Imperfect Querying through Womb Grammars plus Ontologies

Veronica Dahl
Universitat Ulm and Simon Fraser University
Germany and Canada
veronica@cs.sfu.ca

Sergio Tessaris
Free University of Bozen–Bolzano
Italy
stessaris@inf.unibz.it

Thom Fruehwirth
Universitat Ulm
Germany
thom.fruehwirth@uni-ulm.de

Abstract

Womb grammars, or WGs, are a failure-driven constraint-based parsing mechanism specifically developed for cross-language grammar engineering, whose main parsing operation consists of looking for failed constraints between pairs of daughters of a phrasal category. For instance, rather than rejecting those noun phrases where an adjective daughter precedes the noun daughter (a natural mistake for say, an Italian querying in English), a WG checks whether that English ordering requirement fails, and produces a failure indicator if so. Thus, rather than acting solely as filters impeding incorrect sentences from being parsed, the constraints described for a WG can be relaxed to admit mistakes that are personalized to a certain type of user.

Syntactic constraints have been the most studied for WGs, since their first aim was to "repair" a known language's grammar until it reflected that of another language, by modifying constraints that failed with respect to input in the other language. However any other kind of information can also be consulted.

In this article *we extend WG parsing to incorporate semantic information in view of imperfect querying, and we show how the approach lends itself in particular to ontology-driven enhancements*. We assume familiarity with Prolog and in particular, CHR.

Introduction

Constraint Satisfaction has yielded powerful results in many AI areas, including human language processing. Systems that handle multi sets of constraints (as CHR (Frühwirth 1998) and CHRG (Christiansen 2005)) have proved especially suitable for efficiently automating bottom-up sentence analysis through interpreting grammar specifications as directly executable.

The constraint-based approach to parsing typically expresses a language's grammar as a set of linguistic constraints whose *satisfaction* within a given sentence sanctions it as correct (or not) in the language described by the grammar, and associates it with some desired representation (e.g. syntactic, semantic, pragmatic).

Among the linguistic theories that lend themselves the most to constraint-based implementation are those that split the information previously packed into one rewriting rule into several constraints or properties. These *constraint based* or *property-based* theories, such as Property Grammars (PG) (Blache 2005) evolved from IDLP, which unfolds a rewrite rule into the two constraints of immediate dominance (expressing which categories are allowable daughters of a phrasal category) and linear precedence (expressing which of the daughters must precede which others). They deal mostly with syntax, whereas for query answering we obviously need to address meaning representation too.

For example in the PG framework, English noun phrases can be described through a few constraints such as precedence (a determiner must precede a noun, an adjective must precede a noun), uniqueness (there must be at most one determiner), exclusion (an adjective phrase must not coexist with a superlative), obligation (a noun phrase must contain the head noun), and so on. Instead of resulting in either a parse tree or in failure as traditional parsing schemes do, such frameworks characterize a sentence through the list of the constraints a phrase satisfies and the list of constraints it violates, so that even incorrect or incomplete phrases will be parsed. Moreover, it is possible to relax some of the constraints by declaring relaxation conditions in modular fashion. A recent adaptation of this framework into grammar transformation – Womb Grammars (Dahl and Miralles 2012) – automates as well the induction of a language's syntax from that of another.

In such theories, the modularity obtained by splitting grammatical information apart into constraints leads naturally to more *robust parsers*, since it allows us to clearly identify from the parser's output which constraints are satisfied and which fail, which allows us to accept even incomplete or incorrect sentences, instead of simply failing to parse them. We can also produce some indication of the sentence's degree of acceptability by analyzing the failed properties.

While the property-based family of grammars deals mostly with syntax, one of its properties -dependency- has been designed to carry some semantic information. However, this information is restricted to stating how a category's features (such as gender and number) affect the features of another category, so it is patently insufficient to build meaning representations.

In this article we extend WGs to incorporate semantic in-

formation in view of imperfect querying, and we show how this approach lends itself in particular to ontology-driven enhancements, by dynamically exploiting the failure-driven parsing methodology of WGs together with ontological information that allows us to perfect the input.

Motivation

In the PG formalism per se (Blache 2005), from which WG evolved, no parse tree is considered necessary (although at least one implementation offers them in view of user-friendliness (Dahl and Blache 2004)). Instead, sentences are characterized as a list of satisfied constraints (or properties) and a list of unsatisfied ones.

WG, in contrast, associate a parse tree to every successfully parsed sentence (notice however that parsing success no longer implies correctness).

The parse tree is incrementally built bottom-up, from the partial parse trees of its sub phrases. These remain available even when it's impossible to parse all the way up to a sentence node. In that case, the partial parse trees are output, together with a list of failed syntactic properties.

The satisfied properties are left implicit as a complement of the unsatisfied ones, and the syntactic tree (or trees) obtained can then serve for modularly building semantics that might in turn, concomitantly with the failed properties and ontological information, aid in perfecting the input to the point where the analysis can further proceed. This observation motivates the present work.

For example, an Italian person querying in English might mean to ask "How many dessert dishes are there in the menu?", but actually enter "How many dish desserts are there in the menu?". Since "dish" can legitimately act as an adjective, as in "a dish washer", in the absence of semantic constraints the sentence entered could be taken as a request to count the number of desserts of type "dish". Therefore, just relaxing the linear precedence English constraint between noun and adjective in order to cater to Italian users would not help detect the incorrect ordering- it would only result in a wrong parse. However by consulting ontologies, we can detect that "dessert" is a more likely qualifier of "dish" than the other way around, use the failed constraint to correct the number agreement feature in both words, and identify "dessert" as the adjective for the noun "dishes".

Note that for a grammar whose constraints are fully described, the satisfied syntactic properties of a given sentence's parse will be a complement of those failed, and will thus be deducible from them whenever needed. Therefore, by checking only for failure we lose no generality, while obtaining considerable gains in efficiency.

Background

Property Grammars

The PG formalism presently comprises the following seven categories (we adopt the handy notation of (Duchier, Dao, and Parmentier 2013) for readability, and the same example):

Constituency $A : S$, children must have categories in the set S

Obligation $A : \triangle B$, at least one B child

Uniqueness $A : B!$, at most one B child

Precedence $A : B \prec C$, B children precede C children

Requirement $A : B \Rightarrow C$, if B is a child, then also C is a child

Exclusion $A : B \nRightarrow C$, B and C children are mutually exclusive

Dependency $A : B \sim C$, the features of C1 and C2 are the same

Example 1 *For example, if we denote determiners by D, nouns by N, personal nouns by PN, verbs by V, noun phrases by NP, verb phrases by VP and sentences by Se, the context free rules* $NP \rightarrow D\ N$ *and* $NP \rightarrow N$*, which determine what a noun phrase is, can be translated into the following equivalent constraints:* $NP : \{D, N\}, NP : D!, NP : \triangle N, NP : N!, NP : D \prec N, D : \{\}, N : \{\}$.

Incorrect sentences can be "accepted" through declaring some constraints as relaxable. For instance, while from the context-free grammar rules shown we wouldn't be able to parse "the the book" (a common mistake from cutting and pasting in word processors), in the constraint-based formulation we can if we relax the uniqueness of determiner constraint.

Relaxation can be made conditional (e. g. a head noun's requirement for a determiner can be made relaxable in case the head noun is generic and in plural form, as in "Lions sleep tonight"). The failure of relaxable constraints is signalled in the output, but does not block the entire sentence's analysis. Implementations not including constraint relaxation capabilities implicitly consider all properties as relaxable.

Incomplete sentences can be parsed to whatever degree is possible, e.g. input such as "The lion shrewdly" might yield a correct analysis of the noun phrase and the adverb but identify no verb phrase- and hence no sentence-, or alternatively, the input might be parsed into an incomplete sentence, if a sentence's requirement for a verb phrase is relaxed.

Womb Grammars

Womb grammars include the set of properties shown above. They can not only parse sentences from a given grammar, as PG can, but can also induce a target language's grammar from another language's known grammar. The latter functionality shall not concern us here. Interested readers can refer to (Dahl and Miralles 2012; Dahl, Miralles, and Becerra 2012; Becerra, Dahl, and Miralles 2013; Becerra, Dahl, and Jiménez-López 2014).

Previous parsing mechanisms for property-based grammars – as well as for many other constraint-based research areas – focus on constraint satisfaction. In the remainder of this paper, we shall show how our parsing mechanism for WGs, through focusing on constraint failure instead, allows us a useful and elegant extension into semantic properties as well as a clean while dynamic interaction between syntax and semantics, with particularly fruitful ramifications in interaction with ontologies.

CHRG

Our implementation is done in terms of CHR grammar, or CHRG (Christiansen 2005). CHRGs are a grammatical interface to CHR, providing it what DCGs provide to Prolog-namely, they invisibly handle input and output strings for the user. In addition, they include constructs to access those strings dynamically, and the possibility of reasoning in non-classical ways, with abduction or with resource-based assumptions.

For the purposes of this paper, we only use two types of CHRG rules, which parallel the CHR rules of propagation and simplification, and are respectively defined as follows:

A *propagation grammar rule* is of the form

α `-\` β `/-` γ `::>` G `|` δ`.`

The part of the rule preceding the arrow `::>` is called the head, G the guard, and δ the body; α, β, γ are sequences of grammar symbols and constraints so that β contains at least one grammar symbol, and δ contains exactly one grammar symbol which is a nonterminal (and perhaps constraints); α (γ) is called *left (right) context* and β the *core* of the head; G is a conjunction of built-in constraints as in CHR and no variable in G can occur in δ. If left or right context is empty, the corresponding marker is left out and if G is empty (interpreted as **`true`**), the vertical bar is left out. The convention from DCG is adopted that Prolog calls (i.e., non-grammatical stuff) in head and body of a rule are enclosed by curly brackets. Gaps and parallel match are not allowed in rule bodies.

A *simplification grammar rule* is similar to a propagation grammar rule except that the arrow is replaced by `<:>`.

Whereas *propagation* rules add δ' to the constraint store (where δ' denotes δ affected by any substitutions needed for the rule's application), *simplification* rules replace β' by δ', so that β' is removed from the constraint store.

Failure-Driven Parsing

Typically, constraint based programming strives to solve constraints, i.e, to satisfy them. In our problem domain however, the aim is to reach an internal representation of a query even if imperfect.

For perfect queries, if we can assume that the satisfied constraints will be the complement of those that fail (a reasonable assumption, which we make), we can get away with checking that no constraint fails.

For imperfect queries, clearly we cannot allow all constraints to fail at once without significant trouble -or even impossibility- in arriving at any useful meaning representation of the query. However we can capitalize on knowing who will query the system, and admit one type of failed constraint accordingly. For the example in our abstract, relaxing precedence between noun and adjective will cater to romance language speakers.

In rigour, not all constraints are checked only for failure. The constituency constraint is checked for satisfaction. The reason for this is that our parsing is driven by projection (e.g. making a noun phrase out of a noun, a verb phrase out of a verb, etc.) plus category expansion, which will expand a phrase to include any adjacent constituents that are legal for the type of phrase and that satisfy all non-relaxable constraints between them. Since the expansion rule is guided by constituency, senseless expansions do not occur, e.g. NP can be expanded to include an adjacent D (to the left or right!) but not a V. A consequence of this approach is that the constituency constraint is not relaxable.

In order to check for failed constraints efficiently, lexical categories are parsed into a representation that includes their lexical type, word boundaries within the sentence, their list of syntactic attributes (gender, number) and the portion of parse tree that they will contribute to the entire parse. Concretely, our parser expands instantiated categories, which are CHRG grammar symbols of the form

```
iCat(Category, Attributes, Tree)
```

These are compiled into CHR constraints with the word boundaries having been made explicit:

```
iCat(Start, End, Category, Attributes, Tree)
```

Within a CHRG rule, we can spy whenever needed on the dynamic values of the Start and End points, simply by adding them explicitly after the grammar symbol, in the notation :(Start,End). For instance:

```
iCat(Category, Attributes, Tree):(Start,End)
```

Example 2 (Instantiated categories) *Take the noun phrase "an apple", implicitly located (as all phrases input to the analyzer) as from point 0. Parsing it results in the following instantiated categories (shown in the implicit CHR notation):*

```
iCat(0, 1, det, [sing,neutral], det(an))
iCat(1, 2, n,   [sing,neutral], n(apple))
iCat(0, 2, np,  [sing,neutral],
 np(iCat(0, 1, det, [sing,neutral], det(an)),
  iCat(1, 2, n, [sing,neutral], n(apple))))
```

Notice that the NP inherits the attributes of the underlying N. This is useful for checking dependency constraints, which require attributes to match. Later we shall see how to incorporate semantics as well.

Checking for Constraint Violation

Because properties are defined on pairs of daughters of a given phrasal category, they can be checked in very modular fashion. Their relationships with other parts of the sentence, including with further ancestors than the phrase itself, are verified in the same way, phrase by phrase. So each phrase in a sentence to be analyzed is sanctioned bottom-up from its direct daughters, by checking their properties. This allows us to construct semantic representations in equally modular fashion, as we will see in the next section. Properties other than constituency, as we saw, are checked for failure, and their failure is signalled by adding a constraint which carries the information that they have failed.

We next show, through the examples of uniqueness and obligation, that different constraint violation checking may need to take place at different stages of analysis.

Example 3 (Uniqueness) *Violations of uniqueness constraints are checked by a CHRG rule that finds two adjacent words of same category C within the bounds of a phrase of category Cat being parsed, where C has been constrained to appear only once within that phrase. Should such adjacent words be found, the information that uniqueness of a category C under Cat has been violated in the range these categories cover is added as a (fact represented as a CHR) constraint, and the parse tree for the phrase can either delete one occurrence if the repeated category is the same one (as in "the the book"), or include both if different, so that further considerations can be made before choosing one over the other. In slightly simplified form, our CHRG rule that checks for the violation of uniqueness* `Cat : C`*!. looks as follows:*

```
  iCat(C, Attr1, Tree1):(N1, N2), ...,
  iCat(C, Attr2, Tree2):(N3, N4),
  {iCat(N5, N6, Cat, _, Tree)},
  {tpl(uniqueness(Cat, C))}
% tpl/1 describes uniqueness properties
  ::>
% The C's are within the bounds of Cat:
    N5 =< N1, N4 =< N6,
% And they are its direct daughters:
   Tree=..[Cat|T],
   member(iCat(N1,N2,C,Attr1,Tree1),T),
   member(iCat(N3,N4,C,Attr2,Tree2),T)
  | failed(uniqueness(Cat,C)).
```

The first line in the above code finds a category `C` *between word boundaries N1 and N2, with attributes Attr1 and parse tree Tree1. The three dots indicate a skipped substring after N2, before another instance of the same category C is found between the word boundaries N3 and N4. The Prolog calls (between curly brackets) and the guard find a category Cat that dominates both instances of C, and a uniqueness property that is required between a phrase Cat and its immediate daughter C (i.e., a requirement that C appear no more than once as immediate daughter of a phrase of category Cat). Once all this is checked, a grammar symbol (failed/1) is thrown into the constraint store, that states that uniqueness of C within Cat is falsified between word boundaries N1 and N4 (since grammar rules are compiled into CHR rules, what will appear in the constraint store is the equivalent CHR constraint (failed/3), namely failed(N1,N4,uniqueness(Cat,C)).*

This rule can fire even if a phrasal category hasn't been fully expanded, because adding more components to an `iCat` *is not going to modify uniqueness having been violated.*

Example 4 (Obligation) *A constraint such as obligation, in contrast, is designed to only fire after the phrasal category has been fully expanded, since adding one more component into the phrase can result in its becoming satisfied (in the case in which the component added is the one which is obligatory). We now show the CHRG rule that checks for failed obligation:*

```
   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
```

The obligation rule checks a given category for its direct daughters. Since `iCat`*s are retracted when they fail, this check only fires after the* `iCat` *has been fully expanded.*

Incorporating Semantics

Since our pieces of parse tree may end up disconnected (e.g. if noise words that cannot be parsed intervene, impeding us from reaching a "sentence" node), it would be useful to have partial semantic representations which can be further completed if possible, and when not possible, can at least contribute partial meanings.

For this reason we have chosen a compositional semantics, in which each constituent is given a representation built from applying an expression to another. We use the well-known lambda calculus for representing and combining meaning.

For instance, we may wish to associate the Montague based meaning representation no(X,bird(X),sings(X)) to the sentence "No bird sings", given the following word representations (shown in functional notation, and using String' to denote the semantic representation of String):

$\texttt{no'} = \lambda P1.\lambda(P2, \texttt{no}(X, @(P1, X), @(P2, X)))$

$\texttt{bird'} = \lambda X.\texttt{bird}(X)$

$\texttt{sings'} = \lambda X.\texttt{sings}(X)$

The sentence's meaning can be build compositionally, by applying the meaning of the determiner "no" over the meaning of the noun "bird", which results in the following lambda-expression for "no bird":

```
(no bird)' = λ P2.no(X,bird(X),@(P2,X))
```

This lambda-expression is then applied to that of the verb phrase, which yields:

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

Overall System Architecture Our system comprises three main components:

- WG
- ontological
- semantic

These components cooperate with each other as follows:

Faced to an input sentence, the WG component operates bottom-up from the words' syntactic representations, building a parse tree plus a list of failed properties for each phrase it can recognize as such.

For instance, for the sentence "Lions sleep" (more on this example later), it will create the following noun phrase's parse tree:

```
noun_phrase(noun(lions))
```

as well as the list of failed properties

```
Failed= [exigency(noun,determiner),0,1)]
```

(stating that the property of exigency between a head noun and its required determiner fails in the noun phrase recognized between points 0 and 1 of the input sentence).

The failed exigency property will trigger a semantic completion rule which calls the ontological component in order to verify whether "lions" is a generic term, and given that it is, will complete the analysis by making the implicit meaning "every" explicit in the parse tree:

```
noun_phrase(det(every),noun(lions)))
```

Once a phrase has been completed, the semantic component is called. This component combines the meaning representations of the noun phrase's components:

every′ = $\lambda P1.\lambda(P2, \texttt{every}(X, @(P1, X), @(P2, X)))$

lion′ = $\lambda X.\texttt{lion}(X)$

by applying that of "every" over that of "lion", which results in

λ P2.every(X,lion(X),@(P2,X))

Similarly, the WG component will also arrive at a syntactic tree plus a list of failed properties representation for the verb phrase, namely:

```
verb_phrase(verb(sleeps))
 Failed={}
```

Once these two phrases (noun phrase and verb phrase) have been parsed, the semantic component is called to apply the representation of the noun phrase over that of the verb phrase. Note that since the verb phrase contains only an intransitive verb, the verb phrase's semantic representation coincides with that of the verb itself:

$\lambda X.\texttt{sleeps}(X)$

The application of

λ P2.every(X,lion(X),@(P2,X))

over

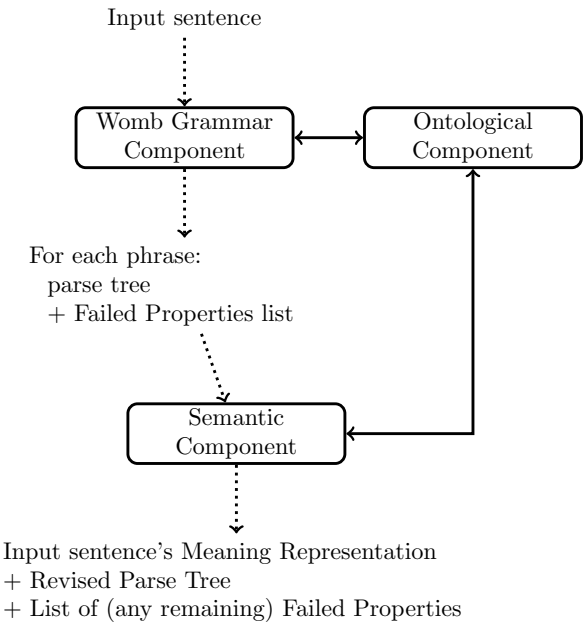
$\lambda X.\texttt{sleeps}(X)$

yields the desired representation

```
every(X,lion(X),sleeps(X))
```

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

Graphically, we can depict this architecture as follows:



The degree of interaction inherent in our architecture serves many purposes, allowing us e.g. to discard an extra determiner if two wrongly made it to the input sentence, to reorder two constituents that appear in non-allowable orderings, to reconstruct implicit meanings, to connect partial subtrees together using ontological consultation in order to arrive at a parse for the complete sentence (this is particularly useful in the case of noisy input intervening within the input sentence).

In short, each phrase that the WG parser outputs will enter a stage of semantic composition, and each phrase thus completed will be combined, if possible, with other thus completed phrases, into a higher level phrase that will in turn check semantic composition rules for combining them, with ontologies being available for consultation at any stage of the syntactic parsing or the semantic analysis process.

Implementation Considerations In terms of implementation, extending WGs to incorporating semantics as described here requires to now have 6-ary instantiated categories whose last argument is the partial semantics associated with that category:

```
iCat(Start, End, Category, Attributes,
Tree,LambdaExpression).
```

Since we are using Prolog, we must transform the above functional representations into relational ones.

Since CHRG does not support true lambda expressions, we represent $\lambda X.P$ by the first-order term $X\backslash P$.

The calls to our (relational equivalent of) "@" use the following Prolog implementation of beta-reduction:

```
at(X\P,X,P).
```

To add the semantic component on each category we can propagate our 5-ary categories into the new 6-ary ones, e.g.:

```
iCat(det, [sing,neutral], det(every))
 ::>
```

```
at(P1,X,Q1), at(P2,X,Q2) | iCat( det,
[sing,neutral], det(every),
P1\P2\every(X,Q1,Q2)).
```

This can in fact be automated by a more general rule which consults modular definitions of semantics for each word, so it becomes easy to experiment with different compositional representations.

Next we need to combine the meanings at appropriate points in our parsing process. We postulate that the appropriate point is every time a phrase is completed (i.e., cannot be further expanded). For our example "no bird sings", once "no bird" has been analyzed into a noun phrase, this noun phrase cannot be further expanded, since "sings" is not allowable as a noun phrase's direct daughter. At this point the parser looks at the parse tree, at any failed properties associated with it, and consults any ontological information needed to take into account the failed properties, in order to construct the meaning representation of the noun phrase.

A distinction between lambda-calculus variables and query representation variables needs to be made, since a query's representation will in general include Prolog rather than lambda variables – i.e., variables that are to remain in the final result, being necessary for evaluating the answer to a query. E.g. the variable X in the formula no(X,bird(X),sings(X)) is not a variable under lambda-abstraction but rather, a part of the sentence's meaning representation, which acts as a placeholder for the answers.

Of course, we could have chosen any other kind of meaning representation while using semantic compositionality together with our techniques for managing failed properties in combination with ontological information. The exemplification in terms of lambda-calculus presented above is merely a proof of concept.

Dynamic Interactions with Failed Properties

Types of Failed Properties Failed properties are placed in the constraint store during the parse of a sentence. There are two kinds of failed properties:

Strongly failed: those that caused an until then potential analysis to fail, and block the said analysis. An example would be the attempt to make "adam eats" a verb phrase; this attempt is blocked because the constraint that a verb must precede its np in a verb phrase does not hold, so the hypothesis that "adam eats" could be a verb phrase is discarded.

Interestingly Failed: those that hold of some constituent that nevertheless does become a part of the result. An example would be the failed but relaxable requirement for a determiner in the subject noun phrase of "Medicines are toxic".

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

Meaning Extraction through Constraint Relaxation and Failed Constraints Property relaxation can work together with failed constraints for various purposes. For instance, a noun's syntactic requirement for a determiner must be relaxed on the syntactico-semantic condition that the noun is in plural form and represents a generic concept. This will be accepted and produce an analysis of the noun into a noun phrase. In particular, the constraint equivalent to the grammar symbol:

```
failed(obligation(np,n,det)).
```

namely:

```
failed(N1,N2,obligation(np,n,det)).
```

will appear in the constraint store.

This (interestingly) failed constraint can be used by the semantic part of our analyzer to correctly extract the meaning of the noun phrase, through reconstructing the missing determiner's meaning (e.g. as "all" or as "most").

The following CHR rule achieves this, through consulting semantic, syntactic and ontological information. Notice that it is applicable only if the noun can be determined to be a generic one. The call to generic(PluralNoun) in the rule's guard serves this purpose, by consulting appropriate domain ontologies.

```
failed(N1,N2,obligation(np,n,det)),
iCat(N1,N2,np, [plural,_],
    np(n(PluralNoun)),_Sem),
  ==>
 generic(PluralNoun)
    | iCat(N1, N1, det,
        [plural,neutral],
        det(every),
        λP1.λP2.every(X,@(P1,X),@(P2,X))) .
```

The semantic argument of iCat above is shown in functional representation for readability (as said, we actually use relational equivalents in our code).

Notice that the non-overtness of the determiner "every" is indicated by its word boundaries being the same: it stretches between point N1 and N1 itself. This easy way of recognizing non-overtness can be exploited further if necessary during other aspects of a sentence's analysis- e.g. relativization, where the antecedent is non-overt.

Since all noun phrase daughters are now explicit for this noun phrase, we can now apply the meaning of "every" over the meaning of "lions" (which will have been propagated from the noun "lions", i.e. NounSem=lions,$\lambda X.lion(X)$) and we obtain (modulo notation):

```
λ P2.every(X,lion(X),@(P2,X))
```

Of course, we could have chosen to materialize the implicit determiner as any other one, e.g as "most" instead of "every". Ontological interactions with a specific semantic domain's ontologies can help determine which is best. For instance, while for "Lions are quadrupeds" the correct determiner is indeed "every', for "Medicines are toxic", "most" might be more appropriate.

Lexical Meaning Extraction through Ontologies and Failed Properties Unknown words can be parsed through Womb Grammar by anonymizing their category and features. These will become efficiently instantiated through constraint satisfaction, taking into account all the syntactic and semantic properties that must be satisfied by the unknown word's interaction with its context.

The clues they can provide regarding syntactic category can serve to guide a subsequent semantic analysis, or to bypass the need for a complete semantic analysis by the concomitant use of ontologies relevant to domain-specific uses of our parser.

For instance, "the resistetente virus" includes an ill-typed word which a human would immediately suspect stands for "resistant". A computer can guess at least its syntactic category through explicitly checking which syntactic constraints fail because of "resistente" not having parsed: given that adjectives must precede nouns, that a noun phrase can have only one head noun, and that determiners are also unique within a noun phrase, the funny word can only be an adjective. The meaning, and perhaps the precise form of the corrected word, can sometimes be determined in consultation with a domain-relevant ontology, in this case of medicine or biology, by marking the word as unknown and letting the semantic construction module consult such ontologies in interaction with the meanings of the known words of the phrase they belong to. In our example for instance, the similarity with the adjective "resistant" which ontological consultation would semantically link to "virus" may result in proposing it as a possible correction.

Similarly, extraneous words that repeat might allow a domain-dependent ontology to help determine their meaning. For instance, from "his humongous fever" and "the humongous white cell count" by consulting the ontology besides the constraints, we can not only determine that "humongous" is an adjective, but also that it probably refers to some quality similar to "high". It would be most interesting to carefully study under which conditions such ontological inferences would be warranted.

In general, we are not necessarily interested in capturing the exact meaning of each unrecognized word; but rather in inferring its relation with known words. The problem can be cast into the (automatic) extraction of a portion of the hypernym relation involving the extraneous word using the actual document or additional sources as corpora (see (Clark et al. 2012)). Starting from existing lexical ontologies (e.g. EuroWordNet (Vossen 2004)) different techniques can be used to expand the ontological knowledge about an unknown word (see e.g. (Gupta and Oates 2007; Snow, Jurafsky, and Ng 2006)). Approaches currently described in the literature can be enhanced by means of the clues that Womb Grammar provides upon failure. E.g., one of the most successful techniques involve the search on corpora (or the web) for specific textual patterns indicating relationships between keywords ("*a* **borogove** *is*") and the fact that the grammar expects a word of a specific category can be used to narrow down the textual patterns to be used to scan the corpus.

Exploiting failed constraints to complete ontologies It is worth noting that ontological information can not only be consulted, but in some cases also potentially augmented by a WG analysis of trustworthy text. E.g. in the absence of lexical information about the word "Absettarov", the parse of a query such as "What illness does the Absettarov virus cause?" should not only classify Absettarov as a proper name (on the grounds of its being capitalized, and of the grammar accepting proper names as adjectives) but should also include Absettarov as a virus (with some indication that this is only postulated, and perhaps some degree of certainty) in the domain-dependent taxonomy the parser is using.

Flexible word order through Constraint relaxation and failure analysis Notice that totally permissive word order is very easy to achieve within our framework. This is interesting because discontinuous constituents such as noun phrases are common in some languages and even even in Latin or Greek very common in verse (and not unusual even in certain prose genres, e.g. Plato's late work, such as the Laws). A contrived example for Latin would be "Puella bona puerum parvum amat" (Good girl loves small boy), where all 5! word permutations are possible, and we certainly do not want to write a separate rule for each of the possible orderings.

In our WG framework, when all permutations are possible, all we have to do is not to include any precedence rule for them! The case in which the contents of constituents may be scrambled up with elements from other constituents can be dealt with by specifying ordering only between those pairs of constituents which must precede each other.

However in our present work we want to describe proper orderings in the host language, say English, while being sensitive to alternative orderings that will eventually be produced by native speakers of another language, say Italian. We deal with this case by stating the English ordering and declaring it as relaxable. This will result in accepting for instance our Italian speaker's query "What illnesses deadly does the Absettarov virus cause?" Imperturbable, our analyser will produce the correct parse tree for the incorrect sentence, while leaving a marker of failed linear precedence between noun and adjective in the form of a constraint. Thus the semantic representation, which is dependent on the parse tree, can be found anyway.

Other uses of semantico-syntactic constraints

While interestingly failed constraints have an obvious use for modularly handling exceptions as just exemplified, we can apply the same methodology we described for them above, for refining analyses of structures even if they do not exhibit any failed constraints. For instance, the correct sentences: "Adam gave an apple to Eve" and "Adam gave Eve an apple" differ only in the ordering of the verb's complements, and in the fact that in the second sentence, such ordering makes the preposition implicit. We can reconstruct it through a constraint that identifies "Eve" as the preposition phrase given that it is animate, while the other candidate ("an apple") is inanimate.

Possible Applications and Extensions

We have proposed an analysis of individual sentences which takes semantic and syntactic constraints into account and obtains a parse tree or trees, a list of failed properties and a semantic representation as a result. The semantic representation is full blown unless it is not possible (e.g. due to the

presence of noise) to produce a complete analysis, and we must be content with partial subtrees which never connect into a full tree for the entire sentence.

An interesting specialization of our work might be to exploit the fact that ontologies can be consulted freely, whether from the semantic or the syntactic component, and instead of aiming at a full-blown representation, aim at complementing syntax in more minimalistic ways, e.g. by obtaining just a semantically annotated parse tree which might be enough for some applications, or even be as much as we can expect given the domain of application chosen.

One important possible application is that of semantically annotating knowledge bases, e.g. those in the semantic web, with ontological information or little more. Typically such applications do not resort to NLU techniques per se but to hand-produced or only semi-automated approaches.

The construction of knowledge from text is crucial to web mining, and WG parsing might allow us to complement the conventional approaches by adding enough semantic information to better guide the web search. For instance, subqueries that can be gleaned from a query can be analysed and either evaluated (e.g. "last year" could concretely evaluate to 2014) or independently submitted to a standard Web search engine, such as Google, and the results can then be combined to produce an answer in many cases more accurate than was possible with previous methods. This may allow us for instance to correctly answer queries containing conjunctions and disjunctions that natural language based systems surprisingly fail to understand, as pointed out in (Gottlob 2009).

Discussion

The idea of extending Womb Grammars with ontologies was first proposed in (Adebara, Dahl, and Tessaris 2015), where it was designed specifically for completing mixed language grammars, and dealt mainly with syntax. The extension we have presented here is tailored to imperfect querying and in particular incorporates a semantic component, which the previous work does not. We have shown how this approach is especially suitable for ontology-driven enhancements. To the best of our knowledge, this is the first time that the parsing power of constraint failure is exploited to the extent that we do in our work.

The resulting search space reduction is significant because deep parsing with the types of constraint grammars we address is theoretically exponential in the number of categories of the grammar and the size of the sentence to parse (van Rullen 2005).

Other than achieving a considerable search space reduction by only having to evaluate constraints for failure, the use of constraint failure in conjunction with semantics and ontologies allows us to parse imperfect queries in repairing ways, so that they can be effectively answered.

As well, this approach promotes easy interaction between different levels of analysis, opening possibilities for other levels of analysis than just syntax and semantics (e.g. pragmatics) to also interact.

The practice of building a parse tree as a side effect of parsing means that interestingly, also ill-formed sentences can generate a parse tree, which can make the source of failure visually clear.

Clearly, while any of a grammar's constraints' failure can be tolerated by our approach (save obligation of a head of phrase, because our parser expands phrases starting from the head), it would not be wise to relax all constraints at the same time: totally erroneous input would be far too intractable to parse. Our compromise solution is to admit those imperfections typical for a given user (e.g. replicating in English the word order due in Italian), which offers a useful enough degree of tolerance without undue extra stress on the parser.

Related Work

Other than the background related work already mentioned in this paper, the previous work that most resembles ours is CDG (Foth, Daum, and Menzel 2005), which also replaces well-formedness rules by declarative constraints that integrate different sources of linguistic knowledge, and includes 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: they handle constraints of specified scores.

The observed constraint violations in (Foth, Daum, and Menzel 2005) 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 in order to actively determine both syntactic and semantic aspects of the analysis.

Another difference is that 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 manifestation in the structure.

Both the CDG approach and our own differ considerably from 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 all other constraints.

With this work we hope to stimulate further research into the uses of ontologies in constraint-based parsing.

Acknowledgments. This paper was supported by NSERC Discovery grant 31611024, and developed during a visit to University of Ulm.

References

Adebara, I.; Dahl, V.; and Tessaris, S. 2015. Completing mixed language grammars through womb grammars plus ontologies. In Henning Christiansen, M. D. J. L., and

Loukanova, R., eds., *Proceedings of the International Workshop on Partiality, Underspecification and Natural Language Processing*, 32–40.

Becerra, L.; Dahl, V.; and Jiménez-López, M. D. 2014. Womb grammars as a bio-inspired model for grammar induction. In *Trends in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection*. Springer International Publishing. 79–86.

Becerra, L.; Dahl, V.; and Miralles, E. 2013. On second language tutoring through womb grammars. IWANN 2013, June 12-14, Tenerife, Spain.

Blache, P. 2005. Property grammars: A fully constraint-based theory. In *Proceedings of the First International Conference on Constraint Solving and Language Processing*, CSLP'04, 1–16. Berlin, Heidelberg: Springer-Verlag.

Christiansen, H. 2005. CHR grammars. *TPLP* 5(4-5):467–501.

Clark, M.; Kim, Y.; Kruschwitz, U.; Song, D.; Albakour, D.; Dignum, S.; Beresi, U. C.; Fasli, M.; and De Roeck, A. 2012. Automatically structuring domain knowledge from text: An overview of current research. *Information Processing & Management* 48(3):552–568.

Dahl, V., and Blache, P. 2004. Directly executable constraint based grammars. *Proc. Journees Francophones de Programmation en Logique avec Contraintes*.

Dahl, V., and Miralles, J. E. 2012. Womb grammars: Constraint solving for grammar induction. In Sneyers, J., and Frühwirth, T., eds., *Proceedings of the 9th Workshop on Constraint Handling Rules*, volume Technical Report CW 624, 32–40.

Dahl, V.; Miralles, E.; and Becerra, L. 2012. On language acquisition through womb grammars. In *7th International Workshop on Constraint Solving and Language Processing*, 99–105.

Duchier, D.; Dao, T.-B.-H.; and Parmentier, Y. 2013. Model-Theory and Implementation of Property Grammars with Features. *Journal of Logic and Computation* 19.

Foth, K.; Daum, M.; and Menzel, W. 2005. Parsing unrestricted german text with defeasible constraints. In Christiansen, H.; Skadhauge, P.; and Villadsen, J., eds., *Constraint Solving and Language Processing*, volume 3438 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 140–157.

Frühwirth, T. W. 1998. Theory and practice of constraint handling rules. *J. Log. Program.* 37(1-3):95–138.

Gottlob, G. 2009. Computer science as the continuation of logic by other means. Keynote at European Computing Summit Sciences 2009, Paris.

Gupta, A., and Oates, T. 2007. Using Ontologies and the Web to Learn Lexical Semantics. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, 1618–1623. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Snow, R.; Jurafsky, D.; and Ng, A. Y. 2006. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, 801–808. Stroudsburg, PA, USA: Association for Computational Linguistics.

van Rullen, T. 2005. Vers une analyse syntaxique a granularite variable. *Ph.D. thesis, Université de Provence (2005)*.

Vossen, P. 2004. Eurowordnet: a multilingual database of autonomous and language-specific wordnets connected via an inter-lingualindex. *International Journal of Lexicography* 17(2):161–173.