Dependency-Based Semantic Interpretation for Answer Extraction

Diego Mollá-Aliod

Ben Hutchinson

Dependency-Based Semantic Interpretation for Answer Extraction

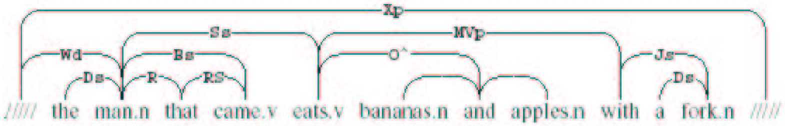
- Dependency-based Parsing Systems
  - Link Grammar
  - Conexor
- Answer Extraction
  - ExtrAns
- Semantic Interpretation
  - Top-down
  - Bottom-up

Dependency-based Parsing Systems

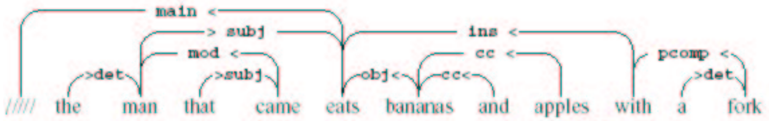
- Parsing systems
  - Parser
  - Comprehensive grammar of English
- Link Grammar and Conexor are dependency-based parsing systems
  - The output is a dependency structure

Dependency Structures

- Link Grammar


Xp
Ss
MVp
Wd
Bs
O^
Js
Ds
R
RS
Ds
///// the man.n that came.v eats.v bananas.n and apples.n with a fork.n /////

- Conexor



Semantic Interpretation

- The Problem
  - Given a dependency structure, how to build the logical form?
    - Building the logical form while parsing is not an option
- Two approaches:
  - Top-down
  - Bottom-up

Dependency-Based Semantic Interpretation for Answer Extraction

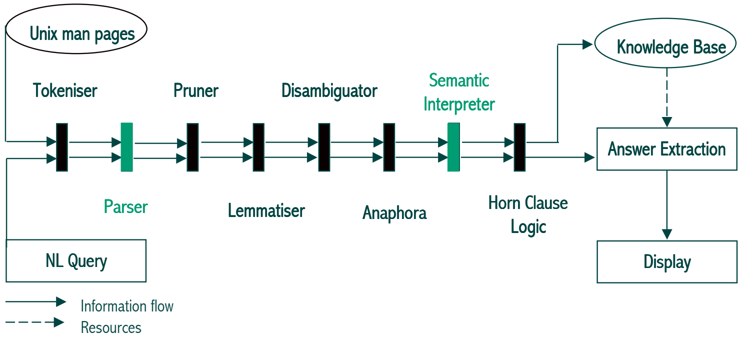
- Dependency-based Parsing Systems
  - Link Grammar
  - Conexor
- Answer Extraction
  - ExtrAns
- Semantic Interpretation
  - Top-down
  - Bottom-up

Answer Extraction

- The Goal of Answer Extraction (AE) is …
  - … to locate exact passages within text documents …
  - … that answer a question worded in natural language.
- Answer Extraction is *not* Information Retrieval (IR)
  - We want answers, not pointers to documents/passages
- Answer Extraction is *not* Question Answering (QA)
  - AE is less ambitious than QA
    - The first editions of TREC-QA are about AE

ExtrAns

- ExtrAns is an AE system that operates over UNIX manual pages
- WebExtrAns operates over Airbus maintenance manuals
  - (SG|X)ML formatting



Unix man pages

Tokeniser

Pruner

Disambiguator

Semantic Interpreter

Knowledge Base

Answer Extraction

Parser

Lemmatiser

Anaphora

Horn Clause Logic

NL Query

Display

Information flow

Resources

ExtrAns' Logical Forms

- Goals of ExtrAns' Logical Forms
  - Expressivity: Be able to express (part of) the meaning of of any sentence
    - Incrementally add more semantic contents if necessary
  - Robustness: Be able to get something out from ungrammatical/unexpected sentences
  - Computability: Be easy to build and to work with
    - Specially for Answer Extraction

ExtrAns' Answer Extraction

A "bag of predicates" approach

- *cp will quickly copy files*

```
1. holds(e4)
2. object(cp,o1,[x1])
3. object(command,o2,[x1])
4. evt(copy,e4,[x1,x6])
5. object(file,o3,[x6])
6. prop(quickly,p3,[e4])
```

- *which command copies files?*

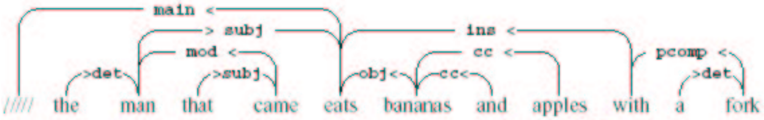
```
?-
1. object(command,O1,[X1]),
2. evt(copy,E4,[X1,X2]),
3. object(file,O2,[X2]).
```

Dependency-Based Semantic Interpretation for Answer Extraction

- Dependency-based Parsing Systems
  - Link Grammar
  - Conexor
- Answer Extraction
  - ExtrAns
- Semantic Interpretation
  - Top-down
  - Bottom-up

Semantic Interpretation

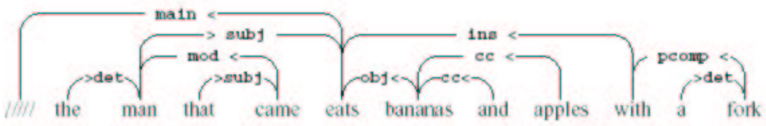
- Input:



- Output:

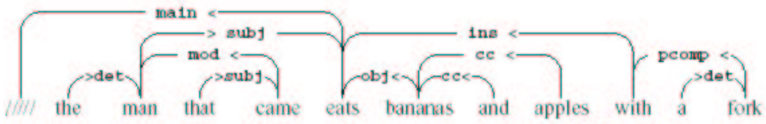
holds(v_e5), object('man',v_o2,[v_x2]), evt('come',v_e4,[v_x2]), evt('eat',v_e5,[v_x2,v_x7]), (v_x6<$v_x7), (v_x8<$v_x7), object('banana',v_o6,[v_x6]), object('apple',v_o8,[v_x8]), prop('with',v_p9,[v_X9,v_x11]), object('fork',v_o11,[v_x11])

Semantic Interpretation: Top-Down



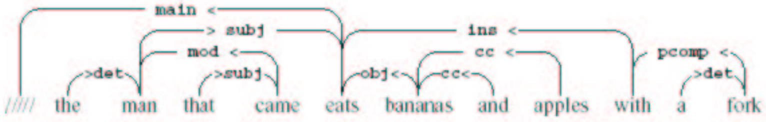
- Starting from the anchor symbol ("/////"), follow the dependencies in reversed direction
- The dependency label indicates the type of dependent
- The far end of the dependency points to the head of the dependent

Semantic Interpretation: Top-Down



1. Find the head of the main sentence
   - follow the link "main" to find *eats*
2. Find the head of the subject
   - follow the link "subj" to find *man*
3. Build the logical form of the subject
   - follow the link "mod" to find the relative clause
   - find the logical form of the clause (recursive call)
     - but this time the subject is found by following "mod"

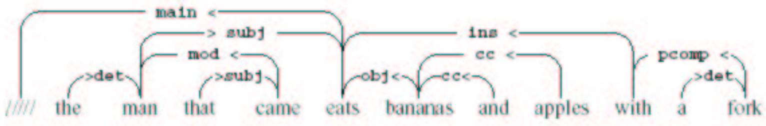
Semantic Interpretation: Top-Down



4. Build the logical forms of the other verb arguments
   - follow the link "obj" to find the head of the direct object
   - build the logical form of the direct object
5. Build the logical forms of other complements and adjuncts
   - follow the link "ins" to find the prepositional phrase
6. Add the logical form of the main event and the **holds** predicate

Semantic Interpretation

- Input:



- Output:

  holds(v_e5), object('man',v_o2,[v_x2]), evt('come',v_e4,[v_x2]), evt('eat',v_e5,[v_x2,v_x7]), (v_x6<$v_x7), (v_x8<$v_x7), object('banana',v_o6,[v_x6]), object('apple',v_o8,[v_x8]), prop('with',v_p9,[v_X9,v_x11]), object('fork',v_o11,[v_x11])

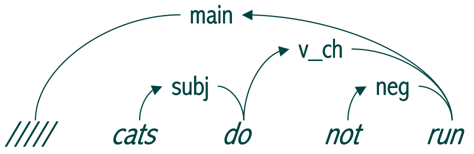
Top-Down and Robustness

- If a dependency structure is incomplete or contains an unexpected dependency, complete sentence constituents will be ignored
  - Some special syntactic structures are handled by the parsing system but are not recognised by the semantic interpreter
- Solution:
  1. Collect the words that have not been covered by the top-down algorithm
  2. Follow the dependencies bottom-up until the heads are found
  3. Use variants of the top-down algorithm starting from the heads
  4. Repeat the procedure until no additional predicates are produced

Semantic Interpretation: Bottom-Up

- The error recovery from the top-down method has a bottom-up component
- Why not do everything bottom-up?
- Three stages in the bottom-up approach
  - Introspection
    - For each word, build the corresponding predicate
    - Some information in the resulting predicates may be missing
  - Extrospection
    - For each word, examine its head and fill the missing information
  - Reinterpretation
    - Do some final adjustments to the logical form

Bottom-Up – Example



- Introspect(cats): object(cat,o2,[x2])
- Introspect(not): object(cat,o2,[x2]), log_op(not,l4,[?])
- Introspect(run): object(cat,o2,[x2]), log_op(not,l4,[?]), evt(run,e5,[?])
- Extrospect(cats): object(cat,o2,[x2]), log_op(not,l4,[?]), evt(run,e5,[x2])
- Extrospect(not): object(cat,o2,[x2]), log_op(not,l4,[e5]), evt(run,e5,[x2])
- Extrospect(run): object(cat,o2,[x2]), log_op(not,l4,[e5]), evt(run,e5,[x2]), holds(e5)
- Re-interpretation: object(cat,o2,[x2]), log_op(not,l4,[e5]), evt(run,e5,[x2]), holds(l4)

Bottom-up and Robustness

- The logical form contains all the basic predicates
  - The introspection stage explores all words in the sentence
- Missing/unexpected dependencies translated into unconnected variables
  - The extrospection stage may fail to follow the dependencies

⇩

The bottom-up approach is robust by nature

Logical Forms and Semantic Interpretation

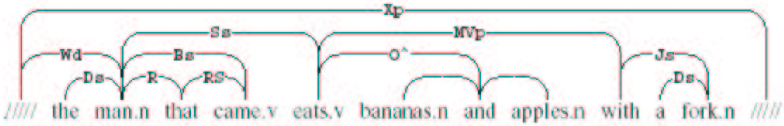
- "Bag of predicates" nature of ExtrAns' flat logical forms
  - Introspection stage: Introduce the bag of predicates
  - Extrospection stage: Add dependency information
- Bottom-up approach:
  - Suitable to ExtrAns' format of logical forms
  - Robust by nature

These conclusions are independent from the dependency-based parsing system

ExtrAns' Answer Extraction

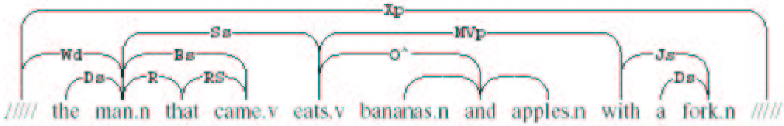
- The text retrieved is not always a logical answer to the question
- The question …
  - *which command copies files?*
- … retrieves the following "answers":
  - *cp will quickly copy the files*
  - *if the user types y, then cp copies the files*
  - *cp refuses to copy a file onto itself*
  - *rm does not copy files*

Semantic Interpretation: Top-Down


Xp
Ss
MVp
Wd
Bs
O^
Js
Ds
R
RS
Ds
///// the man.n that came.v eats.v bananas.n and apples.n with a fork.n /////

1. Find the head of the main sentence
   - follow the links **Wd** and **Ss** to find *eats*
2. Find the head of the subject
   - follow the link **Ss** to find *man*
3. Build the logical form of the subject
   - follow the link **R** to find the relative clause
   - find the logical form of the clause (recursive call)
     - but this time the subject is found by following **Bs**

Semantic Interpretation: Top-Down


Xp
Ss
MVp
Wd
Bs
O^
Js
Ds
R
RS
Ds
///// the man.n that came.v eats.v bananas.n and apples.n with a fork.n /////

4. Build the logical forms of the other verb arguments
   - follow the link **O^** to find the head of the direct object
   - build the logical form of the direct object
5. Create an entity for the main eventuality
   - the entity created is named, say, **e2**
6. Build the logical forms of other complements and adjuncts
   - follow the link **MVp** to find the prepositional phrase
7. Add the logical form of the main event and the **holds** predicate

Answer Extraction over Limited Domains

- Current IR and QA techniques are based on large volumes of data
  - Bag of words approaches
  - Question classification and named-entity extraction
  - Use of patterns
- Small and technical domains have different requirements
  - Little data redundancy: high recall is important!
  - A more comprehensive linguistic analysis is possible and required
    - Full parse
    - Use of logical forms

ExtrAns' Logical Forms

- Use a conjunction of predicates
  - No nested expressions
- Only express what is necessary: use underspecification

- Use reification as a means to represent nested expressions
  - objects
  - events, states ("eventualities")
  - properties
- By default, all variables are existentially quantified
  - Some of the entities may be asserted to exist ("hold") in the world of Unix manual pages