Extracting Structured Data from Web Pages

Summarized by Jaehyok Chong

Algorithm: ExAlg – under revision

```
Input: Set of HTML pages I, Output: Template, Schema, Values
Function ECGM (Equivalence Class Generation Module
        DiffFormat(I, T);// differentiate roles using HTML format
        while (T is not empty) {
                FindEquiv(T, L, Q); // find equivalence classes from T & put LFEQs in L
                HandInv(L, Q); // handle invalid equivalence classes in L
                DiffEq(Q, T); // differentiate roles using equivalence classes
        }
End ECGM
Function AM (Analysis Module)
        ConstTemp(L, S);
        ExVal(I, S, V);
End AM

Main procedure ExAlg(I)
        ECGM(I);
        AM(L);
End ExAlg
```

Example

<html>1<body>2
3Book4Name56Databases
7Reviews89
10
11
12Reviewer13Name1415John
16Rating1718 7
19Text2021 …
22
23
</body>24</html>25

<html>1<body>2
3Book4Name56Query Opt.
7Reviews89
10
11
12Reviewer13Name1415John
16Rating1718 8
19Text2021 …
22
23
</body>24</html>25

<html>1<body>2
3Book4Name56Data Mining
7Reviews89
10
11
12Reviewer13Name1415Jeff
16Rating1718 2
19Text2021 …
22
11
12Reviewer13Name1415Jane
16Rating1718 6
19Text2021 …
22
23
</body>24</html>25

<html>1<body>2
3Book4Name56Transactions
7Reviews89
10
23
</body>24</html>25

DiffFormat

- Extract tokens from HTML pages
- Differentiate roles of tokens using parse tree (structural context)
  - ()
    - [1]([1]): <html><body>()
    - [2]([2]): <html><body>
()
  - Name
    - Name[1]: <html><body>Name
    - Name[2]: <html><body>
Name

DiffFormat

Tokens		Tokens	
<html>		Reviewer	
<body>		**Name² (<html><body> Name)**	
¹ (<html><body>)		**² (<html><body>)**	
Book		John	
Name¹ (<html><body>Name)		Jeff	
¹ (<html><body>)		Jane	
Databases		Rating	
Data		7	
Mining		2	
Query		6	
Opt.		8	
Transactions		Text	
Reviews			
		</body>	
² (<html><body>)		</html>	

FindEquiv – 1st iteration

- Make occurrence vectors for each tokens generated by DiffFormat()
- Find Equivalence classes using occurrence vectors
- Find LFEQs using size & support thresholds
  - SizeThre = 3, SupThres = 3

FindEquiv – 1st iteration

Tokens	Occurrence	Tokens	Occurrence
<html>	1, 1, 1, 1 (*E1*)	Reviewer	1, 2, 1, 0 (*E7*)
<body>	1, 1, 1, 1 (*E1*)	Name[2] (<html><body> Name)	1, 2, 1, 0 (*E7*)
[1] (<html><body>)	2, 2, 2, 2 (*E2*)	[2] (<html><body>)	3, 6, 3, 0 (*E8*)
Book	1, 1, 1, 1 (*E1*)	John	1, 0, 1, 0 (*E9*)
Name[1] (<html><body>Name)	1, 1, 1, 1 (*E1*)	Jeff	0, 1, 0, 0 (*E4*)
[1] (<html><body>	2, 2, 2, 2 (*E2*)	Jane	0, 1, 0, 0 (*E4*)
Databases	1, 0, 0, 0 (*E3*)	Rating	1, 2, 1, 0 (*E7*)
Data	0, 1, 0, 0 (*E4*)	7	1, 0, 0, 0 (*E3*)
Mining	0, 1, 0, 0 (*E4*)	2	0, 1, 0, 0 (*E4*)
Query	0, 0, 1, 0 (*E5*)	6	0, 1, 0, 0 (*E4*)
Opt.	0, 0, 1, 0 (*E5*)	8	0, 0, 1, 0 (*E5*)
Transactions	0, 0, 0, 1 (*E6*)	Text	1, 2, 1, 0 (*E7*)
Reviews	1, 1, 1, 1 (*E1*)		1, 2, 1, 0 (*E7*)
	1, 1, 1, 1 (*E1*)		1, 1, 1, 1 (*E1*)
	1, 2, 1, 0 (*E7*)	</body>	1, 1, 1, 1 (*E1*)
[2] (<html><body>)	3, 6, 3, 0 (*E8*)	</html>	1, 1, 1, 1 (*E1*)

FindEquiv – 1st iteration

- Find LFEQs using SizeThre = 3, SupThres = 3
  - *E1* : <html> <body> Book Name[1] Reviews <ol> </body> </html>
  - *E7* : <li> Reviewer Name[2] Rating Text

Equivalence Classes	SizeThre	SupThres	Equivalence Classes	SizeThre	SupThres
E1	**9**	**4**	*E6*	1	1
E2	2	4	***E7***	**6**	**3**
E3	2	1	*E8*	2	3
E4	6	1	*E9*	1	2
E5	3	1			

HandInv – 1st iteration

- Handle invalid equivalence classes
  - [Observation 4.3] A valid equivalence class is ordered and a pair of two valid equivalence classes is nested
  - Find invalid LFEQs determined by FindEquiv()
- In this iteration, there is no invalid LFEQs.

Ordered Equivalence classes

An equivalence class is **ordered**, if its tokens can be ordered $<t_1,\ldots,t_m>$, such that, for every page p_i ($1 \leq i \leq n$), and every pair of tokens t_j, t_k ($1 \leq j < k \leq m$),

I. If t_j occurs at least l times in p_i, the l^{th} occurrence of t_j in p_i occurs before the l^{th} occurrence of t_k in p_i, and

II. If t_j occurs at least $(l+1)$ times in p_i, the $(l+1)^{st}$ occurrence of t_j in p_i is after the l^{th} occurrence of t_k in p_i.

Nesting of Equivalence classes

A pair of equivalence classes, E_i and E_j is **nested** if,

I. The span of any occurrence of E_i does not overlap with the span of any occurrence of E_j, or

II. The span of all occurrences of E_j is with in Pos(p) of some occurrence of E_i for some fixed p; or vice versa.

Remove either one arbitrarily

- ADEFBDEFC
  - ABC
  - DEF

Causes of invalid equivalence classes

I. Correlation of instantiation of two or more type constructors

II. Frequently Occurring tuples

III. Overlap in the tokens associated with different type constructors

→ Type A (I + II) is considered to be processed by HandInv and Type B (III) is just removed by HandInv

Almost-Ordered

An equivalence class *E* is **almost-ordered** if the tokens in the equivalence class can be partitioned into ordered equivalence classes that do not overlap

- *E* = {*A*, *B*, *C*, *D*}
  ABABCDCD, *ABCD*, *ABABABCDCDCD*
  → divided into two equivalence classes of {*A*, *B*} and {*C*, *D*} which are ordered and do not overlap with each other
- Try All possible partitions
  - Binary Partition only?
  - *ABABCDCDEFEF*, *ABCDEF*, *ABABABCDCDCDEFEFEF*
  - *{A,B} and {C,D,E,F}?*
  - *{A,B} and {C,D} and {E,F}?*

Nearly-Nested

Two equivalence classes, E_i and E_j are said to be **nearly-nested** if for every token $t \in E_i$ (and vice versa) there exists a position p, such that each occurrence of t is either outside the span of any occurrence of E_j, or within Pos(p) of some occurrence of E_j.

- $E_1 = \{A, B, C, D\}$, $E_2 = \{E, F, G, H\}$
  $\underline{ABCD}A(EF)BC(GH)D$, $A(EF)BC(GH)D$, $\underline{EFGH}$
  → divide E_1 into three equivalence classes of $\{A\}$, $\{B, C\}$, $\{D\}$ and E_2 into two equivalence classes of $\{E, F\}$ and $\{G, H\}$
- $E_1 = \{A, B, C, D\}$, $E_2 = \{E, F, G, H\}$
  $A(EF)BC(GH)D$, $A(EF)BC(GH)D$, $\underline{EFGH}$
  → divide E_1 into three equivalence classes of $\{A\}$, $\{B, C\}$, $\{D\}$ and E_2 into two equivalence classes of $\{E, F, G, H\}$

DiffEq – 1st iteration

- Differentiate roles of tokens which are not in the set of LFEQs using LFEQs
- For the tokens satisfies SupThres, differentiate the roles
  - ^{1}, ^{1}, ^{2}, ^{2}
  - Pos(i, *E*) represents i-th position(Pos(i)) in *E* = <t_1, Pos(1), t_2, Pos(2), t_3, Pos(3), …, Pos(n – 1), t_n>

Tokens	Differentiated roles	Dtokens
^{1}	Pos(2, *E1*), Pos(4, *E1*)	$^1_1$, $^1_2$
^{1}	Pos(4, *E1*), Pos(5, *E1*)	$^1_1$, $^1_2$
^{2}	Pos(1, *E7*), Pos(3, *E7*), Pos(4, *E7*)	$^2_1$, $^2_2$, $^2_3$
^{2}	Pos(3, *E7*), Pos(4, *E7*), Pos(5, *E7*)	$^2_1$, $^2_2$, $^2_3$

An Example for DiffEq

- ACDCBDF
  - Assume ABF is in LFEQ, but C and D are not in LFEQ
- ACDCBDF

FindEquiv – 2nd iteration

- For the newly added dtokens by DiffEq(), make occurrence vectors and equivalence classes

FindEquiv – 2nd iteration

Tokens	Occurrence	Tokens	Occurrence
<html>	1, 1, 1, 1 (*E1*)	Reviewer	1, 2, 1, 0 (*E7*)
<body>	1, 1, 1, 1 (*E1*)	Name^{2} (<html><body> Name)	1, 2, 1, 0 (*E7*)
$^{1}_{1}$	**1, 1, 1, 1 (*E1*)**	**$^{2}_{1}$**	**1, 2, 1, 0 (*E7*)**
Book	1, 1, 1, 1 (*E1*)	**$^{2}_{2}$**	**1, 2, 1, 0 (*E7*)**
Name^{1} (<html><body>Name)	1, 1, 1, 1 (*E1*)		
$^{1}_{1}$	**1, 1, 1, 1 (*E1*)**		
$^{1}_{2}$	**1, 1, 1, 1 (*E1*)**	Rating	1, 2, 1, 0 (*E7*)
		$^{2}_{2}$	**1, 2, 1, 0 (*E7*)**
		$^{2}_{3}$	**1, 2, 1, 0 (*E7*)**
		Text	1, 2, 1, 0 (*E7*)
Reviews	1, 1, 1, 1 (*E1*)	**$^{2}_{3}$**	**1, 2, 1, 0 (*E7*)**
$^{1}_{2}$	**1, 1, 1, 1 (*E1*)**		1, 2, 1, 0 (*E7*)
	1, 1, 1, 1 (*E1*)		1, 1, 1, 1 (*E1*)
	1, 2, 1, 0 (*E7*)	</body>	1, 1, 1, 1 (*E1*)
$^{2}_{1}$	**1, 2, 1, 0 (*E7*)**	</html>	1, 1, 1, 1 (*E1*)

FindEquiv – 2nd iteration

- Find LFEQs using SizeThre = 3, SupThres = 3
  - *E1* : <html> <body> $^1_1$ Book Name^1 $^1_1$ $^1_2$ Reviews $^1_2$ </body> </html>
  - *E7* : <li> $^2_1$ Reviewer Name^2 $^2_1$ $^2_2$ Rating $^2_2$ $^2_3$ Text $^2_3$

Equivalence Classes	SizeThre	SupThres	Equivalence Classes	SizeThre	SupThres
E1	**13**	4			
			E7	**12**	3

HandInv – 2nd iteration

- In this iteration, there is no invalid LFEQs.

DiffEq – 2nd iteration

- This time, there is no tokens to be differentiated by the contexts
- Returns 2 LFEQs
  - *E1* : <html> <body> $^{1}_{1}$ Book Name^{1} $^{1}_{1}$ $^{1}_{2}$ Reviews $^{1}_{2}$ </body> </html>
  - *E7* : <li> $^{2}_{1}$ Reviewer Name^{2} $^{2}_{1}$ $^{2}_{2}$ Rating $^{2}_{2}$ $^{2}_{3}$ Text $^{2}_{3}$

ConstTemp

- Using the set of LFEQs found by ECGM (Equivalence Class Generation Module), construct the template for the input Web Pages
- From the root equivalence class (with occurrence vector<1, 1, ...>), for each non-empty positions of each equivalence classes, check the *PosStrings* for the patterns like below table

	Pattern	Template
1	$E_j E_j E_j E_j E_j \ldots$	$\{T_{E_j}\}_e$
2	$E_j S E_j S E_j S E_j S \ldots E_j$	$\{T_{E_j}\}_S$
3	E_j or E_k	$T_{E_j} \mid T_{E_k}$
4	e or E_j	$(T_{E_j})?$
5	String of dtokens and empty equivalence classes	T_B
6	Unknown	T_B

ConstTemp

- Find root equivalence class: *E1*
- Partition E1 with non-empty positions
  - *E1* : (<html><body>Book Name)$_{C1}$ PosString(E1, 6) (Reviews
)$_{C2}$ PosString(E1, 10) (
</body></html>)$_{C3}$
  - *E7* : (
Reviewer Name)$_{C1}$ PosString(E7, 1) (Rating)$_{C2}$ PosString(E7, 2) (Text)$_{C3}$ PosString(E7, 3) ()$_{C4}$
- Check the PosStings to find the patterns

PosString	Pattern	Template
PosString(*E1*, 6)	String of dtokens	T_B
PosString(*E1*, 10)	*E7 E7* …	$\{T_{E7}\}_e$
PosString(*E7*, 5)	String of dtokens	T_B
PosString(*E7*, 8)	String of dtokens	T_B
PosString(*E7*, 11)	String of dtokens	T_B

ConstTemp

- Returns the template
  - <(<html><body>Book Name)$_{C1}$ *(A) (Reviews
)$_{C2}$ {(
Reviewer Name)$_{C4}$ *(B) (Rating)$_{C5}$ *(C) (Text)$_{C6}$ *(D) ()$_{C7}$}$_e$ (
</body></html>)$_{C3}$>

ExVal

- Using the template constructed by ConstTemp, extract values of each web pages

Page	A	B	C	D	
1	Databases	John	7	…	
2	Data Mining	Jeff	2	…	
		Jane	6	…	
3	Query Opt.	John	8	…	
4	Transactions	*e*	*e*	*e*	