



# Web Data Extraction

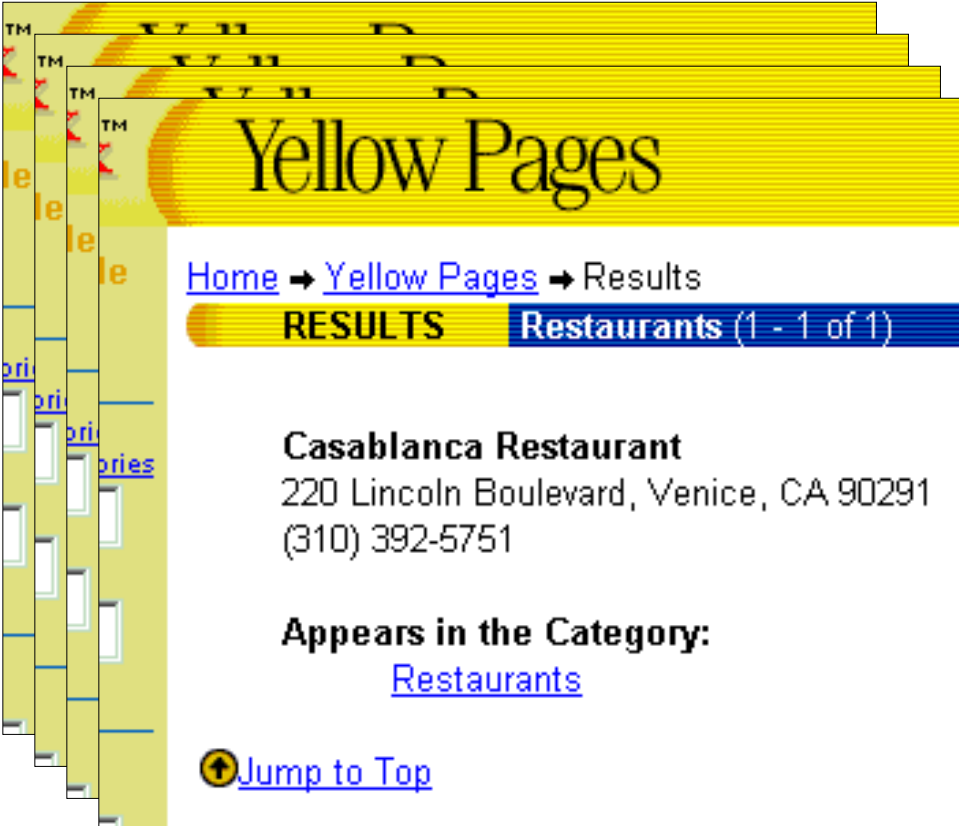
---

Craig Knoblock

University of Southern California

This presentation is based on slides prepared  
by Ion Muslea and Kristina Lerman

# Extracting Data from Semi-structured Sources



Yellow Pages

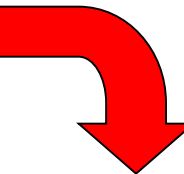
[Home](#) → [Yellow Pages](#) → Results

**RESULTS** Restaurants (1 - 1 of 1)

**Casablanca Restaurant**  
220 Lincoln Boulevard, Venice, CA 90291  
(310) 392-5751

Appears in the Category:  
[Restaurants](#)

[↑ Jump to Top](#)



<b>NAME</b>	Casablanca Restaurant
<b>STREET</b>	220 Lincoln Boulevard
<b>CITY</b>	Venice
<b>PHONE</b>	(310) 392-5751



# Approaches to Wrapper Construction

- Manual Wrapper Construction
- Learning-based Wrapper Construction
- Automatic Wrapper Construction



# Grammar Induction Approach

---

- Pages automatically generated by scripts that encode results of db query into HTML
  - Script = grammar
- Given a set of pages generated by the same script
  - Learn the grammar of the pages
    - Wrapper induction step
  - Use the grammar to parse the pages
    - Data extraction step



# RoadRunner: Towards Automatic Data Extraction from Large Web Sites by Crescenzi, Mecca, & Merialdo



# RoadRunner Overview


---

- Automatically generates a wrapper from large web pages
  - Pages of the same *class*
  - No dynamic content from javascript, ajax, etc
- Infers source schema
  - Supports nested structures and lists
  - Extracts data from pages
- Efficient approach to large, complex pages with regular structure

# Example Pages

- Compares two pages at a time to find similarities and differences
- Infers nested structure (schema) of page
- Extracts fields

<http://www.csbooks.com/author?John+Smith>



The screenshot shows the CSbooks.com website with search results for books by John Smith. The page layout includes a left sidebar with 'Browse Subjects' (Artificial Intelligence, Artificial Life, Computational, Graphics, Computer Graphics, Database Theory, Distributed Computing, Internet Computing, Software Engineering, etc.), a main content area with a search result for 'Database Primer' (1999, 2000 editions), and a 'Book Description' section. The search results list the book title, edition, format, and price, with a 'View Details' button next to each entry.

<http://www.csbooks.com/author?Paul+Jones>



The screenshot shows the CSbooks.com website with search results for books by Paul Jones. The page layout is similar to the previous screenshot, with a left sidebar, a main content area showing search results for 'XML at Work' (1999, 2000 editions) and 'HTML and Scripts' (1999, 2000 editions), and a 'Book Description' section. The search results list the book title, edition, format, and price, with a 'View Details' button next to each entry.

# Extracted Result

total number of SCHEMAs found: 1

Schema Number 1: A | B | C | D | E | F | Total Time: 0" 150 ms

sample1.xml					
A					
John Smith	B	C	D	E	F
	Database Primer	First Edition, Paperback	1998	\$20	This book introduces the reader to the theory and technology... (TRUNCATED)
		Second Edition, Hard Cover	2000	\$30	
	Computer Systems	First Edition, Paperback	1995	\$40	An undergraduate level introduction to computer... (TRUNCATED)
sample2.xml					
A					
Paul Jones	B	C	D	E	F
	XML at Work	First Edition, Paperback	1999	\$30	A comprehensive description of XML, and all related standards... (TRUNCATED)
		HTML and Scripts	null	1993	
	HTML and Scripts	Second Edition, Hard Cover	1999	\$45	A useful HTML handbook, with a good tutorial on the use of sc... (TRUNCATED)
	JavaScripts	null	2000	\$50	A must in every Webmaster's bookshelf ...





# Union-Free Regular Expression (UFRE)

- Web page structure can be represented as *Union-Free Regular Expression* (UFRE)
  - UFRE is Regular Expressions without *disjunctions*
  - If  $a$  and  $b$  are UFRE, then the following are also UFREs
    - $a.b$
    - $(a)^+$
    - $(a)^?$



# Union-Free Regular Expression (UFRE)

---

- Web page structure can be represented as *Union-Free Regular Expression* (UFRE)
  - UFRE is Regular Expressions without *disjunctions*
  - If  $a$  and  $b$  are UFRE, then the following are also UFREs
    - $a.b \rightarrow$  string fields
    - $(a)^+ \rightarrow$  lists (possibly nested)
    - $(a)? \rightarrow$  optional fields
  - Strong assumption that usually holds



# Approach

---

- Given a set of example pages
- Generate the *Union-Free Regular Expression* which contains example pages
- Find the least upper bounds on the RE lattice to generate a wrapper in *linear time*
- Reduces to finding the least upper bound on two UFREs



# Matching/Mismatches

---

Given a set of pages of the same type

- Take the first page to be the *wrapper* (UFRE)
- Match each successive sample page against the wrapper
- *Mismatches* result in generalizations of wrapper
  - String mismatches
  - Tag mismatches

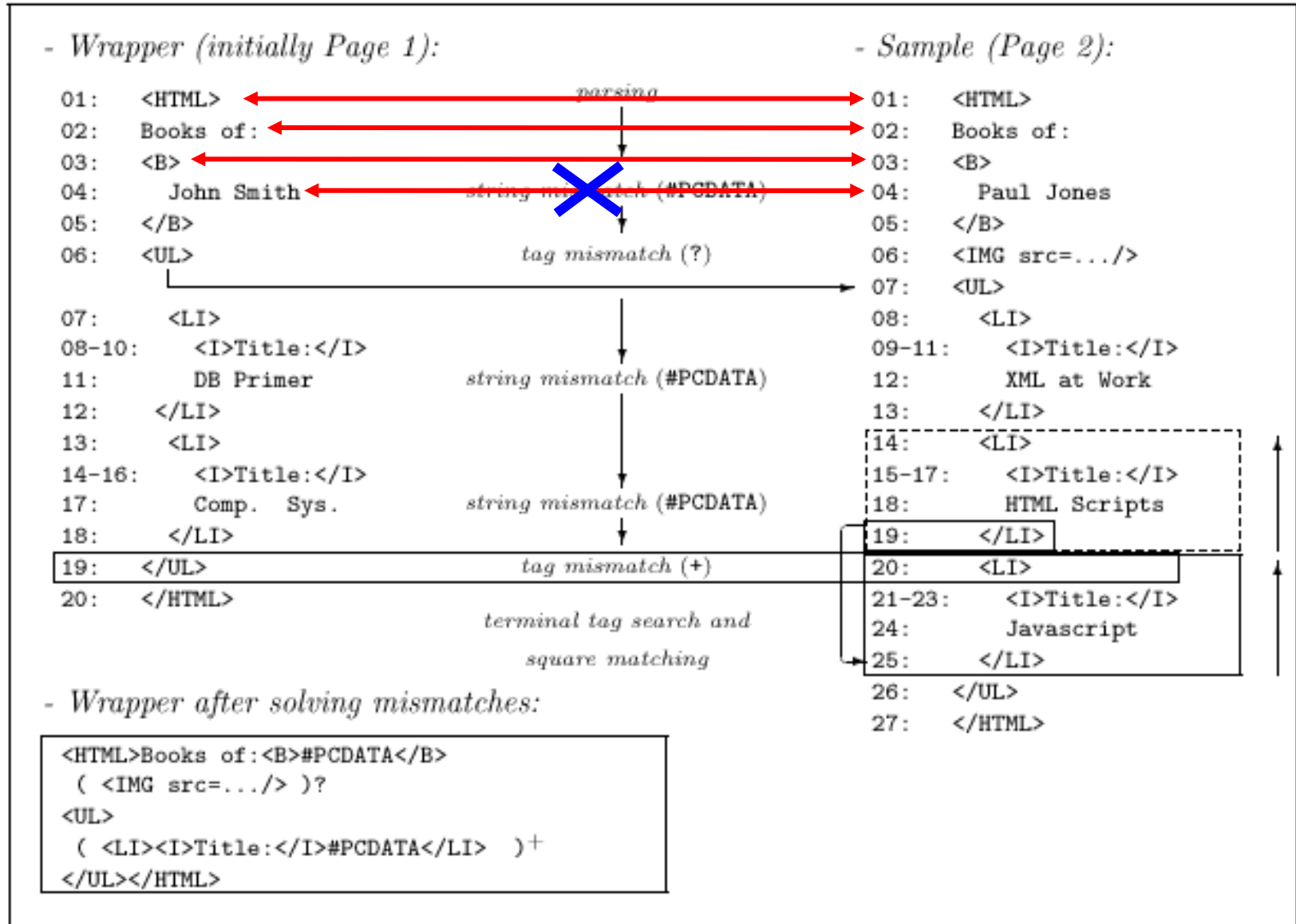


# Matching/Mismatches

Given a set of pages of the same type

- Take the first page to be the *wrapper* (UFRE)
- Match each successive sample page against the wrapper
- *Mismatches* result in generalizations of wrapper
  - String mismatches
    - Discover fields
  - Tag mismatches
    - Discover optional fields
    - Discover iterators

# Example Matching





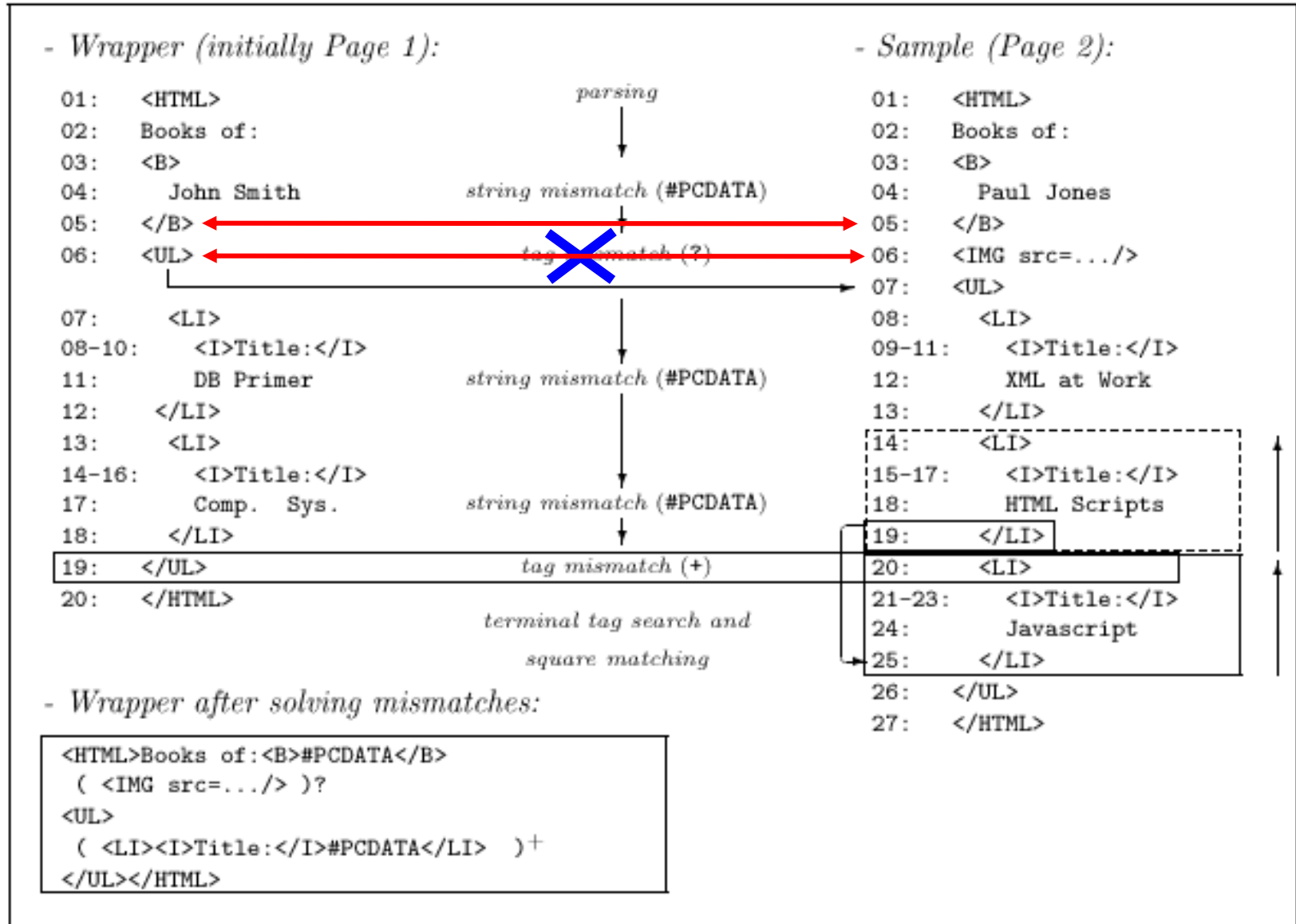
# String Mismatches: Discovering Fields

- String mismatches are used to discover fields of the document
- Wrapper is generalized by replacing “John Smith” with #PCDATA

<HTML>Books of: <B>John Smith

→ <HTML> Books of: <B>#PCDATA

# Example Matching





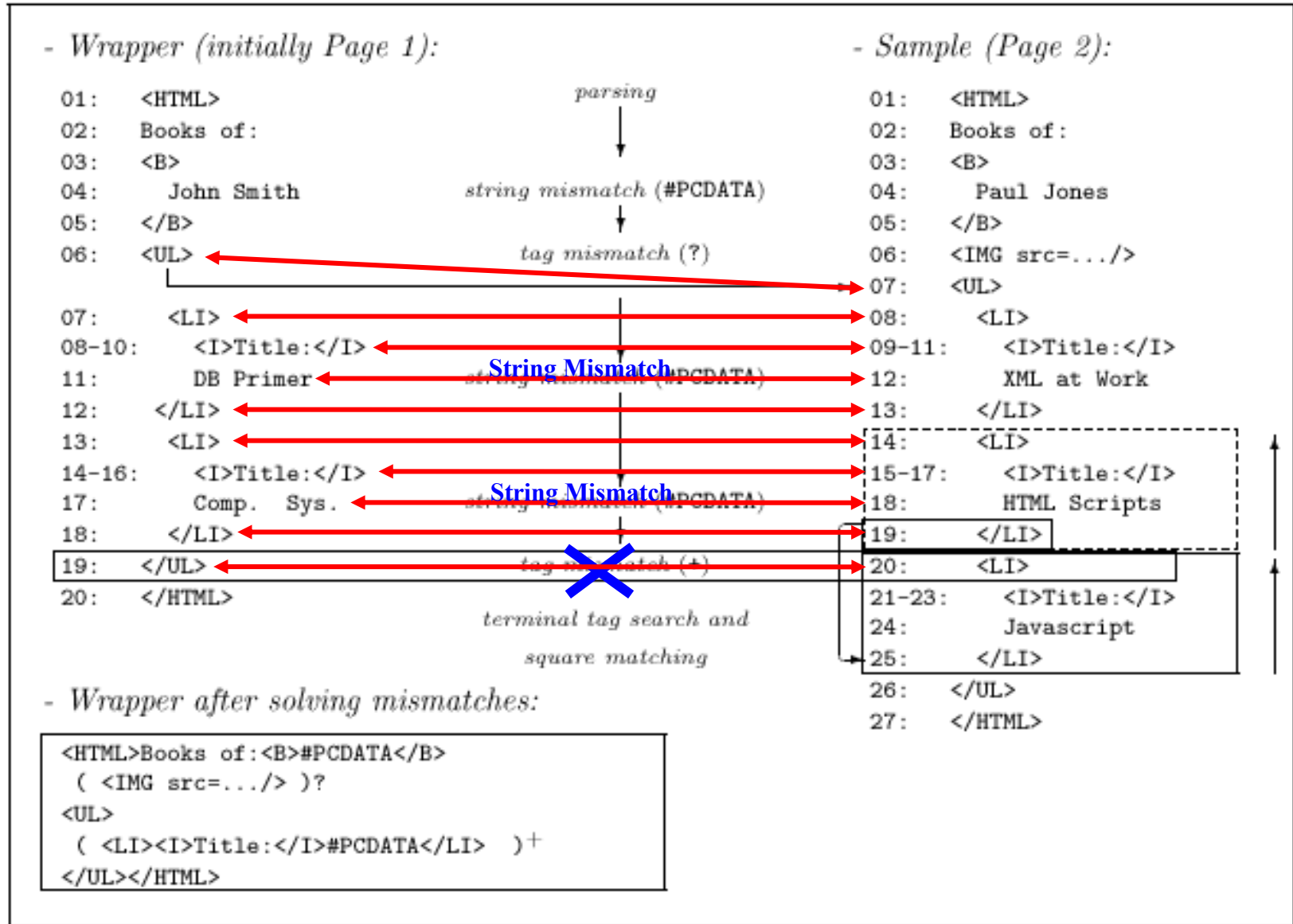


# Tag Mismatches: Discovering Optionals

---

- First check to see if mismatch is caused by an iterator (described next)
- If not, could be an optional field in wrapper *or* sample
- Cross search used to determine possible optionals
- Image field determined to be optional:
  - ( `<img src=.../>`)?

# Example Matching





# Tag Mismatches: Discovering Iterators

- Assume mismatch is caused by repeated elements in a list
  - End of the list corresponds to last matching token: `</LI>`
  - Beginning of list corresponds to one of the mismatched tokens: `<LI>` or `</UL>`
  - These create possible “squares”
- Match possible squares against earlier squares
- Generalize the wrapper by finding all contiguous repeated occurrences:
  - `( <LI><I>Title:</I>#PCDATA</LI> )+`

# Example Matching

- Wrapper (initially Page 1):

```

01: <HTML>
02: Books of:
03: <B>
04:   John Smith
05: </B>
06: <UL>
07:   <LI>
08-10:   <I>Title:</I>
11:     DB Primer
12:   </LI>
13:   <LI>
14-16:   <I>Title:</I>
17:     Comp. Sys.
18:   </LI>
19: </UL>
20: </HTML>
  
```

- Sample (Page 2):

```

01: <HTML>
02: Books of:
03: <B>
04:   Paul Jones
05: </B>
06: <IMG src=.../>
07: <UL>
08:   <LI>
09-11:   <I>Title:</I>
12:     XML at Work
13:   </LI>
14:   <LI>
15-17:   <I>Title:</I>
18:     HTML Scripts
19:   </LI>
20:   <LI>
21-23:   <I>Title:</I>
24:     Javascript
25:   </LI>
26: </UL>
27: </HTML>
  
```

*parsing*  
 ↓  
*string mismatch (#PCDATA)*  
 ↓  
*tag mismatch (?)*  
 ↓  
*string mismatch (#PCDATA)*  
 ↓  
*string mismatch (#PCDATA)*  
 ↓  
*tag mismatch (+)*  
 ↓  
*terminal tag search and square matching*

- Wrapper after solving mismatches:

```

<HTML>Books of:<B>#PCDATA</B>
 ( <IMG src=.../> )?
<UL>
 ( <LI><I>Title:</I>#PCDATA</LI> )+
</UL></HTML>
  
```

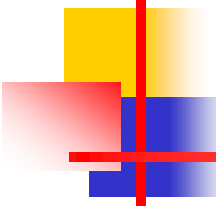


# Internal Mismatches

---

- Generate *internal mismatch* while trying to match square against earlier squares on *the same page*
  - Solving internal mismatches yield further refinements in the wrapper
    - List of book editions
    - `<I>Special!</I>`

# Recursive Example



- Wrapper (initially Page 1):

```

01-05: <HTML>Books of:<B>John Smith</B>
06:   <UL>
07:     <LI>
08:       Computer Systems
09:       <P>
10:         <B>
11:           1st Ed., 1995
12:         </B>
13:       </P>
14:     </LI>
15:     <LI>
16:       Database Primer
17:       <P>
18:         <B>
19:           1st Ed., 1998
20-22:       <I>Special!</I>
23:         </B>
24:         <B>
25:           2nd Ed., 2000
26:         </B>
27:       </P>
28:     </LI>
29-30: </UL></HTML>

```

*internal mismatch* →

- Wrapper after solving mismatches:

```

<HTML>Books of:<B>#PCDATA</B>
<UL>(<LI>#PCDATA<P>
  (<B>#PCDATA
    (<I>Special!</I>)?
    </B>)+ </P></LI>)+
</UL></HTML>

```

- Sample (Page 2):

```

01-05: <HTML>Books of:<B>Paul Jones</B>
06:   <UL>
07:     <LI>
08:       XML at Work
09:       <P>
10:         <B>
11:           1st Ed., 1999
12:         </B>
13:       </P>
14:     </LI>
15:   </UL>
16: </HTML>

```

*external mismatch*

```

28: </LI>
27: </P>
26: </B>
25:   2nd Ed., 2000
24: <B>
23: </B>

```

```

14: </LI>
13: </P>
12: </B>
11:   1st Ed., 1995
10: <B>
09: <P>
08:   Computer Systems:
07: <LI>

```



# Discussion

---

- Assumptions:
  - Pages are well-structured
  - Structure can be modeled by UFRE (no disjunctions)
- Search space for explaining mismatches is huge
  - Uses a number of heuristics to prune space
    - Limited backtracking
    - Limit on number of choices to explore
    - Patterns cannot be delimited by optionals
  - Will result in pruning possible wrappers

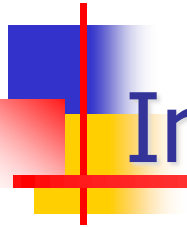


# Limitations

---

- Learnable grammars
  - Union-Free Regular Expressions (RoadRunner)
    - Variety of schema structure: tuples (with optional attributes) and lists of (nested) tuples
    - Does not efficiently handle disjunctions – pages with alternate presentations of the same attribute
  - Context-free Grammars
    - Limited learning ability
- User needs to provide a set of pages of the same type





# Inferlink Web Extraction Software



# Inferlink Web Extraction Software

---

- Two phase processing
  - Step 1: Cluster the pages based on the layout of the pages
  - Step 2: Build a template to extract the data for each cluster



# Inferlink Web Extraction Software: Clustering

- Cluster
  - Based on the visible text
  - Page is broken into chunks
    - These are continuous blocks of text
  - Search for common visible chunks
    - Remove chunks that occur in all pages
    - Remove chunks that occur in less than 10 pages
  - Greedy algorithm to cluster the pages based on the remaining chunks
    - Sort by the size of the clusters created by each chunk



# Inferlink Web Extraction Software: Template Learning

---

- Input: cluster  $\{P_i\}$
- Select 5 random pages to build a template
  - Tokenize on space & punctuation
  - Start with n-grams of tuples of size n,  $n=6$ 
    - Find those n-grams that occur on all pages
    - Keep only those n-grams that occur exactly once per pages
    - Decompose pages based on these n-grams
    - Run algorithm recursive on decomposed page
  - Repeat above for size  $n-1$  down to  $n=2$
  - Construct template based on the decomposition



# Discussion

---

- Inferlink approach solves some of the key limitations of Roadrunner
  - Pages do not all have to be of the same type
  - Multiple optionals would be treated as different page types
  - Scales well with complex pages



# Demonstration

---



# Web Data Extraction Software

---

- Beautiful Soup
  - <http://www.crummy.com/software/BeautifulSoup/>
  - Python library to manually write wrappers
- Jsoup
  - <http://jsoup.org/>
  - Java library to manually write wrappers
- ScrapingHub
  - <http://scrapinghub.com/>
  - Portia provides a wrapper learner
- Others
  - <https://www.quora.com/Which-are-some-of-the-best-web-data-scraping-tools>
  - Tell us if you find a good one!