rangle^L$ ::=	`Html(</A>)`	`Punc(,)`	`Spc(1)`	`Html(<I>)`
3.$s\langle \bar{N}, A\rangle^L$ ::=	`Punc(,)`	`NL(1)`	`Spc(5)`	`Html(<I>)`
4.$s\langle \bar{N}, A\rangle^L$ ::=		`Punc(,)`	`Spc(1)`	`Html(<I>)`

1.$s\langle \bar{N}, A\rangle^R$ ::= `C1Alph(Associate)`
2.$s\langle \bar{N}, A\rangle^R$ ::= `C1Alph(Professor)`
3.$s\langle \bar{N}, A\rangle^R$ ::= `C1Alph(Visiting)`
4.$s\langle \bar{N}, A\rangle^R$ ::= `C1Alph(Gordon)`

□

The generalization algorithm induces contextual rules by taxonomy tree climbing (Michalski 1983). 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. 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`). After the generalization, duplicate instances will be removed and the remaining instances constitute the output contextual rules. The 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 $\exists$ $t'(v')$ $\in$ $c$ such that
     $t$ and $t'$ in the same taxonomy tree THEN
5     replace $t(v)$ w/ 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:

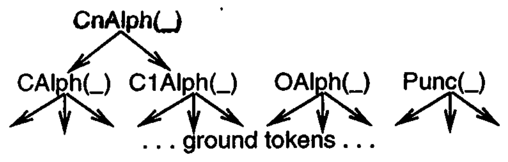


Figure 5: Token Taxonomy Trees

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

Learnability Analysis

This section presents my initial analysis on the learnability of the SoftMealy representation. See (Hsu 1998) for proof of the results and detailed discussion.

This analysis focuses on the sample complexity. In the case of wrapper induction, that amounts to answer how many training items are required to learn an approximately correct wrapper with high confidence. To simplify the analysis, I decompose the problem into two parts: the number of training items required to learn a correct FST graph structure, and the number of items required to learn a correct contextual rule for a separator class.

A FST with a correct graph structure contains necessary edges to cover all possible attribute permutations. Based on the PAC learning theory, the learner needs m training items such that:

$$m \geq \frac{1}{\epsilon}((2K^2 + 2K)\ln 2 + \ln(1/\delta))$$

This is derived from the fact that $2K^2 + 2K$ is the maximal number of edges in a FST for Web sites with K attributes.

An alternative representation to wrap Web pages with variant attribute permutations is to cover each different attribute permutation with a linear FST. This approach requires $m \geq (K^K \ln 2 + \ln(1/\delta))/\epsilon$. This shows the benefit of the compactness of SoftMealy where one edge can be a part of many successful paths. This property allows our learner to cover unseen attribute permutation with sufficient edges. For example, the FST in Figure 3 is induced to cover four different attribute permutations of the example page, but there are six successful paths in that FST. That is, the FST generalizes to two unseen attribute permutations.

The PAC model provides a loose bound on the sample complexity. Since in the worst case, the learner needs to select at least an instance of each attribute permutation to learn the correct graph structure, we can derive a tighter bound by estimating the minimal number of training items required to select at least an instance for each attribute permutation with a probability greater than 0.95. Assuming that the universe of the items is sufficiently large so that this probability follows a multinomial distribution[4], then the upper bound of the training items is the minimal m that makes the probability greater than 0.95:

$$ub(\alpha, p_1, \ldots, p_\alpha) = \min\{m | \sum_{allM_J} m! \prod_{j=1}^{\alpha} \frac{p_j^{m_j}}{m_j!} \geq 0.95\}$$

where α is the number of different attribute permutations, p_j is the proportion of items with attribute permutation j such that $\sum_{j=1}^{\alpha} p_j = 1$, and M_J is a nonzero α partition of m, that is, a set of α positive integers $\{m_1 \ldots m_\alpha\}$ that makes $\sum_{j=1}^{\alpha} m_j = m$.

The upper bound m is a function of α and the distribution $p_1 \ldots p_\alpha$. In general, for a given α, m is small when the distribution is uniform and grows quickly as the distribution becomes skewed. Figure 6(a) illustrates the m surface for $\alpha = 3$. When the distribution is uniform, m grows approximately linearly as α increases, as shown in Figure 6(b).

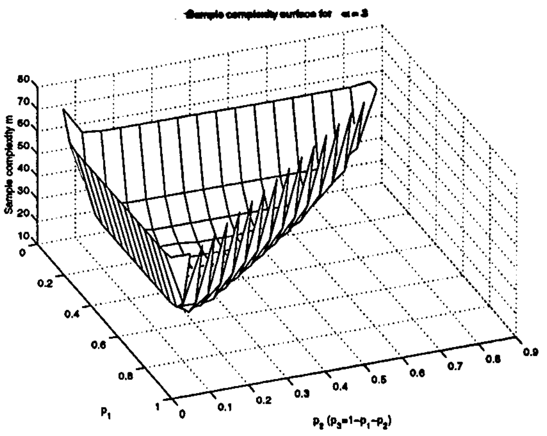
We have discussed the sample complexity to learn a correct graph structure for a FST. A correct wrapper also needs correct contextual rules. Contextual rules are one form of k-DNF formula, from (Haussler 1988), this is polynomially learnable. (Hsu 1998) provides detailed discussion as well as the time complexity of the learning algorithm.

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 items in the Web page with a mouse. The system learns a wrapper from labeled items. If an error appears in the output, the user can provide correct labeling and invoke an error recovery function (Hsu & Dung 1998) 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.

[4]If the size of the universe is small, the probability should follow a hypergeometric distribution.



(a)

$p_1 = \ldots = p_\alpha = 1/\alpha$

$\alpha =$	1	2	3	4	5	6	7	8	9	10
$m =$	1	6	11	16	21	27	33	39	44	51

(b)

Figure 6: Growth of the sample complexity

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[5] 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 items 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 items `IT` in a page. Ideally, `R` should be independent of `IT` but correlated with `SP`. Figure 7(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

[5]The experments were conducted in November, 1997.

URL	IT	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: IT= # of items, 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

correlation -0.031 (≈ 0) between R and IT.

The number of training instances TI is also correlated with SP, with correlation coefficient equal to 0.773 (see Figure 7(c)). This shows that our learning algorithm can learn correct contextual rules for each class of separators with a handful of training items. The outlier at the upper right corner is obtained from the faculty page of the Drexel University[6]. This page is nested and particularly difficult to wrap.

Generalizing over Unseen Pages

We apply SoftMealy to wrap the ASU Web directory site[7]. This Web site provides the information of the faculty, staff and students at ASU with a total of 12 possible attributes. This site is particularly interesting because its output pages contain a large number of different attribute permutations in order to display different personnel categories at a university.

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 items) and selected the largest one as the test page and the others as the training pages. The test page contains 69 items and 17 different attribute permutations while the training pages contain a total of 85 items 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 items in order to successfully wrap the 10 training pages and obtained 92 contextual rule disjuncts. The result is shown in Figure 8, 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 items ($\approx 87\%$) in the test page. We note that because an error may affect

[6]cbis.ece.drexel.edu/ECE/ece_bios.html

[7]www.asu.edu/asuweb/directory/

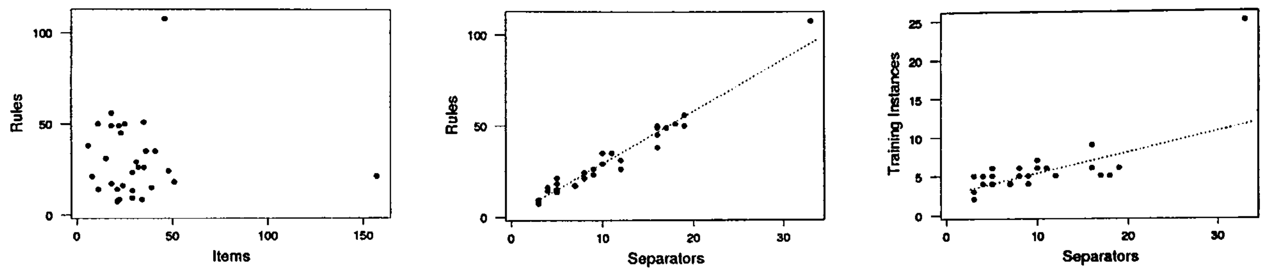


Figure 7: Scatterplots of (a) rules and items, (b) rules and separators, and (c) training instances and separators

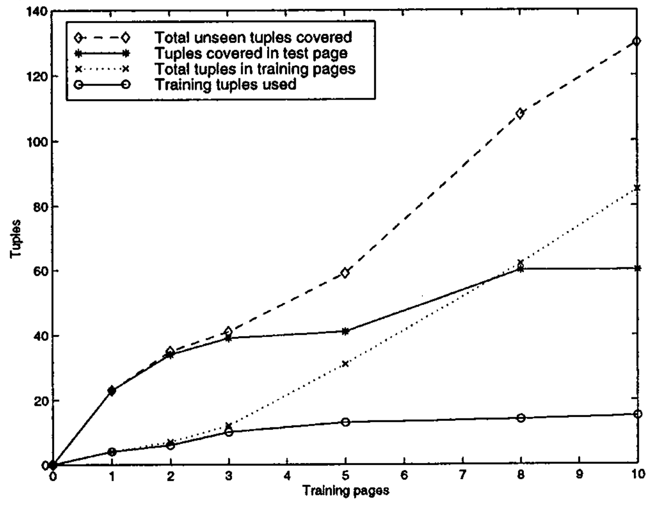


Figure 8: Learning curve on wrapping ASU directory

the extraction of the rest of the page, four items not covered in the test page should have been covered if their previous items had been extracted correctly.

Conclusions and Future Work

This paper presented the SoftMealy wrapper representation for semistructured Web pages. The key features of SoftMealy are as follows. First, the FST allows a wrapper to encode different attribute permutations. Second, the disjunctive contextual rule allows a wrapper to characterize different attribute transition separators. The learning algorithm can bring to bear the token taxonomy trees and induce accurate contextual rules with a handful of examples. The experimental results show that the prototype system performs well on a wide range of Web pages.

The future work includes applying a number of algorithms for removing ambiguity, determinization, and minimization (Roche & Schabes 1997) to optimize learned FSTs. 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 in the Web pages so that they can skip errors and return the most reasonable set of tuples.

References

Arens, Y.; Knoblock, C. A.; and Hsu, C.-N. 1996. Query processing in the SIMS information mediator. In Tate, A., ed., *Advanced Planning Technology*. Menlo Park, CA: The AAAI Press.

Ashish, N., and Knoblock, C. A. 1997. Semi-automatic wrapper generation for internet information sources. In *Proceedings of Coopis-97*.

Buneman, P. 1997. Semistructured data. In *Proceedings of PODS-97*.

Doorenbos, R. B.; Etzioni, O.; and Weld, D. S. 1997. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of AA-97*, 39–48. New York, NY: ACM Press.

Haussler, D. 1988. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence* 36:177–221.

Hsu, C.-N., and Dung, M.-T. 1998. Wrapping semistructured web pages with finite-state transducers. In *Working Notes of CONALD Workshop 6* Pittsburg, PA: Center for Automated Learning and Discovery, Carnegie Mellon University.

Hsu, C.-N. 1998. Wrapper induction: An alternative view. In preparation.

Kirk, T.; Levy, A. Y.; Sagiv, Y.; and Srivastava, D. 1995. The Information Manifold. In *Working Notes of the AAAI Spring Symposium Technical Report SS-95-08*. Menlo Park, CA: AAAI Press.

Kushmerick, N. 1997. *Wrapper Induction for Information Extraction*. Ph.D. Dissertation, Department of Computer Science and Engineering, University of Washington, Seatle, WA.

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

Roche, E., and Schabes, Y., eds. 1997. *Finite-State Language Processing*. Cambridge, MA: MIT press.