

On Precision and Recall of Multi-Attribute Data Extraction from Semistructured Sources

Guizhen Yang Saikat Mukherjee I. V. Ramakrishnan

Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794, U.S.A.
{guizyang,saikat,ram}@cs.sunysb.edu

Abstract

Machine learning techniques for data extraction from semistructured sources exhibit different precision and recall characteristics. However to date the *formal* relationship between learning algorithms and their impact on these two metrics remains unexplored. This paper proposes a formalization of precision and recall of extraction and investigates the complexity-theoretic aspects of learning algorithms for multi-attribute data extraction based on this formalism. We show that there is a tradeoff between precision/recall of extraction and computational efficiency and present experimental results to demonstrate the practical utility of these concepts in designing scalable data extraction algorithms for improving recall without compromising on precision.

1 Introduction

Numerous Web data sources, as illustrated in Figure 1, present organized information about entities and their attributes. For instance, each veterinarian service provider in Figure 1 corresponds to an entity whose attributes include the name of the service provider (*e.g.*, “ABC Animal Hospital”), the resident veterinarian who provides the service (*e.g.*, “John, DVM”), the address and phone number of the service provider. Usually Web pages containing this kind of information have a *consistent* presentation style for each entity.

A common approach for extracting data from Web sources is to build wrappers [28, 23, 8, 36, 15, 2, 32, 26, 7]. Among these assorted approaches learning-based extraction techniques [26, 10, 31, 7, 11] are becoming important since they exhibit a high degree of automation and scalability. Extraction based on learning techniques is a two step process. In the first step, called *labeling*, examples of relevant data to be extracted from the source are supplied. This labeling step can be ei-



Figure 1: A List of Services Web Page

ther manual or completely automated. In the latter case an *ontology*, encoding knowledge of the data domain, applies an attribute *identifier function* (such as pattern matching, keyword-based search) to locate an attribute’s occurrences in the data source. In the second step the labeled examples are used to automatically learn an extraction expression for pulling out all the relevant data in the source.

1.1 Issues of Precision and Recall in Learning-Based Data Extraction

Learning methods synthesize extraction expressions by a process of generalization from the labeled examples and so the set of attribute occurrences that they pull out will be a superset of the labeled examples. Hence these methods are said to increase *recall*¹. But a problem here is that the extraction expression may be too general and pull out incorrect attribute occurrences also, thereby losing *precision*².

¹Recall = $\frac{ce}{ce+fe} \times 100\%$, where *ce* is the number of entities extracted correctly and *te* is the number of true entities not extracted.

²Precision = $\frac{ce}{ce+fe} \times 100\%$, where *ce* is the number of entities extracted correctly and *fe* is the number of false entities

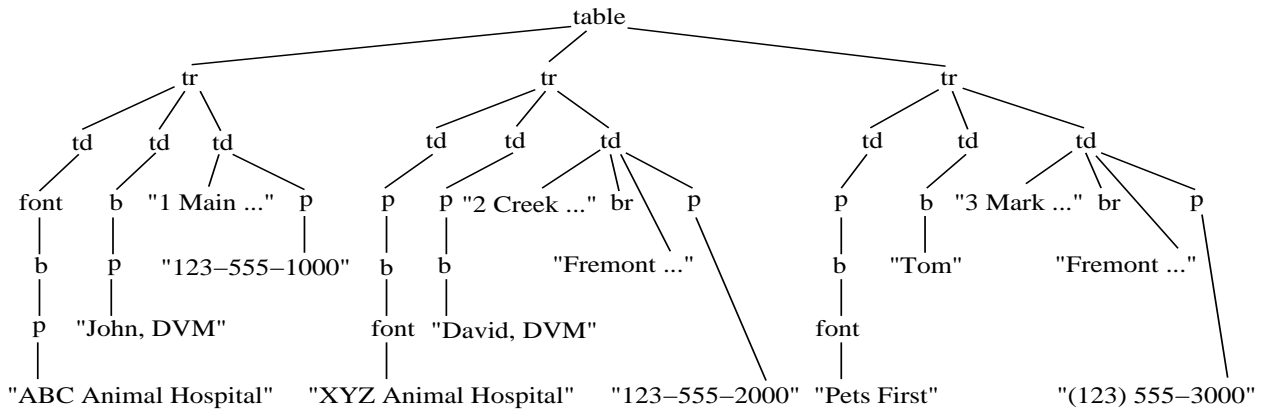


Figure 2: DOM Fragment of Figure 1

For illustration let us consider the following example: Suppose “ABC Animal Hospital” and “XYZ Animal Hospital” are supplied as two examples of the *HospitalName* attribute. In the DOM tree (see Figure 2) corresponding to the Web page in Figure 1, the paths leading to the leaf nodes containing these text strings are $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{font} \cdot \text{b} \cdot \text{p}$ and $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p} \cdot \text{b} \cdot \text{font}$, respectively, where α represents the path string from the root of the document to the *table* tag. We can learn an extraction expression, specifically the regular expression $E_1 = \alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{font}^* \cdot \text{p}^* \cdot \text{b} \cdot \text{p}^* \cdot \text{font}^*$, from these two paths. Observe that if we use this expression E_1 as a path query which is evaluated against the DOM tree, it should return the text string “Pets First”. However, notice that the language³ of E_1 also includes the path string, $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p} \cdot \text{b}$. This path terminates on the leaf node “David, DVM” and hence it will be extracted by E_1 . However, this is an occurrence of a different attribute in the schema, namely the *DoctorName* attribute. By extracting such false positives an extraction expression’s increase in recall can be accompanied by a reduction in precision.

But observe that different extraction expressions can be learned from the same set of examples. For instance, we can learn another expression, $E_2 = \alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p}^* \cdot \text{b}^* \cdot \text{font} \cdot \text{b}^* \cdot \text{p}^*$, from $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{font} \cdot \text{b} \cdot \text{p}$ and $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p} \cdot \text{b} \cdot \text{font}$. The path string $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p} \cdot \text{b}$ is excluded from E_2 ’s language. In fact none of the path strings leading to the *DoctorName* attribute data will be recognized by E_2 . So E_2 retains more precision than E_1 .

The above examples raises several interesting algorithmic questions: *Can we design learning algorithms without losing precision? What is the computational difficulty of such algorithms? Are there tradeoffs between precision/recall and computational efficiency?*

extracted.

³The language of a regular expression is the set of strings recognized by this expression.

As we will discuss later in Section 6, answers to these questions will impact the design of “push-button” technologies for data extraction. To the best of our knowledge a study of the complexity-theoretic aspects of precision and recall of extraction and their impact on the efficiency of learning extractors has remained an unexplored topic in the research literature so far. In this paper we undertake such a study within the context of multi-attribute data extraction.

We introduce a novel formalization of precision and recall of extraction that will serve as the formal model for analyzing an algorithm’s precision/recall characteristics. In our model, extraction expressions are a simple subclass of regular expressions built using the concatenation (“.”) and the Kleene closure (“*”) operators only. We call them *Path Abstraction Expressions* (PAEs)⁴. For instance, the extraction expressions E_1 and E_2 are PAEs. We associate one PAE with each attribute. When a PAE is applied to the DOM tree of a Web page it will match all those path strings that are contained in its language. The leaf nodes with the matched path strings are the attribute occurrences extracted.

To learn a PAE for an attribute we generalize from both its positive and negative examples. The labeling step provides the positive examples for each attribute. The negative examples for an attribute are the positive examples of all the other attributes. Based on the extent to which false positives can be excluded from a PAE’s language, we ascribe *quality* to the PAEs learned. Better quality PAEs eliminate false positives to a larger degree and will therefore possess higher precision. With this brief overview of our model we can now provide a summary of our results as follows.

⁴Although our syntax of PAEs departs slightly from that of union-free regular expressions used for schema learning in [13], our results can be readily extended to handle them. See Appendix B for more details.

1.2 Summary of our Results

1. At the lower end of the quality spectrum sits *nonredundant PAEs*. The language of a nonredundant PAE includes all of its positive examples. Removing any symbol from a nonredundant PAE will result in excluding one or more of the positive examples from its language. We describe polynomial time algorithms for learning nonredundant PAEs.
2. The language of a nonredundant PAE may include negative examples and hence can suffer loss of precision. To improve precision we propose *consistent PAEs*. The language of a consistent PAE includes all the positive examples while excluding all the negative ones. To handle multi-attribute entities we need to learn a set of PAEs — one per attribute. If *every* PAE learned is consistent then we say that this set of PAEs is *unambiguous* w.r.t. the examples. We show, via an intricate reduction (see Appendix A), that the problems of learning a consistent PAE and an unambiguous set of PAEs are NP-complete. We also propose an efficient heuristic for learning a consistent PAE.
3. Note that the above notion of unambiguity is relative to a given set of examples. When a set of PAEs is unambiguous w.r.t. *any* example set we say that it is *inherently ambiguous*. Such a set of PAEs will suffer the least loss of precision in extraction. We show that the problem of learning an inherently unambiguous set of PAEs is decidable.
4. Because learning an unambiguous set of PAEs is computationally difficult, we have to resort to heuristics. However, these heuristics may not guarantee that all of the PAEs learned are consistent. So ambiguities can still occur when using such sets of PAEs for extracting multi-attribute data, resulting in loss of precision. We model ambiguity resolution as an algorithmic problem over bipartite graphs. By combining knowledge about the attribute domains (possibly encoded in an ontology) with this algorithm we resolve the ambiguities as much as possible thereby improving recall without much loss in precision.
5. We engineered a learning algorithm based on 1, 2, 3, 4 above to extract multi-attribute data from 194 different Web pages listing veterinarian service providers and product descriptions. The results, obtained from running this algorithm over these pages, indicate that the overall recall achieved ranges from 58% to 100% with almost no loss in precision.

It is noteworthy pointing out that the classical complexity results in grammar inference, particularly

the seminal works of Gold [19, 20] and Angluin [3] are not applicable to our learning problem. This is because of the differences in the learning bias – we learn consistent PAEs whereas the classical works focus on the smallest size. We refer the readers to Section 5 for a more detailed discussion of related work.

We remark that the notion of unambiguity has a direct bearing on the schema learning problem, which has been a topic of recent papers [13, 6]. An elegant learning algorithm for discovering the implicit schema in template-driven Web documents was proposed in [13]. But the algorithm for discovering a “good” schema in this work can suffer from exponential blow-up. We can view the notion of good schema as corresponding to our notion of unambiguity and our results imply that ambiguity resolution is the root cause of the computational difficulty underlying schema discovery. See the NP-completeness proof for learning unambiguous sets of union-free regular expressions in Appendix B.

The rest of the paper is organized as follows. Section 2 formalizes the notions of PAEs, nonredundant and consistent PAEs, a set of PAEs that is unambiguous w.r.t. an example set, and inherently unambiguous PAEs. Complexity results on learning these classes of PAEs are presented in Section 3. Heuristics for learning unambiguous PAEs and the algorithm for resolving ambiguity are also presented in this section. Experimental results appears in Section 4 and related work in Section 5. Finally, Section 6 concludes this paper.

2 Problem Formalization

We use $|S|$ and $|\alpha|$ to denote the cardinality of a set S and the length of a string α , respectively. A *subsequence* of a given string is obtained by deleting zero or more symbols from this string. The longest common subsequence (LCS) of a set of strings is a subsequence that is common to all of the strings and is the longest such subsequence. A string β is a *supersequence* of another string α if and only if α is a subsequence of β . The shortest common supersequence (SCS) of a set of strings is a supersequence that is common to all of the strings and is the shortest such supersequence. Both the LCS and the SCS of two strings can be computed in quadratic time [12].

A path abstraction expression is just like a regular expression but with two restrictions: (i) it is free of the union operator (“|”); and (ii) the Kleene closure operator (“*”) can only apply to single symbols. Formally, we have the following definition.

Definition 1 (Path Abstraction Expression) *Let Σ be a finite alphabet. A path abstraction expression (abbr. PAE) over Σ is defined inductively as follows:*

(1) Any symbol $c \in \Sigma$ is a PAE; (2) For any $c \in \Sigma$, c^* is a PAE; (3) If E_1 and E_2 are PAEs, so is $E_1 \cdot E_2$.

For instance, $a \cdot b^* \cdot c$ is a PAE whereas neither $a \cdot (b|c)$ nor $a \cdot (b \cdot c)^*$ is a PAE. By disallowing the union operator (“|”) in the syntax of PAEs, we can force *generalization* in the learning algorithms. Otherwise, one can just compose a regular expression by concatenating all the input strings using the union operator. Such techniques do not really capture any regularity in the paths within a DOM tree.

Although we require that the Kleene closure operator (“*”) be applied to single symbols only, this does not really impose any extra technical difficulty. We make this simplification just for the Web domain, since in reality it is very rare that a consecutive sequence of tags would repeat itself in the root-to-leaf paths of a DOM tree.

Note that $a \cdot * \cdot c$ is not a PAE either, although it is a valid XPath [1] query. In the XPath syntax “*” actually stands for the entire alphabet Σ . Because we forbid the union operator in PAEs, therefore we do not allow XPath’s Σ syntax either. However, a query referring to Σ can be easily simulated. For instance, let $\Sigma = \{a, b\}$. Then the XPath query, $a \cdot * \cdot b$, can be simulated using the PAE $a \cdot a^* \cdot b^* \cdot b$.

Following the standard convention, we will usually omit the concatenation operator (“.”) in a PAE. Given a PAE E , the set of strings recognized by E is denoted as $\mathcal{L}(E)$.

Definition 2 (Cover) *Let S be a set of strings and E be a PAE. We say that E covers S , or E is a cover of S , iff $\mathcal{L}(E) \supseteq S$. Similarly, let $\{E_1, \dots, E_n\}$ be a set of PAEs and $\{S_1, \dots, S_n\}$ be a set of sets of strings. We say that $\{E_1, \dots, E_n\}$ covers $\{S_1, \dots, S_n\}$, iff E_i covers S_i for all $1 \leq i \leq n$.*

For instance, ab^*c covers $\{ac, abbc\}$ whereas ab^*c does not cover $\{aac, abbc\}$, since $aac \notin \mathcal{L}(ab^*c)$. $\{ab^*c, aa^*b^*c\}$ covers $\{\{ac, abbc\}, \{aac, abbc\}\}$ whereas $\{aa^*b^*c, ab^*c\}$ does not cover $\{\{ac, abbc\}, \{aac, abbc\}\}$, since $aac \notin \mathcal{L}(ab^*c)$.

Definition 3 (Nonredundancy) *Let S be a set of strings and E be a PAE which covers S . We say that E is nonredundant w.r.t. S , iff we cannot perform either of the following operations on E to obtain a new PAE E' that still covers S : (1) Remove any symbol together with its Kleene closure operator (“*”), e.g., c^* ; (2) Remove a Kleene closure operator (“*”) from a symbol only.*

Given a set of strings S , our goal is to learn a PAE E that covers S . Intuitively, E represents a generalization of all the strings in S . However, if E overgeneral-

izes then it will produce more false positives when we later launch E as a query against the DOM tree. Note that if we can perform either of the two operations in Definition 3 on E and obtain a new E' that still covers S , then $\mathcal{L}(E') \subset \mathcal{L}(E)$. Thus E' should produce less false positives in general. In other words, E' retains more precision than E and so has better quality. We require that one should learn a nonredundant PAE to generalize a set of path strings.

For instance, let $S = \{ab, bc\}$. Then $a^*b^*c^*$ is redundant w.r.t. S , since if we remove the Kleene closure operator from b , then a^*bc^* still covers S . Clearly, a^*bc^* is nonredundant w.r.t. S .

Notice that nonredundant PAEs do not say anything about negative examples. To deal with negative examples we introduce the following definition.

Definition 4 (Consistency) *Let E be a PAE, and POS and NEG be two sets of strings. We say that E is consistent w.r.t. $\langle POS, NEG \rangle$, iff $\mathcal{L}(E) \supseteq POS$ and $\mathcal{L}(E) \cap NEG = \emptyset$.*

In the above Definition 4, the strings in POS serve as positive examples while those in NEG serve as negative examples. Intuitively, a consistent PAE generalizes all positive examples but excludes all negative examples. Therefore, extraction using consistent PAEs will have higher precision than nonredundant PAEs. For instance, aa^*b^* is consistent w.r.t. $\langle \{aa, aab\}, \{ab, cd\} \rangle$ whereas a^*b^* is not, since the negative example $ab \in \mathcal{L}(a^*b^*)$.

Given a pair of sets of positive and negative examples, is there *always* a PAE that is consistent w.r.t. these examples? The answer turns out to be “No”. For instance, it can be shown that there is no PAE that is consistent w.r.t. $\langle \{ab, cd\}, \{aa, aab\} \rangle$.

In practice we often need to extract multiple attributes of an entity. Usually the identifier functions in an ontology for these attributes are able to identify several (albeit incomplete) examples for each attribute. Our goal is to learn a set of PAEs, one for *each* attribute. Note that for any given attribute, the positive examples from other attributes will serve as negative examples for this attribute. If for *each* attribute, we can learn a consistent PAE that covers the positive examples of this attribute but excludes all the positive examples of other attributes combined, then we say that this set of PAEs is unambiguous w.r.t. the given set of sets of examples. Formally, we have the following definition.

Definition 5 (Unambiguity) *Given a set of sets of strings, $\{S_1, \dots, S_n\}$, and a set of PAEs, $\{E_1, \dots, E_n\}$, we say that $\{E_1, \dots, E_n\}$ is unambiguous w.r.t. $\{S_1, \dots, S_n\}$, iff E_i is consistent w.r.t. $\langle S_i, \bigcup_{j \neq i} S_j \rangle$*

for all $1 \leq i \leq n$.

However, even when a set of PAEs is unambiguous w.r.t. the examples, the languages recognized by these PAEs may still overlap. When some or all of these languages overlap, ambiguity may arise since they may identify the same path string in a DOM tree. Therefore, a more desirable, better quality that we want to impose is that these languages be pairwise disjoint. If so, then we say the set of PAEs is *inherently* unambiguous. Clearly, inherently unambiguous PAEs are able to retain more precision than those that are only unambiguous w.r.t. the given examples. This idea is formalized in the following definition.

Definition 6 (Inherent Unambiguity) *Let $\{S_1, \dots, S_n\}$ be a set of sets of strings and $\{E_1, \dots, E_n\}$ be a set of PAEs. We say that $\{E_1, \dots, E_n\}$ is inherently unambiguous w.r.t. $\{S_1, \dots, S_n\}$, iff $\{E_1, \dots, E_n\}$ covers $\{S_1, \dots, S_n\}$ and $\mathcal{L}(E_i) \cap \mathcal{L}(E_j) = \emptyset$ for all $1 \leq i \leq n$, $1 \leq j \leq n$, and $i \neq j$.*

For example, $\{ab^*c, abc^*\}$ is unambiguous w.r.t. the examples $\{\{ac, abbc\}, \{ab, abcc\}\}$, but not inherently unambiguous, because $abc \in \mathcal{L}(ab^*c)$ and $abc \in \mathcal{L}(abc^*)$. As another example, $\{ab^*c, abc^*d\}$ is inherently unambiguous w.r.t. $\{\{ac, abbc\}, \{abd, abccd\}\}$.

Given a pair of sets of examples, is there *always* a pair of PAEs that is inherently unambiguous w.r.t. these examples? The answer also turns out to be “No”. For instance, we have just shown that $\{ab^*c, abc^*\}$ is unambiguous w.r.t. $\{\{ac, abbc\}, \{ab, abcc\}\}$. It can also be shown that there is no pair of PAEs that is inherently unambiguous w.r.t. $\{\{ac, abbc\}, \{ab, abcc\}\}$.

We formally state our learning problems as follows.

Problem 1 (Consistent PAE) *Given two sets of strings POS and NEG, is there a PAE that is consistent w.r.t. $\langle POS, NEG \rangle$?*

Problem 2 (Unambiguous PAEs) *Given a set of sets of strings, $\{S_1, \dots, S_n\}$, is there a set of PAEs, $\{E_1, \dots, E_n\}$, such that $\{E_1, \dots, E_n\}$ is unambiguous w.r.t. $\{S_1, \dots, S_n\}$?*

Problem 3 (Inherently Unambiguous PAEs) *Given a set of sets of strings, $\{S_1, \dots, S_n\}$, is there a set of PAEs, $\{E_1, \dots, E_n\}$, such that $\{E_1, \dots, E_n\}$ is inherently unambiguous w.r.t. $\{S_1, \dots, S_n\}$?*

We should point out that the above three problems are not equivalent problems. Let $\langle S_1, S_2 \rangle$ be a pair of sets of strings. The existence of a PAE that is consistent w.r.t. $\langle S_1, S_2 \rangle$ does not necessarily imply that there is a pair of PAEs that is unambiguous w.r.t. $\{S_1, S_2\}$.

Similarly, the existence of a pair of PAEs that is unambiguous w.r.t. $\{S_1, S_2\}$ does not necessarily imply that there is a pair of PAEs that is inherently unambiguous w.r.t. $\{S_1, S_2\}$.

For instance, aab^* is consistent w.r.t. $\{\{aa, aab\}, \{ab, cd\}\}$. But there is no pair of PAEs that is unambiguous w.r.t. $\{\{aa, aab\}, \{ab, cd\}\}$. Similarly, $\{ab^*c, abc^*\}$ is unambiguous w.r.t. $\{\{ac, abbc\}, \{ab, abcc\}\}$. But there is no pair of PAEs that is inherently unambiguous w.r.t. $\{\{ac, abbc\}, \{ab, abcc\}\}$.

3 Computational Aspects of Learning PAEs

3.1 Learning Nonredundant PAEs

The algorithm *LearnPAE* takes as input a set of positive examples of an attribute (S^+) and returns as output a nonredundant PAE (E) which covers this set of positive examples.

Algorithm LearnPAE(S^+)
input S^+ : a nonempty set of strings
output E : a nonredundant PAE which covers S^+
begin
 1. $n = |S^+|$
 2. Let α_i ($1 \leq i \leq n$) be a string in S^+ .
 3. $E = \alpha_1$
 4. **for** $2 \leq i \leq n$ **do**
 5. $E = SCS(E, \alpha_i)$
 6. **endfor**
 7. Put a * on all the symbols of E .
 8. $E = \text{MakeNonredundant}(E, S^+)$
 9. **return** E
end

In Line 3, the variable E is initialized with the first positive example. In Lines 4–6, the shortest common supersequence (SCS) of the string stored in E and the next positive example is computed and then assigned to E . Clearly, when the loop in Lines 4–6 terminates, E stores a common supersequence for all the strings in S^+ . In Line 7, the string stored in E is generalized to a PAE that covers S^+ by adding * on all the symbols in it. Intuitively, this operation corresponds to generalization beyond the identified positive examples.

The procedure *MakeNonredundant* takes as input a PAE, E , and a set, S^+ , of positive examples that is covered by E . When the procedure ends, it makes E nonredundant w.r.t. S^+ . First, for every symbol in E which contains a *, it checks whether after the symbol along with the * is dropped from E the resulting PAE still covers S^+ . If it does, the symbol together with the * is dropped from E (Lines 4–7). If not, then it is checked whether the PAE obtained by dropping only

the * on the symbol still covers S^+ . If it does, then the * is dropped from the symbol (Lines 9–10).

Algorithm *MakeNonredundant*(E, S^+)

```

input
   $E$  : a PAE which covers  $S^+$ 
   $S^+$  : a nonempty set of strings
output
   $Q$  : a nonredundant PAE which covers  $S^+$ 
begin
1.  $n =$  the number of symbols in  $E$  excluding *
2. Let  $x_i$  ( $1 \leq i \leq n$ ) be the  $i$ -th symbol in  $E$ .
3. for  $1 \leq i \leq n$  do
4.   if a * is attached to  $x_i$  then
5.      $R =$  drop  $x_i$  together with its * from  $E$ 
6.     if  $R$  covers  $S^+$  then
7.        $E = R$ 
8.     else
9.        $R =$  drop the * that is attached to  $x_i$  from  $E$ 
10.      if  $R$  covers  $S^+$  then  $E = R$  endif
11.    endif
12.  endif
13. endfor
14.  $Q = E$ 
15. return  $Q$ 
end

```

Note that if either of the two operations on Lines 4–7 and Lines 9–10 succeeds, then the language recognized by the new PAE is *strictly* smaller than that of the old PAE. We can show that the procedure *MakeNonredundant* indeed returns a PAE, Q , which is nonredundant w.r.t. S^+ . Moreover, the complexity of the algorithm *LearnPAE* is polynomial time.

For illustration, suppose for the *HospitalName* attribute we are given two examples, “ABC Animal Hospital” and “XYZ Animal Hospital”, in Figure 2. Invoking *LearnPAE* with their path strings, $table \cdot tr \cdot td \cdot font \cdot b \cdot p$ and $table \cdot tr \cdot td \cdot p \cdot b \cdot font$, results in computing $table \cdot tr \cdot td \cdot font \cdot p \cdot b \cdot p \cdot font$ as the SCS on exiting the **for**-loop in Lines 4–6 of the algorithm *LearnPAE*. Then *MakeNonredundant* is invoked in Line 8 with $table \cdot tr \cdot td \cdot font \cdot p \cdot b \cdot p \cdot font$ as its input and the procedure returns $table \cdot tr \cdot td \cdot font \cdot p \cdot b \cdot p \cdot font$ as its output, which is the nonredundant PAE learned for extracting the *HospitalName* attribute. Observe that this PAE will also extract “Pets First” in Figure 2.

3.2 Learning Consistent PAEs

Since the algorithm *LearnPAE* only takes into account positive examples, the PAE that it produces will not be consistent in general. A consistent PAE covers all the positive examples for that attribute but excludes all of its negative examples. However, it turns out that the complexity of learning runs up quickly once we need to consider negative examples.

Theorem 1 *The consistent PAE problem is NP-complete.*

Proof. See Appendix A. \square

The algorithm *ConsistentPAE* is a simple heuristic for computing a PAE that is consistent w.r.t. positive and negative examples of an attribute. The central idea behind this heuristic is to find a *distinguishing subsequence* of symbols which are present in all the positive examples but not present in any of the negative examples for that attribute. Along with the set of positive examples (S^+) and the set of negative examples (S^-) for an attribute, it also takes as its input the maximum possible length (K) of the distinguishing subsequence to be searched.

In Line 1 of the algorithm *ConsistentPAE*, the set F contains all common subsequences of S^+ and the length of any string in F is at most K . For each such string α , it is checked if it is also a subsequence of any string in S^- . If it is not, then α is a distinguishing subsequence (Line 3).

Algorithm *ConsistentPAE*(S^+, S^-, K)

```

input
   $S^+$  : a set of strings which serve as positive examples
   $S^-$  : a set of strings which serve as negative examples
   $K$  : the maximum length of a distinguishing subsequence
output
   $E$  : if  $E \neq \varepsilon$ , then  $E$  is a PAE which is consistent
      with respect to  $\langle S^+, S^- \rangle$ .
begin
1.  $F = \{\alpha \mid \alpha \text{ is a common subsequence of } S^+ \text{ and } |\alpha| \leq K\}$ 
2. for each  $\alpha \in F$  do
3.   if  $\alpha$  is not a subsequence of  $\beta$  for all  $\beta \in S^-$  then
4.      $n = |\alpha|$ 
5.      $\alpha = x_1 x_2 \dots x_n$ 
6.     for  $1 \leq i \leq n + 1$  do  $\gamma_i = \varepsilon$  endfor
7.     for each  $\rho \in S^+$  do
8.        $\rho = \rho_1 \cdot x_1 \cdot \rho_2 \cdot x_2 \dots \rho_n \cdot x_n \cdot \rho_{n+1}$ 
9.       for  $1 \leq i \leq n + 1$  do  $\gamma_i = \gamma_i \cdot \rho_i$  endfor
10.    endfor
11.    Put a * on all symbols in  $\gamma_i$  for all  $1 \leq i \leq n + 1$ .
12.     $E = \gamma_1 \cdot x_1 \cdot \gamma_2 \cdot x_2 \dots \gamma_n \cdot x_n \cdot \gamma_{n+1}$ 
13.     $E =$  MakeNonredundant( $E, S^+$ )
14.    return  $E$ 
15.  endif
16. end
17.  $E = \varepsilon$ 
18. return  $E$ 
end

```

Suppose a distinguishing subsequence α consists of the symbols $x_1 x_2 \dots x_n$ (Line 5). The heuristic constructs a (possibly redundant) consistent PAE of the form, $\gamma_1 \cdot x_1 \cdot \gamma_2 \cdot x_2 \dots \gamma_n \cdot x_n \cdot \gamma_{n+1}$, where each γ_i is a concatenation of all the symbols between x_{i-1} and x_i over all the positive examples in S^+ (Lines 7–10). Note that there is a * on all the symbols in each γ_i (Line 11) but not on any of the symbols in α , the distinguishing sequence. As a result, the PAE generated this way does not accept any string in S^- . Finally, this newly constructed PAE (Line 12) is made nonredundant w.r.t. S^+ by invoking the procedure *MakeNonredundant* (Line 13).

Note that the algorithm *ConsistentPAE* is just a heuristic because there may not exist a distinguishing subsequence of size at most K . In such a case, the procedure would fail and return the empty string (Line 17). It can be shown that the complexity of the algorithm *ConsistentPAE* is polynomial time when K is fixed.

For illustration, we will show how to generate a consistent PAE for the *HospitalName* attribute. We invoke *ConsistentPAE* with the path strings leading to “ABC Animal Hospital” and “XYZ Animal Hospital” as the positive examples. The path strings leading to “John, DVM”, “David, DVM”, “123-555-1000”, and “123-555-2000” serve as the negative examples.

Clearly, the *font* symbol distinguishes the two positive examples from the four negative examples. It corresponds to the distinguishing subsequence $\alpha = \textit{font}$ in the algorithm *ConsistentPAE*. The path string for “ABC Animal Hospital” is represented as $\rho_1 \cdot \textit{font} \cdot \rho_2$, where $\rho_1 = \textit{table} \cdot \textit{tr} \cdot \textit{td}$ and $\rho_2 = \textit{b} \cdot \textit{p}$. Similarly, the path string for “XYZ Animal Hospital” is represented as $\rho_1 \cdot \textit{font} \cdot \rho_2$, where $\rho_1 = \textit{table} \cdot \textit{tr} \cdot \textit{td} \cdot \textit{p} \cdot \textit{b}$ and $\rho_2 = \varepsilon$. Concatenation of the respective ρ_i ’s and adding $*$ on every symbol in them yields the redundant, consistent PAE, $\textit{table}^* \cdot \textit{tr}^* \cdot \textit{td}^* \cdot \textit{table}^* \cdot \textit{tr}^* \cdot \textit{td}^* \cdot \textit{p}^* \cdot \textit{b}^* \cdot \textit{font} \cdot \textit{b}^* \cdot \textit{p}^*$. The nonredundant, consistent PAE that is finally generated is $\textit{table} \cdot \textit{tr} \cdot \textit{td} \cdot \textit{p}^* \cdot \textit{b}^* \cdot \textit{font} \cdot \textit{b}^* \cdot \textit{p}^*$. Note that this PAE does not match any of the negative examples for the *HospitalName* attribute.

3.3 Learning Unambiguous PAEs

Our goal is to extract the data values for a set of attributes associated with a concept. This requires learning a set of PAEs, one per attribute. The positive and negative examples used for learning a set of PAEs are obtained in the same way as for learning consistent PAEs. To extract data values from the source with very high recall and precision, it is desirable that this set of PAEs be unambiguous w.r.t. examples. However, the complexity of this problem turns out to be very high.

Theorem 2 *The unambiguous PAEs problem is NP-complete.*

Proof. See Appendix A. \square

From a computational viewpoint learning a set of PAEs that is unambiguous w.r.t. examples amounts to requiring that each PAE in this set be consistent. Therefore we can use the algorithm *ConsistentPAE* repeatedly, once per attribute, as a heuristic for generating an unambiguous set of PAEs.

It is worthwhile pointing out that the PAEs generated by the algorithm *LearnPAE* can at times be consistent (see Section 4). So before resorting to the al-

gorithm *ConsistentPAE*, we can use *LearnPAE* as our initial heuristic due to its relatively lower complexity. For instance, *LearnPAE* generates a consistent PAE, $\textit{table} \cdot \textit{tr} \cdot \textit{td} \cdot \textit{p}$, from the two examples, “123-555-1000” and “123-555-2000”, in Figure 2 for the *PhoneNumber* attribute. This PAE and the consistent PAE above for *HospitalName* form an unambiguous set of PAEs.

Finally, for an inherently unambiguous set of PAEs we require that it be unambiguous w.r.t. any example set. Such a set has the best recall and precision properties. Although the computational complexity of this problem remains open at this time, it can be shown that it is decidable.

Theorem 3 *The inherently unambiguous PAEs problem is decidable.*

Proof. [Sketch] Clearly, given a set of sets of examples, $\{S_1, \dots, S_n\}$, if there exists a set of PAEs, $\{E_1, \dots, E_n\}$, which is inherently unambiguous w.r.t. $\{S_1, \dots, S_n\}$, then the size of each E_i is bounded by the sum of the lengths of all the strings in S_i . We can construct a naive algorithm to enumerate each E_i and check if the resulting set of PAEs is inherently unambiguous w.r.t. $\{S_1, \dots, S_n\}$. \square

3.4 Resolving Ambiguity

Given that learning a set of PAEs that is unambiguous w.r.t. examples is computationally difficult, one has to resort to heuristics. Since heuristics may not guarantee that all the PAEs learned are consistent, ambiguity can occur when using such a set of PAEs for extracting multi-attribute data. We now describe an algorithm based on bipartite graphs that uses domain knowledge to resolve ambiguity as much as possible thereby improving recall while retaining high precision.

For simplicity we assume that record boundaries are already identified and so PAEs are applied to each entity block in a DOM tree (see Section 4.1 for more details). In addition, for expositional convenience we also assume that the attributes are single-valued. Extending the techniques in this section to multivalued attributes is straightforward.

We say that a PAE *matches* an attribute value whenever it accepts the path string terminating on the leaf node in a DOM tree which is labeled with this value. The ambiguity resolution algorithm takes as input a set of PAEs (E) and a set of data values (D) in an entity block and returns a set of 1–1 associations between attributes and data values. Each data value consists of a text string and the path string in the DOM tree that leads to this text string.

Algorithm BipartiteResolution(E, D)**input**

E : a set of PAEs representing attributes
 D : a set of strings representing data values

output

A : a set of pairs in the form of (attribute,value)

begin

```

1.  $A = \emptyset$ 
2.  $E = \langle E_1, \dots, E_n \rangle$ 
3. Let  $E_i$  ( $1 \leq i \leq n$ ) be the PAE for the attribute  $A_i$ .
4.  $m = |D|$ 
5. Let  $\alpha_j \in D$  ( $1 \leq j \leq m$ ) represent the data value  $D_j$ .
6.  $G = \emptyset$  ( $G$  is the set of edges)
7. for  $1 \leq i \leq n$  do
8.   for  $1 \leq j \leq m$  do
9.     if  $\alpha_j \in \mathcal{L}(E_i)$  then  $G = G \cup \{edge(E_i, \alpha_j)\}$  endif
10.  endifor
11. endifor
12. do
13.    $M = \emptyset$ 
14.   for  $1 \leq i \leq n$  do
15.     if  $degree(E_i) = 1$  ( $edge(E_i, \alpha_k) \in G$  for some  $\alpha_k$ ) then
16.        $X = \{E_j \mid j \neq i, edge(E_j, \alpha_k) \in G, degree(E_j) = 1\}$ 
17.       if  $X = \emptyset$  then  $M = M \cup \{E_i\}$  endif
18.     endif
19.   endifor
20.   for each  $E_i \in M$  do
21.     There must exist only one  $edge(E_i, \alpha_k) \in G$ .
22.      $A = A \cup (A_i, D_k)$ 
23.     Remove all edges in  $G$  that are incident on  $\alpha_k$ .
24.   endifor
25. while  $M \neq \emptyset$ 
26. return  $A$ 
end

```

Our ambiguity resolution algorithm works in two steps. In the first step, domain knowledge is used to resolve ambiguity. If a data value D_j has been identified by the ontology as the value for an attribute A_i , then the pair (A_i, D_j) is added to the set of associations for that record. The data value and the corresponding PAE are deleted from D and E (note that we assume all attributes are single-valued), respectively.

In the second step we derive more 1–1 associations between the remaining unresolved data values and PAEs using the algorithm *BipartiteResolution*. This algorithm first constructs a bipartite graph in which the two disjoint sets of vertices are E and D , respectively, and an edge between $E_i \in E$ and $\alpha_j \in D$ is created if E_i matches α_j (Lines 2–11).

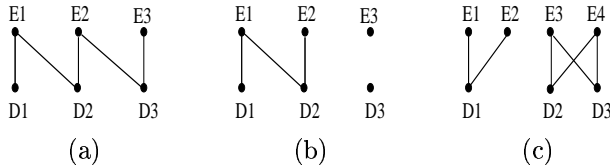


Figure 3: Bipartite Resolution

For example, given the three records of the DOM tree in Figure 2 with the first two records as the examples identified, suppose $E_1 = table \cdot tr \cdot td \cdot p^* \cdot font^* \cdot b \cdot font^* \cdot p^*$ is the PAE learned for *HospitalName*, the

PAE learned for *DoctorName* is $E_2 = table \cdot tr \cdot td \cdot b^* \cdot p \cdot b^*$, and the PAE learned for *PhoneNumber* is $E_3 = table \cdot tr \cdot td \cdot p$. Let D_1, D_2 , and D_3 represent the data values (including their path strings) “Pets First”, “Tom”, and “(123)555-3000” in the third record of the DOM tree, respectively. Then E_1 matches D_1 and D_2 , E_2 matches D_2 and D_3 , and E_3 matches D_3 only. The bipartite graph created from the PAEs and the data values for this record is illustrated in Figure 3(a).

If a PAE E_i uniquely matches (the path string of) a data value α_j and no other PAE uniquely matches α_j , then we make a 1–1 association between E_i and α_j (Lines 14–19). In other words, a high confidence is placed on a match of a data value by a PAE if this particular PAE does not match any other data values and no other PAEs uniquely match this data value. We remove the edges from those PAEs other than E_i that point to α_j (Line 23). For example, in Figure 3(a), since E_3 uniquely matches D_3 the attribute *PhoneNumber* is associated with D_3 and all edges leading into D_3 are deleted. The residual bipartite graph is shown in Figure 3(b).

The computation is repeated until it is not possible to derive any more 1–1 associations. For example, in Figure 3(b), it is still possible to resolve more ambiguity because E_2 now uniquely matches D_2 . As a result, the attribute *DoctorName* is associated with D_2 and all edges leading into D_2 are deleted. In the final residual graph there is a unique matching between E_1 and D_1 . Thus, the attribute *HospitalName* is associated with D_1 and all edges leading into D_1 are deleted. The algorithm terminates now because no more unique associations can be derived (trivially true in this case because there are no longer any edges in the graph).

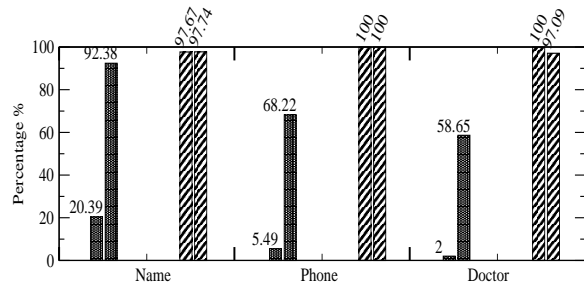
However, it may happen that the ambiguity resolution algorithm based on bipartite graphs is unable to derive any new association at all. For example, in Figure 3(c), the algorithm terminates without any new association because it is not possible to associate D_1 with either E_1 or E_2 (as the condition is violated that a PAE should uniquely match a data value and no other PAEs should uniquely match this data value). Moreover, the ambiguity between E_3, E_4 and D_2, D_3 cannot be resolved either (as there is no unique matching).

4 Experimental Results

Herein we describe our experimental evaluation of our techniques for extracting multi-attribute data. For simplicity we have assumed that all records are flat (*i.e.*, with no nested structures) and an attribute occurrence does not span multiple leaf nodes in a DOM tree. In Section 6 we discuss relaxing these two assumptions.

Attribute Name	Actual Present	Ontology Identified	Nonredundant PAEs	Ambiguity Resolution
Hospital Name	2060	1667	420	1903
Phone Number	1841	963	101	1256
Doctor Name	1705	453	34	1000

(a)



(b)

Figure 4: Aggregated Extraction Results for Veterinarian Sites

Attribute Name	PAE #	Actual Present	Ontology Identified	Consistent PAEs	Recall %	Precision %
Hospital Name	47	338	287	294	86.98	97.35
Phone Number	18	111	47	100	90.09	100
Doctor Name	7	39	26	32	82.05	100

(a)

Attribute Name	Consistent PAEs #	Recall %	Precision %
Hospital Name	47	92.38	97.74
Phone Number	18	68.22	100
Doctor Name	7	58.65	100

(b)

Figure 5: Extraction Results for Consistent PAEs in the Veterinarian Domain

4.1 Experimental Setup

The following sequence of steps were carried out in our experimentation: (1) generating the data sets; (2) labeling the examples; (3) learning the different classes of PAEs from these examples and applying them to extract data from the corresponding individual Web pages; (4) performing ambiguity resolution on the extracted data; (5) manually verifying the recall and precision metrics from the extracted data.

Domains and Data Sets We chose two different domains – veterinarian service providers and lighting products. For the former we focused on *referral* pages such as the one shown in Figure 1. We used a keyword-based Web search engine to retrieve a large collection of veterinarian service provider Web pages. From this collection we randomly selected 170 referral pages from different service provider Web sites. For the product domain we randomly collected 24 Web pages pertaining to lighting products. Each such page lists several product descriptions. These pages were downloaded from 4 different Web sites: 2 from Kmart, 3 from OfficeMax, 13 from Staples, and 6 from Target.

The task of data extraction is considerably simplified by identifying the boundaries of individual entities in a Web page. For instance, in Figure 2, every subtree rooted at *tr* encapsulates a service provider entity. The problem of locating such entity blocks, referred to as *record-boundary discovery* in the literature, has been addressed in a number of previous works [2, 8, 32, 36, 26, 7, 17, 9, 14]. Prior to applying

our data extraction algorithms we identified the boundaries of entities in a Web page.

Labeling We used two ontologies⁵ for labeling examples – one for *Veterinarian Services* and the other for *Lighting Products*. The attributes associated with *Veterinarian Services* in the ontology were *HospitalName*, *PhoneNumber*, and *DoctorName*, while two attributes, *Name* and *Price*, were used for *Lighting Products*.

The keywords *hospital* and *clinic* served as the identifier function for labeling the examples of the *HospitalName* attribute in a Web page. For the *DoctorName* attribute the keyword *DVM* was used. For the *PhoneNumber* attribute we used the regular expression (in Perl syntax), $[0-9]\{3\}-[0-9]\{3\}-[0-9]\{4\}$, as the identifier function. We used the keywords, *lamp*, *bulb*, and *tube*, for the *Name* : attribute of *Lighting Products* and searched the symbol \$ to label examples of the *Price* attribute.

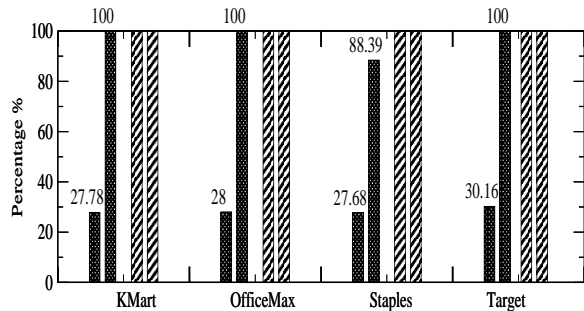
It is relatively easy to come up with such a simple ontology for any domain. In fact, we had to manually inspect only a few Web pages to come with these identifier functions for labeling. However one can use more sophisticated identifier functions. For example, a standard named-entity recognizer can be used for labeling *HospitalName* and *DoctorName* attribute occurrences.

Extraction Process Our experiments were conducted on 1GHz Pentium 4 machines each with 256MB

⁵Using ontologies for data extraction is a known idea [16].

Site	Attribute Name	Actual Present	Ontology Identified	Consistent PAEs
KMart	Name	18	5	18
	Price	18	18	18
OfficeMax	Name	25	7	25
	Price	25	25	25
Staples	Name	112	31	99
	Price	112	112	99
Target	Name	63	19	63
	Price	63	63	63

(a)



(b)

Figure 6: Extraction Results for Individual Product Sites

memory. All algorithms were programmed in Java. All Web pages were parsed into DOM trees and the entity blocks identified. Note that using DOM trees for data extraction is a known idea (*e.g.*, [34]). Next the identifier functions associated with the attributes in the ontology are applied to each DOM tree. The paths leading to the leaf nodes matched by an identifier function become the positive examples for the corresponding attribute. Based on these examples nonredundant PAEs are learned and then applied to each entity block for extracting the attribute data. The two-step ambiguity resolution procedure described in Section 3 was applied to the learned PAEs and the extracted data values to make 1–1 associations between them. This amounts to a strong bias towards high-precision rules [11]. Finally, recall and precision metrics were manually computed from the results of the extraction process.

4.2 Analysis of Experimental Results

Nonredundant PAEs and Ambiguity Resolution Figures 4 summarizes the recall and precision performance of extraction using nonredundant PAEs and the effect of ambiguity resolution. These results were aggregated over the 170 veterinarian Web pages. In Figure 4(a) the total count of the actual occurrences of each attribute (Column 2) over all the pages was ascertained manually. Column 3 shows the number of attribute values which were identified by the corresponding identifier functions in the ontology. For example the identifier function for the *HospitalName* attribute which does a keyword search on the strings “hospital” and “clinic” identified 1667 names. Column 4 is the number of 1–1 associations between a nonredundant PAE and an attribute data value. For example, there were 420 such associations between hospital names and the nonredundant PAE for the *HospitalName* attribute. Column 5 is the number of 1–1 associations between a nonredundant PAE and a attribute data value that were made by the ambiguity resolution procedure. For instance, it resolved 1903 hospital names uniquely.

Correctness of an association was manually verified over all the pages.

Figure 4(b) summarizes as bar charts the recall (shaded bars) and precision (checkered bars) performance of the nonredundant PAEs for each of the three attributes, both before (illustrated by the bar on the left) and after (illustrated by the bar on the right) ambiguity resolution. Observe from the recall/precision bar charts that for all the three attributes there is a significant increase in recall with no loss in precision after ambiguity resolution. This shows that our ambiguity resolution procedure is quite effective.

Consistent PAEs Sometimes the nonredundant PAEs generated by the algorithm *LearnPAE* turn out to be consistent. We used this observation to identify the consistent PAEs among those nonredundant PAEs generated by the algorithm *LearnPAE* on the veterinarian data. We collected the recall and precision numbers only for those Web pages that generated such PAEs. In the table of Figure 5(a), Column 2 is the total number of Web pages where the nonredundant PAE for an attribute was consistent w.r.t. positive and negative examples. Columns 3 and 4 show the actual number of instances of that attribute in these pages and the number of instances identified by the ontology, respectively. Column 5 is the count of correct (manually ascertained) attribute data values extracted by the consistent PAEs. Columns 6 and 7 are the recall and precision numbers for the attributes based on the 1–1 associations made *prior to* ambiguity resolution. In contrast observe the relatively low recall of nonredundant PAEs prior to ambiguity resolution (see the bars on the left in Figure 4(b)). This experimentally validates that consistent PAEs have superior recall and precision than nonredundant PAEs.

For yet another evidence of the superiority of consistent PAEs, observe in Figure 4(b) that after ambiguity resolution the *overall* recall of the *HospitalName* attribute is better than the *PhoneNumber* attribute which in turn is better than that of the *DoctorsName* attribute. The reason can be readily explained by the

number of consistent PAEs generated for the corresponding attributes, which have a high impact on the effectiveness of the bipartite graph resolution procedure. Observe in Figure 5(b) that this number is the highest for the *HospitalName* attribute and is the lowest for the *DoctorName* attribute.

Unambiguous PAEs We ran the algorithm *LearnPAE* to generate a *pair* of PAEs for extracting the *Name* and *Price* attributes from the lighting products pages of four different Web sites. These pages were all “well-structured” in the sense that the pair of PAEs produced by *LearnPAE* for each page turned out to be almost always unambiguous w.r.t. the examples that were identified by the ontology. The statistics for both attributes are shown in Figure 6(a), which can be interpreted the same way as those in the table of Figure 4(a). In Figure 6(b), we compare the recall (shaded) and precision (checkered) numbers of the identifier functions in the ontology (the left bar) and the unambiguous PAEs (the right bar). The statistics in Figure 6(b) are collected on the *Name* attribute only. Observe that precision is 100% while recall is almost close to 100% which experimentally demonstrates the superior quality of unambiguous PAEs.

Efficiency and Scalability Although learning PAEs with high recall and precision is computationally difficult, in practice our simple heuristic based upon *LearnPAE* did generate PAEs of high quality. This implies that the running time of the extraction process on real-life data need not be inordinately high. For instance, the time taken to generate the *HospitalName* PAE ranged from 15 to 172 milliseconds per HTML page. The size of these pages ranged from 1KB to 60KB with some of them having up to 150 entity blocks.

Finally, it was observed that the HTML pages across these four different lighting products Web sites were widely dissimilar. However, in spite of this dissimilarity, the high recall and precision of extraction obtained across all the four sites indicate the scalability of our learning techniques.

5 Related Work

We model the notion of precision and recall that arises in wrapper building as a grammar inference problem. Therefore, to put our work in perspective we now review research in related areas, namely, grammar inference, sequence learning, and wrapper construction.

Grammar Inference This well-known problem was first addressed in the seminal works of Gold and Angluin. Gold [19, 20] showed that the problem of inferring a DFA of *minimum* size from positive examples is NP-complete. In [3] Angluin showed that the problem

of learning a regular expression of *minimum* size from positive and negative examples is NP-complete. In our problem, however, we do not impose any constraint on the size of the PAEs learned.

In [4] Angluin studied the problem of inductive inference of an indexed family of nonempty recursive formal languages from positive examples only. In this work a learner is presented a sequence of positive examples which form some arbitrary enumeration of *all* the elements of the language to be inferred. Such a problem setting is different from ours.

Angluin [5] also proposed a polynomial time algorithm for *actively* learning the minimum DFA of a regular language from a *teacher* who knows the true identity of this regular language. We should point out that such an active learning framework is different from ours, which is passive.

In summary, the problems of learning consistent PAEs and unambiguous sets of PAEs (the problems considered in this paper) do not have equivalent counterparts in either the classical works on grammar inference or in recent works (refer to [35] for a recent survey) and hence none of the known results is applicable.

Sequence Learning There is a large body of work on learning subsequences and supersequences from a set of strings. The following problems are all NP-complete: (1) finding the SCS/LCS of an arbitrary number of strings over a binary alphabet [29, 33]; (2) finding a sequence which is a common subsequence/supersequence of a set of positive examples but not a subsequence/supersequence of any string in a set of negative examples [25, 30]. The semantics of PAEs differs substantially from string matching and hence their results are not applicable. Besides our problem of learning a *set* of consistent PAEs (one per attribute) has no counterpart in these works.

Wrapper Construction Research on wrapper construction for Web sources (see [27] for an extensive survey) has made a transition from its early focus on manual [8, 34, 22] and semi-automatic [2, 32, 23] approaches to fully automated techniques based on machine learning [26, 10, 31, 7, 11, 24, 13]. But the notion of ascribing a precision/recall metric to the learning of extraction expressions and its impact on algorithmic efficiency has not been explored in these works.

Wrapper induction based on machine learning techniques was pioneered by the early work of [26]. The idea of learning a wrapper from a set of positive examples was explored in [26], which proposed algorithms for learning delimiters for tuples and attribute values within each tuple. These algorithms were later extended in [24]. However, the delimiters considered in [26, 24] are mainly based on common suffixes and prefixes of strings preceding and following the attribute

values, respectively. Unlike our PAE setting, there is no notion of generalization using the “*” operator. Consequently in the absence of common suffixes or prefixes their delimiter discovery algorithms will fail. For example, in Fig 2, the *HospitalName* attribute instances “ABC Animal Hospital” and “XYZ Animal Hospital” have no common suffix in their path strings *td.font.b.p* and *td.p.b.font*.

The idea of ambiguity was also explored in [26] in the context of labeling attribute instances in a document. If a data value turns out to be ambiguous (*e.g.*, in the presence of nullable data attributes), then the algorithm in [26] resorts to checking an exponential number of combinations to infer the correct wrapper. However, the reasons behind this computational blowup was not studied in [26]. Our notion of unambiguous PAEs formalizes the computational difficulty of ambiguity resolution. In [10], a similar notion of ambiguity was introduced to guide the learning of a set of extraction expressions. However, [10] did not explore the relationship between the complexity of learning extraction expressions and recall/precision.

The XTRACT system for inferring schemas of XML documents was reported in [18]. However, in the problem settings of [18], the examples are labeled trees instead of strings. Moreover, the main constraint there is minimization of the size of the inferred schema. Consequently, all input examples in [18] are positive examples and our notion of unambiguity w.r.t. positive and negative examples is not explored in [18].

The issue of ambiguity resolution can also commonly arise in learning-based approaches to schema inference and data extraction from Web documents [21, 13, 6] but has remained relatively unexplored in the literature. In [21], several schema inference problems are introduced and shown to be solvable in polynomial time. However, the lower complexity results reported in [21] are due to the assumption of *explicit* encoding of null data values. Our results have shown that if null data values are completely omitted (which can be commonly observed in practice), then the (unambiguous) schema inference problem quickly becomes intractable in general. In [13] the learned schema is represented as a union-free regular expression. But the sophisticated algorithm for discovering a “good” schema can suffer from exponential blowup [13]. We can view the notion of good schema as corresponding to our notion of unambiguity. We have shown that the problem of learning an unambiguous union-free regular expression is computationally difficult (see Appendix B), implying that ambiguity resolution is the root cause of the computational difficulty underlying schema discovery.

Although exponential-time and polynomial-time heuristics have been proposed in [13, 6], respectively, to infer schemas from *unlabeled* Web documents for the

purpose of automated data extraction, our complexity results imply that the schema inference problem is still computationally difficult even when the labels are already known. Moreover, a collection of pages, generated from the *same* template, is required to learn the schema in [13, 6], but our learning techniques can apply to *individual* documents.

In summary, ambiguity appears to be an implicit theme underlying the problems studied in these works. Our complexity results provide a theoretical underpinning for the complexity-theoretic aspects of unambiguous data extraction and schema inference.

6 Conclusion

In our experimental setup an attribute value was associated with a single leaf node in a DOM tree. All we need to do when they span several leaf nodes is to learn a set of PAEs per attribute instead of one as described in this paper. All our results can be readily extended to such a case. However, to deal with nested types we need to use branching semantics for PAEs instead of set semantics as is done currently. This means we will have to learn tree patterns to pull out attribute data. This will be an important and useful extension to the work reported in this paper.

Precision of the extracted data is essential for “push-button” solutions to data extraction that can be envisioned as follows: Starting with a *small*, hand-crafted “seed ontology”, an extraction process driven by PAEs as described in this paper will automatically enrich the ontology with information hidden in the extracted data. However, for this process of bootstrapping an enriched ontology with recalled attribute data to work properly, precision of the extracted data is of paramount importance.

References

- [1] XML path language (XPath). <http://www.w3.org/TR/xpath>.
- [2] B. Adelberg. Nodose: A tool for semi-automatically extracting structured and semi-structured data from text documents. In *ACM SIGMOD Conference on Management of Data*, 1998.
- [3] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.
- [4] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

- [5] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- [6] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD Conference on Management of Data*, 2003.
- [7] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. *ACM SIGMOD Record*, 26(4), 1997.
- [8] P. Atzeni and G. Mecca. Cut & paste. In *ACM Symposium on Principles of Database Systems (PODS)*, 1997.
- [9] D. Butler, L. Liu, and C. Pu. A fully automated object extraction system for the world wide web. In *International Conference on Distributed Computing Systems (ICDCS)*, 2001.
- [10] B. Chidlovskii. Wrapping web information providers by transducer induction. In *European Conference on Machine Learning*, 2001.
- [11] W. Cohen, M. Hurst, and L. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *International World Wide Web Conference*, 2002.
- [12] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
- [13] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *International Conference on Very Large Data Bases (VLDB)*, 2001.
- [14] H. Davulcu, S. Mukherjee, and I. V. Ramakrishnan. Extraction techniques for mining services from web sources. In *IEEE International Conference on Data Mining (ICDM)*, 2002.
- [15] H. Davulcu, G. Yang, M. Kifer, and I. Ramakrishnan. Computational aspects of resilient data extraction from semistructured sources. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 136–144, 2000.
- [16] D. W. Embley, D. M. Campbell, R. D. Smith, and S. W. Liddle. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *International Conference on Information and Knowledge Management (CIKM)*, 1998.
- [17] D. W. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in web documents. In *ACM SIGMOD Conference on Management of Data*, 1999.
- [18] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Shadri, and K. Shim. XTRACT: A system for extracting document type descriptors from XML documents. In *ACM SIGMOD Conference on Management of Data*, 2000.
- [19] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
- [20] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.
- [21] S. Grumbach and G. Mecca. In search of the lost schema. In *Intl. Conf. on Database Theory*, 1999.
- [22] J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting semistructured information from the web. In *Workshop on Management of Semistructured Data*, 1997.
- [23] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. M. Breunig, and V. Vassalos. Template-based wrappers in the tsimmis system. In *ACM SIGMOD Conference on Management of Data*, 1997.
- [24] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.
- [25] T. Jiang and M. Li. On the complexity of learning strings and sequences. In *Theoretical Computer Science*, volume 119, pages 363–371, 1993.
- [26] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, 1997.
- [27] A. Laender, B. Ribeiro-Neto, A. da Silva, and J. Teixeira. A brief survey of web data extraction tools. volume 31, 2002.
- [28] L. Liu, C. Pu, and W. Han. Xwrap: An xml-enabled wrapper construction system for web information sources. In *International Conference on Data Engineering (ICDE)*, 2000.
- [29] D. Maier. The complexity of some problems on subsequences and supersequences. In *Journal of ACM*, volume 25, 1978.
- [30] M. Middendorf. On finding various minimal, maximal, and consistent sequences over a binary alphabet. In *Theoretical Computer Science*, volume 145, pages 317–327, 1995.

- [31] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 190–197, 1999.
- [32] M. Perkowitz, R. B. Doorenbos, O. Etzioni, and D. S. Weld. Learning to understand information on the internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2), 1997.
- [33] K.-J. Rähkä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is np-complete. In *Theoretical Computer Science*, volume 16, pages 187–198, 1981.
- [34] A. Sahuguet and F. Azavant. Web Ecology: Recycling HTML pages as XML documents using W4F. In *ACM SIGMOD Workshop on the Web and Databases (WebDB)*, 1999.
- [35] Y. Sakakibara. Recent advances of grammatical inference. In *Theoretical Computer Science*, volume 185, pages 15–45, 1997.
- [36] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3), 1999.

Appendix

A Proof

Here we will present the proof of Theorem 1. The proof of Theorem 2 is similar to the proof presented here, but is omitted due to want of space.

In the sequel, we will use ε to denote either the empty string or the empty expression. Its intended usage should be clear from the context. We will also use the notation α^k , where α is a string and k an integer, to represent the string obtained by repeating k times the string α . In particular, $\alpha^0 = \varepsilon$.

Proof. Let POS and NEG be two sets of strings. First, deciding whether or not a string is accepted by a PAE can be done in polynomial time. Second, it is easy to see that the size of the shortest PAE that is consistent w.r.t. $\langle POS, NEG \rangle$ is bounded by the sum of the lengths of the strings in POS . Therefore, this problem is in NP.

To prove that this problem is NP-hard, we will reduce SAT to our problem. We will also assume the alphabet $\Sigma = \{\$, 0, 1\}$.

Let F be a propositional formula in *conjunctive* normal form with m clauses C_1, C_2, \dots, C_m and n variables V_1, V_2, \dots, V_n . For $1 \leq i \leq m$ and $1 \leq j \leq n$, let us define:

$$F_{ij} = \begin{cases} \$10, & \text{if } V_j \text{ appears positively in } C_i; \\ \$01, & \text{if } V_j \text{ appears negatively in } C_i; \\ \$00, & \text{if } V_j \text{ does not appear in } C_i. \end{cases}$$

The idea is that in a string we will use $\$01$ and $\$10$ to represent the logical values *true* and *false*, respectively. Thus for all $1 \leq i \leq m$, the string $F_{i1}F_{i2} \dots F_{in}$ encodes the *only* assignment of truth values to the variables, V_1, V_2, \dots, V_n , which makes the clause C_i false. Moreover, define:

$$\begin{aligned} POS &= \{(\$0)^n, (\$1)^n\} \\ NEG &= N_1 \cup N_2 \cup N_3 \\ N_1 &= \{\$^{n+1}, 0\$^n, 1\$^n\} \\ N_2 &= \{\$^k 010\$^{n-k}, \$^k 101\$^{n-k} \mid 1 \leq k \leq n\} \\ N_3 &= \{F_{i1}F_{i2} \dots F_{in} \mid 1 \leq i \leq m\} \end{aligned}$$

We will show that the formula F is satisfiable iff there is a PAE that is consistent w.r.t. $\langle POS, NEG \rangle$.

We will also use two PAEs, $E_t = \$0^*1^*$ and $E_f = \$1^*0^*$, to represent the logical values *true* and *false*, respectively. First, given an assignment of truth values to the variables, V_1, V_2, \dots, V_n , in the formula F , we can construct a PAE, $E = E_1E_2 \dots E_n$, where for all $1 \leq j \leq n$,

$$E_j = \begin{cases} E_t, & \text{iff the truth value assigned to } V_j \text{ is } \textit{true}; \\ E_f, & \text{iff the truth value assigned to } V_j \text{ is } \textit{false}. \end{cases}$$

So if the formula F is satisfiable, then there must be an assignment of truth values to the variables, V_1, V_2, \dots, V_n , which satisfies F . It can be shown that if we construct a PAE, E , as defined above, then E is consistent w.r.t. $\langle POS, NEG \rangle$.

Now suppose that there is a PAE, E , which is consistent w.r.t. $\langle POS, NEG \rangle$. Then it follows that $\mathcal{L}(E) \supseteq POS$ and $\mathcal{L}(E) \cap NEG = \emptyset$. We assume that E is in a compact form in which the consecutive occurrences of 0^* or 1^* are collapsed into one, since the resulted expression will still be equivalent to the original one. For instance, $\$0^*1^*$ is equivalent to $\$0^*0^*1^*$. Since $\mathcal{L}(E) \supseteq POS$, a $*$ operator must be attached to *every* occurrence of 0 and 1 in E . Because $\mathcal{L}(E) \cap N_1 = \emptyset$, E must have the form of $\$\alpha_1\$\alpha_2 \dots \$\alpha_n$, where each α_i is a sequence of 0^* and 1^* only. Moreover, both 0^* and 1^* must appear at least once in each α_i . Because $\mathcal{L}(E) \cap N_2 = \emptyset$, it follows that each α_i is either 0^*1^* or 1^*0^* . Therefore, we can obtain an assignment of truth values to the variables, V_1, V_2, \dots, V_n , as defined above. Because $\mathcal{L}(E) \cap N_3 = \emptyset$, it can be shown that this assignment must satisfy the formula F which is in conjunctive normal form.

Clearly, $|POS| + |NEG| = O(mn)$. Therefore our problem is NP-hard. \square

B Extension to Union-Free Regular Expressions

In this section, we will show that even if we extend the syntax of PAEs to that of *union-free* regular expressions, the corresponding learning problems still remain NP-complete. This complexity result is of particular interest because the union-free regular expressions described here are exactly the syntax used in the schema inference problem in [13].

Formally, we have the following definition of union-free regular expressions.

Definition 7 (Union-Free Regular Expression) Let Σ be a finite alphabet. A union-free regular expression (abbr. UFRE) over Σ is defined inductively as follows:

1. Any symbol $c \in \Sigma$ is a union-free regular expression.
2. If E_1 and E_2 are union-free regular expressions, so is $E_1 \cdot E_2$.
3. If E is a union-free regular expression, so are E^* and $E?$.

In the above definition, E^* means E can appear zero or multiple times while $E?$ means E can appear zero or once, i.e., is optional. For instance, $(ab)^* \cdot d?$ is a valid UFRE. Note that $E? = \varepsilon|E$. Therefore, the above definition actually allows a very restricted use of the union operator.

Now we restate our learning problem w.r.t. union-free regular expressions as follows.

Problem 4 (Consistent UFRE) Given two sets of strings POS and NEG , is there a UFRE that is consistent w.r.t. $\langle POS, NEG \rangle$?

Before proving the NP-completeness result, we need show a bound on the size of a consistent UFRE w.r.t. the size of the positive examples. To be precise about the size of a UFRE, we require that it should be presented in *canonical form*, i.e., the operands of the operators, “ \cdot ”, “ $*$ ”, and “ $?$ ”, be parenthesized. In calculating the size of a UFRE, we count all the characters in its canonical form. For instance, $((a \cdot (b))^*) \cdot (d?)$ is the canonical form of $(ab)^* \cdot d?$. Its size is 17.

Let E be a UFRE and S a set of strings. In the sequel, we will use the notation $\mathcal{L}(E)$ to stand for the set of strings accepted by E , $|E|$ for the size of E , and $\|S\|$ for the sum of the lengths of all the strings in S .

Theorem 4 Let POS and NEG be two sets of strings, $n = \|POS\|$, $m = \|NEG\|$, E the smallest UFRE that is consistent w.r.t. $\langle POS, NEG \rangle$. Then $|E| \leq 12n + 5m + 3$.

Theorem 5 The consistent UFRE problem is NP-complete.

Proof. Let POS and NEG be two sets of strings. First, deciding whether or not a string is accepted by a regular expression can be done in polynomial time. Second, by Theorem 4 we know that the size of the shortest UFRE that is consistent w.r.t. $\langle POS, NEG \rangle$ is bounded by the size of POS and NEG . Therefore, this problem is in NP.

To prove that this problem is NP-hard, we will reduce SAT to our problem. We will also assume the alphabet $\Sigma = \{\$, 0, 1\}$.

Let F be a propositional formula in *conjunctive normal form* with m clauses C_1, C_2, \dots, C_m and n variables V_1, V_2, \dots, V_n . For $1 \leq i \leq m$ and $1 \leq j \leq n$, let us define:

$$F_{ij} = \begin{cases} \$10, & \text{if } V_j \text{ appears positively in } C_i; \\ \$01, & \text{if } V_j \text{ appears negatively in } C_i; \\ \$0, & \text{if } V_j \text{ does not appear in } C_i. \end{cases}$$

The idea is that in a string we will use $\$01$ and $\$10$ to represent the logical values *true* and *false*, respectively. Thus for all $1 \leq i \leq m$, the string $F_{i1}F_{i2} \dots F_{in}$ encodes the *only* assignment of truth values to the variables, V_1, V_2, \dots, V_n , which makes the clause C_i false. Moreover, define:

$$\begin{aligned} POS &= \{(\$0)^n, (\$1)^n\} \\ NEG &= N_1 \cup N_2 \cup N_3 \\ N_1 &= \{\$^k \mid 0 \leq k \leq n-1\} \\ N_2 &= \{\$^k 00\$^{n-k}, \$^k 11\$^{n-k} \mid 1 \leq k \leq n\} \\ N_3 &= \{F_{i1}F_{i2} \dots F_{in} \mid 1 \leq i \leq m\} \end{aligned}$$

We will show that the formula F is satisfiable iff there is a UFRE that is consistent w.r.t. $\langle POS, NEG \rangle$.

We will also use two UFREs, $E_t = \$(0?)(1?)$ and $E_f = \$(1?)(0?)$, to represent the logical values *true* and *false*, respectively. First, given an assignment of truth values to the variables, V_1, V_2, \dots, V_n , in the formula F , we can construct a UFRE, $E = E_1E_2 \dots E_n$, where for all $1 \leq j \leq n$,

$$E_j = \begin{cases} E_t, & \text{iff the truth value assigned to } V_j \text{ is } \textit{true}; \\ E_f, & \text{iff the truth value assigned to } V_j \text{ is } \textit{false}. \end{cases}$$

So if the formula F is satisfiable, then there must be an assignment of truth values to the variables, V_1, V_2, \dots, V_n , which satisfies F . It can be shown that if we construct a UFRE, E , as defined above, then E is consistent w.r.t. $\langle POS, NEG \rangle$.

Now suppose that there is a UFRE, E , which is consistent w.r.t. $\langle POS, NEG \rangle$. Then it follows that $\mathcal{L}(E) \supseteq POS$ and $\mathcal{L}(E) \cap NEG = \emptyset$. We will show that from E we can obtain an assignment of truth values to the variables, V_1, V_2, \dots, V_n , which satisfies the formula F .

Let $E = A_1A_2 \dots A_i$, where each A_k ($1 \leq k \leq i$) is a single symbol ($\$, 0$, or 1), or in the form of $(X)?$ or $(X)^*$. Since $\mathcal{L}(E) \supseteq POS$, it follows that for each A_k , if A_k is a single symbol, then $A_k = \$$. Because $\mathcal{L}(E) \cap N_1 = \emptyset$, so the number of A_k 's that are the single symbol $\$$ must be exactly n . It follows that E must have the form of $B_0\$B_1\$B_2 \dots \$B_n$, where each B_k is a concatenation of expressions in the form of $(X)?$ or $(X)^*$. Moreover, if B_0 is not an empty expression, then we can remove B_0 from E , because the resulting expression would still be consistent w.r.t. $\langle POS, NEG \rangle$. Therefore, in the following we will assume that $E = \$B_1\$B_2 \dots \$B_n$.

Next we will show that each B_k in E can be transformed into either $(0?)(1?)$ or $(1?)(0?)$ and the resulting new expression E is still consistent w.r.t. $\langle POS, NEG \rangle$.

We define two transformation operations on the B_k 's as follows: (i) remove a “ $?$ ” operator; (ii) remove a “ \cdot ” operator together with one of its operands. We will keep performing either of these two operations on each B_k unless it gives rise to inconsistency. So when this process stops, we cannot remove any operator from any of the B_k 's and still get a new consistent expression.

We claim that when the above process ends, each B_k must be either $(0?)(1?)$ or $(1?)(0?)$. Because $\mathcal{L}(E) \supseteq POS$, it must be true that $\{\varepsilon, 0, 1\} \subseteq \mathcal{L}(B_k)$ for each B_k . Since

$\mathcal{L}(E) \cap N_2 = \emptyset$, it follows that $00 \notin \mathcal{L}(B_k)$, $11 \notin \mathcal{L}(B_k)$, for each B_k . Therefore, B_k cannot have the form of $(X)^*$. Moreover, B_k cannot have the form of $(X)?$ either; otherwise we could remove the “?” operator and the resulting new expression would still be consistent, because $\{0, 1\} \subseteq \mathcal{L}(X) \subseteq \mathcal{L}((X)?)$.

Therefore, it must be true that $B_k = (C_{k1}) \cdot (C_{k2})$. Note that $\mathcal{L}(C_{k1})$ and $\mathcal{L}(C_{k2})$ must contain only one of 0 and 1 but not both; otherwise the “.” operator together with one of C_{k1} and C_{k2} could be removed. Since $\varepsilon \in \mathcal{L}(B_k)$, it follows that $\varepsilon \in \mathcal{L}(C_{k1})$, $\varepsilon \in \mathcal{L}(C_{k2})$.

Let us suppose $0 \in \mathcal{L}(C_{kj})$ ($j = 1$ or $j = 2$). Since $00 \notin \mathcal{L}(B_k)$, it follows that $00 \notin \mathcal{L}(C_{kj})$. So C_{kj} cannot be a single symbol, or have the form of $(X)^*$ or $(X) \cdot (Y)$. Therefore C_{kj} must have the form of $(D)?$. By a similar argument, we can show that D must be the single symbol 0. So C_{kj} must be $(0)?$. Similarly, we can show that if $1 \in \mathcal{L}(C_{kj})$ ($j = 1$ or $j = 2$), then C_{kj} must be $(1)?$. Therefore, B_k must be either $(0?)(1?)$ or $(1?)(0?)$.

We have shown that we can obtain a consistent UFRE, $E = \$B_1\$B_2 \dots \$B_n$, where each B_k is either $(0?)(1?)$ or $(1?)(0?)$. We define an assignment of truth values to the variables, V_1, V_2, \dots, V_n , as follows: if $B_k = (0?)(1?)$, then assign *true* to V_k ; if $B_k = (1?)(0?)$, then assign *false* to V_k . Because $\mathcal{L}(E) \cap N_3 = \emptyset$, we can verify that this truth value assignment must satisfy the formula F .

Clearly, $|POS| + |NEG| = O(mn)$. Therefore this problem is NP-hard. \square