Processing complex question in the commercial domain

Presented by : Amine Hallili

Advisors : Fabien Gandon

Catherine Faron Zucker

Headlines

- Introduction & motivations
- SynchroBot overview
- Question analysis and modeling
- Learning regex for property value identification
- Evaluation
- Future work

Introduction

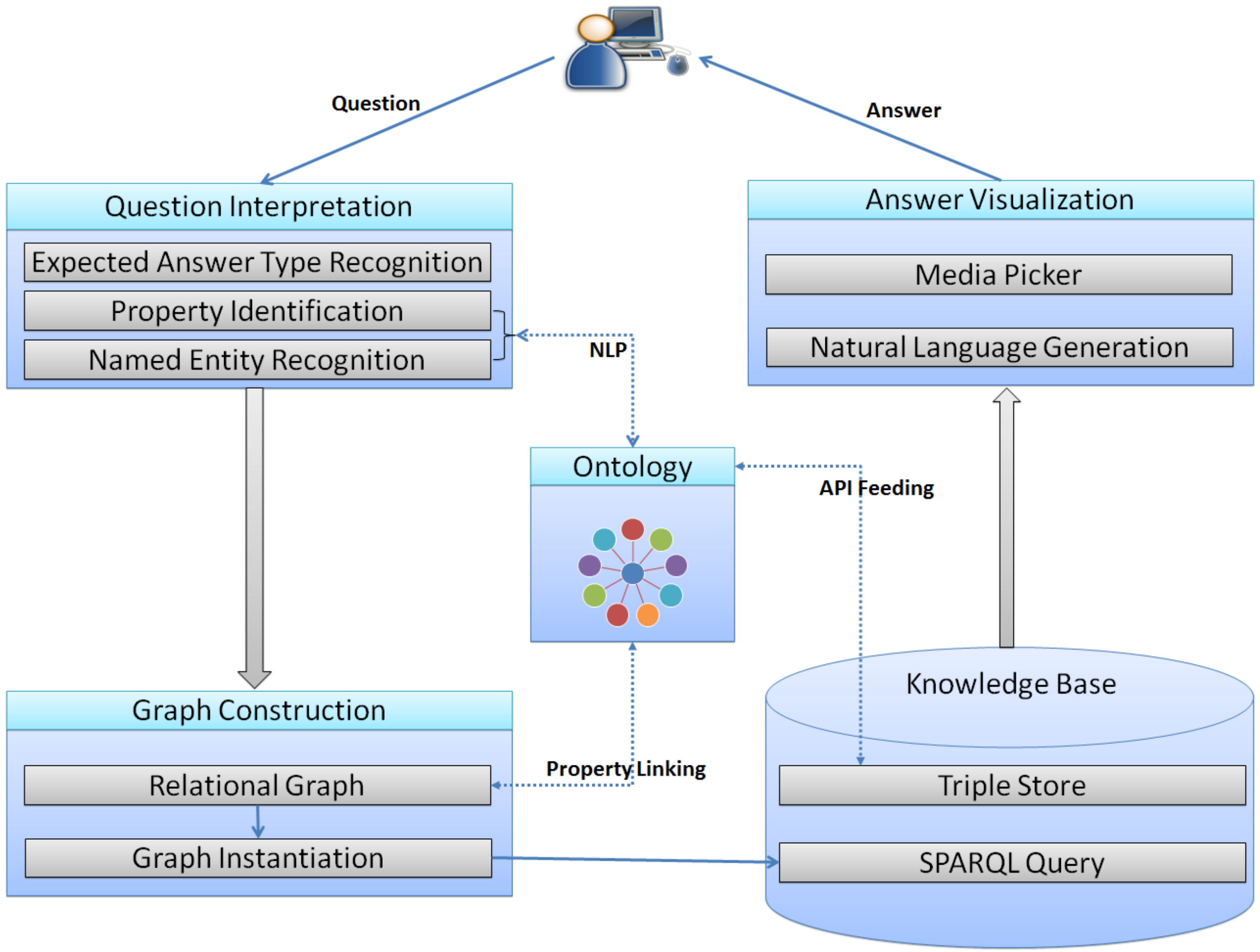
- Huge evolution of the e-Commerce
- Huge amount of data generated every second
- User needs are getting more complex and specific
- Several systems try to satisfy these needs
  - Search engines, comparative shopping systems, question answering systems

- **Research question** : how can a system understand and interpret complex natural language (NL) questions (also known as n-relation questions) in a commercial context?

SynchroBot

- Natural Language Question Answering system for commercial domain
- From QAKiS (open domain)
  => domain specific (e-Commerce)

SynchroBot


Question
Answer
Question Interpretation
Expected Answer Type Recognition
Property Identification
Named Entity Recognition
NLP
Answer Visualization
Media Picker
Natural Language Generation
Ontology
API Feeding
Graph Construction
Relational Graph
Graph Instantiation
Property Linking
Knowledge Base
Triple Store
SPARQL Query

Question Analysis and modeling

Expected Answer Type (EAT) Recognition
Named Entity Recognition (NER)
Property identification

Example : Give me the price of Nexus 5 phone !

EAT Recognition

- Detecting types in NL questions
  - Specifying the type of Named Entities

  Ex : Give me the price of Nexus 5 phone # Give me the price of Nexus 5
  - Specifying the type of resources

  Ex : Give me the price of available phones
- Why ?
  - To improve precision
  - To limit the number of retrieved Named Entities

EAT Recognition

Give me the price of phones cheaper than 200$

```
<rdfs:Class rdf:ID="Phone">
    <rdfs:label xml:lang="fr">Telephone</rdfs:label>
    <rdfs:comment xml:lang="fr">Un produit de type phone, smartphone, cellulaire,...</rdfs:comment>
    <rdfs:label xml:lang="en">phone</rdfs:label>
    <rdfs:comment xml:lang="en">A product which is a phone, smartphone, cellphone,...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="http://i3s.unice.fr/MerchantSiteOntology#Product" />
</rdfs:Class>
```

Give me the address of Nexus 5 seller

```
<rdfs:Class rdf:ID="Seller">
    <rdfs:label xml:lang="en">Seller</rdfs:label>
    <rdfs:label xml:lang="fr">Vendeur</rdfs:label>
    <rdfs:comment xml:lang="Seller">Un vendeur qui peut mettre un ensemble de produits en vente.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="http://i3s.unice.fr/MerchantSiteOntology#Organization" />
</rdfs:Class>
```

Named Entity Recognition

- Classic definition
  - (persons, organizations, locations, times, dates)
- Commercial domain ?
  - More types (Phones, Cases, …)

Named Entity Recognition

mso:legalName	Samsung Galaxy S5
mso:name	AT&T GoPhone - **Samsung Galaxy S5** 4G LTE No-Contract Cell Phone - Dark Gray
mso:description	The 4.5" WVGA Super AMOLED Plus touch screen on this AT&T GoPhone **Samsung Galaxy S5** SGH-i437 cell phone makes it easy to navigate features. The 5.0MP rear-facing camera features a 4x digital zoom and an LED flash for clear image capture.

Give me the price of Samsung Galaxy S5 ?

Give me the price of Samsung S5 ?

Give me the price of Samsung 5 ?

Named Entity Recognition : Algorithm

```
var score = 0
var match // contains the matched string (occurrence in the NL question)
List namedEntities
var stringToMatch // we put the first word. Our goal is to find the
    largest match

for( word in question )
begin

   if (findMatch(stringToMatch)) then
      update(match)
      update(score)
      addNamedEntities(namedEntities)
      stringToMatch = concat(stringToMatch, word)
   else
      if (findMatch(word)) then
         update(match)
         update(score)
         addNamedEntities(namedEntities)
         stringToMatch = word
      endif
   endif

end

cleanNamedEntities()
sortNamedEntities()
computeScore(NamedEntites) // computing score for each named entity
    according to the general number of retrieved named entities
```

Named Entity Recognition : Algorithm

- Example : ”*What is the battery life time of Nokia - Lumia Icon 4G LTE Cell Phone - White (Verizon Wireless)*”

- **Cleaned sentence :** What Nokia Lumia Icon 4G LTE Cell Phone White Verizon Wireless

[What, 0] **->** [Nokia, n] **->** [Nokia Lumia, n] **->** … [Nokia Lumia Icon 4G LTE Cell Phone White Verizon Wireless, n]

- **Cleaned sentence* :** What Nexus 5 Nokia Lumia

[What, 0] **->** [Nexus, n] **->** [Nexus 5, n] **->** [Nexus 5 Nokia, 0] **->** [Nokia, n] **->** [Nokia Lumia, n]

Property Identification

Label based method
Value based method

Label based property identification

```
<rdf:Property rdf:ID="price">
    <rdfs:label xml:lang="en">price</rdfs:label>
    <rdfs:label xml:lang="en">cost</rdfs:label>
    <rdfs:label xml:lang="en">value</rdfs:label>
    <rdfs:label xml:lang="en">worth</rdfs:label>
    <rdfs:label xml:lang="en">tariff</rdfs:label>
    <rdfs:label xml:lang="en">amount</rdfs:label>
     <rdfs:label xml:lang="fr">prix</rdfs:label>
    <rdfs:label xml:lang="fr">couter</rdfs:label>
    <rdfs:label xml:lang="fr">coute</rdfs:label>
    <rdfs:label xml:lang="fr">tarif</rdfs:label>
    <rdfs:label xml:lang="fr">valeur</rdfs:label>
    <rdfs:comment xml:lang="en">The price of a product.</rdfs:comment>
    <rdfs:subPropertyOf rdf:resource="http://schema.org/price" />
    <rdfs:domain rdf:resource="http://i3s.unice.fr/MerchantSiteOntology#Product"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>

    <rdfs:comment xml:lang="fr">Le prix d'un produit.</rdfs:comment>
    <rdfs:comment xml:lang="en">The price of a product.</rdfs:comment>

    <sbmo:responsePattern xml:lang="fr">Le prix de _resource_ est de : _value_ .</
    <sbmo:responsePattern xml:lang="en">The price of _resource_ is  _value_ .</sbm

    <sbmo:regexExtractionPattern><![CDATA[(?i)[0-9]+([,|.][0-9]+)?(euro(s?)|£|\$|€

    <sbmo:mediaType>text</sbmo:mediaType>
    <sbmo:valueType>unit</sbmo:valueType>

</rdf:Property>
```

Give me the price of Nexus 5 !

Value based property identification

```
<rdf:Property rdf:ID="price">
    <rdfs:label xml:lang="en">price</rdfs:label>
    <rdfs:label xml:lang="en">cost</rdfs:label>
    <rdfs:label xml:lang="en">value</rdfs:label>
    <rdfs:label xml:lang="en">worth</rdfs:label>
    <rdfs:label xml:lang="en">tariff</rdfs:label>
    <rdfs:label xml:lang="en">amount</rdfs:label>
     <rdfs:label xml:lang="fr">prix</rdfs:label>
    <rdfs:label xml:lang="fr">couter</rdfs:label>
    <rdfs:label xml:lang="fr">coute</rdfs:label>
    <rdfs:label xml:lang="fr">tarif</rdfs:label>
    <rdfs:label xml:lang="fr">valeur</rdfs:label>
    <rdfs:comment xml:lang="en">The price of a product.</rdfs:comment>
    <rdfs:subPropertyOf rdf:resource="http://schema.org/price" />
    <rdfs:domain rdf:resource="http://i3s.unice.fr/MerchantSiteOntology#Product"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>

    <rdfs:comment xml:lang="fr">Le prix d'un produit.</rdfs:comment>
    <rdfs:comment xml:lang="en">The price of a product.</rdfs:comment>

    <sbmo:responsePattern xml:lang="fr">Le prix de _resource_ est de : _value_ .</
    <sbmo:responsePattern xml:lang="en">The price of _resource_ is  _value_ .</sbm

    <sbmo:regexExtractionPattern><![CDATA[(?i)[0-9]+([,|.][0-9]+)?(euro(s?)|£|\$|€

    <sbmo:mediaType>text</sbmo:mediaType>
    <sbmo:valueType>unit</sbmo:valueType>

</rdf:Property>
```

Give me details of the products cheaper than 200$

Value based property identification

- Constraints :
- A value can correspond to multiple properties
  - 200$ -> [price, cost]
- A property can have multiple values
  - Storage [4GB, 8GB]

Must be handled during the graph construction

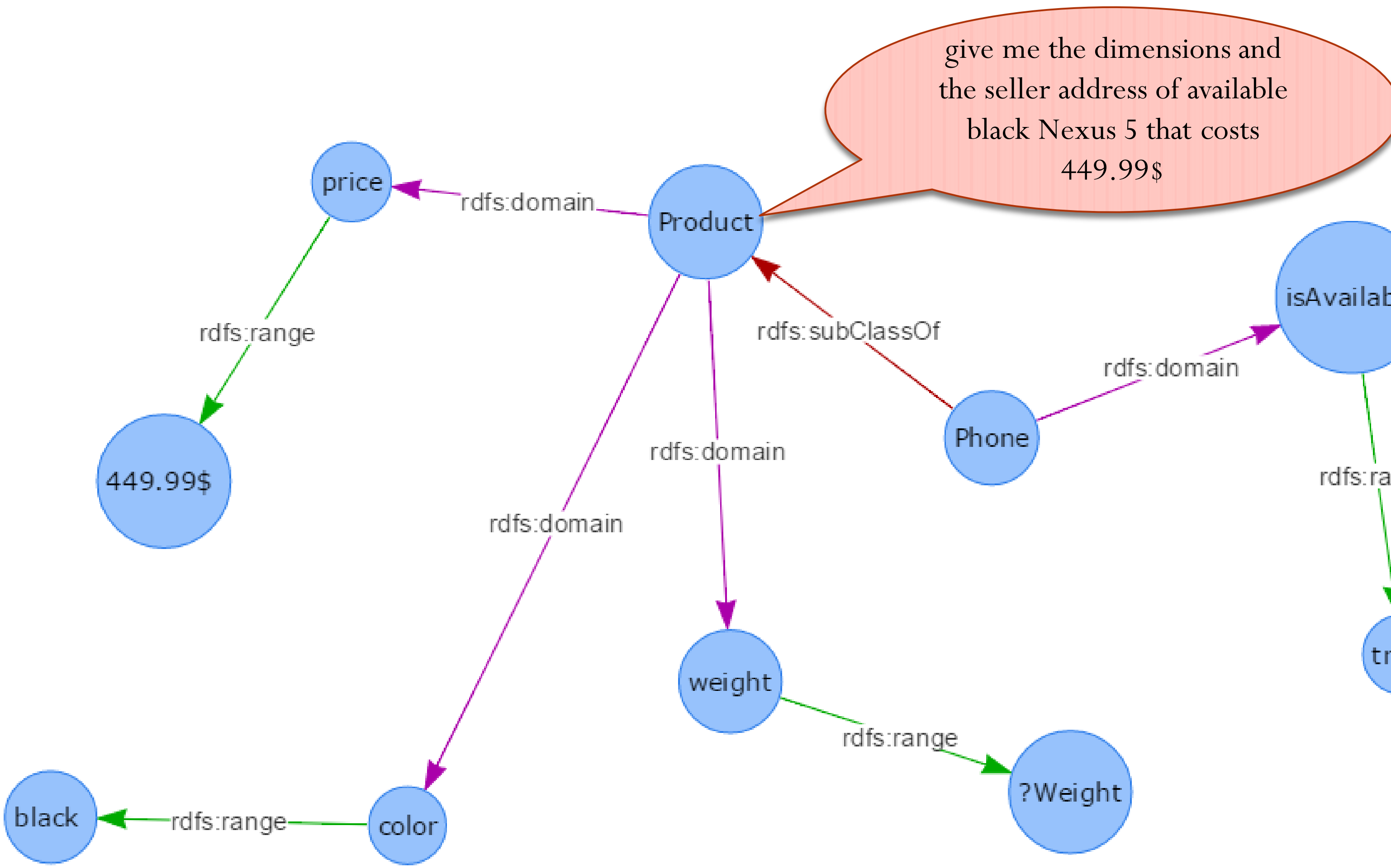
Graph construction

Relational graph creation

Graph instantiation

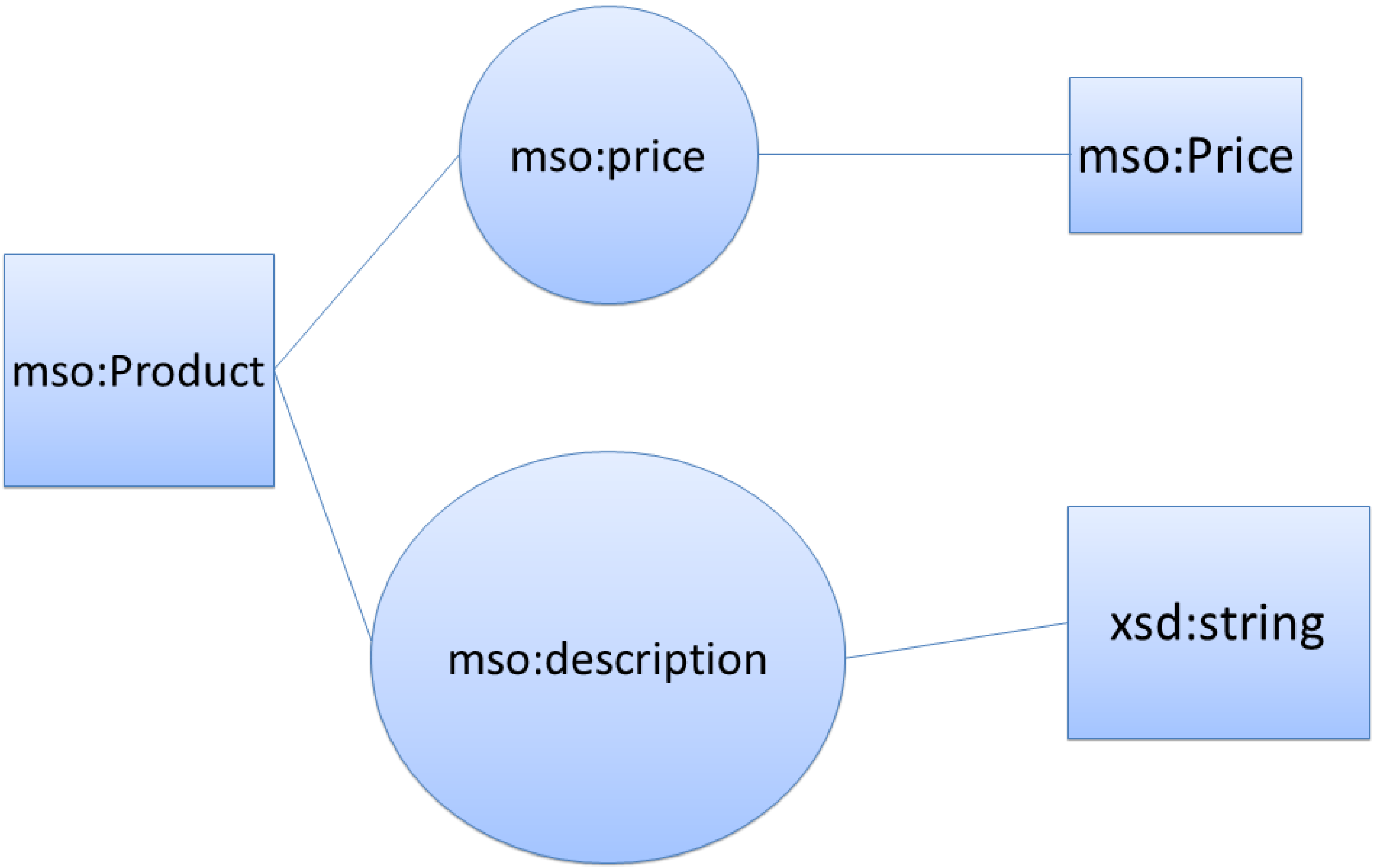
Graph construction

Goal : creating one connected graph to generate SPARQL query



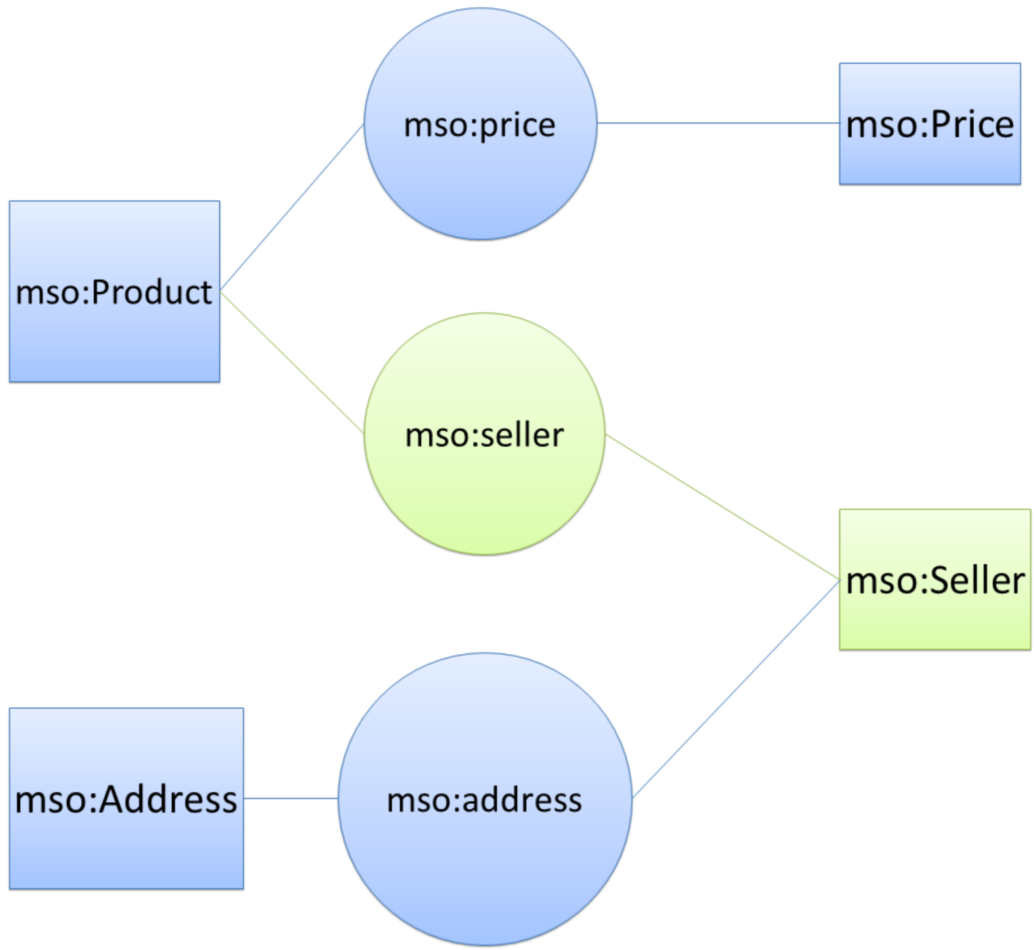
Relational graph creation

Give me details about the products cheaper than 200$



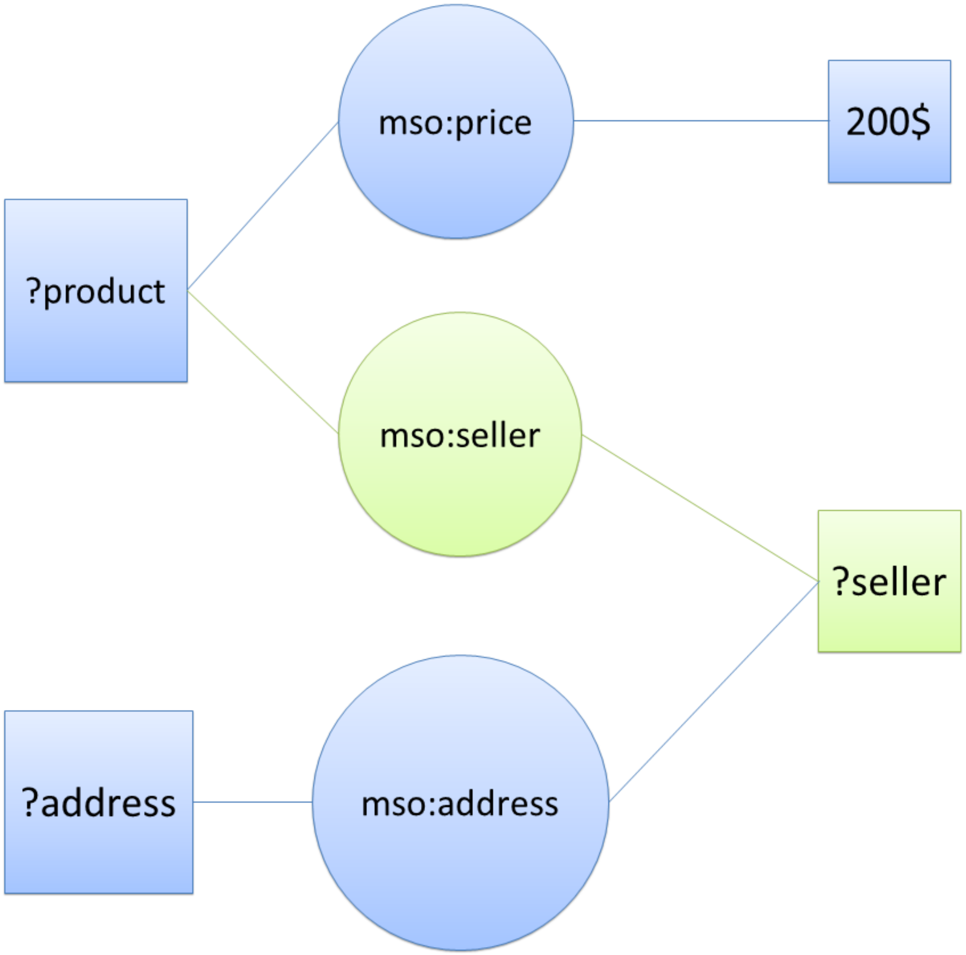
Relational graph creation

Give me the address of the products cheaper than 200$



Graph instantiation

Give me details about the products cheaper than 200$



SPARQL query

Give me details about the products cheaper than 200$

```
Select distinct *
where {
?ne a <http://i3s.unice.fr/MerchantSiteOntology#Product>
?ne <http://i3s.unice.fr/MerchantSiteOntology#name> ?n
optional {
    ?ne <http://i3s.unice.fr/MerchantSiteOntology#description> ?var1
  }
optional {
    ?ne <http://i3s.unice.fr/MerchantSiteOntology#price> ?v
    ?v rdf:value ?var2
    filter (contains (?var2, lcase(str("200"))))
  }
bind( IF(bound(?var1),1,0)+ IF(bound(?var2),1,0) as ?c)
}
order by desc (?c) limit 20
```

Learning regex Automatically

Why ?

Anticipating most forms of property values

In case new properties are introduced

In case the domain is changed

Learning regex Automatically

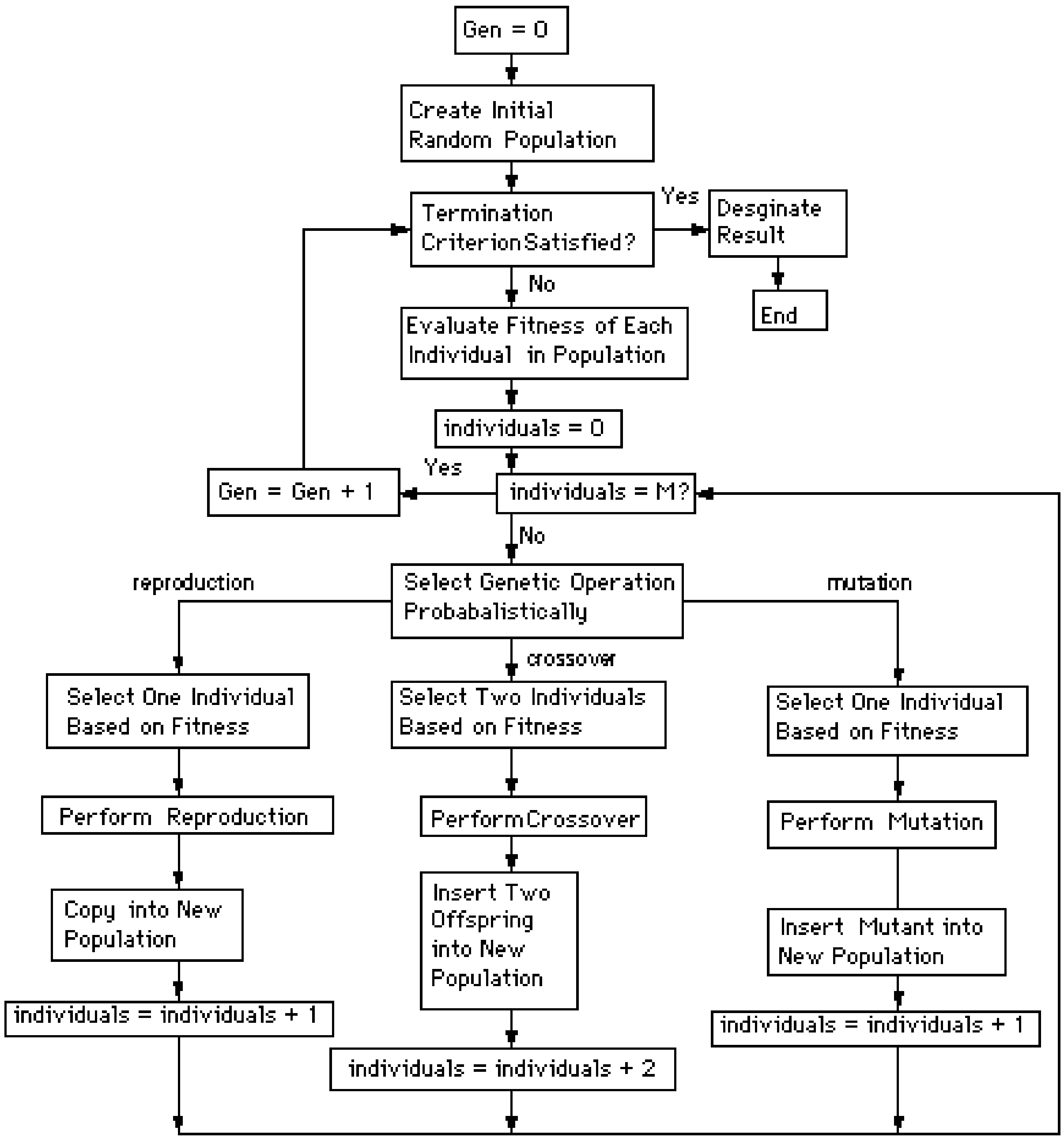
- Genetic Programming (GP) approach :
- "In artificial intelligence, **genetic programming (GP)** is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user-defined task" - *Wikipedia*

Genetic Programming : Goal

[Petrovski et al. 2014][Bartoli et al. 2012]

Text	Value to extract	regex
Patriot Memory - FUEL+ 5200 mAh Rechargeable Lithium-Ion Battery and Signature Series **8GB** microSDHC Memory Card & 8GB	8GB	?
Apple - iPhone 4s 8GB **499.99$** Cell Phone - Black (Verizon Wireless)	499.99$	?
Nokia - **Lumia 1520** 4G Cell Phone - Black (AT&T)	Lumia 1520	?
HTC - One (M7) 4G LTE with 32GB Memory Cell Phone - **Black** (Sprint) & 32GB	Black	?

Flowchart for Genetic Programming



Genetic programing : algorithm

- Create population (500 individuals)
- Repeat 150 or precision = 1
  - For each individual
    - For each example
      - Compute individual fitness
  - While new population < 500
    - Select 2 individuals
    - crossover

Genetic programming

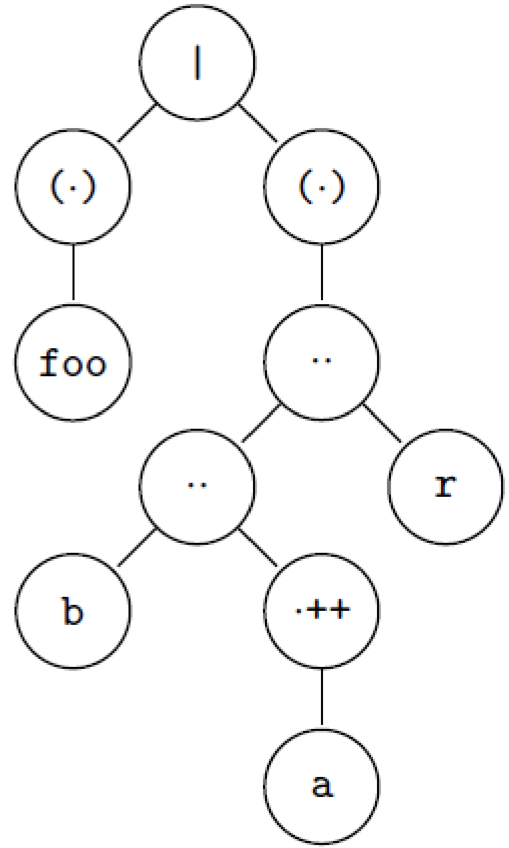
- Individuals : valid regex represented by a tree

Operators :

- concatenate node : a binary node to concatenate two leaves.
- possessive quantifiers : {"*+", "++", "?+", "m,n"}
- Group operator : "()"
- Class operator : "[]"

Terminals :

- constants : a single character, a number or a string.
- Ranges : "a-z", "0-9", "a-z0-9", "A-Z"
- Character class : {"\w", "\d"}
- White space : "\s"
- Wildcard character : "."



(foo)|(ba++r)

Genetic programming

- Population :
  - Half of the population derived from the examples by replacing : (characters, \w) and (numbers, \d)
    - (''200$" -> ''\d\d\d\w") (32GB -> \d\d\w\w)
  - The other Half is generated randomly using the ramped half-and-half method
    - Generate random trees with different depth

Genetic programming

- Fitness function :
  - Precision

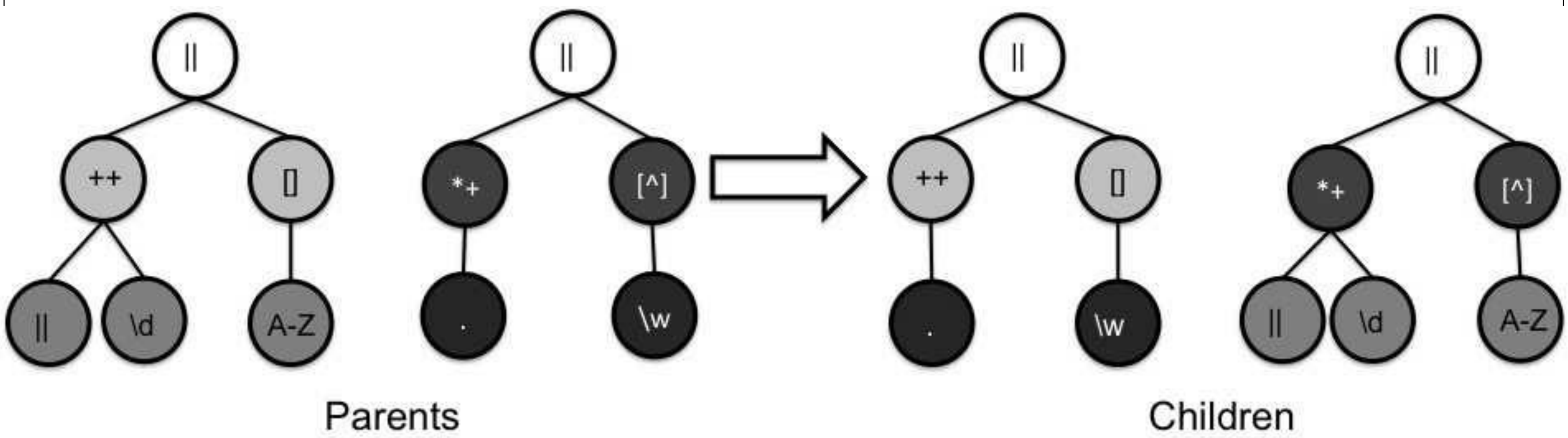
$$\text{Precision} = \frac{tp}{tp + fp}$$

  - Matthews Correlation Coefficient (MCC)

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Genetic Operation

Crossover | Mutation | Reproduction



P.S : Before performing genetic operation, node compatibility must be checked

Selection

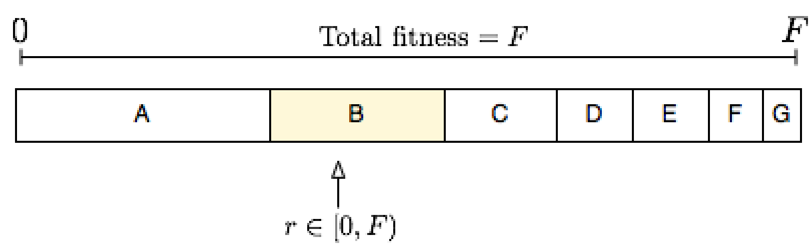
- Fitness proportionate selection also known as the roulette wheel selection

$$p_i = \frac{f_i}{\Sigma_{j=1}^{N} f_j}$$

where N is the number of individuals

- The selection token (r) is randomly generated

$$0 < r < \Sigma_{j=1}^{N} f_j$$



Evaluation

Genetic programming result

SynchroBot performances

GP : result

<table>
<tr><th>Property</th><th>Precision</th><th>Automatic Regex</th><th>Manually regex</th></tr>
<tr><td>storage</td><td>100%</td><td>[0-9]++G[a-zA-Z]</td><td>\d++[Gg][Bb]</td></tr>
<tr><td>price</td><td>97,33%</td><td>\d++.\d++\D</td><td>(?i)[0-9]+([,|.][0-9]+)?(euro(s?)|£|\$|€|dollar(s?))</td></tr>
<tr><td>Release date</td><td>~60%</td><td>\d\d\W[0-9]++\D\d?+</td><td>((19|20)\d\d)[\-/](0?[1-9]|1[012])[\-/](0?[1-9]|[12][0-9]|3[01])</td></tr>
<tr><td>…</td><td></td><td></td><td></td></tr>
<tr><td>model</td><td>~20%</td><td>(?:[^\d]+\s[a-z0-9]+)*+</td><td>([A-Z]\w++)+*([A-Z]\d)</td></tr>
<tr><td>color</td><td>~11%</td><td>\w\w\w\w</td><td>(?i)aliceblue|antiquewhite|aqua|aquamarine|azure|beige|bisque|black…</td></tr>
</table>

SynchroBot

QALM [Hallili et al 2014] : Question Answering Linked Merchant data)

Benchmark for evaluating question/answering systems that use commercial data

Questions / Number	Training	Goldstandard	Handled		
			v1	v2	v3
1-relation questions	15	12	yes	yes	yes
2-relations questions	9	8	no	yes	yes
N-relations questions	2	5	no	yes	yes
Named-Entityless questions	19	11	no	yes	yes
Boolean questions	7	4	partially	partially	partially
Aggregations questions	8	4	no	no	no

Table 3. Question analysis

	Training			Goldstandard		
	v1	v2	v3	v1	v2	v3
Answered questions	21/40	25/40	25/40	13/30	17/30	17/30
Right answers	4/21	7/25	7/25	3/13	5/17	5/17
Partially right answers	10/21	16/25	10/17	16/25	10/17	10/17

Table 4. General analysis

SynchroBot

	Precision		
	Version 1	Version 2	Version 3
Limited set	19%	25, 44%	38%
Whole set	10,23%	21,01%	35,56%

Conclusion & future work

- Proposing generic NE classification for domain specific systems
- Optimizing the learning of regular expression (LRE)
- Applying the LRE to other topical domains