Grammar Error Detection with Best Approximated Parse

Jean-Philippe Prost

Grammar Error Detection with Best Approximated Parse

Jean-Philippe Prost
LIFO, Université d'Orléans
INRIA Lille - Nord Europe
Jean-Philippe.Prost@univ-orleans.fr

Abstract

In this paper, we propose that grammar error detection be disambiguated in generating the connected parse(s) of optimal merit for the full input utterance, in overcoming the cheapest error. The detected error(s) are described as violated grammatical constraints in a framework for Model-Theoretic Syntax (MTS). We present a parsing algorithm for MTS, which only relies on a grammar of well-formedness, in that the process does not require any extra-grammatical resources, additional rules for constraint relaxation or error handling, or any recovery process.

1 Introduction

Grammar error detection is a crucial part of NLP applications such as Grammar Checking or Computer-Assisted Language Learning (CALL). The problem is made highly ambiguous depending on which context is used for interpreting, and thus pinpointing, the error. For example, a phrase may look perfectly fine when isolated (*e.g. brief interview*), but is erroneous in a specific context (*e.g.* in **The judge grants brief interview to this plaintiff*, or in **The judges brief interview this plaintiff*). Robust partial parsing is often not enough to precisely desambiguate those cases. The solution we prescribe is to point out the error(s) as a set of violated (atomic) constraints of minimal cost, along with the structural context used for measuring that cost. Given an ungrammatical input string, the aim is then to provide an approximated rooted parse tree for it, along with a description of all the grammatical constraints it violates. For example, Figure 1 illustrates an approximated parse for an ill-formed sentence in French, and the error being detected in that context. Property Grammar (Blache, 2001) provides an elegant framework for that purpose.

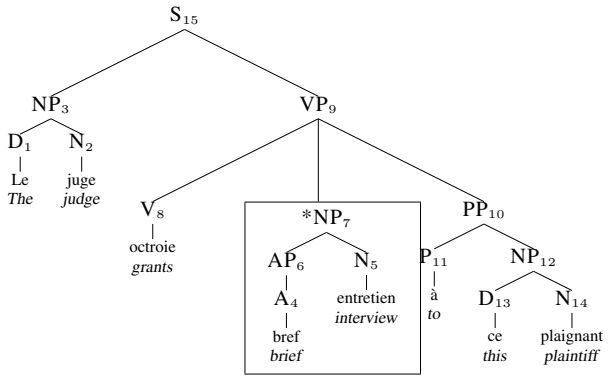

Figure 1: Approximated parse for an erroneous French sentence (the Noun '*entretien*' requires a Determiner).

Most of the relevant approaches to robust knowledge-based parsing addresses the problem as a recovery process. More specifically, we observe three families of approaches in that respect: those relying on grammar *mal-rules* in order to specify how to correctly parse what ought to be ungrammatical (Bender et al., 2004; Foster, 2007); those relying on constraint relaxation according to specified relaxation rules (Douglas and Dale, 1992); and those relying on constraint relaxation with no relaxation rules, along with a recovery process based on weighted parsing (Fouvry, 2003; Foth et al., 2005). The first two are actually quite similar, in that, through their use of extra-grammatical rules, they both extend the grammar's coverage with a set of ought-to-be-ungrammatical utterances. The main drawback of those approaches is that when faced with unexpected input at best their outcome remains unknown, at worst the parsing process fails. With robust weighted parsing, on the other hand, that problem does not occur. The recovery process consists of filtering out structures with respect to their weights or the weights of the constraints being relaxed. However, these strategies usually can not discriminate between grammatical and ungrammatical sentences. The reason for that comes

from the fact that grammaticality is disconnected from grammar consistency: since the grammar contains contradicting (universal) constraints, no conclusion can be drawn with regard to the grammaticality of a syntactic structure, which violates part of the constraint system. The same problem occurs with Optimality Theory. In a different fashion, Fouvry weighs unification constraints according to "how much information it contains". However, relaxation only seems possible for those unification constraints: error patterns such as word order, co-occurrence, uniqueness, mutual exclusion, ... can not be tackled. The same restriction is observed in VanRullen (2005), though to a much smaller extent in terms of unrelaxable constraints.

What we would like is (i) to detect *any* type of errors, and present them as conditions of well-formedness being violated in solely relying on the knowledge of a grammar of well-formedness—as opposed to an *error grammar* or *mal-rules*, and (ii) to present, along-side the violated constraints, an approximated parse for the full sentence, which may explain which errors have been found and overcome. We propose here a parsing algorithm which meets these requirements.

2 Property Grammar

The framework we are using for knowledge representation is Property Grammar (Blache, 2001) (PG), whose model-theoretical semantics was formalised by Duchier et al. (2009). Intuitively, a PG grammar decomposes what would be rewriting rules of a generative grammar into atomic syntactic properties — a *property* being represented as a boolean constraint. Take, for instance, the rewriting rule NP $\rightarrow$ D N. That rule implicitely informs on different properties (for French): (1) NP has a D child; (2) the D child is unique; (3) NP has an N child; (4) the N child is unique; (5) the D child precedes the N child; (6) the N child requires the D child. PG defines a set of axioms, each axiom corresponding to a constraint type. The properties above are then specified in the grammar as the following constraints: (1) NP : $\triangle$ D; (2) NP : D!; (3) NP : $\triangle$ N; (4) NP : N!; (5) NP : D $\prec$ N; (6) NP : N $\Rightarrow$ D. These constraints can be independently violated. A PG grammar is traditionally presented as a collection of Categories (or Constructions), each of them being specified by a set of constraints. Table 1 shows an example of a category. The class of models we are working

NP (*Noun Phrase*)	
Features	Property Type : Properties
[AVM]	*obligation* : NP: $\triangle$(N $\vee$ PRO) *uniqueness* : NP: D! : NP: N! : NP: PP! : NP: PRO! *linearity* : NP: D $\prec$ N : NP: D $\prec$ PRO : NP: D $\prec$ AP : NP: N $\prec$ PP *requirement* : NP: N $\Rightarrow$ D : NP: AP $\Rightarrow$ N *exclusion* : NP: N $\Leftrightarrow$ PRO *dependency* : NP: N$\left[\begin{smallmatrix}\text{GEND } \boxed{1}\\ \text{NUM } \boxed{2}\end{smallmatrix}\right] \rightsquigarrow$ D$\left[\begin{smallmatrix}\text{GEND } \boxed{1}\\ \text{NUM } \boxed{2}\end{smallmatrix}\right]$

Table 1: NP specification in Property Grammar

with is made up of trees labelled with categories, whose surface realisations are the sentences σ of language. A syntax tree of the realisation of the well-formed sentence σ is a *strong* model of the PG grammar $\mathcal{G}$ iff it satisfies every constraint in $\mathcal{G}$. The loose semantics also allows for constraints to be relaxed. Informally, a syntax tree of the realisation of the ill-formed sentence σ is a *loose* model of $\mathcal{G}$ iff it maximises the proportion of satisfied constraints in $\mathcal{G}$ with respect to the total number of evaluated ones for a given category. The set of violated constraints provides a description of the detected error(s).

3 Parsing Algorithm

The class of models is further restricted to constituent tree structures with no pairwise intersecting constituents, satisfying at least one constraint. Since the solution parse must have a single root, should a category not be found for a node a wildcard (called *Star*) is used instead. The Star category is not specified by any constraint in the grammar.

We introduce an algorithm for Loose Satisfaction Chart Parsing (LSCP), presented as Algorithm 1. We have named our implementation of it *Numbat*. LSCP is based on the probabilistic CKY, augmented with a process of *loose constraint satisfaction*. However, LSCP differs from CKY in various respects. While CKY requires a grammar in Chomsky Normal Form (CNF), LSCP takes an ordinary PG grammar, since no equivalent of the CNF exists for PG. Consequently, LSCP generates n-ary structures. LSCP also uses scores of *merit* instead of probabilities for the constituents. That score can be optimised, since it only factors through the influence of the constituent's immediate descendants.

Steps 1 and 2 enumerate all the possible and

Algorithm 1 Loose Satisfaction Chart Parsing

```
/* Initialisation */
Create and clear the chart π: every score in π is set to 0

/* Base case: populate π with POS-tags for each word */
for i ← 1 to num_words
    for (each POS-category T of w_i)
        if merit(T) ≥ π[i, 1, T] then
            Create constituent w_i^T, whose category is T
            π[i, 1, T] ← {w_i^T, merit(w_i^T)}

/* Recursive case */
/* Step 1: SELECTION of the current reference span */
for span ← 1 to num_words
    for offset ← 1 to num_words − span + 1
        end ← offset + span − 1
        K ← ∅
/* Step 2: ENUMERATION of all the configurations */
        for (every set partition P in [offset, ..., end])
            K_P ← buildConfigurations(P)
            K ← K ∪ K_P
/* Step 3: CHARACTERISATION of the constraint system from the grammar */
            for (every configuration A ∈ K_P)
                χ_A ← characterisation(A)
/* Step 4: PROJECTION into categories */
                /* C_A is a set of candidate constituents */
                C_A ← projection(χ_A)
                checkpoint(C_A)
/* Step 5: MEMOISATION of the optimal candidate constituent */
                for (every candidate constituent x ∈ C_A, of construction C)
                    if merit(x) ≥ π[offset, span, C] then
                        π[offset, span, C] ← {x, merit(x)}
        if π[offset, span] = ∅ then
            π[offset, span] ← preferred forest in K
```

legal configurations of optimal sub-structures already stored in the chart for a given span and offset. At this stage, a configuration is a tree with an unlabelled root. Note that Step 2 actually does not calculate all the set partitions, but only the legal ones, *i.e.* those which are made up of subsets of contiguous elements. Step 3 evaluates the constraint system, using a configuration as an assignment. The characterisation process is implemented with Algorithm 2. Step 4 consists of making a category judgement for a configuration, on the basis of which constraints are satisfied and violated, in order to label its root. The process is a simple table lookup, the grammar being indexed by properties. Step 5 then memoises the optimal sub-structures for every possible category. Note that the uniqueness of the solution is not guaranteed, and there may well be many different parses with exact same merit for a given input utterance.

Algorithm 2 Characterisation Function

```
function characterisation(A = ⟨c_1, ..., c_n⟩ : assignment,
                          G: grammar)
        returns the set of evaluated properties relevant to A,
                and the set of projected categories for A.

  /* For storing the result characterisation: */
  create and clear χ_A [property]: table of boolean, indexed by property
  /* For storing the result projected categories: */
  create and clear C_A: set of category
  /* For temporarily storing the properties to be evaluated: */
  create and clear S: set of property

  for (mask ∈ [1 ... 2^n − 1])
      key ← applyBinaryMask(A, mask)
      if (key is in the set of indexes for G) then
          /* Properties are retrieved from the grammar, then evaluated */
          S ← G[key].getProperties()
          χ_A ← evaluate(S)
          /* Projection Step: fetch the categories to be projected */
          C_A ← G[key].getDominantCategories()
  return χ_A, C_A
```

The *key* is a hash-code of a combination of constructions, used for fetching the constraints this combination is concerned with.

Should the current cell in the chart not being populated with any constituents, a preferred forest of partial parses (= Star category) is used instead. The preferred forest is constructed on the fly (as part of `buildConfigurations`); a pointer is maintained to the preferred configuration during enumeration. The preference goes to: (i) the constituents with the widest span; (ii) the least overall number of constituents. This translates heuristically into a *preference score* p_F computed as follows (where F is the forest, and C_i its constituents): $p_F = \mathit{span} \cdot (\mathtt{merit}(C_i) + \mathit{span})$. In that way, LSCP always delivers a parse for any input. The technique is somehow similar to the one of Riezler et al. (2002), where *fragment parses* are allowed for achieving increased robustness, although their solution requires the standard grammar to be augmented with a *fragment grammar*.

4 Evaluation

In order to measure *Numbat*'s ability to (i) detect errors in an ungrammatical sentence, and (ii) build the best approximated parse for it, *Numbat* should, ideally, be evaluated on a corpus of both well-formed and ill-formed utterances annotated with spannnig phrase structures. Unfortunately, such a Gold Standard is not available to us. The development of adequate resources is central to future works. In order to (partially) overcome that problem we have carried out two distinct evaluations: one aims to measure *Numbat*'s performance on grammatical sentences, and the other one on ungrammatical sentences. Evaluation 1, whose results are reported in Table 2, follows the protocol devised for the EASY evaluation campaign of parsers of French (Paroubek et al., 2003), with a subset of the campaign's corpus. For comparison, Table 3 reports the performance measured under the same circumstances for two other parsers: a shallow one (VanRullen, 2005) also based on PG, and a stochastic one (VanRullen et al., 2006). The grammar used for that evaluation was developed by VanRullen (2005). Evaluation 2 was run on

	Precision	Recall	F
Total	**0.7835**	**0.7057**	**0.7416**
general_lemonde	0.8187	0.7515	0.7837
general_mlcc	0.7175	0.6366	0.6746
general_senat	0.8647	0.7069	0.7779
litteraire	0.8124	0.7651	0.788
mail	0.7193	0.6951	0.707
medical	0.8573	0.678	0.757
oral_delic	0.6817	0.621	0.649
questions_amaryllis	0.8081	0.7432	0.7743
questions_trec	0.8208	0.7069	0.7596

Table 2: EASY scores of *Numbat* (Eval. 1)

	Precision	Recall	F
shallow parser	0.7846	0.8376	0.8102
stochastic parser	0.9013	0.8978	0.8995

Table 3: Comparative EASY scores

a corpus of unannotated ungrammatical sentences (Blache et al., 2006), where each of the ungrammatical sentences (amounting to 94% of the corpus) matches a controlled error pattern. Five expert annotators were asked whether the solution trees were possible and acceptable syntactic parses for their corresponding sentence. Specific instructions were given to make sure that the judgement does not hold on the grammatical acceptability of the surface sentence as such, but actually on the parse associated with it. For that evaluation VanRullen's grammar was completed with nested categories (since the EASY annotation scheme only has chunks). Given the nature of the material to be assessed here, the Precision and Recall measurements had to be modified. The total number of input sentences is interpreted as the number of *predictions*; the number of COMPLETE structures is interpreted as the number of *observations*; and the number of structures evaluated as CORRECT by human judges is interpreted as the number of correct solutions. Hence the following formulations and scores: Precision=CORRECT/COMPLETE=0.74; Recall=CORRECT/Total=0.68; F=0.71. 92% of the corpus is analysed with a complete structure; 74% of these complete parses were judged as syntactically correct. The Recall score indicates that the correct parses represent 68% of the corpus. In spite of a lack of a real baseline, these scores compare with those of grammatical parsers.

5 Conclusion

In this paper, we have proposed to address the problem of grammar error detection in providing a set of violated syntactic properties for an ill-formed sentence, along with the best structural context in the form of a connected syntax tree. We have introduced an algorithm for Loose Satisfaction Chart Parsing (LSCP) which meets those requirements, and presented performance measures for it. Future work includes optimisation of LSCP and validation on more appropriate corpora.

Acknowledgement

Partly funded by ANR-07-MDCO-03 (CRoTAL).

References

E. M. Bender, D. Flickinger, S. Oepen, A. Walsh, and T. Baldwin. 2004. Arboretum: Using a precision grammar for grammar checking in CALL. In *Proc. of InSTIL/ICALL2004*, volume 17, page 19.

P. Blache, B. Hemforth, and S. Rauzy. 2006. Acceptability Prediction by Means of Grammaticality Quantification. In *Proc. of CoLing/ACL*, pages 57–64. ACL.

P. Blache. 2001. *Les Grammaires de Propriétés : des contraintes pour le traitement automatique des langues naturelles*. Hermès Sciences.

S. Douglas and R. Dale. 1992. Towards Robust PATR. In *Proc. of CoLing*, volume 2, pages 468–474. ACL.

D. Duchier, J-P. Prost, and T-B-H. Dao. 2009. A Model-Theoretic Framework for Grammaticality Judgements. In *To appear in Proc. of FG'09*, volume 5591 of *LNCS*. FOLLI, Springer.

J. Foster. 2007. Real bad grammar: Realistic grammatical description with grammaticality. *Corpus Linguistics and Lingustic Theory*, 3(1):73–86.

K. Foth, W. Menzel, and I. Schröder. 2005. Robust Parsing with Weighted Constraints. *Natural Language Engineering*, 11(1):1–25.

F. Fouvry. 2003. Constraint relaxation with weighted feature structures. pages 103–114.

P. Paroubek, I. Robba, and A. Vilnat. 2003. EASY: An Evaluation Protocol for Syntactic Parsers. www.limsi.fr/RS2005/chm/lir/lir11/ (08/2008).

S. Riezler, T. H. King, R. M. Kaplan, R. Crouch, J. T. III Maxwell, and M. Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques. In *Proc. of ACL*, pages 271–278. ACL.

T. VanRullen, P. Blache, and J-M. Balfourier. 2006. Constraint-Based Parsing as an Efficient Solution: Results from the Parsing Evaluation Campaign EASy. In *Proc. of LREC*, pages 165–170.

T. VanRullen. 2005. *Vers une analyse syntaxique à granularité variable*. Thèse de doctorat.