Gestão e Tratamento da Informação

Web Data Extraction:
Automatic Wrapper Generation

Departamento de Engenharia Informática
Instituto Superior Técnico

1º Semestre
2010/2011
Outline

Automatic Wrapper Generation

String Matching and Tree Matching

Multiple Alignments

Building DOM Trees

Extraction of Flat Data Records

Extraction of Nested Data Records
Outline

Automatic Wrapper Generation

String Matching and Tree Matching

Multiple Alignments

Building DOM Trees

Extraction of Flat Data Records

Extraction of Nested Data Records
Problems of wrapper induction

Wrapper induction (supervised) has two main shortcomings:

- It is unsuitable for a large number of sites due to the manual labeling effort
- Wrapper maintenance is very costly.
  - The Web is a dynamic environment.
  - Sites change constantly.
  - Since rules learnt by wrapper induction systems mainly use formatting tags, if a site changes its formatting templates, existing extraction rules for the site become invalid.
Due to these problems, automatic (or unsupervised) extraction has been studied. Automatic extraction is possible because data records (tuple instances) in a Web site are usually encoded using a very small number of fixed templates. It is possible to find these templates by mining repeated patterns.
Two data extraction problems

- We have described an abstract model of structured data on the Web (i.e., nested relations), and a HTML mark-up encoding of the data model.
- The general problem of data extraction is to recover the hidden schema from the HTML mark-up encoded data.
- We study two extraction problems, which are really quite similar:
  1. Extraction given a single list page
  2. Extraction given multiple pages
Problem 1: Extraction given a single list page

- **Input:** A single HTML string $S$, which contains $k$ non-overlapping substrings $s_1, s_2, \ldots, s_k$ with each $s_i$ encoding an instance of a set type. I.e., each $s_i$ contains a collection $W_i$ of $m_i \geq 2$ non-overlapping sub-substrings encoding $m_i$ instances of a tuple type.

- **Output:** $k$ tuple types $\sigma_1, \sigma_2, \ldots, \sigma_k$, and $k$ collections $C_1, C_2, \ldots, C_k$, of instances of the set types such that for each collection $C_i$ there is a HTML encoding function $enc_i$ such that $enc_i : C_i \rightarrow W_i$ is a bijection.
Problem 2: Data extraction given multiple pages

- **Input:** A collection $W$ of $k$ HTML strings, which encode $k$ instances of the same type

- **Output:** A type $\sigma$, and a collection $C$ of instances of type $\sigma$, such that there is a HTML encoding $enc$ such that $enc : C \rightarrow W$ is a bijection
Regular expressions and extraction

- Regular expressions are often employed to represent templates (or encoding functions)
- However, templates can also be represented as string or tree patterns as we will see later
- Extraction:
  - Given a regular expression, a nondeterministic finite-state automaton can be constructed and employed to match its occurrences in string sequences representing Web pages
  - In the process, data items can be extracted, which are text strings represented by \#text
Templates as regular expressions

- A regular expression can be naturally used to model the HTML encoded version of a nested type.

- Given an alphabet of symbols $\Sigma$ and a special token $\#text$ that is not in $\Sigma$,

- A regular expression over $\Sigma$ is a string over $\Sigma \cup \{\#text, *, ?, |, (, )\}$ defined as follows:
  - The empty string $\epsilon$ and all elements of $\Sigma \cup \{\#text\}$ are regular expressions.
  - If $A$ and $B$ are regular expressions, then $AB$, $(A|B)$ and $(A)$? are regular expressions, where $(A|B)$ stands for $A$ or $B$, and $(A)$? stands for $(A|\epsilon)$.
  - If $A$ is a regular expression, $(A)^*$ is a regular expression, where $(A)^*$ stands for $\epsilon$ or $A$ or $AA$ or . . .

Note: we use $(A)^+$ as a shortcut for $A(A)^*$.
Outline

Automatic Wrapper Generation

String Matching and Tree Matching

Multiple Alignments

Building DOM Trees

Extraction of Flat Data Records

Extraction of Nested Data Records
Some useful algorithms

- The key is to find the encoding template from a collection of encoded instances of the same type.
- A natural way to do this is to detect repeated patterns from HTML encoding strings.
- **String edit distance** and **tree edit distance** are obvious techniques for the task.
String edit distance

- String edit distance: the most widely used string comparison technique.
- The edit distance of two strings, $s_1$ and $s_2$, is defined as the minimum number of point mutations required to change $s_1$ into $s_2$, where a point mutation is one of:
  1. change a letter
  2. insert a letter
  3. delete a letter
A formal definition

For two strings $s_1$ and $s_2$, the edit distance $d(s_1, s_2)$ is defined as:

- $d(\epsilon, \epsilon) = 0$
- $d(s, \epsilon) = d(\epsilon, s) = |s|$
- $d(s_1 + c_1, s_2 + c_2) = \min(d(s_1, s_2) + r(c_1, c_2),
  d(s_1 + c_1, s_2) + 1,
  d(s_1, s_2 + c_2) + 1)$

where $r(c_1, c_2) = 0$ if $c_1 = c_2$ and $r(c_1, c_2) = 1$ otherwise.
Dynamic programming

- We can use a matrix $m[0..|s_1|, 0..|s_2|]$ to hold the edit distances
- The value in each cell is computed iteratively, from $m[0, 0]$
- The last cell, $m[|s_1|, |s_2|]$ will hold the required value of edit distance

- The matrix is defined as:
  - $m[0, 0] = 0$
  - $m[i, 0] = i$
  - $m[0, j] = j$
  - $m[i, j] = min(m[i-1, j-1] + r(s_1[i], s_2[j]), m[i-1, j] + 1, m[i, j-1] + 1)$
An example

- To compute the edit distance between “XGYXYXYX” and “XYXYXYTX”

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>G</th>
<th>Y</th>
<th>X</th>
<th>Y</th>
<th>X</th>
<th>Y</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>s₂</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

- The string alignment is

XGYXYXY-X
X-YXYXYTX
Tree Edit Distance

- Tree edit distance between two trees $A$ and $B$ (labeled ordered rooted trees) is the cost associated with the minimum set of operations needed to transform $A$ into $B$.
- The set of operations used to define tree edit distance includes three operations:
  - node removal
  - node insertion
  - node replacement
- A cost is assigned to each of the operations.
Definition

Let $X$ be a tree and $X[i]$ be the $i$-th node of $X$.

A mapping $M$ between a tree $A$ and a tree $B$ is a set of ordered pairs $(i, j)$, where $i$ is a node in tree $A$ and $j$ is a node in tree $B$, such that, for every $(i_1, j_1), (i_2, j_2) \in M$:

1. $i_1 = i_2$ iff $j_1 = j_2$
2. $A[i_1]$ is on the left of $A[i_2]$ iff $B[i_1]$ is on the left of $B[i_2]$
3. $A[i_1]$ is an ancestor of $A[i_2]$ iff $B[i_1]$ is an ancestor of $B[i_2]$
An Example Mapping
Simple tree matching

- In the general setting:
  - mapping can cross levels, e.g., node a in tree A and node a in tree B.
  - replacements are also allowed, e.g., node b in A and node h in B.
- We describe a restricted matching algorithm, called Simple Tree Matching (STM), which has been shown quite effective for Web data extraction:
  - STM is a top-down algorithm
  - Instead of computing the edit distance of two trees, it evaluates their similarity by producing the maximum matching through dynamic programming.
Algorithm: STM(A, B)
1. if the roots of the two trees A and B contain distinct symbols then
2. return (0)
3. else $m :=$ the number of first-level sub-trees of $A$;
4. $n :=$ the number of first-level sub-trees of $B$;
5. Initialization: $M[i, 0] := 0$ for $i = 0, \ldots, m$;
   $M[0, j] := 0$ for $j = 0, \ldots, n$;
6. for $i = 1$ to $m$ do
7. for $j = 1$ to $n$ do
8. $M[i, j] := \max(M[i, j-1], M[i-1, j], M[i-1, j-1]+W[i, j])$;
   where $W[i, j] = STM(A_i, B_j)$
9. end-for
10. end-for
11. return ($M[m, n]+1$)
12. end-if
An Example

Matching = 5
Outline

Automatic Wrapper Generation

String Matching and Tree Matching

Multiple Alignments

Building DOM Trees

Extraction of Flat Data Records

Extraction of Nested Data Records
Multiple alignment

- Pairwise alignment is not sufficient because a web page usually contains more than one data record.
- We need **multiple alignment**
- We discuss two techniques
  - Center Star method
  - Partial tree alignment
Center star method

This is a classic technique, and quite simple. It is commonly used for multiple string alignments, but can be adapted for trees.

Let the set of strings to be aligned be \( S \). In the method, a string \( s_c \) that minimizes,

\[
\sum_{\forall s_i \in S} d(s_c, s_i)
\]

is first selected as the center string. \( d(s_c, s_i) \) is the distance of two strings.

The algorithm then iteratively computes the alignment of rest of the strings with \( s_c \).
The algorithm

\textbf{CenterStar}(S)
1. choose the center star $s_c$ using Equation (3);
2. initialize the multiple sequence alignment $M$ that contains only $s_c$;
3. for each $s$ in $S-\{s_c\}$ do
4. \hspace{1em} let $c^*$ be the aligned version of $s_c$ in $M$;
5. \hspace{1em} let $s'$ and $c^{**}$ be the optimally aligned strings of $s$ and $c^*$;
6. \hspace{1em} add aligned strings $s'$ and $c^{**}$ into the multiple alignment $M$;
7. \hspace{1em} add spaces to each string in $M$, except, $s'$ and $c^{**}$, at locations where new spaces are added to $c^*$
8. endfor
9. return multiple string alignment $M$
Example: We have three strings, i.e., \( S = \{\text{ABC}, \text{XBC}, \text{XAB}\} \). ABC is selected as the center string \( s_c \). Let us align the other strings with ABC.

Iteration 1: Align \( c^* (= s_c) \) with \( s = \text{XBC} \):

\[
\begin{align*}
    c^*: & \quad \text{A B C} \\
    s*: & \quad \text{X B C} \\
\end{align*}
\]

Update \( M \):

\[
\begin{align*}
    \text{A B C} & \rightarrow \text{A B C} \\
    \text{X B C} & \rightarrow \text{X B C} \\
\end{align*}
\]

Iteration 2: Align \( c^* \) with \( s = \text{XAB} \):

\[
\begin{align*}
    c^*: & \quad - \text{A B C} \\
    s*: & \quad \text{X A B} - \\
\end{align*}
\]

Update \( M \):

\[
\begin{align*}
    \text{A B C} & \rightarrow - \text{A B C} \\
    \text{X B C} & \rightarrow - \text{X B C} \\
    \text{X A B} & \rightarrow - \\
\end{align*}
\]
Problems

- Assume there are $k$ strings in $S$ and all strings have length $n$, finding the center takes $O(k^2 n^2)$ time and the iterative pair-wise alignment takes $O(k n^2)$ time. Thus, the overall time complexity is $O(k^2 n^2)$
- If the center string does not have a particular data item, other strings that contain the item may not be aligned properly.
  - E.g., the letter “X” in the example
Giving the cost of 1 for “changing a letter” in edit distance is problematic because of optional data items in data records.

The problem can be partially dealt with by disallowing “changing a letter” (e.g., giving it a larger cost). However, this introduces another problem.

For example, if we align only “ABC” and “XBC”, it is not clear which of the following alignment is better:

\[
\begin{align*}
A-BC & \quad -ABC \\
-XBC & \quad X-BC
\end{align*}
\]
The partial tree alignment method

- **Choose a seed tree:** A seed tree, denoted by $T_s$, is picked with the maximum number of data items
  - The seed tree is similar to center string, but without the $O(k^2n^2)$ pair-wise tree matching to choose it.
- **Tree matching:** For each unmatched tree $T_i, (i \neq s)$,
  - match $T_s$ and $T_i$
  - Each pair of matched nodes are linked (aligned)
  - For each unmatched node $n_j$ in $T_i$ do
    - expand $T_s$ by inserting $n_j$ into $T_s$ if a position for insertion can be uniquely determined in $T_s$
- The expanded seed tree $T_s$ is then used in subsequent matching
Partial tree alignment of two trees

New part of $T_s$

Insertion is possible

Insertion is not possible
The Algorithm

**Algorithm** PartialTreeAlignment($S$)

1. Sort trees in $S$ in descending order of the number of unaligned data items;
2. $T_s \leftarrow$ the first tree (which is the largest) and delete it from $S$;
3. $R \leftarrow \emptyset$;
4. **while** ($S \neq \emptyset$) **do**
5. $T_i \leftarrow$ select and delete next tree from $S$; // follow the sorted order
6. STM($T_s$, $T_i$); // tree matching
7. AlignTrees($T_s$, $T_i$); // based on the result from line 6
8. **if** $T_i$ is not completely aligned with $T_s$ **then**
9. **if** InsertIntoSeed($T_s$, $T_i$) **then** // True: some insertions are done
10. $S = S \cup R$;
11. $R \leftarrow \emptyset$
12. **endif**;
13. **if** there are still unaligned items in $T_i$ that are not inserted into $T_s$ **then**
14. $R \leftarrow R \cup \{T_i\}$
15. **endif**;
16. **endif**;
17. endwhile;
18. Output data fields from each $T_i$ to a data table based on the alignment results.

**Fig. 21.** The partial tree alignment algorithm
A complete example

$T_s = T_1 \circ p$

$T_2 \circ p$

$T_3 \circ p$

No node inserted

New $T_s \circ p$

c, h, and k inserted

$T_2$ is matched again
## Output Data Table

<table>
<thead>
<tr>
<th></th>
<th>(T_1)</th>
<th>(T_2)</th>
<th>(T_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(n)</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>(c)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(k)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline

Automatic Wrapper Generation

String Matching and Tree Matching

Multiple Alignments

Building DOM Trees

Extraction of Flat Data Records

Extraction of Nested Data Records
Building DOM trees

- We now start to talk about actual data extraction
- The usual first step is to build a **DOM tree** (tag tree) of a HTML page
  - Most HTML tags work in pairs. Within each corresponding tag-pair, there can be other pairs of tags, resulting in a nested structure.
  - Building a DOM tree from a page using its HTML code is thus natural.
- In the tree, each pair of tags is a **node**, and the nested tags within it are the **children** of the node.
Two steps to build a tree

1. **HTML code cleaning:**
   - Some tags do not require closing tags (e.g., `<li>`, `<hr>` and `<p>`) although they have closing tags.
   - Additional closing tags need to be inserted to ensure all tags are balanced.
   - Ill-formatted tags need to be fixed. One popular program is called *Tidy*, which can be downloaded from http://tidy.sourceforge.net/.

2. **Tree building:** simply follow the nested blocks of the HTML tags in the page to build the DOM tree. It is straightforward.
Correcting errors in HTML can be hard.
There are also dynamically generated pages with scripts.
Visual information comes to the rescue.
As long as a browser can render a page correct, a tree can be built correctly.
  Each HTML element is rendered as a rectangle.
  Containments of rectangles representing nesting
Outline

Automatic Wrapper Generation

String Matching and Tree Matching

Multiple Alignments

Building DOM Trees

Extraction of Flat Data Records

Extraction of Nested Data Records
Flat Data Records

- Given a single list page with multiple data records,
  - Automatically segment data records
  - Extract data from data records.
- Since the data records are flat (no nested lists), string similarity or tree matching can be used to find similar structures.
  - Computation is a problem
  - A data record can start anywhere and end anywhere
Two important observations

- **Observation 1:** A group of data records that contains descriptions of a set of similar objects are typically presented in a contiguous region of a page and are formatted using similar HTML tags. Such a region is called a data region.

- **Observation 2:** A set of data records are formed by some child sub-trees of the same parent node.
An example
The DOM tree
The Approach

Given a page, three steps:

1. Building the HTML Tag Tree
   - Erroneous tags, unbalanced tags, etc

2. Mining Data Regions
   - String matching or tree matching

3. Identifying Data Records
Definition: A generalized node (a node combination) of length \( r \) consists of \( r \geq 1 \) nodes in the tag tree with the following two properties:

1. the nodes all have the same parent
2. the nodes are adjacent.

Definition: A data region is a collection of two or more generalized nodes with the following properties:

1. the generalized nodes all have the same parent
2. the generalized nodes all have the same length
3. the generalized nodes are all adjacent
4. the similarity between adjacent generalized nodes is greater than a fixed threshold.
Example of data regions
Mining data regions

- We need to find where each region starts and where it ends.
  - perform string or tree matching
- Computation is not a problem anymore
  - Due to the two observations, we only need to perform comparisons among the children nodes of a parent node.
  - Some comparisons done for earlier nodes are the same as for later nodes
String comparisons:
- (1,2), (2,3), (3,4), ..., (9,10), (1-2, 3-4), (3-4, 5-6), ..., (1-2-3, 4-5-6), (4-5-6, 7-8-9)
- (2-3, 4-5), (4-5, 6-7), ..., (2-3-4, 5-6-7), (5-6-7, 8-9-10)
- (3-4-5, 6-7-8)
A possible configuration of data records
Extracting Data from Data Records

- Once a list of data records is identified, we can align and extract data items from them.
- Approaches (align multiple data records):
  - Multiple string alignment
    - Many ambiguities due to pervasive use of table related tags.
  - Multiple tree alignment (partial tree alignment)
    - Together with visual information is effective
Generating extraction patterns and data extraction

Once data records in each data region are discovered, we align them to produce an extraction pattern that can be used to extract data from the current page and also other pages that use the same encoding template.

Partial tree alignment algorithm is just for the purpose.
Outline

Automatic Wrapper Generation

String Matching and Tree Matching

Multiple Alignments

Building DOM Trees

Extraction of Flat Data Records

Extraction of Nested Data Records
Nested Data Records

Problem with the previous method

- Not suitable for nested data records, i.e., data records containing nested lists.
- Since the number of elements in the list of each data record can be different, using a fixed threshold to determine the similarity of data records will not work.
Solution

- The problem, however, can be dealt with as follows:
  - We can traverse the DOM tree bottom-up.
  - This ensures that nested lists at lower levels are found first based on repeated patterns before going to higher levels.
  - When a nested list is found, its records are collapsed to produce a single template.
  - This template replaces the list of nested data records.
- When comparisons are made at a higher level, the algorithm only sees the template. Thus it is treated as a flat data record.
The NET algorithm

Algorithm NET(\textit{Root}, \tau)
1. TraverseAndMatch(\textit{Root}, \tau);
2. for each top level node \textit{Node} whose children have aligned data records do
3. \hspace{1em} PutDataInTables(\textit{Node});
4. endfor

TraverseAndMatch (\textit{Node}, \tau)
1. if Depth(\textit{Node}) \geq 3 then
2. \hspace{1em} for each \textit{Child} \in \textit{Node}.Children do
3. \hspace{2em} TraverseAndMatch(\textit{Child}, \tau);
4. \hspace{1em} endfor
5. \hspace{1em} Match(\textit{Node}, \tau);
6. endif

Fig. 31. The overall NET algorithm
The MATCH algorithm

- It performs tree matching on child sub-trees of Node and template generation. \( \tau \) is the threshold for a match of two trees to be considered sufficiently similar.

```
Match(Node, \( \tau \))
1  Children \leftarrow Node.Children;
2  while Children \neq \emptyset do
3      ChildFirst \leftarrow \text{select and remove the first child from Children};
4      for each ChildR in Children \leftarrow Children – \{ChildFirst\} do
5          if TreeMatch(ChildFirst, ChildR) > \( \tau \) then
6              AlignAndLink();
7              Children \leftarrow Children – \{ChildR\}
8          endfor
9      if some alignments (or links) have been made with ChildFirst then
10         GenNodeTemplate(ChildFirst)
11     endwhile
12 If consecutive child nodes in Children are aligned then
13     GenRecordTemplate(Node)
```
An example
Algorithm details

- **GenNodeTemplate**
  - It generates a node template for all the nodes (including their sub-trees) that match ChildFirst.
  - It first gets the set of matched nodes ChildRs
  - then produces a template which is the final tree.

- **AlignAndLink**
  - aligns and links all matched data items in ChildFirst and ChildR.
Generating Record Templates

- Produces a regular expression pattern for each data record.
- This is a grammar induction problem.
- Grammar induction is to infer a regular expression given a finite set of positive and negative example strings.
  - However, we only have a single positive example.
  - Fortunately, structured data in Web pages are usually highly regular which enables heuristic methods to generate “simple” regular expressions.
- We need to make some assumptions
Assumptions

► Three assumptions

1. The nodes in the first data record at each level must be complete.
2. The first node of every data record at each level must be present.
3. Nodes within a flat data record (no nesting) do not match one another.

► On the Web, these are not strong assumptions. In fact, they work well in practice.
Generating the NFA

**GenRecordPattern(Node)**

1. `String ← Assign a distinctive symbol to each set of matched children of Node;
2. create an NFA \( N = (Q, \Sigma, \delta, q_0, F) \), where \( Q = \{q_0\} \), \( q_0 \) is the start state, \( \Sigma \) is the symbol set containing all symbols appeared in \( String \), \( \delta \) is the transition function, \( F = \emptyset \) is the set of accept states;
3. \( q_c ← q_0; \) // \( q_c \) is the current state
4. for each symbol \( s \) in \( String \) in sequence do
5.   if \( \exists \) a transition \( \delta(q_c, s) = q_n \) then
6.     \( q_c ← q_n \) // transit to the next state;
7.   else if \( \exists \delta(q_i, s) = q_j, \) where \( q_i, q_j ∈ Q \) then // \( s \) appeared before
8.     if \( \exists \delta(q_f, ε) = q_i, \) where \( \delta(q_i, s) = q_j \) and \( f ≥ c \) then
9.       TransitTo\( (q_c, q_f); \)
10. else create a transition \( \delta(q_c, ε) = q_i, \) where \( \delta(q_i, s) = q_j \)
11. \( q_c ← q_j \)
12. else create a new state \( q_{c+1} \) and a transition \( \delta(q_c, s) = q_{c+1}; \)
13. \( Q = Q ∪ \{q_{c+1}\}; \)
14. \( q_c ← q_{c+1} \)
15. if \( s \) is the last symbol in \( String \) then
16.   Assign the state with the largest subscript the accept state \( q_r, F = \{q_r\}; \)
17.   TransitTo\( (q_c, q_r); \)
18. endfor
19. generate a regular expression pattern based on the NFA \( N; \)
20. Substitute all the node templates into the regular expression pattern.
An example

- The algorithm simply produces a string for generating a regular expression, and then applies grammar induction.
- For our example, we obtain:

<table>
<thead>
<tr>
<th>t1</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>t2</th>
<th>N5</th>
<th>N6</th>
<th>N7</th>
<th>N8</th>
<th>N9</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

- The final NFA and the regular expression are:

\[ (a (b \ c^2)^+)^+ \]
We finally obtain the following
The final call outputs data items in a table, which is simple after the data record templates are found.

An example:

<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>t6</td>
<td>t7</td>
<td>t8</td>
</tr>
<tr>
<td>t2</td>
<td>t9</td>
<td>t10</td>
<td>t11</td>
</tr>
<tr>
<td>t2</td>
<td>t12</td>
<td>t13</td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td>t14</td>
<td>t15</td>
<td>t16</td>
</tr>
</tbody>
</table>
An more complete example

Aligned data nodes are linked

<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
<th>t4</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>t5</td>
<td>t6</td>
</tr>
<tr>
<td>t2</td>
<td>t8</td>
<td>t9</td>
</tr>
</tbody>
</table>

Alignment after collapsing

The output data table for the example i
Summary

Wrapper induction

◮ Advantages:
  ▶ Only the target data are extracted as the user can label only data items that he/she is interested in.
  ▶ Due to manual labeling, there is no integration issue for data extracted from multiple sites as the problem is solved by the user.

◮ Disadvantages:
  ▶ It is not scalable to a large number of sites due to significant manual efforts. Even finding the pages to label is non-trivial.
  ▶ Wrapper maintenance (verification and repair) is very costly if the sites change frequently.
Automatic extraction

- **Advantages:**
  - It is scalable to a huge number of sites due to the automatic process.
  - There is little maintenance cost.

- **Disadvantages:**
  - It may extract a large amount of unwanted data because the system does not know what is interesting to the user. Domain heuristics or manual filtering may be needed to remove unwanted data.
  - Extracted data from multiple sites need integration, i.e., their schemas need to be matched.
Conclusions

- In terms of extraction accuracy, it is reasonable to assume that wrapper induction is more accurate than automatic extraction. However, there is no reported comparison.

Applications

- Wrapper induction should be used in applications in which the number of sites to be extracted and the number of templates in these sites are not large.
- Automatic extraction is more suitable for large scale extraction tasks which do not require accurate labeling or integration.

Still an active research area!
Questions?