# Extracting partial structures from HTML documents

Sakamoto, Hiroshi
Department of Informatics, Kyushu University

Arimura, Hiroki
Department of Informatics, Kyushu University

Arikawa, Setsuo
Department of Informatics, Kyushu University



出版情報：DOI Technical Report. 181, 2000-11. Department of Informatics, Kyushu University
バージョン：
権利関係：

# Extracting partial structures from HTML documents

**Hiroshi Sakamoto, Hiroki Arimura, and Setsuo Arikawa**

Department of Informatics, Kyushu University
Hakozaki 6-10-1, Hizashi-ku, Fukuoka-shi 812-8581, Japan
{hiroshi, arim, arikawa}@i.kyushu-u.ac.jp
phone: +82-92-642-2693, fax: +82-92-642-2698

**Abstract**

The new wrapper model for extractiong text data from HTML documents is introduced. The Kushmerick's wrapper class (Kusshmerick 2000) may be unsuccessful in the case that sufficiently long delimiters are not found. The wrapper class introduced in this paper partially overcomes this difficulty by using the tree structures of HTML documents. The learning problem to learn such a wrapper program from given text is considered. Moreover, we try to expand our wrapper to extract a portion of HTML not only text attributes.

**keywords:** data extraction, wrapper induction, semi-structured data, learning from examples

## Introduction

The HTML documents currently distributed on the world-wide-web can be regarded as a very large semi-structured database. These texts are structured by many tags that have their own special meanings beforehand. A markup text is expressed by a rooted ordered tree. The root is the unique node which denotes the whole document, the other internal nodes are labeled by tags, and the leaves are labeled by the contents of the document or the attributes of the tags.

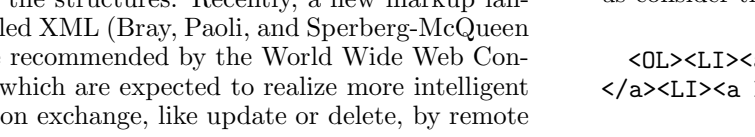
Although computers can not understand the meaning of languages, they can perform a complicated rendering using the structures. Recently, a new markup language called XML (Bray, Paoli, and Sperberg-McQueen 1998) are recommended by the World Wide Web Consortium, which are expected to realize more intelligent information exchange, like update or delete, by remote operation.

The XML documents are used for data translation or extraction. For example, an XML document is translated to an HTML for the browsing or it may be translated to another XML document in different format for data exchange. These translations are realized by the stylesheet language XSLT, which is recommended by W3C in 1999 (Clark 1999). This language is very powerful because regular expression and recursion are allowed to use, and thus, it is hard to construct an algorithm for generating such languages.



For this purpose, we introduced more restricted classes for simple translation of semi-structured data and studied the learnability of these classes (Sakamoto, Arimura, Arikawa 2000). We considered the problem of finding a translation rules for given input and output tree with respect to the classes, and we presented an effective learning algorithm for the problem. However, our algorithm depends on the expressiveness of XML tags. For example, if we want to extract only family names from the data containing the texts "Hiroshi Sakamoto", "Hiroki Arimura", "Setsuo Arikawa", etc., then we can do so by using an agreement that family names must be inserted between a pair of specific tags like "Hiroshi <family>Sakamoto</family>". It is impossible to describe such a logical structure in HTML.

On the other hand, Kushmerick introduced a framework for extracting attributes from HTML documents called *wrapper induction* (Kusshmerick 2000). Any plain text rather than tags in an HTML document is said to be an attribute. The wrapper is a program to extract such attributes from given documents and wrapper induction is a framework to learn such program from given example. His algorithm additionally takes the points of strings to be extract. Thus, the main work of the algorithm is to compute the demimiter strings common to all $i$-th attributes. For example, let us consider the following text.




There are two attributes to be extracted and the algorithm knows the position of each attribute. Then, it searches the left and right hand of such attributes and find the longest common suffix and longest common preffix of them. In this case, they are the `">` and the `</a><`, respectively. By using these delimiters, we can extract the same attribute of the form `...">`*attribute*`</a><...`, automatically.

However, as is shown in this example, in case that some tags contain different tag attributes values (e.g., two tags of `<a>` contain defferent values www1 and www2), we can not find sufficiently long suffix or prefix for delimiters. It follows that undesirable text attribute

may be extracted. Thus, we need a new method for wrapper induction.

The idea is to construct trees from HTML documents and extract the common structure to these trees. Any HTML document is corresponding to an ordered labeled tree such that a pair of begin tag <t> and end tag </t> is an internal node labeled by <t>. Other empty tags, tag attributes, and text attributes are labels of leaves. Using this data model, we define the delimiters to extract text attributes from tree structures, and we present the algorithm to compute such delimiters from HTML documents. Give any document $P$ and the positions of attributes, this algorithm runs in $O(n \cdot m)$ time, where $n$ is nearly equal to the length of $P$ and $m$ is the number of types of the attributes (more precisely, $n$ is the number of nodes of the parsing tree of $P$).

Finally, we mention the application of such wrapper class. One direction of HTML is to grow toward richer document format. The recommendation HTML 4.0 includes such a new additional features like the CSS(Cascading Style Sheets). On the other hand, we focus on the other direction for small information appliances such as smart phones, smart communicators, mobile PDAs, and etc. There are several hardware restrictions such as small memory, low power CPU, small or no secondary storage, small display, mono-color, single character font, and restricted input method (no keyboard and mouse). Thus, we need other simple document format and recently, the *Compact HTML* is proposed by (Kamada 1998) which defines a subset of HTML for small information appliances.

The most hopeful usage of the language is to exist together with HTML, i.e., using HTML documents by translating to Compact HTML documents as occasion demands. All tags in Compact HTML is defined in regular HTMLs, but it is not easy to realize such translations because the amount of texts is restricted by the size of a small display. Thus, we must select the important attributes from the large texts. However, the usual wrapper can not extract a portion of all attributes in a document. Hence, we would study this problem with the aim of constructing the automatical translate system.

## Basic Idea

In this section, we briefly explain the notion of wrapper induction introduced by Kushmerick (Kusshmerick 2000) and consider the difficult case not covered by his method. In the next section, we introduce the new wrapper class in order to overcome the difficulty.

An HTML document is a string over a finite alphabet $\Sigma$, where $\Sigma$ is the ASCII character set. An HTML document is defined by the specific tag set and other plain text. Then, there are two types of attributes for the problem in this paper, one is the *tag attribute* and the other is the *text attribute*. The tag attribute is the ordinary one which is defined in any HTML tag. For example, <a href="Mailto:hiroshi@i.kyushu-u.ac.jp"> contains the tag attribute value "Mailto:hiroshi@i.kyushu-u.ac.jp" and <font face="Arial"> contains the tag attribute value "Arial".

The *text attribute* is a string not containing any tags and appearing between two tags such that <a>w, where <a> and  are tags. For examples, <h3>H. Sakamoto</h3> contains the text attribute "H. Sakamoto". Here, we note that it is not necessary that the <a> and  are not beginning and end tags. For example, if a document contains the string Kyushu Univ.</font><br>, then it contains the text attribute "Kyushu Univ.".

Query language receives some queries and outputs some HTML documents. Such documents are described by the repetition of a typical structure. For example, when we find Web sites by using a search engine, the resulting HTML documents may be described by a format which consists of three text atrributes which denote the ranking number of the site, the title of the head, and the abstract no longer than 100 words in the site, respectively. Such HTML documents are called *pages*. The aim of our system is to extract information from pages.

In a page, a sequence of attribute values of the form $\langle t_1, \ldots, t_k \rangle$ is called *tuple*. Thus, a page can be described by a set of tuples such that the $i$-th tuple consists of the text attributes for the $i$-th record in the page. For the simplicity, we denote any tuple $\langle t_1, \ldots, t_k \rangle$ by another description of tuple $\langle \langle b_1, e_1 \rangle, \ldots, \langle b_k, e_k \rangle \rangle$, where $\langle b_i, e_i \rangle$ is the pair of the begin and end indices of $t_i$. Thus, in general, the contents of text attributes of a page, called *label* is described by a set of such tuples as follows.

$$\left\{ \begin{array}{c} \langle \langle b_{1,1}, e_{1,1} \rangle, \ldots, \langle b_{1,j}, e_{1,j} \rangle, \ldots, \langle b_{1,k}, e_{1,k} \rangle \rangle, \\ \langle \langle b_{i,1}, e_{i,1} \rangle, \ldots, \langle b_{i,j}, e_{i,j} \rangle, \ldots, \langle b_{i,k}, e_{i,k} \rangle \rangle, \\ \langle \langle b_{n,1}, e_{n,1} \rangle, \ldots, \langle b_{n,j}, e_{n,j} \rangle, \ldots, \langle b_{n,k}, e_{n,k} \rangle \rangle, \end{array} \right\}$$

Given a page $P$ and a label $L$ of $P$, a *wrapper* $W$ for $(P, L)$ is a program such that $W(P) = L$. The *wrapper induction* problem is, given a set of examples, $\{\ldots, (P_i, L_i), \ldots\}$, to output a wrapper $W$ such that $W(P_i) = L_i$ for all $i$, where $P_i$ is the label of $L_i$.

**Definition 1** (Kusshmerick 2000) The LR-wrapper is a sequence of the left and the right delimiters of the form $\langle \ell_1, r_1, \ldots, \ell_k, r_k \rangle$, where $\ell_i, r_i \in \Sigma^+$.

The procedure $exec_{LR}$ takes input an LR-wrapper and a page $P$, and it returns the label $\{\ldots, \langle \langle b_{m,1}, e_{m,1} \rangle, \ldots, \langle b_{m,k}, e_{m,k} \rangle \rangle, \ldots\}$ as follows.

The $exec_{LR}$ continues the followings until there are more occurrences of $\ell_1$ in $P$. For each $\langle \ell_i, r_i \rangle$ $(1 \leq i \leq k)$, it finds the first occurrence of $\ell_i$ and the first $r_i$ after the $\ell_i$, and it returns the string between the $\ell_i$ and $r_i$ as to the attribute $\langle b_{m,i}, e_{m,i} \rangle$, where $m$ is at most the number of tuples in $P$.

**Example 1** Let us show an example of LR-wrapper for an HTML documents.

```
<html><b>sakamoto</b> <i>hiroshi</i><br>
<b>arimura</b> <i>hiroki</i><br>
<b>arikawa</b> <i>setsuo</i></html>
```

In order to extract the attributes from this document, for example, we can find the LR-wrapper ⟨, , , ⟩. The first left-right delimiters  and  cut the attribute "Sakamoto" and the second delimiters cut the attribute "Hiroshi". Thus, the first tuple is ⟩Sakamoto, Hiroshi⟩. Similarly, the wrapper finds the second and third tuples.

Several wrapper classes are defined and the expressiveness is studied in (Kusshmerick 2000). The wrapper classes he introduced are LR, HLRT, OCLR, HOCLRT, N-LR, and N-HLRT. For example, HLRT is the class in which the procedure takes additional information $(h, t)$, where $h, t$ are any indices in $P$ and it excutes the wrapper in the part from $h$ to $t$. The OCLR is similar but it can takes the information $(o, c)$ which indicates the beginning and the end of each tuple.

The problem in this paper is to extract a portion of an input HTML. For this purpose, the usal wrapper classes are not powerful as follows.

- Let $t = \langle t_1, \ldots, t_k \rangle$ be a tuple of a page $P$. We can not define a wrapper to extract any subsequence $\langle t_{i_1}, \ldots, t_{i_j} \rangle$ of $t$.
- Since any HTML document is regarded as a flat text, the information of tree structure is not used for the wrapper induction.

**Example 2** Let us consider the problem to extract only the text attributes for email address from the following page $P$. The algorithm to learn the LR-wrapper class searches the left hand and right hand of the text attributes. Then, it find the longest common suffix "> and the longest common prefix </a></h3>.

```
<html><body>
sakamoto<br>
<h2><a href="www.i.kyushu-u.ac.jp/~hiroshi">
www.i.kyushu-u.ac.jp/ hiroshi</a></h2><br>
<h3><a href="mailto:hiroshi@i.kyushu-u.ac.jp">
hiroshi@i.kyushu-u.ac.jp</a></h3><br>
...
...
arimura<br>
<h2><a href="www.i.kyushu-u.ac.jp/~arim">
www.i.kyushu-u.ac.jp/ arim</a></h2><br>
<h3><a href="mailto:arim@i.kyushu-u.ac.jp">
arim@i.kyushu-u.ac.jp</a></h3>
</body></html>
```

However, the ⟨">, </a></h3>⟩ is not correct LR-wrapper for this document. This LR-wrapper can not cut the text attributes for email addresses because there are two occurrences of "> in one tuple of $P$, one is before the string www and the other is before the text attribute for email. Thus, the first delimiter matches with the first occurrence of ">. Hence, the wrapped string is the one between the first "> and the occurrence of the second delimiter </a></h3>. This string is not equal to the required text attribute.

The learning algorithm for LR-wrapper class finds the longest common suffix/prefix for each $i$-th text attributes of all tuples in $P$. However, in general, many tags have their own tag attributes and these tag attributes are different on each occurrence. In the above example, all occurrences of <a> contain different tag attributes. Thus, the algorithm can not find sufficiently long common suffix/prefix. To overcome this difficulty, we focus on the tree structure of HTML and introduce the data model for extracting text attributes from tree structures.

## T-wrapper and the algorithm

We denote an HTML is represented by an ordered labeled tree such that each tag name, tag attribute, and text attribute are defined as a label on a node. Thus, all tag attributes, text attributes, and empty tags are considered as leaves.

First, we define the method to parse an HTML document into a first-order term. Note that a first-order term is corresponding to a labeled tree in noe-to-one.

**Definition 2** An HTML document is corresponding to an ordered labeled tree over an alphabet consisting of all tag names, tag attributes, text attributes. The correspondence is defined as follows.

1. Any empty tag <e> is equal to the term $\langle e \rangle$ which is a function symbol with arity zero.
2. For each beginning tag <t>, end tag </t>, and a string $w$ not containing any tag, the string <t>$w$</t> is equal to the term $\langle t \rangle(w)$. If tag attributes $b_1, \ldots, b_n$ are defined in <t>, then the term is $\langle t \rangle(b_1, \ldots, b_n, w)$.
3. For each beginning tag <t>, end tag </t>, and a string $w$, the string <t>$w$</t> is equal to the term $\langle t \rangle(f(w))$, where $f(w)$ is the term recursively defined in the above. Moreover, if tag attributes $b_1, \ldots, b_n$ are defined in <t>, then the term is $\langle t \rangle(b_1, \ldots, b_n, f(w))$.

Let $P_t$ be the term expresstion of a page $P$ defined in the above. Next, we define the notion of the label $L_t$ of $P_t$ corresponding to the label $L$ of $P$. Since $P_t$ equal to an ordered labeled tree, without lossing generality, we can identity $P_t$ with the tree. Then, let $N_t = \{0, \ldots, n\}$ be the set of node of $P_t$, where 0 is the root.

**Definition 3** Let $(P, L)$ be a pair of page and its label and let $L = \{t_1, \ldots, t_m\}$, where $t_i = \langle \ldots, \langle b_{i,j}, e_{i,j} \rangle, \ldots \rangle$. Then, the label $L_t$ of $P_t$ is defined as to $L_t = \{n_1, \ldots, n_m\}$, where $n_i = \langle \ldots, n_{i,j}, \ldots \rangle$ and $n_{i,j} \in N_t$ such that $label(n_{i,j})$ in $P_t$ is the text attribute from $b_{i,j}$ to $e_{i,j}$ in $P$.

Intuitively, $n_{i,j}$ in the above denotes the node number whose node label corresponds to the text attribute indicated by the positions $b_{i,j}$ and $e_{i,j}$. Thus, for a $P_t$ of a page $P$, the label $L_t$ denotes the set of sequence of such node number to be wrapped. Thus, we next expand the notion of LR-wrapper to extract such node labels from $P_t$ and introduce the *T-wrapper* class.

$exec_T$(T-wrapper $\langle p_1, \ldots, p_k \rangle$, page $P_t$)
$p_i = \langle p_{i,1}, \ldots, p_{i,j}, x_i \rangle$
m=0
while there are more occurrence of $p_{1,1}$ in $P_t$
  m=m+1
  for each $p_i \in \{p_1, \ldots, p_k\}$
    find a path matches with $p_{i,1} \cdots p_{i,j}$ on $n_i$ in $P_t$;
    save the label of $x_i$-th child of $n_i$ as $n_{m,i}$
return the label $\{\ldots, \langle n_{m,1}, \ldots, n_{m,k} \rangle, \ldots\}$

Table 1: The procedure specifies how an T-wrapper is executed.

**Definition 4** *The T-wrapper is of the form* $W = \langle p_1, \ldots, p_k \rangle$ *such that* $p_i = \langle p_{i,1}, \ldots, p_{i,j}, x_i \rangle$*, where all* $p_{i,1}, \ldots, p_{i,j}$ *are node labels and* $x_i$ *is a positive integer.*

To explain the semantics of the T-wrapper, we define some notation for trees. Let $T$ be an ordered labeled tree and let $\{0, \ldots, n\}$ be the set of nodes of $T$, where 0 is the root. The $label(n_i)$ denotes the label of the node $n_i$. A path in $T$ is a sequence of nodes of the form $(n_1, \ldots, n_k)$ such that $n_i$ is a child of the $n_{i-1}$. A sequence $(w_1, \ldots, w_k)$ of texts is said to *match with* the path $(n_1, \ldots, n_k)$ in $T$ if $(w_1, \ldots, w_k) = (label(n_1), \ldots, label(n_k))$. We also said that the sequence matches with the path on the node $n_k$ more precisely.

Given a page $P_t$, for each $p_i \in \{p_1, \ldots, p_k\}$, the procedure first try to find a path which matches with $(p_{i,1}, \ldots, p_{i,j})$ on $n_i$ in $P_t$. Next, the procedure extract the label of the $x_i$-th child of $n_i$. While there are more tuples not searched, the procedure find the next path. The procedure is exactly defined in Table 1.

For example, the HTML document $P$ not extracted by LR-wrapper in Example 2 can be handled by the T-wrapper $W = \langle$`<html>`, `<body>`, `<h3>`, `<a>`, $2\rangle$. The $P_t$ is the tree having the path `<html>`$\cdot$`<body>`$\cdot$`<h3>`$\cdot$`<a>` and the last node $n$ of such a path has a child whose label is an email address. Moreover, such children are exactly the 2nd child of $n$. Note that the first child of $n$ is the tag attribute of the tag `<a>`. Thus, the T-wrapper can extract only the email addresses form $P_t$. The model for extracting text attributes and learning problem of such wrappers are briefly summarized as follows.

- A page is represented by an ordered labeled tree. T-wrapper $W$ is a sequence of $p_i$ and $p_i = \langle q_i, x_i \rangle$. The $q_i$ denotes a sequence of some texts and $x_i$ is an integer.
- The $W$ finds the first occurrence of a path that matches with $q_i$ and extract the label of the next node. The node is the $x_i$-th children.
- The example given to the learning algorithm is a set of $(P_n, L_n)$, where $L_n$ is the label of the page $P_n$. The goal of the learning algorithm is to output such a T-wrapper $W$ such that $W(P_n) = L_n$.

We present the algorithm to compute a T-wrapper $W$ from given pairs $(P_i, L_i)$ of pages and labels. We assume

procedure `Learn`$_T(E = \{\ldots, (P, L), \ldots\})$
Let $N$ be the set of nodes
Let $L = \{n_i \mid n_i = \langle n_{i,1}, \ldots, n_{i,k} \rangle,\ n_{i,j} \in N,\ 1 \le i \le m\}$
Initialize $W = (q_1, \ldots, q_k)$ such that
  $q_i = (\ )$ for all $1 \le i \le k$

For each $(P, L) \in E$,
  let $(q'_1, \ldots, q'_k)$ be output of `path-finder`$(P, E)$
  compute the longest common suffix of $q_i$ and $q'_i$ and
    let it be the latest $q_i$

If a $q_i$ is empty sequence, then return FALSE
Else return $W$

procedure `path-finder`$(P, E)$
For each $1 \le j \le k$
  compute all paths $p(n_{1,j}), \ldots, p(n_{m,j})$ and
  compute the longest common path $p_j$ of them
    If $|p_j| = 0$, then return FALSE
    else if $n_{1,j}$ is the $x_i$-th child, then
      return $q_j = (p_j, x_j)$ as a candidate for $q_j$

Table 2: The algorithm to learn T-wrapper $W$ such that $W(P) = L$.

that pages $P_i$ are parsed into tree expressions. We also assume a standard partial order of node numbers as follows. Let $n$ and $m$ be two nodes of a tree $T$. (1) If $m$ is a descendant of $n$, then $n < m$. (2) Else if $n$ and $m$ are children of a node such that $n$ is in left for $m$, then $n < m$. (3) Else if $n, m$ are descendants of $n', m'$,respectively such that $n' < m'$, then $n < m$. Let $n$ be a node in $T$. Then, let $p(n)$ denote the path from the root to the parent of the $n$. Let $p(n) = (n_1, \ldots, n_i)$ and $p(m) = (m_1, \ldots, m_j)$. Then, a sequence $(w_1, \ldots, w_k)$ is said to be a *common suffix* of $p(n)$ and $p(m)$ if $label(n_{i-k'+1}) = label(m_{j-k'+1}) = w_{k'}$ for all $1 \le k' \le k$. The longest common suffix of $p(n)$ and $p(m)$ is the longest one of them.

Here, we estimate of the time to compute a T-wrapper for given example $E = \{\ldots, (P, L), \ldots\}$. Let $K$ be the number of attributes in a tuple of $L$ and let $L = \{n_i \mid n_i = \langle n_{i,1}, \ldots, n_{i,K} \rangle,\ 1 \le i \le m\}$. Let $|P|$ be the size of $P$, that is the number of nodes in $P$. The $|P|$ is at most $d \cdot h$, where $d$ is the length of the longest path and $h$ is the number of leaves of $P$.

We can find each node in $P$ in $O(d)$ by binary search. Thus, we can make the pointers from attributes in $L$ from nodes in $P$ in $O(m \cdot d \cdot K) = O(|P| \cdot K)$ because $K + m \le h$ and $d \cdot h \le |P|$. By using this pointers, we can find the longest common suffix of $m$ nodes in $O(|P|)$, which is the delimiter of the $i$-th attribute ($1 \le i \le K$). Thus, for each $(P, L) \in E$, the time to find all delimiters for $L$ is $O(|P| \cdot K)$.

## Concluding remark

We define a new wrapper class for extracting attributes from HTML documents. We present the effective algorithm to compute such wrappers from given pages and the positions of required attributes in the pages. This wrapper class can extract attributes for some cases which the Kushmerick's wrapper class can not. However, it is not shown the relation ship between our wrapper class and the Kushmerick's. Now we are verifying the performance of our algorithm with respect to many Web sites.

## References


Angluin, D. 1988. Queries and concept learning. *Machine Learning* 2:319–342.

Bray, T., Paoli, J., and Sperberg-McQueen, C. M. 1998. Extensible Markup Language (XML) Version 1.0. *W3C Recommendation 1998.* `www.w3.org/TR/REC-xml`

Buneman, P., Davidson, S., Hillebrand, G., and Suciu, D. 1996. A query language and optimization techniques for unstructured data. *University of Pennsylvania, Computer and Information Science Department, Technical Report MS-CIS 96-09.*

Clark, J. 1999. XSL Transformations (XSLT) Version 1.0. *W3C Recommendation 1999.* `www.w3.org/TR/xslt`

Dershowitz, N. and Jouannaud, J.-P. 1990. Rewrite Systems. *Chapter 6, Formal Models and Semantics, Handbook of Theoretical Computer Science*, Vol. B, Elsevier.

Drewes, F. 1996. Computation by tree transductions. *Ph D. Thesis*, University of Bremen, Department of Mathematics and Informatics.

Hirata, K, Yamada, K., and Harao, M. 1999. Tractable and intractable second-order matching problems. *Proc. 5th Annual International Computing and Combinatorics Conference.* LNCS 1627:432–441.

Ikeda, D. 1999. Characteristic sets of strings common to semi-structured documents. *Proc. 2nd International Conference on Discovery Science.* LNAI 1721:13–23.

Kakada, T. 1998. Compact HTML for small information appliances. *W3C NOTE 09-Feb-1998.* `www.w3.org/TR/1998/NOTE-compactHTML-19980209`

Khardon, R. 1998. Learning function-free Horn expressions. *Proc. COLT'98*, 154–165.

Kilpelainen, P and Mannila, H. 1995. Ordered and unordered tree inclusion. *SIAM J. Comput.* 24: 340–356.

Kushmerick, N. 2000. Wrapper induction: efficiency and expressiveness. *Artificial Intelligence* 118:15–68.

Sakamoto, H., Arimura, H., and Arikawa, S. 2000. Identification of tree translation rules from examples. *Proc. 5th International Colloquium on Grammatical Inference.* LNAI 1891:241–255.

Valiant, L. G. 1984. A theory of the learnable. *Commun. ACM* 27:1134–1142.