Imperfect Querying through Womb Grammars plus Ontologies

Veronica Dahl[(1,3)], Sergio Tessaris [2] and Thom Fruehwirth [3]

(1) Simon Fraser University

(2) Free University of Bozen-Bolzano

(3) University of Ulm

Motivation

- Need for more flexible parsers:
  - Social media promotes *jargons with mixed lexicons and syntax*
  - In some domains, non-standard syntactic arrangements are acceptable
  - Jargons and English words are infiltrating technical and even non-technical communications in many other languages
  - Yet we are quite far from the ability to parse creative text and queries

Motivation

- Need for more flexible parsers:
  - *Traditional parsers focus on constructing syntactic trees* for complete and correct sentences in a given language.
  - More flexible models can be economically achieved by focusing instead on *grammar constraints, or properties*.
  - However, no need to throw the baby with the bath water: *we can still obtain* (perhaps only partial) *parse trees*
  - Using ontologies and a list of which properties failed, *can aid in perfecting the input to the point that the analysis can proceed*.

An example

CORRECT: How many dessert dishes are there in the menu?

SERGIO ☺: How many dish desserts are there in the menu?

(think “dish washer”, where dish acts as an adjective)

Observation:

- By consulting ontologies we can determine that “dessert” is a more likely qualifier of “dish”, thus signaling that *the English precedence constraint: adj<noun has failed*.
- The failed constraint serves to correct the number agreement feature in both words, identifying “dessert” as the adjective for the noun “dishes”.

Constraint vs. tree-based parsing

- In the tree-based paradigm, adaptation of a rule such as

  np → det, adj, n

  into languages where nouns must precede adjectives *would require changing every rule where the two are involved*.

- In the constraint-based paradigm, all we need to do is *change the precedence constraint from adj < n into n < adj.*

Property Grammars (Blache 2001)

Example: We can describe the ***noun phrase*** rules:

***np -> d*, n**

np -> n

by stating ***which properties must hold in an np***:

- *Constituency:* **d** and **n** are allowable
- *Obligation*: **n** is obligatory
- *Uniqueness*: at most one **d,** at most one **n**
- *Precedence:* **d** must precede **n**

Other possible constraints:

- *Exclusion* (a determiner excludes a numeral),
- *Exigency* (an adjective requires a noun),
- *Dependency* (the features of an adjective and a noun are the same)

Property Grammar Parsing Results- Example

$_0$ le $_1$ le $_2$ livre $_3$

Outmost parsed category = cat(0,3,np,[sing,masc])

(intermediate results are available too)

Satisfied Properties = [prec(1,det,2,n,3),dep(1,det,2,n,3),unicity(det,1,3),exige(n,det,1,3),prec(0,det,1,n,3),dep(0,det,1,n,3),exige(n,det,0,3)]

Unsatisfied Properties = [unicity(det,0,3)])

Failure Driven (WG) Parsing

Results for the same Example

$_0$ le $_1$ le $_2$ livre $_3$

Outmost parsed category= cat(0,3,np,[sing,masc])

Parse Tree= np(det(le),det(le),n(livre))

Unsatisfied Props= [unicity(det,0,3)])

N.B. Satisfied Props are NOT CALCULATED (left implicit, as a complement of the Unsatisfied properties)

Computational Backbone-CHR (Thom Fruehwirth)

- concurrent committed-choice constraint programming language, developed for the implementation of constraint solvers.

- Guarded rules that transform multi-sets of constraints (atomic formulae) until no more change happens.

- Types of rules

  Simplification: H <=> C | B. (replace H by B in the store)

  Propagation: H ==> C | B. (add B, leave H in the store)

  (there is also Simpagation- we do not use it here)

Grammatical Counterpart-CHRG (H. Christiansen,05)

- Grammar symbol contiguity is indicated by hidden word boundaries
- Types of rules

Propagation: H ::> C | B. (add B, leave H in the store)

Eg. np(S), verb(V), obj(B) ::> sent(s(S,V,B).

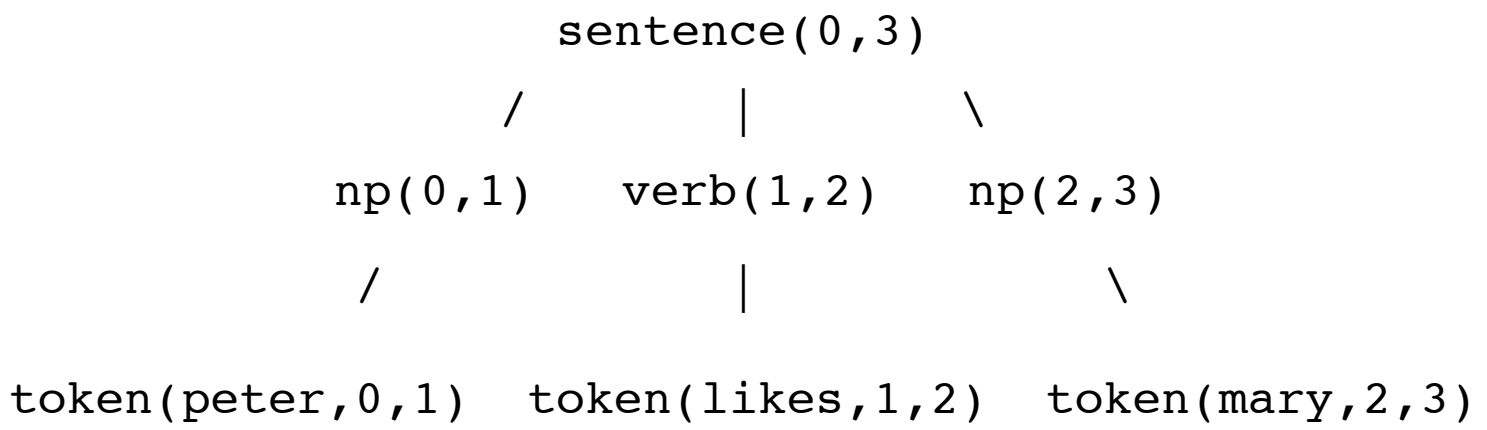
compiles into:

np(N1,N2,S), verb(N2,N3,V), obj(N3,N4,B) ==>
sent(N1,N4,s(S,V,B)

Simplification: H <::> C | B. (rewrite H by B in the store)

Parsing with CHR

```
np(N0,N1), verb(N1,N2), np(N2,N3) ==>
  sentence(N0,N3).
token(peter,N0,N1) ==> np(N0,N1).
token(mary, N0,N1) ==> np(N0,N1).
token(likes,N0,N1) ==> verb(N0,N1).
```



Why CHR/G?

- high level while executable descriptions.
- facilitates interaction between different language and data levels
- ***popular linguistic formalisms are based on constraints***
- robustness: partial results possible even for imperfect input
- inherent treatments of ambiguity and memoing
- non-classical inference (abduction (w/Henning Christiansen), assumptions (w/ Paul Tarau)

Failure-Driven Parsing

Womb Grammars

- A novel constraint-based framework developed with Emilio Miralles (CHR'12), implemented in CHRG (Christiansen 2005), useful for *inducing a target language's constraints from those of a source language,* from a sufficiently representative input corpus. Here we use it for *parsing*.
- They describe phrases in terms of properties or constraints on pairs of daughters of a given category (along the lines of IDLP (Gazdar and Pullum 81), 5P (Bes 99), Property Grammars (Blache 05))
- They focus on failed constraints, and can be tailored to a specific user's linguistic ability, by admitting a small number of failed constraints to occur, and still producing results.

Failure-Driven Parsing

- Example 1: check for violation of obligation (C must occur in Cat)

```
iCat(Cat, Attr, Tree) ,                          % Found Cat
{tpl(obligation(Cat, C))}                        % Cat should have C
::> Tree=..[Cat|T],                              % Get children
   not(member(iCat(_,_,C,_,_), T))               % There isn't a child C
| failed (obligation(Cat, C)).                   % Obligation is
                                                    violated!
```

Failure-Driven Parsing

- Example 2: check for violation of uniqueness (no more than one occurrence of C in Cat)

```
iCat(C, Attr1, Tree1) :(N1, N2),
iCat(C, Attr2, Tree2) :(N3, N4),
{iCat(N5,N6,Cat ,__, Tree)},
{tpl(uniqueness(Cat, C))},
        ::> N5=<N1, N4=<N6,          % The C's are within Cat
Tree=..[Cat|T],                      % Get children
member(iCat(N1,N2,C,Attr1,Tree1), T))
member(iCat(N3,N4,C,Attr2,Tree2), T))
| failed(uniqueness(Cat, C)).
```

Incorporating Semantics

- Partial output is possible even if no final representation for the whole sentence is available → compositional semantics

Example: using λ-calculus for creating Montague-like representations (let e' denote: the semantic representation of e)

(No bird sings)'= no(X,bird(X),sings(X)

(no)'= λP1. λ(P2,no(X,@(P1,X),@(P2,X)))
(bird)'= λX.bird(X)
(sings)'= λX.sings(X)

Incorporating Semantics

(no)'= λP1. λ(P2,no(X,@(P1,X),@(P2,X)))
(bird)'= λX.bird(X)
(sings)'= λX.sings(X)

Applying (no)' over (bird)':

(no bird)'= λP2. no(X,bird(X),@(P2,X))

Applying this λ-expression on the verb's yields:

(no bird sings)'= no(X,bird(X),sings(X)

Overall System Architecture

Three main components:

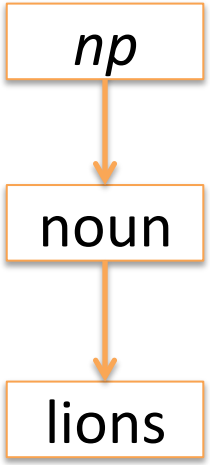
WG

Ontological

Semantic

The WG component operates bottom-up from the words' syntactic representation, building a parse tree plus a list of failed properties for each phrase.

E.g. the np "Lions sleep" generates the parse tree



In functional notation: np(noun(lions))

Example (cont.)

As well as the list of failed properties:

Failed= {exigency(noun,determiner,0,1)

This triggers a semantic completion rule which calls the ontological component, to verify that “lions” is a generic term, and makes the implicit meaning “every” explicit in the parse tree:

- Next, the semantic component is called, and combines:

(every)’= λP1. λ(P2,every(X,@(P1,X),@(P2,X)))
(lion)’= λX.lion(X)

Example (cont.)

- By applying (every)' over (lion)', we obtain:

(every lion)'= λP2. every(X,lion(X),@(P2,X))

For the verb phrase, the output is

Tree= verb_phrase(verb(sleep))

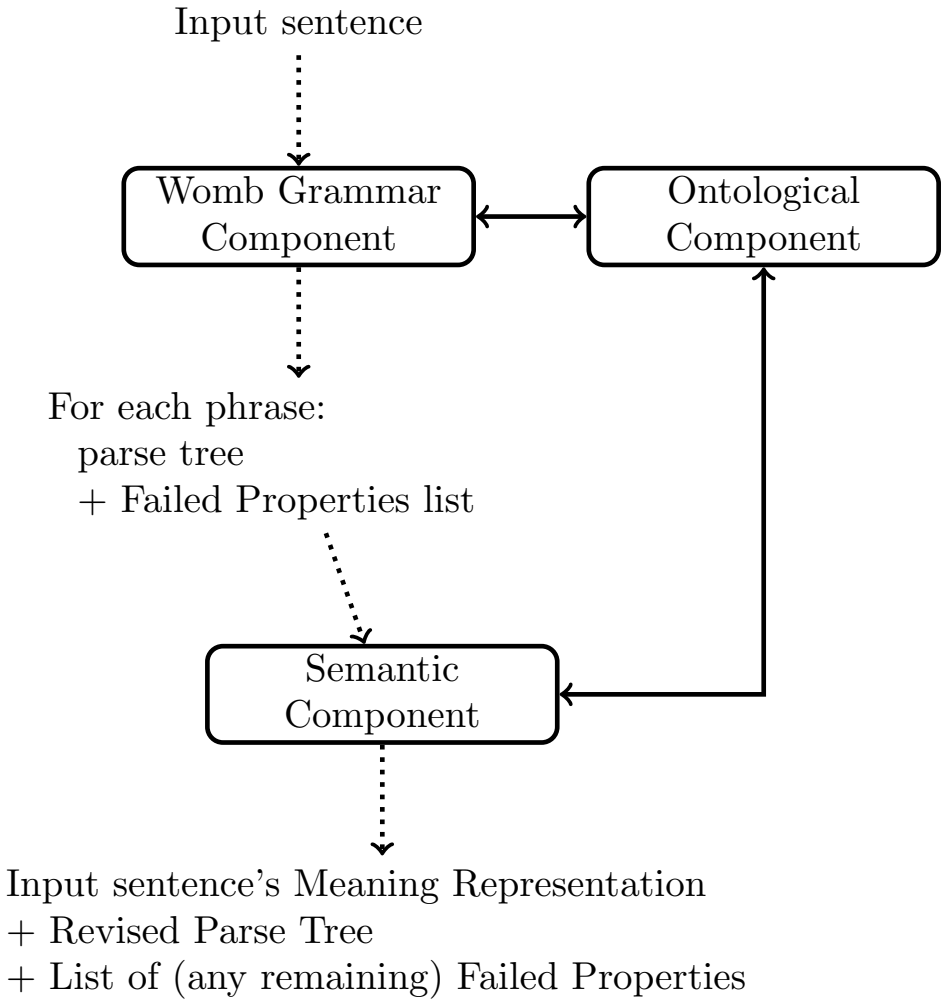
Failed= {}

- We next apply that np' over vp= λX.sleeps(X), obtaining the desired representation:

no(X,bird(X),sings(X)

Further interactions with ontologies

- Also the WG parser can call the ontological component, by expressing the call in the guard on any of its rules.
- Likewise, the semantic component can, other than controlling the order of applications of semantic representations, also call explicitly for ontological information that might allow it to make better decisions at any point.



Implementation Considerations, Notation

- Our category constraint now needs to include one 6th argument for the meaning representation:

  iCat(Start,End,Cat,Attrs,Tree1,MeaningRepr)

- Representatiions must be relational rather than functional.
- We define beta-reduction in Prolog as:

  at(X\P,X,P).

where X\P stands for λX.P

Implementation Considerations

- When a phrase cannot be further expanded, the parser looks at its parse tree, and takes any failed properties into account to consult ontological information that may help zoom in on a more accurate meaning for the phrase.

Dynamic Interactions with Failed Properties

- Types of Failed Properties

1. Strongly Failed: those that caused an --until then-- potential analysis to fail. E. g. An attempt to make “adam eats” into a verb phrase blocks because the constraint that a verb must precede its np modifiers does not hold.
2. Interestingly failed: those that hold of some constituent that nevertheless becomes part of the result. E.g. the failed but relaxable requirement for a determiner en the subject noun phrase “Medecines are toxic”.

Only the interestingly failed constraints become a part of the sentence’s characterization

Meaning Extraction through Constraint Relaxation and Failed Constraints

- A noun's syntactic requirement for a determiner must be relaxed on the syntactico-semantic condition that the noun be in plural form and represent a generic concept. In this case, a constraint is generated:

failed (N1,N2,obligation(np, det)))

- The semantic component then reconstructs the missing determiner and its meaning (as either "all" or "most"), after consulting appropriate ontologies, through the following CHR rule:

Meaning Extraction through Constraint Relaxation and Failed Constraints

failed (N1,N2,obligation(np, det)))

iCat(N1,N2,np, [plural,_], np(n(PluralNoun)),_Sem) ,
==> generic(PluralNoun) |

Tree=..[Cat|T], % *Get children*

iCat(N1,N1,det, [plural,neutral]det(every), λP1.
λ(P2,every(X,@(P1,X),@(P2,X)))

)

(semantic argument shown in functional notation for better readability)

Which determiner is made explicit can be a function of the noun's ontological info ("lions are mammals" points to "all", while "Medecines are toxic" points to "most")

Lexical Meaning Extraction through Ontologies and Failed Constraints

- Unknown words can be parsed by WGs by anonymyzing their category and features, which will become instantiated through constraint satisfacion.

E.g. “a slo virus”

Using constituency plus the constraints:

- adjectives must precede nouns,
- a noun phrase can have only one head noun,
- a noun phrase can have one determiner at most,

the funny word is identified as an adjective.

Looking up “virus” on WordNet returns “slow virus” as an hyponym, proposing it as a possible correction

Exploiting Failed Constraints to Complete Ontologies

- Ontological info can not only be consulted, but also potentially augmented by a WG analysis of trustworthy text.

E.g. In the absence of lexical info about the word "Absettarov", parsing a query such as "What illnesses does the Absettarov virus cause?" should

- help classify Absettarov as a virus (because of its being capitalized, and of the grammar accepting proper names as adjectives)
- Include Absettarov as a virus (marking it as postulated) in the domain dependent taxonomy being used.

Other uses of semantico-syntactic constraints

- Refining analysis of structures even if they do not exhibit any failed constraints. E.g we can identify “Eve” as a prepositional phrase in “Adam gave Eve an apple” (given that “Eve” is animate, while the other candidate (an apple) is inanimate). Thus it will yield the same analysis as the (also correct) sentence “Adam gave an apple to Eve”.

Possible Applications and Extensions: Partial but useful Analyses

- The semantic representation we obtain for a sentence will be full-blown unless it is not possible to arrive at a complete analysis (e.g. due to noise), in which case we get partial subtrees which never connect into a sentence node.
- Therefore an interesting specialization of our work might be to exploit the fact that we can consult ontologies from either the syntactic or the semantic component, to aim at *obtaining a semantically annotated parse tree* which, while not being a full-blown analysis, might be enough for some applications.

Possible Applications and Extensions: Web Mining

- Constructing knowledge from text is crucial to *web mining*. WG parsing might allow us to complement the conventional approaches by adding enough semantic info to better guide the web search.

  E.g. subqueries that can be gleaned from a query can be analysed and either evaluated (e.g, “last year” could evaluate to “2014”), or independently submitted to a standard Web search engine, for its results to be combined in order to produce more accurate answers. This may allow us for instance to correctly answer queries containing conjunctions and disjunctions - a difficult problem in NL based systems.

Discussion

- Ontologies could provide support for robust parsing
  - Lexical ontologies (e.g. WordNet) can be exploited for improving word recognition and disambiguation
    - E.g. suggesting the correction of “slo virus” into “slow virus”
  - Domain specific ontologies can be used to disambiguate by further deduction
    - E.g. “a virus plant” doesn’t make sense because “plants” and “viruses” are disjoint (domain knowledge), but “plant virus” is a subclass of “virus”; a more likely interpretation

Related Work

- CDG (Foth, Daum, Menzel 2005): as we do, they
  - replace well-formedness rules by declarative constraints that integrate different sources of linguistic knowledge
  - Include a related mechanism to that of relaxable constraints (defeasible constraints) in order to accept incomplete or incorrect input
- Defeasible constraints are more informative than constraint relaxation because they are weighted, handling constraints of specified scores.
- The observed constrain violations serve to diagnose the input but serve no active role in coming up with appropriate semantics. In our approach, we exploit the combination of these different sources to actively determine both syntactic and semantic aspects of our analysis.

Related Work (cont.)

- In CDG, structure buildup is incrementally achieved by pruning out and adding substructures as a consequence of failed properties.
- In contrast, we start with minimalistic trees by selecting appropriate subsets of words (just a phrase node and its immediate daughters) as basis for constraint application, and express violated properties explicitly rather than deleting their manifestations in the structure.

Related Work (cont.)

- Both the CDG approach and our own differ considerably fro constraint-based unification grammars such as HPSG, because neither of them include explicit generative rules such as

$$s \rightarrow np, vp$$

- As discussed, phenomena like free word order is therefore easier to implement, because linear precedence is clearly separated from other constraints.