A Grammar Correction Algorithm

Deep Parsing and Minimal Corrections for a Grammar Checker

Lionel Clément[1] Kim Gerdes[2] Renaud Marlet[3]

[1] LaBRI, Université Bordeaux 1
[2] ILPGA, LPP, Sorbonne Nouvelle
[3] LaBRI, INRIA Bordeaux – Sud-Ouest

Abstract. This article presents the central algorithm of an open system for grammar checking, based on deep parsing. The grammatical specification is a context-free grammar with flat feature structures. After a shared-forest analysis where feature agreement constraints are relaxed, error detection globally minimizes the number of corrections and alternative correct sentences are automatically proposed.

1 Introduction

Grammar checkers are among the most common NLP technologies used by the general public. Yet they have attracted comparatively little research, at least w.r.t. the number of publications. There are most likely various reasons for this. First, from a practical point of view, the systems appear to be highly dependent on the language they correct. In addition, a large part of grammatical definitions is a collection of idiosyncratic errors, that may depend on the language of the user and his level of knowledge. This task is difficult to automate due to the unavailability of large error corpora. Finally, the usefulness of such a system relies heavily on its integration into a word processor, and a grammar checker can easily be made available to the public only since the rise of OpenOffice. It is now conceivable that a research community form around grammar checking, openly sharing resources and results, just as in other areas of NLP. There may also be deeper reasons: linguistics has taken a long time to define itself as a science of the language that is actually spoken, and normativity is no longer at the center of interest, although some research on deviations from the norm (in sociolinguistics, psycholinguistics, lexicology, or in research on foreign language teaching) can have an indirect interest for the development of error grammars.

After showing the necessity for quality grammar checking based on deep syntactic analysis, as opposed to the frequently used superficial and local grammars, we present the algorithm at the heart of our system. We show how to detect, locate, and order minimal correction proposals, using a tweaked-up context free grammar parser. This is the central module of the open grammar checker that we are developing, using, for the moment, a French grammar, that is also used here for illustration purposes.

2 How to check grammars

A grammar checker has two main functions:

– It notifies the user of possibly incorrect sentences (or fragments).
– It proposes corrections, possibly with a "linguistic" explanation of the error.

In practice, additional properties also are desirable:

– It should be fast: checking should be perform as the user types.
– It should minimise noise: too many false alarms would make it unusable.
– It should minimise silence: only few errors should remain unnoticed.

2.1 Minimal correction cost

Uszkoreit, quoted in [1], distinguishes 4 steps in the grammar correction process: (1) identification of possibly ungrammatical segments, (2) identification of the possibly infringed constraints, (3) identification of the possible source of the error, and (4) construction and ordering of the correct alternatives.

In this paper, we focus on this last point. The problem is to find alternative correct sentences in the *neighborhood* of an incorrect sentence and to order them by *plausibility*. We show how to model the plausibility of a correction in terms of its *minimal correction cost*.

The notion of minimality of a proposed correction is non trivial and different definitions can be given. Consider the following example, keeping in mind that French has an agreement in gender and number between the subject and the predicative adjective:

Example 1.

(a) * Les cheval blanc sont arrogants.
The-⟨*plur,masc/fem*⟩ horse-⟨*sing,masc*⟩ white-⟨*sing,masc*⟩ are-⟨*plur,3rd person*⟩ arrogant-⟨*plur,masc*⟩
(b) Le cheval blanc est arrogant.
The-⟨*sing,masc*⟩ horse-⟨*sing,masc*⟩ white-⟨*sing,masc*⟩ is-⟨*sing,3rd person*⟩ arrogant-⟨*sing,masc*⟩
(c) Les chevaux blancs sont arrogants.
The-⟨*plur,masc/fem*⟩ horses-⟨*plur,masc*⟩ white-⟨*plur,masc*⟩ are-⟨*plur,3rd person*⟩ arrogant-⟨*plur,masc*⟩

With a local decision principle, the subject noun phrase "les cheval blanc" has a best correction with a singular determiner ("le" instead of "les"). For the whole sentence, we end up with 3 singular words and 2 plural words, which tips the balance towards correction (b). With a global decision principle, however, we started out with 5 words that needed number agreement: 3 are plural, 2 are singular. The optimal correction, minimal in the number of words to change, is therefore (c). This shows that the correction is best taken globally, as local decisions can imply iterative suboptimal proposals.

Two proximity models allow for plausible corrections: we can count the number of words to be changed, or the number of features to be changed. Although the two approaches often agree, they are incomparable:

Example 2.

(a) * C'est une histoire de cliente arrivée mécontent mais repartis satisfaits.
It is a story of client-⟨*sing,fem*⟩ arrived-⟨*sing,fem*⟩ unhappy-⟨*sing,masc*⟩ but left-⟨*plur,masc*⟩ satisfied-⟨*plur,masc*⟩

(b) C'est une histoire de client arrivé mécontent mais reparti satisfait.
It is a story of client-⟨*sing,masc*⟩ arrived-⟨*sing,masc*⟩ unhappy-⟨*sing,masc*⟩ but left-⟨*plur,masc*⟩ satisfied-⟨*plur,masc*⟩

(c) C'est une histoire de cliente arrivée mécontente mais repartie satisfaite.
It is a story of client-⟨*sing,fem*⟩ arrived-⟨*sing,fem*⟩ unhappy-⟨*sing,fem*⟩ but left-⟨*sing,fem*⟩ satisfied-⟨*sing,fem*⟩

(d) C'est une histoire de clients arrivés mécontents mais repartis satisfaits.
It is a story of client-⟨*plur,masc*⟩ arrived-⟨*plur,masc*⟩ unhappy-⟨*plur,masc*⟩ but left-⟨*plur,masc*⟩ satisfied-⟨*plur,masc*⟩
"It's a story of client(s) that came in unhappy but that left satisfied."

In (a), the "client" noun phrase is rather masculine, 3 votes against 2, and rather singular, equally 3 votes against 2. Minimizing the number of features to correct affects 4 occurrences of features and 4 words (b). But two alternatives are more economical w.r.t. the number of corrected words: (c) and (d) both correct 5 features but only 3 words. Our algorithm chooses the minimization of features to correct, that seem more plausible from a "cognitive" point of view. Moreover, the word minimization choice can be encoded in our algorithm as well.

2.2 Positive grammar vs negative grammar

We can identify incorrect sentences with a *positive* grammar, generating all grammatical sentences, or with a *negative* grammar, describing ungrammaticalities.

However, we believe that a complete grammar checker actually needs both a positive and a negative grammar, for different types of error detection. The global structure of the sentence and the agreement phenomena is easier to described positively, because there are more ungrammatical variations than grammatical structures and agreement possibilities. Yet, other types of errors do not integrate as easily into a positive grammar. Common stylistic errors like barbarisms (e.g., the Gallicism 'to assist to a meeting') or pleonasms (e.g., 'free gift') are much more easily detected negatively by specific rules.

This separation in two types of grammars can also be motivated from linguistic or engineering standpoints: the positive grammar describes the basic general language properties, that are independent of the speaker, whereas the negative grammar can be adapted for the user of the system, e.g., to detect false cognates that are specific to a language pair (e.g., 'affair' vs "affaire").

Although we currently only support a positive grammar, we plan to support a negative grammar as well, which would be based on deep syntactic information gathered during the (relaxed) positive grammar parsing.

2.3 Shallow and deep parsing

Grammar checking can be done with a variety of tools: part-of-speech tagging, chunking, regular expressions, unification grammars, and so on. The more an analysis is superficial, the more it is prone to noise and silence — not only because syntactic information is partial, but also because these tools have a high error rate on whole sentences. Shallow parsing is thus confined to negative grammars, where rule locality reduces these effects [2].

Most grammar checkers (e.g., for French: Antidote, Cordial, Microsoft Office, and Prolexis) are commercial products whose functioning is opaque. Although the parsing depth can sometime be known or guessed, the actual computation remains undisclosed. LanguageTool [3] is one of the few open grammar checkers, integrated in the upcoming version of OpenOffice (3.1). The grammar correction infrastructure is based on a part-of-speech tagger and on local rules built with regular expressions. More than 1600 rules are available for French. Consider:

Example 3.

a * Le chien de mes voisins mordent.
The dog of my neighbors bite.
b Les voitures qu'il a font du bruit.
The cars that he has make noise.

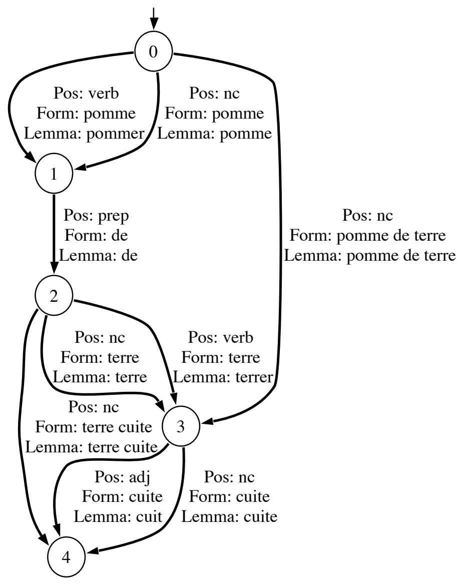
With its local rules, LanguageTool remains silent on (a) and detects two false errors in (b). Microsoft Office [4], which seemingly makes a deeper analysis, flags the error in (a), but asks, wrongly, to replace "font" ('make') by "fait" ('made') in (b). In order to verify the global structure and the (possibly long-distance) agreements, we cannot do without a deep and complete analysis of the sentence.

In the rest of this article, we describe the deep parsing as well as the error detection and correction which form the central part of our grammar checker.

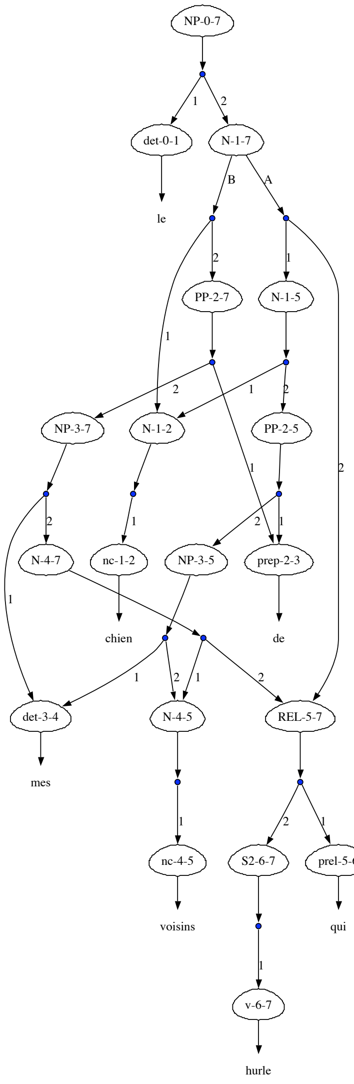
3 Positive error parsing and correction

In short, the correction process of our checker is as follows. A sentence is first segmented into a lattice of inflected forms, i.e., a directed acyclic graph (DAG) with unique initial and final nodes, to which we add extra lemmas representing plausible substitutions (e.g., homophones). Figure 1a shows a simple DAG produced by the the lexical analysis. In French, "pomme de terre" and "terre cuite" are idioms ('potato' and 'terracotta') as well as phrases ('apple of earth' and 'cooked earth'). Besides, the lexer also gives two analyses to words like "du": as a partitive determiner and as a contraction of the preposition "de" ('of') and the definite determiner "le" ('the').

The parser then constructs a shared forest of analyses ignoring agreement phenomena. Next, a bottom-up pass through the forest attributes to alternative parses the cost of the minimal correction that satisfy the agreements, by modifying features or of using substituted lemmas. A sentence that has a minimal cost of zero is grammatical; otherwise, it is ungrammatical and a top-down pass determines the inflections and substitutions that reconstruct correct sentences.



(a) Word lattice for "pomme de terre cuite"



(b) Shared forest for "Le chien de mes voisins qui hurle" 'The dog of my neighbours that howls'

Fig. 1. Examples of lexical and syntactic analyses

3.1 Lexical correction

The notion of proximity significantly applies to word level: the lexicon has to define what forms can correct others and at what price.

A first type of corrections is based on inflection, i.e., different word forms of the same lemma, e.g., "jolie" ('pretty'-⟨*fem*⟩) for "joli" ('pretty'-⟨*masc*⟩). But not all features may vary freely, e.g., we can get the plural form of "crayon" ('pencil'-⟨*sing,masc*⟩) but we cannot make it feminine; it somehow has an infinite cost.

A second type of corrections corresponds to plausible substitutions of lemmas, in particular homophones: "on" ('one') / "ont" ('have'), "est" ('is') / "ait" ('has') / "ai" ('have'), "à" ('to') / "a" ('has'), etc. As categories vary in this case, contrary to the previous situation with inflection, the parsing of these corrections adds additional alternative analyses, adding to the complexity of the overall process. The addition of such alternative lemmas therefore has to be done carefully.

Such corrections might first seem as an intrusion on the field of negative grammars, but the integration of similar lemmas is coherent with our goal: we want to include in the neighborhood of correct sentences (positive grammar) the most plausible incorrect corresponding sentences. Besides, the choice of the grouping of words under the same lemma is to a certain extent dependent on linguistic and theoretical considerations. For example, the frequently confused relative pronouns "que" ('which'-⟨*accusative*⟩) and "dont" ('of which'-⟨*genitive*⟩) can be seen either as different lemmas, or as a common *pronoun lemma* with different values for the case feature.

At the border between positive and negative grammar, we use a similar procedure for prepositions that are complements of verbs: we encode frequently subcategorized prepositions ("à", "de", "par", ...) as variations governed by a feature. Alternatively, we could have placed the detection of theses cases in the negative grammar. The choice between the two grammars depends on the difficulty to describe the possible error. Also, an advantage of expressing a phenomenon in the positive grammar is to automatically provide correction proposals.

For the positive grammar that we discuss here, the correction cost can depend on the type of lemma, on the type of forms, and may even be user-specific. It constitutes an independent resource. We thus have three lexical resources[4]:

– a classical *inflectional lexicon*, containing the lemmas, their word forms and their features constraints,
– a *substitution lexicon*, listing substitution groups of different lemmas/forms,
– a *model of proximity*, associating substitution costs to features and lemmas.

Our inflectional lexicon has the following extensional format:

```
# Form    Lemma    Categories and features
heureux   heureux  adj[gen=masc; nb=sing,plur]
portions  portion  nc[gen=fem!; nb=plur]
portions  porter   v[pers=1; nb=plur; mode=ind,subj; tps=imp]
```

In the case of different lemmas sharing the same form, we have an entry for each lemma. For each feature, a status concerning its possible values has to be provided: a value is either *prescribed* (i.e., compliant with the inflected form, thus assignable at a null cost), *plausible* (not covered by the inflected form but assignable to reach an agreement, at a non-null cost), or *forbidden* (not assignable, or equivalently, assignable at an infinite cost). These assignments

[4] Not counting a separate definition of features, shared by the lexicon and grammar.

constraints are specified by a list of values. By default all mentioned values are prescribed. The mark "`!`" forbids values other than those that are listed[5]. This syntax is light weight and furthermore allows for an easy static recognition of incoherence in the lexicon w.r.t. the feature definitions.

To a large extent, these kinds of lexicons can be build automatically [5], which is valuable for languages that are poor in free resources. In fact, our French lexicon is compiled from the Lefff, the free lexicon of French inflected forms, which was built on this basis. Our substitution lexicon, containing substitutable lemmas as "`on|ont`" or "`a|à`", stems mainly from the homophones of the *Lexique 3* [6]. The lexer systematically introduces these forms as alternatives into the lexical lattice before parsing.

Work in progress concerns a parameterizing tool for the cost model, including feature-dependant and substitution costs, as well as an extension based on a kind of Levenshtein distance to possibly balance correction costs according to lexicographic proximity. Presently, the cost model is a simple ad hoc program attributing costs following a fixed scheme: prescribed values have a cost of zero; plausible values have a cost of 1; and forbidden values have an infinite cost.

Substitutions, too, currently have a cost of 1. Contrary to other feature value assignments, this cost has to be paid as soon as the corresponding alternative is chosen. This can be modeled by an additional implicit feature, here named "$", with only one possible value: it thus always has to be assigned. N.B. All these arbitrary values are to be adjusted by a practical evaluation of the grammar.

Currently, most systems make spelling correction and grammar checking separate. The user's best strategy is to correct spelling first. Our correction scheme allows a tighter integration of a spell checker and a grammar checker: correction proposals for unknown words can be added into in the lattice built by the lexer. A single operation can then correct wrong inflections like "chevals" (an imaginable but wrong plural form of "cheval" ('horse'), the correct plural form being "chevaux") or "faisez" (an imaginable but wrong second person plural form of "faire" ('to do'), the correct form being "faites").

3.2 The principle of constraint relaxation

The notion of *proximity* for the correction of an ungrammatical sentence is subjective. The hypothesis that we make is that the proximity can be modeled with a positive grammar in which we relax some constraints. We thus obtain more analyses of the input sentence, and a cost assignment for the unsatisfied constraints measures the proximity to a correct sentence.

For most constraint-based formalisms, experiments have been done on constraint relaxations. Some formalisms as Property Grammars [7] even consider it as a key point of their approach. This idea also has been applied to HPSG [8]. And error ranking can be refined using weights representing feature importance [9]. For parsers of simpler grammars, constraint relaxation also has proved robust.

[5] Marks "`?`" and "`#`" (not exemplified here) identify respectively plausible and forbidden values, allowing more explicit or finer specifications.

Real-time grammar checking requires simple and efficient parsing mechanisms. Our system for parsing with constraint relaxation is based on a well known formalism: we use a feature-based context-free grammar, the features being flat and with finite values. Here is a simplified example of a rule:

```
sn[nb=N;gen=G;pers=3] -> det[nb=N;gen=G] sadj[nb=N;gen=G;
     type=anté]* nc[nb=N;gen=G] sadj[nb=N;gen=G;type=post]*
     pp[]? rel[nb=N; gen=G]? ;
```

We relax the agreement constraints related to features, but impose that the sentence be parsable by the bare (feature-free) context-free grammar; the correction cost computations are then performed on the context-free backbone. Note that the grammar can also be specifically written to relax some requirements and accept incorrect phrases with simple cases of structural incoherence, e.g., considering as grammar axioms not only sentences but also a few given phrases, or making some terms optional. Moreover, as explained earlier, we systematically add paths into the lexical lattice for words that are closed to the actual input words (homophones or words that are often mistaken one for another).

We keep all structural ambiguities of our context-free analysis in a shared forest [10], using a variant of an Earley parser [11] that can directly handle Kleene stars and optional terms. The complexity is the same as for standard Earley parsing: the worst case complexity is cubic in the length of the input sentence. An example of such a shared parsing forest is given in figure 1b.

Moreover, our parser considers each word of a sentence as a potential starting point of the grammar axiom. We thus can construct partial analyses even for sentences and phrases that do not have a global analysis in our grammar, and flag local feature agreement errors in noun phrases, verb clusters, etc. This not only is useful for robustness, but also to signal meaningful errors even with a grammar with limited coverage.

We end up with two types of grammatical errors: *structural errors* (for which we do not find a single constituent structure) and *non structural errors* (for which unification fails) [12]. Note that adding an alternative to the substitution lexicon can transform a structural error into a non structural error. We can even manage to handle certain word order constraints by means of additional features and lemmas, e.g., for the anteposition and postposition of adjectives.

4 Signaling Errors

The text to analyze is assumed to be segmented into individual sentences[6]. As explained above, each such sentence is itself segmented by a lexer that constructs a lattice of inflected word-forms. Alternatives in this lattice are due to lexical ambiguities (including compounds) as well as possible word substitutions.

Parsing this lattice then constructs a shared forest of parse trees (see figure 1b). Features are entirely ignored at this stage; we only build the derivation

[6] Splitting heuristics based on punctuation provide a reasonable segmentation. We are studying a more robust technique that would exploit the fact that our parser is able to work on a stream of characters.

trees of the context-free skeleton of the grammar, with sharing. The resulting parse forest can be represented by a finite number of equations, that belong to one of the following three types:

- $f = \{t_1, \dots, t_m\}$ is a forest of m alternative trees, each tree being associated to the same non-terminal in the grammar.
- $t = \rho(f_1, \dots, f_n)$ is the node of a tree, constructed using grammatical rule ρ, where n is the number of terms used in the right-hand side of ρ. (The number n refers to a rule instance, taking into account variants due to the option sign `?` and the Kleene star `*`.)
- $t = l$ is a leaf node, constructed using lexical entry l.

The parser puts up with infinite analyses, as may occur (unwittingly) with rules such as "`np -> np pp?`". In this case, it generates cycles in the forest, that can be easily removed. For what follows, the forest is assumed to be a lattice.

4.1 Error Mining

Looking for errors requires traversing the parse forest to determine parse alternatives with minimum correction costs. A forest that contains a rooted tree with a null correction cost is considered as error free. Otherwise, plausible corrections associated to the parse alternatives are ranked according to their cost.

A correction is represented as a *feature assignment* $\alpha = (u_\varphi)_{\varphi \in \Phi}$ such that:

- Φ is a set of features that each can be assigned a value.
- For each feature $\varphi \in \Phi$, $u_\varphi \in \mathit{Dom}(\varphi)$ is a feature value.

A feature assignment example is $\boldsymbol{\alpha} = (\mathsf{gender} \mapsto \mathsf{masc}, \mathsf{number} \mapsto \mathsf{pl})$. Moreover, a *feature assignment cost* $\kappa = (c_\varphi)_{\varphi \in \Phi}$ is a family of *cost functions* such that:

- $c_\varphi : \mathit{Dom}(\varphi) \to \mathbb{N} \cup \{\infty\}$ associates an cost $c_\varphi(u_\varphi)$ to a feature value u_φ.

An example of a feature assignment cost for a masculine singular adjective is $\boldsymbol{\kappa} = (\mathsf{gender} \mapsto (\mathsf{masc} \mapsto 0, \mathsf{fem} \mapsto 1), \mathsf{number} \mapsto (\mathsf{sg} \mapsto 0, \mathsf{pl} \mapsto 1))$. An infinite cost corresponds to an impossible feature assignment, such as imposing the masculine gender to 'mother' or, in French, imposing a feminine gender to "crayon" ('pen').

Last, to perform the feature assignment $\alpha = (u_\varphi)_{\varphi \in \Phi}$ with the feature assignment cost $\kappa = (c_\varphi)_{\varphi \in \Phi}$ has a total cost of $\kappa(\alpha) = \sum_{\varphi \in \Phi} c_\varphi(u_\varphi)$. For instance, with $\boldsymbol{\alpha}$ and $\boldsymbol{\kappa}$ as defined above, the total cost is $\boldsymbol{\kappa}(\boldsymbol{\alpha}) = 0 + 1 = 1$.

To look for corrections in the forest and rank them, we investigate the feature assignment costs of parse alternatives. There can be an exponential number of such alternatives (w.r.t. the sentence length). In the rest of this subsection, we consider that we scan all these alternatives; in the following subsection, we show how to reduce and control this potential combinatorial explosion. Still, even though we examine each alternative, we exploit the sharing in the forest to factorize the computation of alternative feature assignment costs.

For each node t or forest f in the parse structure, we compute a set of *feature assignment costs* $(\kappa_i)_{i \in I} = ((c_{i,\varphi})_{\varphi \in \Phi_i})_{i \in I}$ representing the set of minimum costs for using t or f in an alternative parse i, for the different feature assignments. This computation operates bottom-up (from leafs to the roots) as follows[7].

(a) Let $t = l = s[\varphi = ...]_{\varphi \in \Phi}$ be a leaf in the parse forest. It represents a single alternative. Its feature assignment cost $(c_\varphi)_{\varphi \in \Phi}$ is defined by the cost model (cf. §3.1).

(b) Let $f = (t_i)_{i \in I}$ be a forest. Each alternative i is itself based on parse alternatives $j \in J_i$, whose feature assignment costs are $((c_{i,j,\varphi})_{\varphi \in \Phi_{i,j}})_{j \in J_i}$. The alternative feature assignment costs for f are thus $((c_{i,j,\varphi})_{\varphi \in \Phi_{i,j}})_{i \in I, j \in J_i}$, i.e., the set union of all alternative costs. (No global decision regarding assignment costs can be taken at this stage.)

(c) Let $t = \rho(f_i)_{i \in I}$ be a node built on the grammar rule ρ, $((c_{i,j,\varphi})_{\varphi \in \Phi_{i,j}})_{j \in J_i}$ be the alternative feature assignment costs of forests f_i, $c_0[\varphi = x_{0,\varphi}]_{\varphi \in \Phi_0}$ be the left-hand side of the instance of ρ, and $c_i[\varphi = x_{i,\varphi}]_{\varphi \in \Phi_i}$ be the term corresponding to f_i in the right-hand side of ρ.

1. For each forest f_i and each alternative cost $(c_{i,j,\varphi})_{\varphi \in \Phi_{i,j}}$ in f_i, we rule on features $\varphi \in \Phi_{i,j}$ that are not propagated in ρ for lack of equations of the form "$\varphi = x$" in $c_i[...]$. The minimum assignment cost of such a feature φ, for this alternative (i, j), is $\min(c_{i,j,\varphi}) = \min_{u \in \mathit{Dom}(\varphi)} c_{i,j,\varphi}(u)$. E.g., with cost $\boldsymbol{\kappa}$ as above, we have $\min(\boldsymbol{c}_{\mathsf{number}}) = \min(0, 1) = 0$.
   As it is not possible to rule on propagated features, as going up the forest may add new constraints, it is not yet possible to tell which alternative costs less. The minimum assignment cost $\min(c_{i,j,\varphi})$ must thus be preserved for when the other features of alternative (i, j) are evaluated. For this, we use the special feature named \$ (that is also used in the substitution lexicon, cf. §3.1), that we systematically propagate up the forest: the \$ feature is *implicitly* considered as present at each non-terminal occurrence, associated with the same feature variable, named $x_\$$. In other words, each rule $s_0[...] \to s_1[...] \ldots s_n[...]$ actually represents the rule $s_0[...; \$ = x_\$] \to s_1[...; \$ = x_\$] \ldots s_n[...; \$ = x_\$]$. For the rest of the computations, ruling on non-propagated features amounts to replacing each feature assignment cost $(c_{i,j,\varphi})_{\varphi \in \Phi_{i,j}}$ by $(c'_{i,j,\varphi})_{\varphi \in \Phi'_{i,j}}$ where:
   - The set of propagated features of alternative (i, j) is: $\Phi'_{i,j} = \Phi_{i,j} \cap \Phi_i$.
   - The set of non-propagated features of alternative (i, j) is: $\Phi_{i,j} \setminus \Phi_i$.
   - For propagated features $\varphi \in \Phi'_{i,j}$, except \$, we have $c'_{i,j,\varphi} = c_{i,j,\varphi}$.
   - Feature \$ accumulates the minimum costs of non-propagated features of alternative (i, j): $c'_{i,j,\$} = c_{i,j,\$} + \sum_{\varphi \in \Phi_{i,j} \setminus \Phi_i} \min(c_{i,j,\varphi})$.

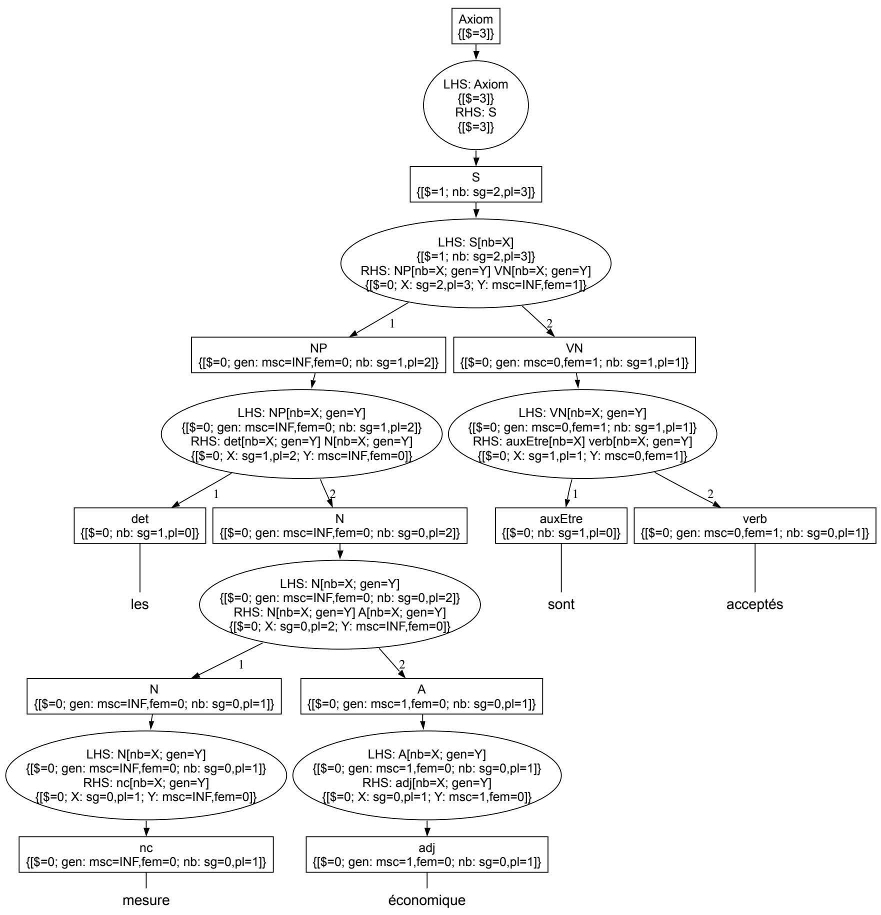
[7] For lack of space, we do not describe here the treatment of a number of grammar constructs: filtering (equivalent to $=_c$ in LFG), constant equality constraint ("$\varphi = v$" in the left- or right-hand side of a rule), joint constant and variable equality constraint (written "$\varphi = x \,\&\, v$)". The "$\varphi = x$" case, that we describe here, is at the heart of the minimal correction algorithm.

2. We then construct the *agreement cost* of the variables of rule ρ. We consider each alternative $\bar{\jmath} = (j_i)_{i\in I} \in \bar{J} = \prod_{i\in I} J_i$, i.e., all combinations of an alternative from each forest f_i. (See §4.2 for combinatorial reduction.) For each feature variable x, whose value domain is $\mathit{Dom}(x)$, in the set X_r of feature variables occurring on the right-hand side of ρ, we define a cost function $c_x : \mathit{Dom}(x) \to \mathbb{N} \cup \{\infty\}$ that represents the sum of the feature assignment costs that are associated to x in the different alternatives j_i in which x occurs in $s_i[\dots]$. In other words, let Φ_i^x be the set of features $\varphi \in \Phi_i$ such that $x_{i,\varphi} = x$. Then $c_{\bar{\jmath},x} = \sum_{i\in I,\ \varphi\in\Phi_i^x\cap\Phi'_{i,j_i}} c'_{i,j_i,\varphi}$.
   This sum of functions only concerns features that are both present in $s_i[...]$ and f_i. A feature that is present in $s_i[...]$ but not in f_i, i.e., in $\Phi_i \setminus \Phi_{i,j_i}$, is implicitly considered as having a null cost function. The computation of assignment costs $(c_{\bar{\jmath},x})_{x\in X_\mathsf{r}}$ can be implemented using side effects, with a single traversal of the costs functions of alternative $\bar{\jmath}$.
3. Finally, for each cost $(c_{\bar{\jmath},x})_{x\in X_\mathsf{r}}$ of an alternative $\bar{\jmath} \in \bar{J}$, which somehow represents an alternative instance of the right-hand side of ρ, we construct the feature assignment cost corresponding to the left-hand side. Given that some variables on the right-hand side (in X_r) are not necessarily present in the left-hand side (in X_0), some feature assignment costs are not propagated at this stage either. As above, they are accumulated in feature \$. Besides, if a variable is only present in the left-hand side, it is useless to propagate the corresponding feature. (Alternatively, it is possible to consider the associated cost function as null.)
   More formally, the feature assignment cost constructed for the left-hand side $s_0[...]$ of ρ, corresponding to alternative $\bar{\jmath}$, is $(c'_{\bar{\jmath},\varphi})_{\varphi\in\Phi'_0}$ where:
   - The propagated features are the feature on the left-hand side, except for the features that are associated to a variable that does not occur on the right-hand side, but including feature \$: $\Phi'_0 = \{\varphi \in \Phi_0 \mid x_{0,\varphi} \notin X_\mathsf{r}\} \cup \{\$\}$.
   - For each propagated feature $\varphi \in \Phi'_0$ besides \$, the alternative cost for the corresponding variable $x_{0,\varphi}$ is propagated: $c'_{\bar{\jmath},\varphi} = c_{\bar{\jmath},x_{0,\varphi}}$.
   - Feature \$ accumulates the minimum costs of the feature variables that are not propagated from the right-hand side to the left-hand side: $c'_{\bar{\jmath},\$} = c_{\bar{\jmath},x_\$} + \sum_{x\in X_\mathsf{r}\setminus X_0} \min(c_{\bar{\jmath},x})$.

When the roots of the parse forest are reached, a set of alternative costs $(\kappa_i)_{i\in I} = ((c_{i,\varphi})_{\varphi\in\Phi_i})_{i\in I}$ is available. The minimum correction cost is $\min_{i\in I}(\min(\kappa_i))$. This cost may correspond to several alternatives and several feature assignments.

A simple example of such an analysis is shown in figure 2. The global cost for the sentence is 3 ("\$" feature): 2 for the cost of assigning the number feature of "mesure économique" to singular, 1 for the cost of assigning the gender feature of "acceptés" to feminine. (Other features have been discarded for readability.)

Once a feature value assignment with minimum cost is chosen, an opposite top-down traversal (from the roots to the leafs) enumerates actual correction alternatives that implement this minimum cost. More alternatives can be listed using cost for ranking. Conversely, if there are too many correction alternatives



Axiom {[$=3]}
LHS: Axiom {[$=3]} RHS: S {[$=3]}
S {[$=1; nb: sg=2,pl=3]}
LHS: S[nb=X] {[$=1; nb: sg=2,pl=3]} RHS: NP[nb=X; gen=Y] VN[nb=X; gen=Y] {[$=0; X: sg=2,pl=3; Y: msc=INF,fem=1]}
1
2
NP {[$=0; gen: msc=INF,fem=0; nb: sg=1,pl=2]}
VN {[$=0; gen: msc=0,fem=1; nb: sg=1,pl=1]}
LHS: NP[nb=X; gen=Y] {[$=0; gen: msc=INF,fem=0; nb: sg=1,pl=2]} RHS: det[nb=X; gen=Y] N[nb=X; gen=Y] {[$=0; X: sg=1,pl=2; Y: msc=INF,fem=0]}
LHS: VN[nb=X; gen=Y] {[$=0; gen: msc=0,fem=1; nb: sg=1,pl=1]} RHS: auxEtre[nb=X] verb[nb=X; gen=Y] {[$=0; X: sg=1,pl=1; Y: msc=0,fem=1]}
det {[$=0; nb: sg=1,pl=0]}
N {[$=0; gen: msc=INF,fem=0; nb: sg=0,pl=2]}
auxEtre {[$=0; nb: sg=1,pl=0]}
verb {[$=0; gen: msc=0,fem=1; nb: sg=0,pl=1]}
les
sont
acceptés
LHS: N[nb=X; gen=Y] {[$=0; gen: msc=INF,fem=0; nb: sg=0,pl=2]} RHS: N[nb=X; gen=Y] A[nb=X; gen=Y] {[$=0; X: sg=0,pl=2; Y: msc=INF,fem=0]}
N {[$=0; gen: msc=INF,fem=0; nb: sg=0,pl=1]}
A {[$=0; gen: msc=1,fem=0; nb: sg=0,pl=1]}
LHS: N[nb=X; gen=Y] {[$=0; gen: msc=INF,fem=0; nb: sg=0,pl=1]} RHS: nc[nb=X; gen=Y] {[$=0; X: sg=0,pl=1; Y: msc=INF,fem=0]}
LHS: A[nb=X; gen=Y] {[$=0; gen: msc=1,fem=0; nb: sg=0,pl=1]} RHS: adj[nb=X; gen=Y] {[$=0; X: sg=0,pl=1; Y: msc=1,fem=0]}
nc {[$=0; gen: msc=INF,fem=0; nb: sg=0,pl=1]}
adj {[$=0; gen: msc=1,fem=0; nb: sg=0,pl=1]}
mesure
économique

Fig. 2. Correction cost evaluation for "* Les mesure économique sont acceptés"

for a given cost, some may be (randomly) ignored, not to clutter the user with too many choices.

4.2 Combinatorial Reduction and Control

Thanks to the sharing in the parse forest, the algorithm in §4.1 also shares some computations and data structures. Still, it enumerates all parsing alternatives, whose number can be exponential (w.r.t. to the length of the sentence) in the worst case. It is crucial to reduce the number of combinations to consider, without altering the optimum, or using heuristics. For this, we can first set minimum and maximum bounds for a cost function c_φ (cf. example §4.1):

- $\min(c_\varphi) = \min_{u \in Dom(\varphi)}(c_\varphi(u))$ e.g., $\min(\boldsymbol{c}_{\mathsf{genre}}) = \min(0,1) = 0$
- $\max(c_\varphi) = \max_{u \in Dom(\varphi)}(c_\varphi(u))$ e.g., $\max(\boldsymbol{c}_{\mathsf{genre}}) = \max(0,1) = 1$

Besides, the minimum and maximum bounds of a feature assignment cost are:

- $\min(\kappa) = \sum_{\varphi \in \Phi} \min(c_\varphi)$ e.g., $\min(\boldsymbol{\kappa}) = \min(0,1) + \min(0,1) = 0$,
- $\max(\kappa) = \sum_{\varphi \in \Phi} \max(c_\varphi)$ e.g., $\max(\boldsymbol{\kappa}) = \max(0,1) + \max(0,1) = 2$,

For any feature assignment α, we are sure that $\min(\kappa) \leq \kappa(\alpha) \leq \max(\kappa)$.

Last, two relations $\leq_\forall$ and $\leq^{\min}_{\max}$ are defined on feature assignment costs. $\leq_\forall$ expresses an upper bound on the costs for each feature value that can be assigned; an absent feature is treated as present with null costs. $\leq^{\min}_{\max}$ expresses the fact that the maximum cost of one assignment is less than the minimum cost of the other one. More precisely, for each $\kappa_1 = (c_{1,\varphi})_{\varphi \in \Phi_1}$ and $\kappa_2 = (c_{2,\varphi})_{\varphi \in \Phi_2}$,

- $\kappa_1 \leq_\forall \kappa_2$ iff $\forall \varphi \in \Phi_1, \forall u \in \mathit{Dom}(\varphi)$, if $\varphi \in \Phi_2$ then $c_{1,\varphi}(u) \leq c_{2,\varphi}(u)$ else $c_{1,\varphi}(u) = 0$,
- $\kappa_1 \leq^{\min}_{\max} \kappa_2$ iff $\max(\kappa_1) \leq \min(\kappa_2)$.

Relation $\leq_\forall$ is a partial order and $\leq^{\min}_{\max}$ is transitive. But, more importantly, they satisfy the following property:

- if $\kappa_1 \leq_\forall \kappa_2$ or $\kappa_1 \leq^{\min}_{\max} \kappa_2$, then for any assignment α: $\kappa_1(\alpha) \leq \kappa_2(\alpha)$.

In this case, if κ_1 and κ_2 are in the same set of alternative costs, whatever the final choice of feature assignment may be, κ_2 will always cost more than κ_1. Cost κ_2 can thus be discarded from the alternatives to consider. This leads to a combinatorial reduction.

To implement this, if $\trianglelefteq$ is one of the relations $\leq_\forall$, $\leq^{\min}_{\max}$, or $\leq_\forall \cup \leq^{\min}_{\max}$ (i.e., the disjunction of both relations $\leq_\forall$ and $\leq^{\min}_{\max}$), we define $\inf_\trianglelefteq$ as the function that takes a set of feature assignment costs $(\kappa_i)_{i \in I}$ and returns its minimal elements with respect to $\trianglelefteq$. In other words, we have:

- $\inf_\trianglelefteq((\kappa_i)_{i \in I}) = (\kappa_j)_{j \in J}$ where $J = \{j \in I \mid \forall i \in I \setminus \{j\}, \kappa_i \not\trianglelefteq \kappa_j\}$

To reduce the number of alternative costs to take into account, without affecting the optimum (cf. §4.1), it is possible to replace a set of alternative feature assignment costs $(\kappa_i)_{i \in I}$ by its minimal elements $\inf_\trianglelefteq((\kappa_i)_{i \in I})$. This applies to:

- case (b): for a forest $f = (t_i)_{i \in I}$, we can reduce using $\inf_\trianglelefteq$ the set union $((c_{i,j,\varphi})_{\varphi \in \Phi_{i,j}})_{i \in I, j \in J_i}$ of the alternative costs corresponding to each tree t_i.
- case (c1): When we rule on features that are not propagated into the right-hand side of ρ, we can $\inf_\trianglelefteq$-reduce the union $((c'_{i,j,\varphi})_{\varphi \in \Phi'_{i,j}})_{i \in I, j \in J_i}$ of the alternative costs that, on the contrary, are propagated.
- case (c2): When we construct the agreement cost $(c_x)_{x \in X_\mathsf{r}}$ of variables that are in the right-hand side of a rule, we can $\inf_\trianglelefteq$-reduce it, too.
- case (c3): Finally, when we construct the feature assignment cost $(c'_{\bar{\jmath},\varphi})_{\bar{\jmath} \in \bar{J}, \varphi \in \Phi'_0}$ for the left-hand side of a rule, we can also $\inf_\trianglelefteq$-reduce it.

These operations reduce the number of alternative combinations to consider without altering the globality of the optimum.

Although they seem *very* efficient on the kind of sentences and grammars on which we have experimented so far, these optimisations do not guarantee a polynomial time error analysis. To prevent a combinatorial explosion, the following heuristics may also (or alternatively) be used:

- optimistic heuristics: discard any alternative $\kappa = (c_\varphi)_{\varphi \in \Phi}$ such that $\min(\kappa)$ is above a given threshold,
- pessimistic heuristics: discard any alternative $\kappa = (c_\varphi)_{\varphi \in \Phi}$ such that $\max(\kappa)$ is above a given threshold,

– safe heuristics: set an upper bound on the size of any set of alternatives. (In case this upper bound is set to 1, the error analysis computes a local optimum, as opposed to a global one.)

If heuristics are used, the optimum is not global anymore, but the worst case globlal complexity becomes polynomial in the length of the sentence. Only the "safe heuristics" guarantees that a solution is always found.

5 Related Works

Our proposal substantially differs form approaches based on *mal-rules* (aka error productions), where specific, extra rules are added to the parser to treat ill-formed input [13]: ordinary parsing rules are used to detect ill-formed input and mal-rules are meta-rules applying to ordinary rules to relate the structure of ill-formed input to that of well-formed structures, possibly allowing error recovery. With our terminology, ordinary parsing rules form a positive grammar, and mal-rules are specific rules, written in the spirit of a negative grammar, to specify how to relax violated constraints in parsing the positive grammar. By contrast, in our approach, constraint relaxation is automatic, based on feature value accommodation; the linguist does not have to write any specific rule — although s/he has to design the positive grammar to control syntactic proximity.

Mal-rules also result in an explosion of the number of possible parses and raise an issue regarding performance [14]. This can only partially be dealt with using a number of extra rules, many of which are variants of existing rules, which raises another issue regarding grammar maintenance. Although we also use context-free rules augmented with features, we directly address performance with a parser that does enumerate alternatives but construct a shared forest of parse trees. Our cost evaluation also keeps under control the possible combinatorial explosion.

There are proposals to deal with ambiguities using a mal-rule scoring mechanism, the score being defined based on a layered view of stereotypical levels of language ability [15]. Our approach is more compact: it consists in providing a cost model for the plausibility of various feature assignment.

LFG parsing also has been adapted to grammar checking [16]: c-structures are built using a rich error recovery mechanism, and f-structures construction flags unification clashes but always succeeds. Like in our approach, specific rules do not have to be written: constraint relaxation is automatic. Although performance is not mentioned, this could be an issue because LFG grammaticality is NP-complete. Moreover, adding loose error recovery and making all unifications successful dramatically generate ambiguities, which is not mentioned either. We deal with these issues by using context-free rules equiped with flat feature structures, which guaranties a cubic parsing. Although the expressive power may seem poorer, it practically proves to be very flexible and powerful.

The approach in [17] is quite different: parsing amounts to eliminating impractical dependencies using graded constraints (akin to our costs), from an initial state where *all* (exponentially many) dependencies are assumed. A single structural interpretation which violates as few and as weak constraints as possible is picked up by solving a partial constraint satisfaction problem. Remaining dependencies that still violate constraints trigger errors. As for our proposal, the reported error correspond to a global optimum. The design choices regarding performance are quite different though.

6 Conclusion

This paper tackles the central problem of grammar correction and, contrary to common approaches based on negative grammars, argues in favor of a combination of both positive and negative grammars. Moreover, in order to obtain a satisfactory grammar checker, the positive grammars have to provide global and deep analyses supporting feature agreement. We have presented an easy and expressive formalism to precisely do that. The use of a lattice as entry point to our parsing allows the addition of homophones and other "similar" words. Our Earley parser on this lattice relaxes constraints if necessary and provides a shared forest, for which we showed how to compute the most plausible corrections without losing all the combinatorial advantages of the sharing – in a single pass, as opposed to systems such as [18]. We have laid out in greater detail this algorithm, as well as heuristics that control potential combinatorial explosion.

The currently implemented system can, e.g., correct (a) with (b), including the long distance agreement of "cerise" ('cherry') and "cueillis" ('picked'):

Example 4.

(a) * Les enfants ont mangé ces cerise rouge qui étaient juteuses et sucrées et qu'ils ont vu que j'avais cueillis.
The children have eaten these-⟨*plur*⟩ cherry-⟨*fem,sing*⟩ red-⟨*sing*⟩ that were-⟨*plur*⟩ juicy-⟨*fem,plur*⟩ and sweet-⟨*fem,plur*⟩ and that they have seen that I have picked-⟨*masc,plur*⟩.

(b) Les enfants ont mangé ces cerises rouges qui étaient juteuses et sucrées et qu'ils ont vu que j'avais cueillies.
The children have eaten these-⟨*plur*⟩ cherry-⟨*fem,plur*⟩ red-⟨*plur*⟩ that were-⟨*plur*⟩ juicy-⟨*fem,plur*⟩ and sweet-⟨*fem,plur*⟩ and that they have seen that I have picked-⟨*fem,plur*⟩.

(c) * Les enfants ont mangé cette cerise rouge qui était juteuse et sucrée et qu'ils ont vu que j'avais cueilli.
The children have eaten this-⟨*sing*⟩ cherry-⟨*fem,sing*⟩ red-⟨*sing*⟩ that was-⟨*sing*⟩ juicy-⟨*fem,sing*⟩ and sweet-⟨*fem,sing*⟩ and that they have seen that I have picked-⟨*masc,sing*⟩.

Only a global and deep analysis can yield this minimal correction (3 revised features). Interestingly, Kleene star and optional terms provide shared forests with few ambiguities. By comparison, Microsoft Office, that often seems to have a somewhat global view of the sentence, actually makes cascading suboptimal choices and corrects (a) into (c), which is not minimal (5 revised features), and is even syntactically wrong as the past participle in the relative phrase is not in agreement. LanguageTool does not pick up any errors in this sentence.

Work is in progress to write a larger grammar and test it on a corpus of errors to assess the quality of error corrections. A crucial extension is likely to be a support for syntactic disambiguation, or more precisely a weighting of alternative parses. We also want to test the system more thoroughly regarding performance and ease in writing grammars. Experiments with other languages are planed too, as well as the integration into OpenOffice.

References

1. Sågvall Hein, A.: A chart-based framework for grammar checking – initial studies. In: 11th Nordic Conference in Computational Linguistic. (1998) 68–80
2. Souque, A.: Vers une nouvelle approche de la correction grammaticale automatique. In: RECITAL, Avignon (June 2008) 121–130
3. Naber, D.: Integrated tools for spelling, style, and grammar checking. OpenOffice.org Conference, Barcelona (2007) http://www.languagetool.org.
4. Fontenelle, T.: Les nouveaux outils de correction linguistique de Microsoft. In: TALN conference, Louvain (April 2006) 3–19
5. Clément, L., Sagot, B., Lang, B.: Morphology based automatic acquisition of large-coverage lexica. In: LREC. (May 2004) http://alpage.inria.fr/s̃agot/lefff.html.
6. New, B.: Lexique 3 : une nouvelle base de données lexicales. In: TALN conference. (April 2006) 892–900
7. Prost, J.P.: Modélisation de la gradience syntaxique par analyse relâchée à base de contraintes. Phd thesis, Univ. de Provence et Macquarie Univ. (December 2008)
8. Vogel, C., Cooper, R.: Robust chart parsing with mildly inconsistent feature structures. Edinburh Working Papers in Cognitive Science: Nonclassical Feature Systems **10** (1995)
9. Fouvry, F.: Constraint relaxation with weighted feature structures. In: 8th International Workshop on Parsing Technologies. (2003)
10. Billot, S., Lang, B.: The structure of shared forests in ambiguous parsing. In: 27th meeting on Association for Computational Linguistics, ACL (1989) 143–151
11. Earley, J.: An efficient context-free parsing algorithm. CACM **13**(2) (1970)
12. Bustamante, F.R., León, F.S.: Gramcheck: A grammar and style checker. In: COLING. (1996) 175–181
13. Sondheimer, N.K., Weischedel, R.M.: A rule-based approach to ill-formed input. In: 8th conference on Computational linguistics, Tokyo, Japan, ACL (1980) 46–53
14. Schneider, D., McCoy, K.F.: Recognizing syntactic errors in the writing of second language learners. In: 17th international conference on Computational linguistics, Montreal, Quebec, Canada, ACL (1998) 1198–1204
15. Mccoy, K.F., Pennington, C.A., Suri, L.Z.: English error correction: A syntactic user model based on principled "mal-rul" scoring. In: 5th International Conference on User Modeling (UM), Hawaii, USA (1996) 59–66
16. Reuer, V.: Error recognition and feedback with lexical functional grammar. CALICO journal **20**(3) (2003) 497–512
17. Menzel, W., Schröder, I.: Constraint-based diagnosis for intelligent language tutoring systems. In: IT&KNOWS conf. at IFIP '98, Wien/Budapest. (1998) 484–497
18. Richardson, S.D., Braden-Harder, L.C.: The experience of developing a large-scale natural language text processing system: CRITIQUE. In: 2nd conference on Applied Natural Language Processing, Austin, Texas, ACL (1988) 195–202