Towards Robust PATR

Shona Douglas[*] and **Robert Dale**[†]
Centre for Cognitive Science
University of Edinburgh
Edinburgh EH8 9LW, Scotland

Abstract

We report on the initial stages of development of a robust parsing system, to be used as part of *The Editor's Assistant*, a program that detects and corrects textual errors and infelicities in the area of syntax and style. Our mechanism extends the standard PATR-II formalism by indexing the constraints on rules and abstracting away control of the application of these constraints. This allows independent specification of grouping and ordering of the constraints, which can improve the efficiency of processing, and in conjunction with information specifying whether constraints are necessary or optional, allows detection of syntactic errors.

Introduction

The *Editor's Assistant* [Dale 1989, 1990] is a rule-based system which assists a copy editor in massaging a text to conform to a house style. The central idea is that publishers' style rules can be maintained as rules in a knowledge base, and a special inference engine that encodes strategies for examining text can be used to apply these rules. The program then operates by interactively detecting and, where possible, offering corrections for those aspects of a text which do not conform to the rules in the knowledge base. The expert-system-like architecture makes it easy to modify the system's behaviour by adding new rules or switching rule bases for specific purposes.

Our previous work in this area has been oriented towards the checking of low-level details in text: for example, the format and punctuation of dates, numbers and numerical values; the punctuation and use of abbreviations; and the typefaces and abbreviations to be used for words from foreign languages. In this paper, we describe some recent work we have carried out in extending this mechanism to deal with syntactic errors; this has led us to a general mechanism for robust parsing which is applicable outside the context of our own work.

[*]E-mail addess is `S.Douglas@ed.ac.uk`.

[†]Also of the Department of Artificial Intelligence at the University of Edinburgh; e-mail address is `R.Dale@ed.ac.uk`.

Syntactic Errors

Categories of Errors

Ultimately, the aim of *The Editor's Assistant* is to deal with *real* language—unrestricted natural language text in all its richness, with all its idiosyncracies. The system is therefore an experiment in what we call **intelligent text processing**: an intersection of techniques from natural language processing and from more mundane text processing applications, with the intelligence being derived from the addition of language sensitivity to the basic text processing mechanisms.

Many of the corrections made routinely in the course of human proofreading require subleties of semantic and pragmatic expertise that are simply beyond current resources to emulate. However, examination of common syntactic errors and infelicities, both as described in the literature (see, for example, [Miller 1986]) and as appearing in data we have analysed, has led us to distinguish a number of tractable error types, and we have based the development of our system on the various requirements imposed by these classes. The error types are defined very much with processing requirements in mind; orthogonal categorisations are of course possible. We give summary descriptions of these classes here; examples are provided in Figure 1.

Constraint Violation Errors: These involve what, in most contemporary syntactic theories, are best viewed as the violation of constraints on feature values. All errors in agreement fall into this category.

Lexical Confusion: These involve the confusion of one lexical item with another. We specifically include in this category cases where a word containing an apostrophe is confused with a similar word that does not, or *vice versa*.

Syntactic Awkwardness: We include here cases where the problem is either stylistic or likely to cause processing problems for the reader. Note that these 'errors' are not syntactically incorrect, but are constructions which, if overused, may result in poor writing, and as such are often included in style-checker 'hit-lists'; thus, we would include multiple embedding constructions, potenti-

Constraint Violation Errors:

(1) Subject-verb number disagreement:

a. ∗John and Mary runs.
b. ∗The dogs runs.

(2) Premodifier-noun number disagreement:

a. ∗This dogs runs.
b. ∗All the dog run.

(3) Subject-complement number disagreement:

a. ∗There is five dogs here.
b. ∗There are a dog.

(4) Wrong pronoun case:

a. ∗He and me ran to the dog.
b. ∗This stays between you and I.

(5) Wrong indefinite article:

a. ∗A apple and an rotten old pear.
b. A NeXT workstation and ∗a NEC laptop.[1]

Lexical Confusion:

(6) Confusion of *its* and *it's*:

a. ∗Its late.
b. ∗The dog ate it's bone.

(7) Confusion of *there*, *their*, and *they're*:

a. ∗Their is a dog here.
b. ∗They're is a dog here.
c. ∗There dog was cold.
d. ∗They're dog was cold.
e. ∗There here now.
f. ∗Their here now.

(8) Confusion of possessive *'s* and plural *s*:

a. ∗The dog's are cold.
b. ∗The boy ate the dogs biscuit.

Syntactic Awkwardness:

(9) Too many prepositional phrases:

a. The boy gave the dog in the window at the end with the red collar with the address on the back of it a biscuit.

(10) Passive constructions:

a. The boy was seen by the dog.

Missing or extra elements:

(11) Unpaired delimiters:

a. ∗The dog, which was in the garden was quiet.

(12) Missing delimiters:

a. ∗The dog I think was in the garden.
b. ∗In the garden dogs are a menace.

(13) Missing list separators:

a. ∗There were two dogs three cats and a canary.

(14) Double syntactic function:

a. ∗It seems to be is a dog.
b. ∗I think know I've been there before.

Figure 1: Examples of Syntactic Errors

ally ambiguous syntactic structures, and garden path sentences in this category. These problems are detectable by simple counting or recognition of syntactic forms.

Missing or Extra Elements: These are cases where elements (either words or punctuation symbols) are omitted or mistakenly included in a text. An interesting sub-category here, which is surprisingly frequent, is the presence of two constituents which serve the same or a similar purpose; by analogy with double-word errors (where a word appears twice in succession when only one occurrence was intended), we refer to these as cases of **double syntactic function**.

The errors dealt with in this paper all fall into the first class, i.e, those that can be seen as breaking constraints on feature values. At the end of the paper we make some observations on how the mechanism can be extended to the other classes.

Previous Work

Of course, there exists a significant body of work dealing with computational approaches to syntactic errors like those just discussed. Broadly, work dealing with ungrammatical input falls into two categories: approaches where the principal objective is to determine what meaning the speaker intended, and approaches where the principal objective is to construct an appropriate correction. The first kind of approach is most appropriate in the development of natural language interfaces, where syntactic dysfluencies can often be ignored if the user's intentions can be determined by means of other evidence. However, these approaches (in the simplest cases, based on detecting content words) are inappropriate where the system must also propose a correction for the hypothesised error.

Of the different techniques that have been proposed under the second category, the most useful is that usually referred to as **relaxation**. This is a rather elegant method for extending a grammar's coverage to include ill-formed input, while retaining a principled connection between the constructions accepted by the more restrictive grammar and those accepted by the extended one. If a grammar expresses information in terms of constraints or conditions on features, a slightly less restrictive grammar can be constructed by **relaxing** some subset of these constraints. Work commonly referred to in this context includes Kwasny and Sondheimer [1981] and Weischedel and Black [1980], but very many systems use some kind of relaxation process, whether of syntactic or semantic constraints. The most well known is IBM's work on the Epistle and Critique systems [Heidorn et al. 1982; Jensen et al. 1983; Richardson and Braden-Harder 1988].

[1]In British English, *NEC* is spelled out, rather than being pronounced like the word *neck*; thus, the correct form here is *an NEC*.

Epistle parses text in a left-to-right, bottom-up fashion, using grammar rules written using an *augmented phrase structure grammar* (APSG). In APSG, each grammar rule looks like a conventional context-free phrase structure rule, but may have arbitrary tests and actions specified on both sides of the rule. So, for example, we might have a rule like the following:

(15) NP VP (NUMB.AGREE.NUMB(NP)) $\rightarrow$
 VP(SUBJECT = NP)

This rule states that a noun phrase followed by a verb phrase together form a VP,[2] provided the number of the NP and the original VP agree. The resulting VP structure then has the original NP as the value of its SUBJECT attribute.

Using rules like these, the system attempts to parse a sentence as if it were completely grammatical. Then, if no parse is found, the system relaxes some conditions on the rules and tries again; if a parse is now obtained, the system can hypothesise the nature of the problem on the basis of the particular condition that was relaxed. Thus, if the above rule was used in analysing the sentence *Either of the models are acceptable*, no parse would be obtained, since the number of the NP *Either of the models* is singular whereas the number of the VP *are acceptable* is plural. However, if the number agreement constraint is relaxed, a parse will be obtained; the system can then suggest that the source of the ungrammaticality is the lack of number agreement between subject and verb.

One thing that must be borne in mind when considering the merits and demerits of relaxation methods is that they depend crucially on how much of the particular grammar's information is expressed as constraints on feature values. Where the basic form of a grammar is, say, complex phrase structure rules, the use of features may be confined to checking of number and person agreement. If, on the other hand, more of the informative content of the grammar is represented as constraints, as in recently popular unification-based grammars [Sheiber 1986], relaxation can be used to transform grammars to less closely related ones.

In the remainder of this paper, we show how a unification-based formalism, PATR-II, may be extended by a declarative specification of relaxations so that it can be used flexibly for detecting syntactic errors. Under one view, what we are doing here is rationally reconstructing the Epistle system within a unification-based framework. A useful consequence of this exercise is that the adoption of a declarative approach to the specification of relaxations makes it much easier to explore different processing regimes for handling syntactic errors.

[2]This second, higher-level VP plays the role of what we would normally think of as an S node.

```
X0   → X1 X2
     ⟨X0 cat⟩       =   VP
     ⟨X1 cat⟩       =   NP
     ⟨X2 cat⟩       =   VP
     ⟨X0 subject⟩   =   X1
     ⟨X1 num⟩       =   ⟨X2 num⟩
```

Figure 2: PATR version of the Epistle rule

Making PATR Robust

The Basic Mechanism

In this section, we describe an experimental system, written in Prolog, that is designed to support the mechanisms necessary to apply PATR-type rules to solve constraints selectively. The major components of the system are (a) the parsing mechanism; (b) the underlying PATR system; and (c) the rule application mechanism that mediates between these two.

The parser encodes the chosen strategy for applying particular grammar rules in a particular order. At this stage, the parser is not a crucial component of the system; all we require is that it apply rules in a bottom-up fashion. Accordingly, we use a simple shift-reduce mechanism. The parser will be the focus for many of the proposed extensions discussed later; in particular, we are in the process of implementing a chart-based mechanism to allow handling of errors resulting from missing or extra elements.

The basic PATR system provides a unification based mechanism for solving sets of constraints on feature structures. A PATR rule corresponding to the grammar rule discussed in the context of Epistle above is shown in Figure 2.

It is fairly obvious that, given some mechanism that allows us to remove the final constraint in this rule, we can emulate the behaviour of the Epistle system. In our model, the rule application mechanism provides the interface between the parsing mechanism, which accesses the lexicon and decides the order in which to try rules, and the PATR system. To see how this works, we will consider a slightly more complex rule, shown in Figure 3; the use of the numbers on the constraints will be explained below.

Given this rule, a constituent of category NP will be found given two lexical items which are respectively a determiner and a noun, provided all the constraints numbered 1 through 6 are found to hold. Note the constraint numbered 4: we suppose that the features addressed by ⟨X1 agr precedes⟩ and ⟨X2 agr begins⟩ may have the values vowel and consonant. This allows us to specify the appropriate restrictions on the use of the two forms *a* and *an*.[3]

[3]Of course, the implication here that *a* is used before words beginning with a vowel and *an* is used before words beginning with a consonant is an oversimplification. There are also, of course, other means by which this constraint could be checked; however, we include it

```
X0  → X1 X2
    1  ⟨X0 cat⟩             =  NP
    2  ⟨X1 cat⟩             =  Det
    3  ⟨X2 cat⟩             =  N
    4  ⟨X1 agr precedes⟩    =  ⟨X2 agr begins⟩
    5  ⟨X1 agr num⟩         =  ⟨X2 agr num⟩
    6  ⟨X0 agr num⟩         =  ⟨X2 agr num⟩
```

Figure 3: Simple NP rule in the PATR formalism

Relaxing Constraints

Given the rule in Figure 3, and a standard parsing mechanism, there will be no problem in parsing correct NPs like *these dogs*. However, consider our target errors in (16a–c):

(16) a. ∗this dogs
  b. ∗an dog
  c. ∗an dogs

Example (16a) exhibits premodifier–noun number disgreement; (16b) exhibits use of the wrong indefinite article; and (16c) contains both of these errors. If the parser is to make any sense of these strings, we must introduce a more elaborate control structure.

Premodifier-noun number agreement is enforced by constraint 5; constraint 4 enforces the use of the proper indefinite article. We need to be able to relax constraint 5 to parse (16a), and to relax constraint 4 to parse (16b); to parse (16c), we want to relax both constraints 5 and 4 at once.

To deal with this, we make use of the notion of a **relaxation level**. Instead of applying all constraints associated with a rule, we specify for every rule, at any given relaxation level, those constraints that are **necessary** and those that are **optional**. At relaxation level 0, which is equivalent to the behaviour of the standard PATR system, all constraints are deemed necessary. At relaxation level 1, however, constraints 4 and 5 are optional. Optional constraints, if violated, need not result in a failed parse, but do correspond to particular errors.

The algorithm in Figure 4 applies all constraints appropriately, given a specification as just described. Here, N is the set of necessary constraints and O is the set of optional constraints, both for a given relaxation level L; R is the set of constraints which have to be relaxed in order for the rule to be used. R will always be a subset of O, of course; we return the actual value of R as a result of parsing with the rule. The outer conditional ensures that all the necessary constraints are satisfied. The inner conditional takes appropriate action for each relaxable constraint whether or not it is satisfied: if the constraint is satisfied, it has exactly the same effect as a necessary

here as a constraint on the application of the rule for expository purposes.

```
When applying rule $r$ at relaxation level $L$:
$N \leftarrow$ necessary constraints on $r$ at $L$
$O \leftarrow$ optional constraints on $r$ at $L$
$R \leftarrow \{\}$
**if** all $n \in N$ can be solved
**then** incorporate any instantiations required
    **for** each $o_i \in O$ **do**
        **if** $o_i$ can be solved
        **then** incorporate instantiations
        **else** $R \leftarrow C \cup o_i$
        **endif**
    **next**
**else return** failure
**endif**
**return** $C$
```

Figure 4: The relaxation algorithm, version 1

Relaxation level 0:
  necessary constraints = {1,2,3,4,5,6}
  optional constraints = {}

Relaxation level 1:
  necessary constraints = {1,2,3,6}
  optional constraints = {5,4}

Figure 5: The relaxation specification for the NP rule, version 1: optional constraints

Relaxation level 1:
  necessary constraints: {1,2,3}
  relaxation packages:
   (a) {5, 6}: Premodifier-noun number disagreement
   (b) {4}: *a/an* error

Figure 6: The relaxation specification for the NP rule, version 2: grouped constraints

constraint; if not, the constraint is recorded as having been relaxed.

Once parsing is complete, the information in R can then be used to generate an appropriate error message.

The operation of this algorithm is supported by explicitly indexing each constraint within a rule, as in Figure 3, and abstracting out the specification of which constraints may be relaxed at a given relaxation level. The constraint application specification for the NP rule is given in Figure 5.

Grouping Constraints

This is not the whole story, however. Consider the NP *this dogs*, which would be correctly parsed at relaxation level 1 as exhibiting premodifier-noun number disagreement under the system described so far. The instantiation of X0 resulting from this rule application would be as follows:

```
When applying rule $r$ at relaxation level $L$:
$N \leftarrow$ necessary constraints on $r$ at $L$
$O \leftarrow$ relaxation packages for $r$ at $L$
$R \leftarrow \{\}$
if all $n \in N$ can be solved
then incorporate any instantiations
    for each relaxation package $P_i \in O$ do
        if all constraints $c_i \in P_i$ can be solved
        then incorporate any instantiations
        else $R \leftarrow R + P_i$
        endif
    next
else return failure
endif
return $R$
```

Figure 7: The relaxation algorithm, version 2

$$(17)\ \mathsf{X0} = \begin{bmatrix} \mathsf{cat:}\ \mathsf{np} \\ \mathsf{agr:}\ \begin{bmatrix}\mathsf{num:}\ \mathsf{plu}\end{bmatrix} \end{bmatrix}$$

Note in particular that ⟨X0 agr num⟩ has the value plu. This results from the solution of constraint 6, which is one of the necessary constraints at relaxation level 1 as specified in Figure 5. This 'feature transport' constraint propagates the number of the head noun to the superordinate noun phrase. It is not appropriate to perform such a propagation under the current circumstances, however, because once a case of premodifier-noun number disagreement has been identified, we cannot tell whether it is the number of the noun or the number of the determiner that is in error. One might argue that one of the two is more likely than the other, but such a heuristic belongs in the mechanism that offers replacements rather than in the relaxation mechanism itself. If the number of the noun is always propagated to the noun phrase, spurious error reports may emerge in subsequent parsing: for example, in the text *This dogs runs*, a subject-verb number disagreement will be flagged in addition to the premodifier-noun number disagreement error. This will be at best misleading.

We would like to be able to express the intuition that it is not really meaningful to apply constraint 6 if constraint 5 has failed; these constraints should be grouped together, to be applied together or not at all. So we introduce an addition to the specification for relaxation level 1, shown in Figure 6.

We refer to a group of constraints to be relaxed together or not at all, plus the error message that corresponds to the failure of the group of constraints, as a **relaxation package**. The algorithm of Figure 4 has been adapted to apply such relaxation packages, resulting in the algorithm in Figure 7. Here, R is the set of relaxation packages required in order to complete the parse.

Note that if all the constraints in a relaxation package can be applied successfully, they have exactly the same effect as necessary ones, in terms of contributing to the building of structure. Thus, if the number agreement condition constraint 5 is satisfied, as in the case of the text *an dogs*, then the associated feature percolation constraint, 6, will add the feature ⟨agr num⟩ to X0, with value ⟨X2 agr num⟩.

Ordering Constraints

In the previous section, we altered the mechanism to allow for the fact that it is not meaningful to apply some constraints if others have failed; in the worst case, this avoided confusing error diagnoses. Even if no such confusion would result, however, considerable efficiency gains can be made by ordering constraints in such a way as to minimise unnecessary structure building. A similar point is made by Uszkoriet [1991], who talks of the need for a flexible control strategy for efficient unification based parsers, to ensure that the conditions that are most likely to fail are tried first.

Ideally, the ordering of constraints would be derived automatically from other information; but it is unclear how this would be done. Currently, we make use of one central ordering principle:

(18) Category constraints on RHS items come first.

In the bottom-up parsing system we use, all RHS items will be instantiated with feature structures corresponding to lexical entries, or to syntactic categories built up by rule from lexical entries; it is a discipline on our lexicon and our structure building rules that all such feature structures will have a cat feature. This means that a query about the cat value will involve no structure building. However, if, before checking the category, we were to enquire about the ⟨agr num⟩ feature, we might involve ourselves in some unnecessary structure building, because if applied to a feature structure that does not have an ⟨agr num⟩ feature, what was thought of as a conditional constraint will in fact result in structure building. For example, the constraint in (19) applied against the structure in (20) will result in the structure shown in (21); this is clearly not desirable.

(19) ⟨X1 agr num⟩ = plu

$$(20)\ \mathsf{X1} = \begin{bmatrix} \mathsf{cat:}\ \mathsf{conjunction} \\ \mathsf{lex:}\ \mathsf{and} \end{bmatrix}$$

$$(21)\ \mathsf{X1} = \begin{bmatrix} \mathsf{cat:}\ \mathsf{conjunction} \\ \mathsf{lex:}\ \mathsf{and} \\ \mathsf{agr:}\ \begin{bmatrix}\mathsf{num:}\ \mathsf{plu}\end{bmatrix} \end{bmatrix}$$

These considerations give rise to the ordering of constraints given in Figure 8; we assume that when the algorithm in Figure 7 tests whether all members of a constraint set can be solved, the constraints are solved in the order given in the specification, and the test halts as soon as any member of the constraint set cannot be solved.

Relaxation level 0:
necessary constraints: {2,3,5,4,1,6}
relaxation packages: {}
Relaxation level 1:
necessary constraints: {2,3,1}
relaxation packages:

(a) {5, 6}: Premodifier-noun number disagreement
(b) {4}: *a/an* error

Figure 8: The relaxation specification for the NP rule, version 3: constraint ordering

Discussion

We have argued that combining the relaxation technique for syntactic error correction with a grammar (such as is found in recent unification formalisms) that expresses most of its information in the form of constraints provides a good starting point for a flexible mechanism for detecting and correcting syntactic errors. Our work in this area so far raises a number of interesting questions which need to be pursued further.

Dependencies between Constraints: As we have seen, the ordering of constraints in the relaxation specifications is very important. However, the particular role a specific constraint performs will of course depend on the particular parsing strategy being used. Ideally, we would like to generate the ordering information automatically, although it is not entirely clear how this might be done. One source of some ordering constraints might come from using typed feature structures in the lexicon, so that the rule application mechanism can determine ahead of time what the primary source of information is. Another approach might be to require the grammar writer to specify the constraints on rules as belonging to specific categories, and then to allow the rule application mechanism to impose a predefined ordering between categories; in particular, the most troublesome constraints are those which transport feature values around a structure, since they may transport the wrong values, as we saw in the example discussed earlier.

Generation of Replacement Text: A topic we have not addressed in the present paper is the generation of corrections for hypothesised errors. The result of parsing using relaxation provides sufficient information to generate such replacements, but once again we need to maintain information about the dependencies between elements of a structure so that, when a new structure is created, any conflicts that arise can be resolved: for example, if generating a correction involves changing the num feature of a noun from plural to singular, we need to encode the information that the lex feature is dependent upon the num feature and some specification of the root form, so that the replacement mechanism knows which features take priority and which may be overridden.

Deciding between Error Hypotheses: When a constraint unifying two incompatible values v_1 and v_2 has to be relaxed, then in the absence of further information there are two equally likely error hypotheses: one, that v_1 is the correct value and v_2 is wrong, and the other that v_2 is correct and v_1 is wrong. However, there are two types of situation in which further information available during parsing may enable one hypothesis to be preferred. The first is where the absolute likelihood of one error seems greater than that of the other. For example, in the case of the noun phrase *these dog* it might prove to be much more likely for a writer to mistakenly omit the single letter *s* than to choose the wrong determiner, which involves a change of two letters—there may be quantifiable difference between the assumptions behind the two hypotheses. The second is where a number of possible errors are linked, for example if the whole sentence was *These dog are fierce*. Here, two possible errors involving different rules are interdependent, and once again it is possible to argue that one error hypothesis requires a quantifiably different set of assumptions; here, both *these* and *are* would have to be wrong if *dog* were to be assumed correct.

To a certain extent, it may be possible to rely on unification to deal with these conflicts. The relaxation package dealing with the noun phrase number disagreement might 'hold its fire'—not signal an error immediately—leaving the number feature of the noun phrase uninstantiated. Then there will be no clash with the number of the verb phrase, which will be propagated down to the noun phrase. It may be possible to hook this value up to the subsequent processing of the error suggestion from the noun phrase rule.

Alternatively, the idea that there are a number of assumptions behind a given error hypothesis could be formalised, perhaps by using an ATMS [de Kleer 1986a, 1986b] to keep track of inconsistencies. Hypotheses could be weighted both by their absolute likelihood and the contextual evidence (i.e., the number and weight of related errors consistent and inconsistent with the hypotheses).

Much depends on where during the parsing process errors arise and are notified, and so detailed consideration of this issue has been deferred until our chart parser extension to this system has been explored.

Levels of Relaxation: The examples we have provided have only explicitly mentioned one level of relaxation. One can imagine situations where other, further levels of relaxation are available. In particular, note that, since categorial information can be specified by means of constraints, we can also consider handling instances of words misspelled as words of other syntactic categories by means of the same mechanism; relaxing category feature constraints might be an appropriate candidate for a further level of relaxation. There is of course the question of how one decides what relaxations should be available at what levels; determining this requires more detailed statistical analysis of the frequencies of different kinds of errors.

It is also likely to be required that individual error rules, spread across a number of grammar rules, be capable of being treated as a unit, that is, switched on or off together, orthogonal to the idea of relaxation levels.

Different Kinds of Relaxation: In the foregoing, we assumed that relaxing a constraint simply meant removing it. There are other notions of constraint relaxation that could be used, of course; for example, if a constraint assigns a value to some feature, we could relax this constraint by assigning a less specific value to that feature. There may be other cases where we would want to generalise the notion of relaxation to include the possibility that a constraint could be replaced by a quite different constraint.

Conclusions and Future Work

We have described a simple extension to the PATR-II formalism which allows us to provide declarative specifications of possible relaxations on rules. This provides a good starting point for a flexible mechanism for detecting and correcting syntactic errors. One reason for this is that relaxation provides a precise and systematic way of specifying the relationship between errorful and 'correct' forms, making it easier to generate suggestions for corrections. A second reason is that the very uniform representation of linguistic information will allow flexible strategies for relaxation to be applied; this is particularly important when dealing with text that may contain unpredictable errors.

As we have shown, the mechanism described here can be applied straightforwardly to Constraint Violation Errors as described at the beginning of the paper. At the moment we have a rather *ad hoc* mechanism that deals with cases of Lexical Confusion by providing alternative lexical entries in the case of parse failure, but this needs to be integrated better with the relaxation mechanism. Cases of Stylistic Awkwardness simply require the addition of a critic that walks over the structures produced by the parser. The major focus of our current work is the replacement of the shift-reduce parser by a chart parser, to enable us to handle cases of Missing or Extra Elements.

Acknowledgements

This work was carried out as part of IED Project 1679, *The Editor's Assistant*; Douglas is supported by SERC grant GRF 35654. Much of our thinking on this topic was inspired by conversations with Pablo Romero-Mares, who constructed an early version of the parser as an MSc project.

References

R Dale [1989] Computer-based Editorial Aids. Pages 12–20 in *Recent Developments and Applications of Natural Language Understanding*, edited by Jeremy Peckham. Kogan Page, London.

R Dale [1990] A Rule-based approach to Computer-Assisted Copy Editing. *Computer Assisted Language Learning*, **2**, 59–67.

G E Heidorn, K Jensen, L A Miller, R J Byrd, and M S Chodorow [1982]] The Epistle text-critiquing system. *IBM Systems Journal*, **21**, 305–326.

J de Kleer [1986a] An Assumption-based Truth Maintenance System. *Artificial Intelligence*, **28**, 127–162.

J de Kleer [1986b] Extending the TMS. *Artificial Intelligence*, **28**, 163–196.

S C Kwasny and N K Sondheimer [1981] Relaxation Theories for Parsing Ill-Formed Input. *American Journal of Computational Linguistics*, **7**, 99–108.

K Jensen, G E Heidorn, L A Miller, and Y Ravin [1983] Parse fitting and prose fixing: getting a hold on ill-formedness. *American Journal of Computational Linguistics*, **9**, 147–160.

L A Miller [1986] Computers for Composition: A Stage Model Approach to Helping. *Visible Language*, **XX**(2), 188–218.

S D Richardson and L C Braden-Harder [1988] The Experience of Developing a Large-Scale Natural Language Text Processing System: CRITIQUE. In *Proceedings of the 2nd Applied Natural Language Processing Conference*, pp195–202.

S M Shieber [1986] *An Introduction to Unification-based Approaches to Grammar.* The University of Chicago Press, Chicago, Illinois.

H Uszkoreit [1990] Strategies for Adding Control Information to Declarative Grammars. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pp237–245.

R M Weischedel and J E Black [1980] Responding Intelligently to Unparsable Inputs. *American Journal of Computational Linguistics*, **6**, 87–109.