ScarCheck – a Software for Word and Grammar Checking

Per Starbäck

1 Introduction

ScarCheck is an extension to the chart parser UCP (Uppsala Chart Parser) (Carlsson, 1981; Sågvall Hein, 1983, 1987).

UCP creates a chart, given a string[1] and a grammar as input. Each edge in the chart has an associated feature structure of features with values. If there is an edge spanning the whole chart the feature structure of that edge is the result of the parse, but if there is no such edge the parse failed, and there is no result.

In ScarCheck the aim is not to return parses but to find and isolate errors in the text. Therefore the grammatical rules are extended so that they describe the errors they find, and a special program traverses the chart afterwards, reporting on some of the error descriptions.

The errors are described with a feature ERR that technically works just like normal grammatical features like NUMBER or SUBJECT, so no special mechanism is needed for this. Also the grammatical rules are written more bottom-up than usual, since it may be fruitful to find an ill-formed phrase even if the grammar can't handle the whole larger unit that phrase occurs in.

This way, there will be useful information in smaller edges even when there is no edge spanning the whole chart. The component for traversing the chart, looking for the errors, and reporting on them is called ReportChart.

[1] Earlier the input string was always a string of characters. In the current version it can be a sequence of syntactic codes instead. ScarCheck is available in three versions, using characters (Sågvall Hein and Starbäck, this issue) or syntactic codes (Sågvall Hein, this issue) in UCP2 (implemented in Common Lisp), or using syntactic codes (only) in the newer UCP light (implemented in C) (Weijnitz, this issue).

2 Only Top-Level Errors

Only edges that have a feature ERR themselves are regarded as error edges, not those that have ERR at a lower level in the feature structure.

This is important to notice for grammar writers. For example a rule for prepositional phrases usually would just set two features of that phrase to the feature structures of the preposition and the noun phrase, respectively. Then the resulting prepositional phrase *wouldn't* be an error edge, even if the included noun phrase is an error edge, and the prepositional phrase thus actually contains an error! The grammar must take care of this either by propagating errors to larger units, or by not allowing building-blocks that are error edges.

The rationale for this is to give maximum flexibility to grammar writers, since sometimes it may be clear when constructing a larger unit that what looked like an error really wasn't. (Of course this policy also makes the work of ReportChart much easier, but instead the grammar has to do more work, so that is no net win.)

3 Traversal Principles

Every edge with an error description in the chart doesn't correspond to an actual error in the text, so there needs to be a strategy for determining which of them to report on.

3.1 The Right-Before-Wrong Principle

If there are several edges spanning over the same text, none of the errors in those edges should be reported if at least one of those edges doesn't contain an error.

As an example "the fish swim" might get two parses. One with singular "fish", and one with plural "fish". The first one might generate a congruence error, but the second not, so no error should be reported. (None of the examples given here are actual examples for which error grammars are constructed, since currently only grammars for Swedish have been written for ScarCheck.)

3.2 The Longest-Span Principle

There may be error edges spanning over something that looks like a phrase with an error, where actually those words don't belong together in a phrase,

which will be seen by other edges.

As an example, when "two sea horses" is parsed an error edge might be generated for "two sea", stating that the noun should be plural. But since there is also a longer edge spanning all three words, no error should be reported.

3.3 The Left-To-Right Principle

The chart is scanned from left to right. At each point only edges starting at the current vertex are considered. When the longest edge has been chosen, the new search will be made from where that edge ended. So if the only edges are an edge E_1 from vertex 1 to 3 and an edge E_1 from 2 to 4, an error will be reported if E_1 is an error edge, but not if E_2 is, but E_1 isn't.

If E_1 and E_2 both are error edges, only one of them (E_1) is reported. The reason left-to-right is used instead of right-to-left is of course because this corresponds to the order the text is written and read.

4 Algorithm

More exactly it works like this:

1. Set V to the first vertex.
2. If V is the last vertex we are done.
3. If there are no edges starting at V, advance V to the next vertex and go back to step 2.
4. Collect all the longest edges starting at vertex V
5. Set V to the end vertex of those edges.
6. If all those edges are error edges, report on those errors. [2]
7. Go back to step 2.

[2] Actually, ScarCheck currently only reports on one of those edges, which seems to give nicer results.

References

Mats Carlsson. Uppsala chart parser 2: System documentation. Technical Report UCDL-R-81-1, Uppsala University. Center for Computational Linguistics, 1981.

Anna Sågvall Hein. A parser for Swedish. status report for sve.ucp. Technical Report UCDL-R-83-2, Uppsala University. Center for Computational Linguistics, 1983.

Anna Sågvall Hein. Parsing by means of Uppsala chart processor (UCP). In Leonard Bolc, editor, *Natural Language Parsing Systems*, pages 203–266. Springer, Berlin, Heidelberg, 1987.

Anna Sågvall Hein. A specification of the required grammar checking machinery. *Working Papers in Computational Linguistics & Language Engineering*, this issue.

Anna Sågvall Hein and Per Starbäck. A test version of the grammar checker for Swedish, deliverable 6.5.1. *Working Papers in Computational Linguistics & Language Engineering*, this issue.

Per Weijnitz. Uppsala chart parser light system documentation. *Working Papers in Computational Linguistics & Language Engineering* , this issue.