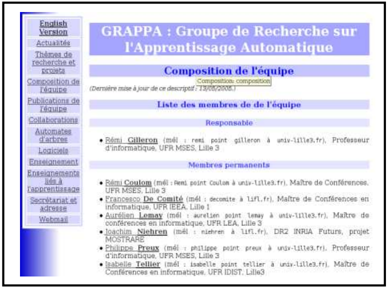
Web Wrapper Specification Using Compound Filter Learning

Julien Carme

May 25, 2006

Plan

Web Information Extraction



Task:

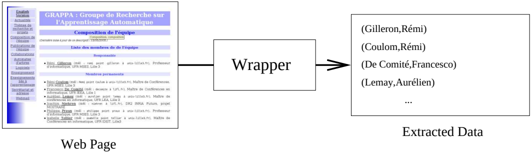
- Automatically access Information available on the Web
- Example: extract member names on labs web sites.

Difficulty: data organization is

- In a layout description language, not adapted to automatic processing
- Specific to each Web site

Wrappers

Standard approach: writing wrappers, programs which turn HTML pages into XML or databases.



Wrappers are:

- Specific to a given website
- Specific to a given task

→ Need for an efficient way to write wrappers

Writing Wrappers

Different ways:

- Using a programming language
  - Generic: Perl, Python...
  - Specialized: XPATH, XSLT
- Using a specification GUI
  - Graphical interface over a wrapper description language
  - Lixto, W4F . . .

In both cases, writing wrappers:

- Is time-consuming
- Needs some knowledge from the user

Learning Wrappers

To solve these problems, we *learn* wrappers:

- The user provides some examples (via mouse clicks in a browser)
- The system *infers* a wrapper from these examples

Advantages:

- Fast
- The user needs no knowledge

Difficulty: the choice of a formalism for wrapper description

- Expressive enough
- Efficiently learnable

Learning Wrappers: Related Works

Machine learning in wrapper design has been experimented in several works:

- WIEN
  - Delimiters computation
- Stalker
  - Document structure inference
- Roadrunner
  - Sequence matching
- Squirrel
  - Tree automata inference
- . . .

Learning Wrappers in Lixto

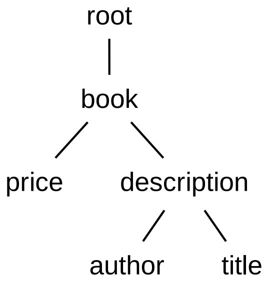
We are interested in the integration of wrapper learning in Lixto.
Constraints:

- Constraints of interactive learning:
  - High performances with few examples
  - Very small computing time
- Constraints specific to Lixto
  - Good intelligibility, for human checking and edition.

Plan

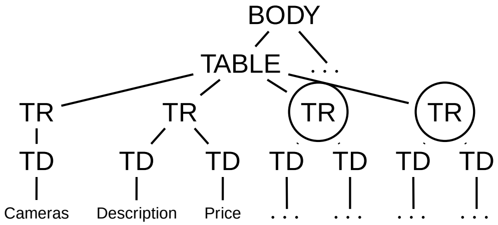
Framework (1)

- Wrapper: hierarchy of patterns
- Pattern: defines the extraction of one type of elements
- Each pattern is defined independantly



Framework (2)

- Documents are trees
- Patterns are nodes selection functions in these trees



Defining patterns

How do we define these functions? Two objects:

- Filters:
  - Simple node selection functions
  - Poorly expressive
  - Very intelligible
- Compound filters:
  - Combination of filters
  - Very expressive
  - Still very intelligible

Filters

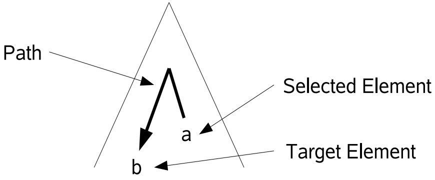
Defined by:

- A path p
  - Input a *starting node*
  - Output a set of *target nodes*
- A test t
  - Input a *node*
  - Output *true* or *false*

A node a is selected if and only if there exists a node b such that:

- $b \in p(a)$ (the path p leads from a to b)
- $t(b)$ (b satisfies t)

Filter: Example

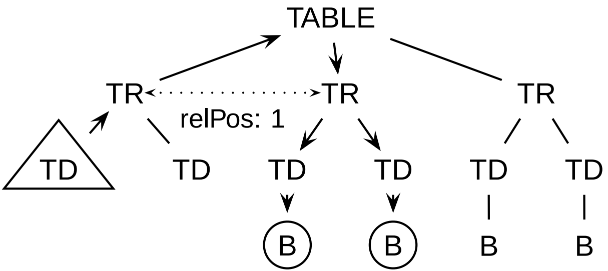


The node a is selected by the filter (p, t) iff there exists b such that p leads from a to b and b satisfies the test t

Paths

Describe a way to walk from a node to another, in a way similar to XPATH. Defined by:

- An ascending path (ex: /par::TD/par::TR)
- A relative position (ex: +1, or *)
- A descending path (ex: /TR[1]/TD/B[1])



Tests

Output *true* or *false* depending on the properties of input node.
Different kind of tests:

Null test:	Always output *true*
Text test:	Output *true* iff the Text Data of the input node is equal to a given value
Attribute test:	Output *true* if the input node contains a given attribute
Attribute value test:	Output *true* if the input node contains a given attribute with a given value

Example (1)

Selects a node in a list whose left brother contains the string "Price":

- Path:
  - Ascending: $/par :: LI$
  - Relative position: -1
  - Descending: $/LI$
- Test:
  - Text test
  - Value: "Price"

Example (2)

Selects a node which *ID* attribute is equal to “description”:

- Path:
  - Ascending: $/$
  - Relative position: 0
  - Descending: $/$
- Test:
  - Attibute value test
  - Attribute: *IS*
  - Value: “description”

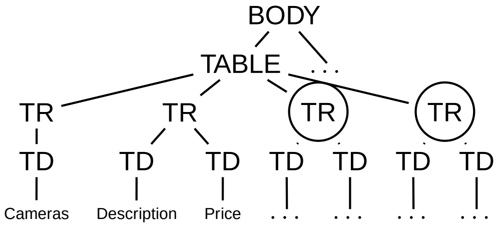
Example (3)

Selects a node addressed by the XPATH $/HTML/BODY/H1[1]$ from the root.

- Path:
  - Ascending: $/par :: H1[1]/par :: BODY/par :: HTML$
  - Relative position: 0
  - Descending: $/$
- Test:
  - Null test

Compound Filters

Combinations of filters, with operators OR, AND and NOT



C_1: brother of "Cameras"
C_2: second child of its parent
C_3: target of path /BODY/TABLE/TR
Result: C_1 AND C_3 AND (NOT C_2)

Defining Wrappers: Conclusion

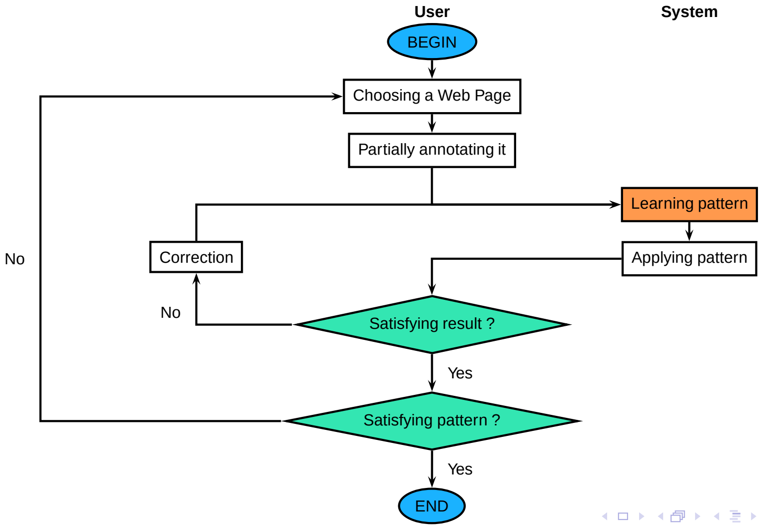
We have defined objects defining patterns which are:

- Intelligible
- Expressive

We show now how to learn these objects from a user interaction.

Plan

Interactive Learning



Learning from positive and negative example

Interactive learning
$\longrightarrow$ Learning from positive and negative examples

Two kind of interactions:

- The user adds a new element
  $\longrightarrow$ provides a positive example
- The user removes a selected element
  $\longrightarrow$ provides a negative example

At each interaction, a new pattern is learned from examples provided by all previous interactions.

Learning algorithm

Goal: given a set of examples, find a compound filter, which:

- is consistant with all examples
- is as small as possible
- is constituted with filters as “simple” as possible

Overview of the algorithm:

- Exhaustive generation of all filters consistant with at least one example
- Selection of a set of optimal filters
- Generation of an optimal combination of filters of this set

Learning algorithm: filter generation

We generate the set of all filters consistant with at least one example

All filters (p, t) such that:

- p leads from a to b, where a is an example
- t is satisfied for b

Learning algorithm: filter selection (1)

Filters are compared considering their behaviour on examples.

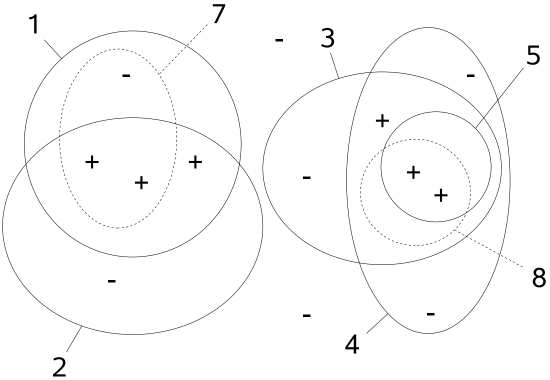
Let:

- E be the set of examples
- f_1 and f_2 be two filters
- E_{f_1} and E_{f_2} be subset of E satisfied by f_1 and f_2.

Then:

- If $E_{f_1} \subsetneq E_{f_2}$, then f_1 is discarded.
- If $E_{f_1} = E_{f_2}$, then f_1 or f_2 is discarded.

Learning algorithm: filter selection (2)



- 7 is discarded, because $E_7 \subsetneq E_1$
- 8 (or 5) is discarded, because $E_5 = E_8$

Learning algorithm: filter selection (3)

When two filters behave similarily on examples, a choice based on heuristics is done between them.

These heuristics tend to choose the “simplest” filter:

- The shortest path
- The most generic path
- The simplest test. In order of preference:
  - Null test
  - Text test defined on the shortest possible text
  - Attribute test
  - Attribute value test

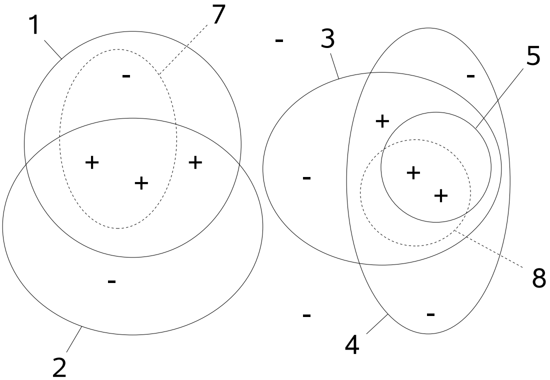
Learning Algorithm: combination

From the remaining filters, an optimal combination is computed.

This can be reduced to a standard boolean function learning problem:

- Each example x is a vector of boolean $(x_1 \ldots x_n)$
- Each x_i is the consistance of x with filter i.
- The target function:
  - Inputs a vector $(x_1 \ldots x_n)$
  - Outputs *true* for positive examples, *false* for negative examples

Learning algorithm: combination



Optimal compound filter: (1 AND 2) OR (3 AND 4)

Plan

1. Introduction
2. Defining Wrappers
3. Learning Wrappers
4. Conclusion

Conclusion

- Work in progress:
  - Integration in Lixto
  - Extensive tests
- Preliminary results:
  - Fast enough for interactive learning
  - Resulting wrappers close to manually written ones in Lixto
  - Very good results on standard benchmarks
- Perspectives:
  - Combination with a textual information extraction system