Expressive Power of Tree and String Based Wrappers

Ikeda, Daisuke
Computing and Communications Center, Kyushu University

Yamada, Yasuhiro
Graduate School of Information Science and Electrical Engineering, Kyushu University

Hirokawa, Sachio
Computing and Communications Center, Kyushu University

https://hdl.handle.net/2324/2960

出版情報：Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), pp.21-26, 2003-08. International Joint Conference on Artificial Intelligence
バージョン：
権利関係：

Expressive Power of Tree and String Based Wrappers

Daisuke Ikeda
Computing and Communications Center,
Kyushu University, Fukuoka 812-8581, Japan
daisuke@cc.kyushu-u.ac.jp

Yasuhiro Yamada
Graduate School of Information Science
and Electrical Engineering,
Kyushu University, Fukuoka 812-8581, Japan
yshiro@matu.cc.kyushu-u.ac.jp

Sachio Hirokawa
Computing and Communications Center,
Kyushu University, Fukuoka 812-8581, Japan
hirokawa@cc.kyushu-u.ac.jp

Abstract

There exist two types of wrappers: the string based wrapper such as the LR wrapper, and the tree based wrapper. A tree based wrapper designates extraction regions by nodes on the trees of semi-structured documents. The tree based wrapper seems to be more powerful than the string based one. There exist, however, many HTML documents on the Web such that a standard tree based wrapper fails to extract contents because they are structured by presentational tags, punctuation symbols, and white spaces. Moreover, some of such documents use multi-byte characters for structuring. To treat some of such documents, we propose automatic wrapper generation based on common substring detection and to use input documents without any modification. In this framework, a part of text elements including white spaces and multi-byte characters can be a part of a wrapper. We show the superiority such wrappers to usual wrappers created after document are parsed and modified. However, there still exist HTML documents such that wrappers with text elements fail to extract contents. Thus, we propose another class of wrappers, called the regional tree wrapper, which utilize the tree structures of input documents as well as addressing functions on strings.

1 Introduction

There are useful information hidden in enormous documents on the Web. It is difficult, however, to restructure them and integrate contents on different sites because the documents do not have rigid structures like database systems. To extract contents from semi-structured documents on the Web, we need the wrapper which is a procedure to divide contents according to the their types and to extract them.

Due to the enormous pages on the Web, writing wrappers manually is tedious and error-prone. There have been a lot of researches on semi- or full- automatic wrapper generation algorithms: some of them are machine learning based approaches using training examples [Cohen and Fan, 1999; Cohen and Jensen, 2002; Kushmerick *et al.*, 1997; Muslea *et al.*, 1998; Sakamoto *et al.*, 2001] and some of them find regularities among given documents [Embley *et al.*, 1999; Buttler *et al.*, 2001; Crescenzi *et al.*, 2001; Yamada *et al.*, 2001; 2002]. XWrap [Liu *et al.*, 2001] and Lixto [Baumgartner *et al.*, 2001] provide graphical user interfaces to indicate a region which should be extracted. A good survey of data extraction from the Web is found in [Laender *et al.*, 2002].

Some of these algorithms first find substrings common to a set of given documents [Crescenzi *et al.*, 2001; Yamada *et al.*, 2001; 2002]. Common substrings correspond to templates of documents, so we then identify the regions of contents. Finding common substrings framework enables an algorithm to generate wrappers full-automatically. Another benefit of such algorithms is independence from markup and natural languages. For example, consider the following example.

```
<tag>■ Daisuke Ikeda </tag>
<tag>■ Yasuhiro Yamada</tag>
```

This is a part of HTML document. "■" is a multi-byte character and a part of a text element. This is used for an item heading, so it should be treated as a part of a template. An algorithm based on common substrings easily deals with templates which contains tags and parts of text elements.

The above "■" example seems to be appropriate for wrappers based on the string. Each name is extracted if it is surrounded between "<tag> ■" and "</tag>." The other type of wrapper is based on the tree structure. Selecting a class of wrapper is important. A class of simple wrappers is easy to be generated but can not describe many Web sites. A class of complex wrappers can deal with large number of Web sites but it is difficult to generate such a wrapper automatically. We call a string (resp. tree) based wrapper a *string wrapper* (resp. *tree wrapper*) for short.

Algorithms which generate string wrappers treat a semi-structured document just as a string [Ashish and and Knoblock, 1997; Crescenzi *et al.*, 2001; Kushmerick *et al.*, 1997; Lerman *et al.*, 2001; Yamada *et al.*, 2001; 2002]. Basically, these wrappers consist of strings surrounding contents which are variable. The simplest string wrapper is an LR wrapper [Kushmerick *et al.*, 1997] consisting of a set of pairs of left and right *delimiters*. In other words, a pair of delimiters surrounds one type of data. The string wrapper is too simple to describe complex structures such as tree structures[1].

Algorithms for tree wrappers utilize the tree structure of semi-structured documents [Baumgartner *et al.*, 2001; Buttler *et al.*, 2001; Liu *et al.*, 2001; Muslea *et al.*, 1998; Sakamoto *et al.*, 2001]. Basically, such algorithms find nodes on a tree which designate the regions of data to be extracted. Some of them are machine learning and require training examples [Cohen and Fan, 1999; Cohen and Jensen, 2002; Sakamoto *et al.*, 2001]. The algorithm in [Buttler *et al.*, 2001] utilizes regularities of trees, such as depths of contents nodes, and find record boundaries automatically.

In this paper, we discuss expressive power of wrappers. We consider full automatic wrapper generation and assume that a part of text elements can be a part of template. Many string and tree wrappers are studied in this viewpoint [Kushmerick, 2000]. But, considered wrappers are only string ones.

In general, the tree wrapper seems to be more expressive than the string wrapper if input semi-structured documents are well-structured like XML documents. In this case, it is enough for the tree wrapper to point a node or set of nodes to be extracted. However, still huge amount of documents on the Web are HTML files and many of them are ill-structured. In fact, many HTML files allow for different types of contents or multiple contents to be mixed in one structure tag. These contents are separated by presentational layout tags such as <p> and <br>, white spaces[2], and punctuation symbols.

For such ill-structured documents, pointing nodes on a tree structure is not sufficient and we need to divide a portion inside one tag into more small pieces according to their data type. Semi automatic or interactive methods can deal with such documents. For example, Lixto [Baumgartner *et al.*, 2001] has a predicate *subtext*, so it describes contents formatted such characters. However, we want to extract data from such documents automatically. Thus, we pay attentions to both string and tree wrappers, especially string wrappers which can include any characters, such such as multi-byte characters, white spaces, comment tags, and layout tags.

An Emacs lisp package “shimbun[3]” also supports the importance of white spaces and comments. The package enables some mail/news readers to see Web pages in a mail/news manner. Target Web pages of the package are those on online news sites[4], mail archives, or bulletin boards. In other words, shimbun is a set of wrappers written by hand. White spaces and comments are used to indicate regions of contents in many wrappers in shimbun package.

This paper is organized as follows: First, we introduce the LR wrapper in Section 2 in the framework of common substring detection. In this section, we show sample LR wrappers which are created from real data on the Web and include characters of text elements. Then, we give some definitions of tree wrappers in the literatures in Section 3. In this section, we show some HTML files gathered from the Web such that they are not applicable to the string wrapper without a part of text elements and even to the tree wrapper. Finally, we propose a new wrapper class which works well for such HTML files. A proposed wrapper, we call it a regional tree wrapper, have both features from the string wrapper and the tree wrapper.

2 Wrapper Generation by Common Parts Detection

In this section, first we introduce the LR wrapper according to [Kushmerick *et al.*, 1997; Kushmerick, 2000]. Then, we define the wrapper generation problem as common substring detection. This framework enables a wrapper generation algorithm to work full-automatically. The wrapper is very simple and seems to be week for real and diverse data on the Web. However, we show by example that the LR wrapper is enough powerful if we treat input documents as is.

We assume some alphabet Σ with fixed size for each set of input documents.

Definition 1 ([Kushmerick *et al.*, 1997]) *An LR wrapper is a K-tuple $(l_1, r_1, \ldots, l_K, r_K)$, where K is some constant and $l_i, r_i \in \Sigma^*$ for any $1 \leq i \leq K$.*

Each l_i (resp. r_i) is called a left (resp. right) delimiter.

Although any strings can be a delimiter by Definition 1, many implementations restrict a delimiter to be a sequence of tags. There were no such restrictions in [Kushmerick *et al.*, 1997; Kushmerick, 2000], but only the ASCII character set was considered as Σ. On the other hand, Yamada *et al.* implemented a wrapper generation algorithm in which any characters can be parts of a delimiter [Yamada *et al.*, 2002] and these characters are not restricted to the ASCII character set. They use Unicode[5] as a character set, so that the algorithm deals with any documents coded in Unicode. Moreover, the algorithm can generate wrappers including a part of text elements which can be written in any of natural languages if the part is coded in Unicode (see Table 1).

A *pattern* is a string over Σ and V, where V is infinite set of *variables*. Let $p = w_0 x_1 w_1 \cdots x_m w_m$ be a pattern, where $x_i \in V (1 \leq i \leq m)$ and $w_i \in \Sigma^* (0 \leq i \leq m)$. Then the sequence $s = (w_0, \ldots, w_m)$ of strings over Σ is called a *template*.

Definition 2 *The wrapper generation problem is, given a set of positive documents, to find a template for the set.*

Unlike the machine learning of pattern languages, we do not consider variables.

[1] Some classes of wrappers considered in [Kushmerick, 2000] and RoadRunner [Crescenzi *et al.*, 2001] allow nesting and repetition to describe the complex structure.

[2] A *white space* denotes a space, tab, or new-line character.

[3] `http://emacs-w3m.namazu.org/`

[4] The news site is the main target of “shimbun” which stands for a new paper in Japanese.

[5] `http://www.unicode.org/`

According to this framework, Yamada *et al.* developed two automatic wrapper generation algorithms [Yamada *et al.*, 2001; 2002]. Both algorithms find pairs of delimiters and generate LR wrappers. These delimiters are parts of a template. Both algorithms do not require any training examples due to the algorithm developed in [Ikeda *et al.*, 2001]. An output of the algorithm corresponds to a set of training examples. In other words, the algorithm in [Ikeda *et al.*, 2001] automatically indicates positions of the contents in given documents.

We briefly explain how the algorithm in [Ikeda *et al.*, 2001] works. It outputs a pair (n, a) of integers, where $n \geq 2$ is a length of strings and $0 \leq a \leq 100$ is a percentage. The algorithm finds a common parts of a given set of documents. A common parts is expressed as the following: counts all substrings with length n of given documents, sorts them by their occurrences in decreasing order, and then substrings in the first a percent constitute common parts. The algorithm decides appropriate (n, a) automatically using the new notion "alternation counts." An alternation count on some (n, a) is the number of boundaries between common an uncommon parts.

The algorithm initially sets $(n, a) = (2, 1)$, then it increases n or a by one. It compares three alternation counts on current (n, a), $(n, a+1)$, and $(n+1, a)$, then chooses one of them provided the smallest alternation count. If the current value gives the smallest alternation count, then the algorithm stops and outputs current (n, a).

Created wrappers by both algorithms [Yamada *et al.*, 2001; 2002] are string based, but they are different from standard string wrappers on the following points: (1) These algorithms treat input documents as-is. No modification on input documents is applied. Comments and tags with some spell mistake can be left. Useful information hidden in comment tags or in attributes of tags were found by these algorithms. (2) The search space of delimiters is not restricted to a set of tag sequences. Both algorithms do not utilize the grammar of any markup languages. Instead, they try to find substring common to the documents. Generated delimiters can include strings in contents parts of HTML documents. In fact, sometimes, such strings were multi-byte characters if given documents contains such characters (see Table 1).

In the viewpoint of expressiveness, there are important difference between two algorithms in [Yamada *et al.*, 2001; 2002]. A delimiter considered in [Yamada *et al.*, 2002] can include white spaces. On the other hand, the algorithm in [Yamada *et al.*, 2001] converts successive white spaces into a single space, so that tab and new-line characters were ignored in the result. In HTML and XML, a sequence of successive white spaces equals to a single space, so a user can add white spaces for readability of a document. For example, "<tagA>Hello</tagA>" is the same as the following notation:

```
<tagA>
  Hello
</tagA>
```

Therefore, most algorithms ignore successive white spaces when they parse HTML/XML files. This difference yields interesting pairs of delimiters such as "␣(␣" and "␣)</p>". These delimiters were found in the data set of XML files gathered from "Mainichi INTERACTIVE NewsML"[6]. However, expressive power of white spaces was not discussed in [Yamada *et al.*, 2002].

Table 1 is the wrapper created on HTML documents gathered from "Sankei Web,"[7] which is one of the major online news sites in Japan. We see that an HTML page on online news outlet contains an instance of an article record. In general, an article contains the body of the article, the date and hour, the heading, and the credit. Sometimes, additional subheadings appear. "■" in the left delimiter of Headline is a multi-byte character.

From Table 1, we see that white spaces play an important role. For example, the body of an article of this site is surrounded by "<BLOCKQUOTE>\n\n" and "<p>\n\n\n." The right delimiter shows that two blank lines after a <p> tag are used to indicate the end of an article.

A weakness of the LR wrapper is shown in Example 1, where "Ikeda" and "Yamada" are the contents but "COMMERCIAL" is not.

Example 1

```
<tagA>
 <a href="http://abc.edu/">Ikeda</a>
 <a href="http://def.ac.jp/">Yamada</a>
</tagA>
 ...
<a href="http://hi.com/">COMMERCIAL</a>
```

A string wrapper generation algorithm can not generate a wrapper extracting the above contents because attribute values of "href" are different from each other and delimiters turn out to be """>" and "</a>." But these delimiters extract "COMMERCIAL" too.

One solution for the above example is to use white spaces and don't-care symbol "*", then we have the pair of delimiters "\n␣<a href="*">" and "</a>." However, this does not work well if all <a> tags have the same depth in the tree representation, or the indentation is not used. Another solution is to use the tree structure described in Section 3.

3 Tree Wrappers

In this section, we consider expressive power of tree wrappers and some of their limitations. Basically, such a tree wrapper recognizes regions by a node of the tree structure. A node is represented by the path expression, which is a sequence of tags from the root to the correspondent tag. A standard expression is XML Path Language (XPath)[8]. XPath is a language for addressing parts of an XML document, designed to be used by XSLT. For example, the contents in Example 1 of Section 2 are represented by "tagA/a."

Tree wrappers seem to be more powerful than string wrappers. In fact, any string wrappers proposed in the literature can be expressed by some tree wrapper if input documents are well-structured and one tag at the lowest level does not

[6] http://www.mainichi.co.jp/digital/newsml/

[7] http://www.sankei.co.jp/

[8] http://www.w3.org/TR/xpath

Table 1: Wrapper created from “Sankei Web.” Each row consists of left (upper) and right (bellow) delimiters. A row corresponds to one data type

Data Type	Created wrapper
Headline	`<font color="#8b0000">■</font><b>`
	`</font><b>`
Sub headline	`\n<b>`
	`<BR></b>`
Body text	`<BLOCKQUOTE>\n\n`
	`<p>\n\n\n`

contain several types of data simultaneously. However, there are many HTML files which do not obey the above assumption.

Figure 1 is a simplified search result from “Google”[9]. Indentation is for readability. The whole result is surrounded by “<p class="g">” and “</p>.” The node “p/a” corresponds to the page title and some words in the title are emphasized by bold face if they are query words. Next, the result contains some parts of the page with the query words’ highlighting. In Figure 1, two lines just bellow <font size="-1"> correspond to that. We can also find such a structure at “Lycos”[10].

```
<p class="g">
  <a href="URL">
    <b>KEYWORD</b>-TITLE OF A PAGE
  </a><br>
  <font size="-1">
    <b>KEYWORD</b>-....,<br>
    including <b>KEYWORD</b>- <br>
    <span class="f">
      <font size="-1">Description:</font>
    </span>
    Foundation which....
    ...
  </font>
</p>
```

Figure 1: One search result on Google

It is impossible to represent these two lines by a simple node representation because <font size="-1"> contains another types of data. <b> tags increase the difficulty to represent desired nodes because they are used to emphasize query words and so the number of appearance of query words is not fixed in advance.

To represent these two lines precisely, we need an additional substring function that can point them by “from just after <font size="-1"> to just before the second occurrence of <br>,” or a span function that can point them “from the first child of <font size="-1"> to the previous node of <span class="f">.”

Next, we consider an article file on “washingtonpost.com – News Front –[11].” In each article file, a heading of the article is in the title tag like “<title> *Article’s Heading* (washingtonpost.com)</title>.” “(washingtonpost.com)” is immovable, that is, all title tags contain it, while *Article’s Heading* is variable. Thus, it is necessary for a wrapper to extract the part “Article’s Heading” only. However, it is impossible for the tree wrapper without string based functionalities. Articles on “Mainichi Shimbun”[12] also have title tags line <title>Mainichi INTERACTIVE *heading*</title>.

We see the date of the article at the end of the article’s body in an article of many news sites, like the following example. This is taken from “Mainichi Shimbun.”

Example 2

```
<font size="4">
...
<p> PARAGRAPH </p>
<p> PARAGRAPH </p>
<p> [Mainichi 03/05] ( 2003-3-5-22:56 ) </p>
</font>
```

The pair “␣(␣” and “␣)</p>” of delimiters extracts the date and hour of an article, such as “2003-3-5-22:56.” On the other hand, “03/05” part is not extracted. Because these files are gathered in a day and this part of every file is constantly “03/05,” so “03/05” is treated as a part of structures.

In addition to Mainichi Shimbun, we also find similar articles on “YOMIURI ON-LINE”[13] and “asahi.com”[14].

Next, we again consider result pages dynamically created at search engines. In a result page of “Yahoo!”[15] and “Lycos”, the short description of a page follows “␣-”. Such a heading symbol is often used for readability. The symbol “■’ in Table 1 is also such a heading symbol. We want to remove these symbols from the contents since such heading symbols have nothing to do with the contents. “AltaVista”[16] also used “URL:” to show the URL for a page[17].

From above observations, we know that there two types of mixture in one tag. One is mixture of different types of data, such as the date following a news article. The other is

[9] http://www.google.com/

[10] http://www.lycos.com/

[11] http://www.washingtonpost.com/

[12] http://www.mainichi.co.jp/

[13] http://www.yomiuri.co.jp/

[14] http://www.asahi.com/

[15] http://www.yahoo.com/

[16] http://www.altavista.com/

[17] Currently, AltaVista does not use such symbols.

mixture of data and a part of templates, such as "(washingtonpost.com)." It is difficult to divide different types of data automatically. So, we only consider the latter case.

Some algorithms for the tree wrapper in the literature can deal with such templates in a text elements. For example, Lixto [Baumgartner *et al.*, 2001] has a predicate *subtext* to extract substrings from a text element. But, it is an interactive wrapper generation method and regions to be extracted are specified by a user. The authors do not know a wrapper generation algorithm which generates a tree wrapper and treats a part of text elements as a part of the wrapper *full-automatically*.

4 Combining String and Tree Wrappers

In the previous two sections, we described advantages and disadvantages of both string and tree wrappers. In this section, we introduce another wrapper class which comprises these two wrapper classes.

As described in Section 3, we need additional substring and span functions which are used to specify parts of a node instead of the entire node. Therefore, a subset of XML Pointer Language (XPointer)[18] is a good candidate for such a class of wrappers.

XPointer is the language to be used as the basis for a fragment identifier for any URI reference. XPointer have functions to address a range of nodes as well as those to address a node. XPointer also have string based functions to indicate a range of nodes. These functions are enough powerful to handle documents described in this paper.

For a string x, a substring w of x, and an integer i ($1 \leq i \leq |x|$), a *left* (resp. *right*) *position* of w on x is the first index $j + |w| - 1$ (resp. j) such that $x[j..j + |w| - 1] = w$. For example, consider a string $w_l x w_r$, where x is a content and w_l, w_r are delimiters. x is pointed by the right of w_l and the left of w_r.

For a node n and a string w, a *left (resp. right) position* of w under n is a left (resp. right) position of w on the string defined by n.

We introduce a *region* to express a fragment. A region is a pair (l, r) of nodes, positions, or their combination. If l (resp. r) is a position, it is a left (resp. right) position. Now we define a regional tree wrapper as follows.

Definition 3 (regional tree wrapper) *A regional tree wrapper is a set of rules, where a rule is either is a node or region.*

The semantic of regional tree wrapper is as follows. If a rule is a node, the contents under the node are extracted. For a region (l, r), it points the whole string from a node or position l to a node or position r.

5 Conclusion

We discussed expressive power of string and tree based wrappers according to the framework of common substring detection. Especially, we focused on string wrappers with a part of text elements including white spaces and multi-byte characters. We showed HTML files to which standard tree wrappers are not applicable. Then, we proposed the regional tree wrapper which comprises both string and tree wrappers.

The regional tree wrapper is a small subset of XPointer. XPointer is a rich language, so it might have some functions or definitions which are useless for the purpose of wrapper generation. It is important future works to prove expressive power of the proposed class of wrappers by theoretically and empirically.

[18] `http://www.w3.org/TR/xptr`

References

[Ashish and and Knoblock, 1997] Naveen Ashish and Craig Knoblock. Wrapper Generation for Semi-structured Internet Sources. In *Proceedings of Workshop on Management of Semistructured Data*, 1997.

[Baumgartner *et al.*, 2001] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web Information Extraction with Lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 119–128, September, 2001.

[Buttler *et al.*, 2001] David Buttler, Ling Liu, and Calton Pu. A Fully Automated Object Extraction System for the World Wide Web. In *Proceedings of the 21th International Conference on Distributed Computing Systems*, pages 361–370, 2001.

[Cohen and Fan, 1999] William W. Cohen and Wei Fan. Learning Page-Independent Heuristics for Extracting Data from Web Pages. In *Proceedings of the 8th International World Wide Web Conference*, 1999.

[Cohen and Jensen, 2002] William W. Cohen and Lee S. Jensen. A Structured Wrapper Induction System for Extracting Information from Semi-structured Documents. In *Proceedings of IJCAI 2001 Workshop on Adaptive Text Extraction and Mining*, 2001.

[Crescenzi *et al.*, 2001] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109–118, September, 2001.

[Embley *et al.*, 1999] David W Embley, Yuan Jiang, and Yiu-Kai Ng. Record-Boundary Discovery in Web Documents. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 467–478, 1999.

[Ikeda *et al.*, 2001] Daisuke Ikeda, Yasuhiro Yamada, and Sachio Hirokawa. Eliminating Useless Parts in Semi-structured Documents using Alternation Counts. In *Proceedings of the 4th International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence, Vol. 2226, pages 113–127, Springer-Verlag, November, 2001.

[Kushmerick, 2000] Nicholas Kushmerick. Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence*, Vol. 118, pages 15–68, 2000.

[Kushmerick *et al.*, 1997] Nickolas Kushmerick, Daniel S. Weld and Robert B. Doorenbos. Wrapper Induction for

Information Extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 729–737, August, 1997.

[Laender *et al.*, 2002] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva and Juliana S. Teixeira, Automatic Data Extraction from Lists and Tables in Web Sources. SIGMOD Record Vol. 31, No. 2, pages 84–93, 2002.

[Lerman *et al.*, 2001] Kritina Lerman, Craig Knoblock and Steve Minton. Automatic Data Extraction from Lists and Tables in Web Sources. In *Proceedings of IJCAI 2001 Workshop on Adaptive Text Extraction and Mining*, 2001.

[Liu *et al.*, 2001] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *Proceedings of the 16th International Conference on Data Engineering*, pages 611–621, 2000.

[Muslea *et al.*, 1998] Ion Muslea, Steve Minton, and Craig Knoblock. STALKER: Learning Extraction Rules for Semistructured Web-based Infomation Sources. In *Proceedings of AAAI-98 Workshop on AI and Information Integration*, pages 74–81, 1998.

[Sakamoto *et al.*, 2001] Hiroshi Sakamoto, Hiroki Arimura, and Setsuo Arikawa. Extracting Partial Structures from HTML Documents. In *Proceedings of the 14th International Florida Artificial Intelligence Research Symposium (FLAIRS'2001): Knowledge Discovery and Data Mining*, AAAI Press, pages 264–268, 2001.

[Yamada *et al.*, 2001] Yasuhiro Yamada, Daisuke Ikeda, and Sachio Hirokawa. SCOOP: A Record Extractor without Knowledge on Input. In *Proceedings of the 4th International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence, Vol. 2226, pages 428–487, Springer-Verlag, November, 2001.

[Yamada *et al.*, 2002] Yasuhiro Yamada, Daisuke Ikeda, and Sachio Hirokawa. Automatic Wrapper Generation for Multilingual Web Resources. In *Proceedings of the 5th International Conference on Discovery Science*, Lecture Notes in Computer Science, Vol. 2534, pages 332–339, Springer-Verlag, November, 2002.