

1. Introduction of Stalker

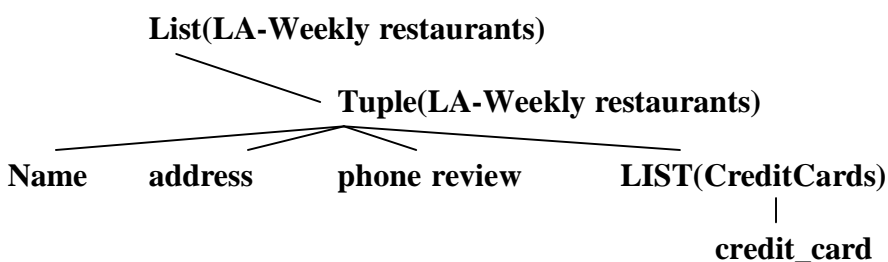
Today we have a lot of information around us, the best example is the www where we can find any information we want. But is there any possibility to get information without seeking in google or a similar search engine?

Yes there is a possibility, it is called STALKER algorithm, which is based on user labelled training examples.

The problem is, that there are thousand of html pages that contain maybe relevant data. STALKER shows how to get out information very quickly.

2. Describing the Content of a Page

Each HTML page follows its own structure; there are a lot of different tags. The information is often presented in tuples, so there is a formalism needed to describe these tags. This formalism is called ϵC formalism. This formalism splits the content of a page into several tuples k-tuples where each tuple can be either a leaf (l) or a list L. So it is possible to get an embedded list.



3. Extracting Data from a document

First I will explain the rule generation, or better what is a rule and for what is it used for? A document is a sequence of tokens ST and it follows the content root. A key idea behind is the extraction rules can base on “landmarks” –

Example: `<p> Name: xxx <p>`

To get xxx we must determine a rule – R1 and this rule means `SkipTo()`

A second rule we define is `SkipTo()` so we identify the end of the information we want to have. To get a better rule, because R1 is not unique we use `R3 = SkipTo(Name) SkipTo()` instead. We generalize now R3 to `SkipTo(Name Symbol HtmlTag)`.

The explanation of R4 is, ignoring all tokens until you find a 3-token landmark that consist of token Name immediately followed by a punctuation sign and an HTML tag. This is quite a simple example of defining rules.

R4 can be extended to gain more complexity – for example if you want to extract lists and iterations with missing elements.

If the ϵC is available for a document and an extraction rule and a list iteration rule is defined as well, any item of interest can be extracted by simple determining the path P from the root to the corresponding leaf and by successively extraction each node from its parent.

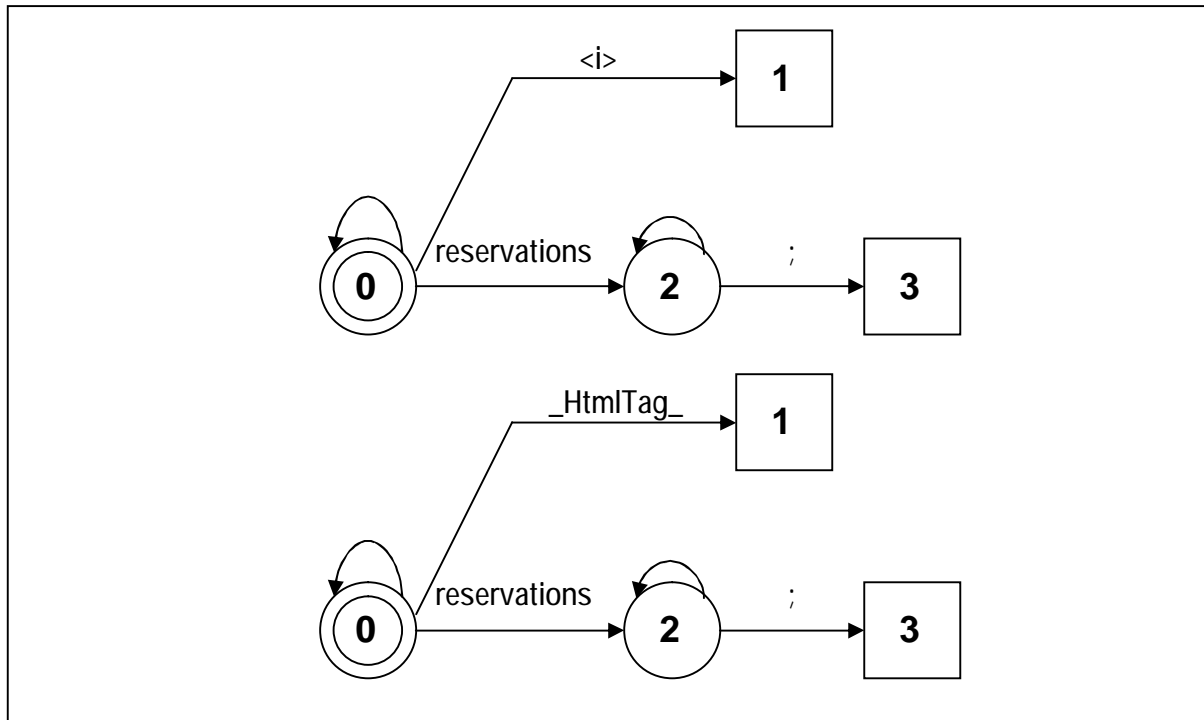
Graphical Example of an extraction rule based on Grammar

D1: wine; parking; reservations suggested; Visa

D2: full bar; no reservations; major cards

D3: full bar; reservations accepted <i> no credit

D4: beer; parking; reservations suggested <i> cash



This grammar shows how we can get out if credit cards are being accepted or not.

4. What is a linear landmark?

It is a normal landmark described by a sequence of tokens that contains wildcards (HTML tags, Number, Signs). Linear landmarks are interesting for two reasons. They will allow efficient navigation within the εC structure of the documents and on the other hand it is quite simple to generate them.

5. Learning Extraction Rules

First the user has to decide which content he is interested in. Therefore a GUI provides the user the document structure displayed as tokens (sequences).

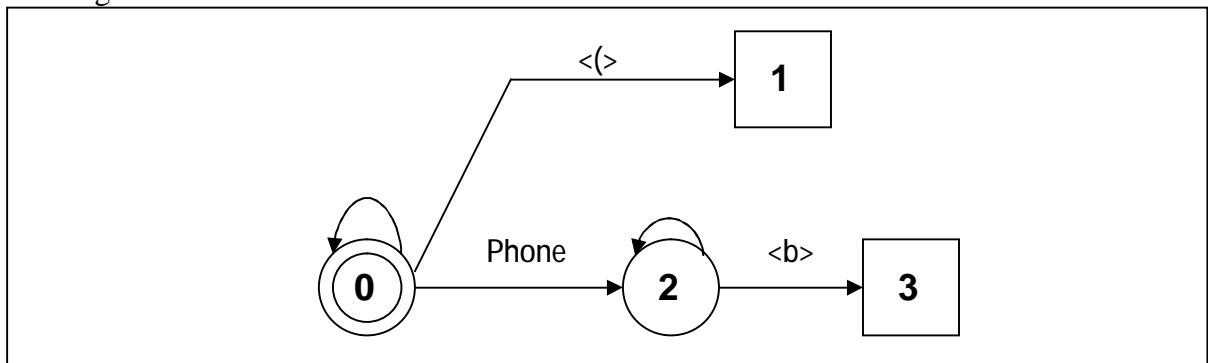
For example have a closer look at these four data entries, we want to extract the prefixes of the phone numbers.

E1: 513 Pica, Venice, Phone: l-800-555-1515
E2: 90 Colfax, Palms , Phone: (818) 508-1570
E3: 523 1st St., LA c/b>, Phone: l-888-578-2293
E4: 403 Vernon, Watts c/b>, Phone: (310)1310 798-0008

The algorithm now creates the first Rule R1::= SkipTo() which accepts E2 and E4 and rejects E1 and E3 because R1 cannot be matched on them.

In the second step there are only uncovered examples E1 and E3 taken.

So the grammar looks like this:



Stalker is a typical sequential covering algorithm –as long as there are uncovered positive examples it tries to learn the perfect disjunct.

5.1 Learning function

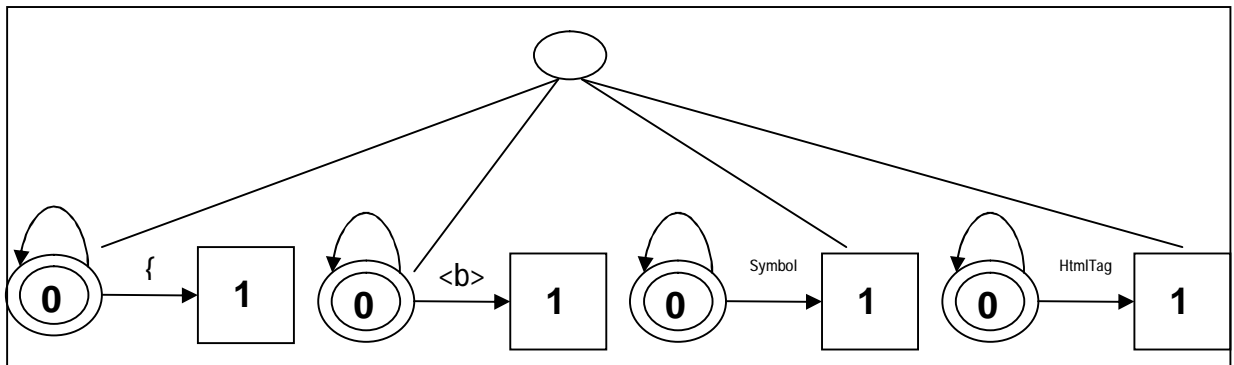
```
STALKER( Examples )
- let RetVal be an empty SLG
- WHILE Examples ≠ 0
- aDisjunct = LearnDisjunct(Examples)
- remove all examples covered by aDisjunct
- add aDisjunct to RetVal
- return RetVal
```

```
LearnDisjunct( Examples )
- Terminals = Wildcards U
GetTokens(Examples)
- Candidates = GetInitialCandidates(
'Ezamples')
- WHILE Candidates ≠ 0 DO
- let D = BestDisjunct(Candidates)
- IF D is a perfect disjunct THEN return D
- FOR EACH t ∈ Terminals DO
Candidates = Candidates U Refine(D, t)
- remove D from Candidates
- return best disjunct
```

LearnDisjunct is a brute force algorithm. It generates a initial set of candidates and repeatedly selects and refines the best candidates.

An initial set of candidates can have the following style

{ Phone : , . HtmlTag Word Symbol }



The Refine() function tries to obtain better disjuncts by making the landmark more specific or by adding new states in the automaton – that called topology refinements. For this refinement STALKER uses a refining terminal which can be either a token or a wildcard – in this point the user can define wildcards.

6. Experimental results

STALKER was compared within Kushmericks WIEN algorithm.

The example was to get out a list extraction rule out of small (4 leaves) up to big documents (18 leaves)

SRC	leaves	Docs	Examples
S1 OKRA	4	252	3335
S2 BigBook	6	235	4299
S3 Address Finder	6	10	57
S4 Quote Server	18	10	22

The experiment has the main goal to get out if STALKER produces “good” rules based on just a few traingsessions. Because no user normally has the patience to run dozen of samples. The amazing result was, that STALKER usually needs just 10 trainingsessions to produce a good result – within an accuracy of 97%.

It is quite a good algorithm – fast an stable.