Natural Language Engineering **10** (1): 91–94. © 2004 Cambridge University Press
DOI: 10.1017/S1351324903003309 Printed in the United Kingdom

Industry watch

ROBERT DALE
Centre for Language Technology, Macquarie University, Sydney, Australia

(*Received 4 December 2003*)

If you wanted to make some money out of natural language processing, an appropriate strategy might be to identify an area of technology that was relatively mature—one where the more fundamental technical problems had been resolved through a significant amount of research activity—and then identify potential applications for that technology.

You might adopt a 'research activity bell-curve' model of identifying technology maturity: see whether the number of published research papers in a specific area has peaked and is now sloping off. On that basis, a technology that might be ripe for commercialisation would be parsing technology. This is a very well-explored area theoretically, and we now have a very well-established body of techniques that can be pressed into service.

Of course, one technology doesn't make an application. Anyone who teaches an introductory NLP class will have spent time explaining to students that you can't do much with parsing technology on its own. The output of a parser serves as fodder for the next stage of an application. It's just one component amongst many, and most conventional language technology applications will also require some semantic analysis before they can do anything useful. But, provided you believe in the autonomy of syntax, there is one application for which parsing would appear to be the only language technology you need: grammar checking.

Grammar checking technology goes back, in one form or another, at least to the late 1970s, when the Unix Writer's Workbench tools were developed (Macdonald 1983). That early system used simple pattern matching to detect a range of basic grammatical and stylistic errors; but the same technology formed the basis of a number of commercial products over the next 10 years, perhaps the most well known of these being the early incarnations of Grammatik.[1] There are still other products on the market today that appear to be based on this technology.

It doesn't take much reflection to see that simple pattern-matching approaches quickly reach their limits when you try to apply them to more complex errors. But if you thought there might be an opening here where you could apply all that stuff you know about syntax and parsing, you'd be too late. George Heidorn and Karen Jensen were already applying parsing techniques to the grammar checking problem in the early 1980s, resulting in IBM's Critique system (Heidorn et al 1982). Heidorn

[1] The more recent versions that have surfaced insider Corel's WordPerfect appear to be more sophisticated.

and Jensen subsequently moved to Microsoft and led the development of the Word grammar checker, first released as part of Word 97.[2] Damn, another window of opportunity missed.

But you've invested a lot of your career in parsing, and you're not one to give up easily. So, where else might you try to deploy grammar checking technology to commercial advantage? You might consider, for example, focussing on the needs of specific groups, such as those who want or need to learn a language. Or you might try to build a grammar checker for a language that doesn't have one. Again, not surprisingly, there are already active players on both of these fronts. Indeed, an apparently popular strategy is to create a grammar checker for a language not already covered in Word and then to sell or licence this technology to Microsoft. Given Word's international penetration, that's a good way to reach the largest possible user base: there are around 400 million legitimate copies of Microsoft Office out there, and probably three or four times that number of copies when you take software piracy into account.

Another direction you might pursue is to develop grammar checking capabilities for other platforms. We might take 'platform' here to mean 'application', but that's unlikely to get you very far. On the Windows desktop, there'd be little point in trying to develop another word processor just to sell your neat grammar checking ideas; the two leaders, Word and WordPerfect, are too entrenched, and customers are unlikely to switch just because of the grammar checker. Other standard office applications from the same vendors can just use the same components, so there's also little point in developing a grammar checker for PowerPoint. Or you might consider 'platform' in the sense of operating system: last time I looked, Star Office didn't have a grammar checker, so there is an opportunity there: Linux users are waiting for your contribution. With a broader interpretation of 'platform', there might also be possibilities on other devices. For example, the inevitable slowness of handwriting in pen interfaces makes it sensible to consider functionalities like automatic word completion, which would be far less useful on the desktop. Maybe new devices also throw up possibilities for grammar checking that aren't so obvious on the desktop?

But these don't seem like killer apps. So you might consider the more obvious route to competing with products in the existing marketplace: build something that is faster, cheaper, or better. Faster or cheaper won't work in this case, since the speed of current grammar checkers seems quite fast enough, and few people will pay you for something that is part of something they've paid for already.

But better might work. You don't have to look hard to find complaints about existing grammar checking technology; the web abounds with pieces on the flaws of grammar checkers. More often than not, these are thinly disguised anti-Microsoft rants, and they're often more about user interface issues than the underlying technology. People who are addicted to passive sentences seem to get particularly upset. Not surprisingly, some marketing pitches play on the street perception of grammar checkers. For example, StyleWriter, which claims not to be a grammar checker,

[2] See (Heidorn 2000) for an interesting inside look at the development of the Word grammar checker.

includes in its web pages the statement 'we all know how annoying grammar checkers are to use'. There are more serious critical pieces; for example, Daniel Kies' *Modern English Grammar* page considers the results of running instances of the 20 most frequent errors in English through a number of grammar checkers, with less than encouraging results;[3] but such empirical analyses are relatively rare.

A digression: around 15 years ago—this was before the time of Word's grammar checker, and in the era of the simple pattern-matching checkers—I recall giving a talk at a conference on computers and writing, in which I heavily criticised the grammar checkers of the day as being more than slightly brain-dead. I bolstered my arguments with a wide range of examples where grammar checkers would give erroneous advice or fail to catch egregious errors. At the end of the talk, someone who taught writing to engineering students at a further education college came up to me expressing concern. For years he'd been using the very tools I had been criticising, and he was of the view that, despite their acknowledged limitations, they resulted in a significant improvement in the writing of those students. I'd only started teaching at that point and hadn't a real idea of how awful students' writing could be. I've marked enough student essays in the intervening years to understand where he was coming from.

The moral of that little story is that, if you're reading this journal, you're probably better educated than the average member of the population, and you can probably write reasonably well. Your writing is probably sufficiently good that, more often than not, the errors the grammar checker identifies will be false positives. But there are plenty of people who can and do benefit from the current abilities of the technology. The point is that the grammar checker wasn't designed for literate language technologists. Tools like Word's grammar checker represent a substantial feat of technology transfer and engineering—we're talking here about a parser that you can throw absolutely any sentence at without it falling over[4]—and they provide a valuable service to many people.[5]

Nonetheless, it's clear that the current grammar checking technology out there in the marketplace is far from perfect, and there's lots of room for improvement. Microsoft knows that, of course, but there's no business case to support the development of improvements, so the technology has remained relatively static for a number of years. We're left with a technology base that can address a wide range of simpler grammatical errors, but leaves a number of aspects of writing support untouched. This is particularly the case for 'grammatical' errors that are more contextual in nature, requiring that attention be paid to the subtleties of lexico-semantics, rhetorical structure and discourse analysis. My guess is that people won't upgrade

[3] See http://papyr.com/hypertextbooks/engl_126/gramchek.htm.

[4] Well, almost. I recall, but can no longer track down, an article on ZDNet or something similar a few years ago that reported that too many prepositions or conjunctions in one sentence could cause Word to crash. If you have a pointer to this piece, I'd be very pleased to receive it.

[5] At this point, I should make it clear that I used to work for Microsoft—although not on the grammar checker—and that I still own shares in the company; but don't worry, this text has been run through a soon-to-be-released bias-checking program, and received a high objectivity rating.

their grammar checkers until support is provided at these broader levels. Now, there is a fair bit of research out there that might play a role in developing such tools. But to make use of it you need two things: first, you need the syntactic substratum that a truly broad coverage parser can provide; and you need that substratum provided in an interactive environment so that you can build writing advisory tools around it. These infrastructural elements are very costly to build.

And why *should* you build them? If you're a Microsoft Office user, you already have them on your desktop, inside Word. The only problem, of course, is that you can't get at them; there's no API that lets you get at the results of the parser, and Microsoft has no current plans to make an API available. If there were such an API, we could expect a wide range of experiments built on top of it, and it's quite likely that some really smart tools for writing assistance would be developed. Some life would be injected into the moribund grammar checking market. But without the API, there's no point in trying: any innovation you might want to build is crippled without this infrastructure.

What are the lessons of all of this? There are at least four. First, if you really want to use the research activity bell-curve model for identifying commercialisable technologies, you still won't be able to avoid a bit of crystal ball gazing. If you wait for the peak to pass, you'll be too late. Second, stop complaining, learn how to switch off the specific grammar checking features you don't like, and appreciate the technology for what it is. Third, choose your target natural language well, so as to avoid head-on competition with the really big guys. Fourth, next time you have dinner with some Microsoft NLPers at a conference dinner, remind them that you'd really like to have access to an API for the parser that's inside Word.

If you have views on any of the above, drop a note to `rdale@acm.org`, and I'll follow up in a future column.

References

Heidorn, G., Jensen, K., Miller, L., Bird, R. and Chodorow, W. (1982) The EPISTLE Text Critiquing System. *IBM Systems Journal*, **21**(3), 305–326.

Heidorn, G. (2000) Intelligent writing assistance. In *A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text*, R. Dale, H. Moisl and H. Somers (Eds.). New York: Marcel Dekker.

Macdonald, N. H. (1983) The UNIX Writer's Workbench Software: Rationale and Design. *Bell System Technical Journal*, **62**(6), 1891–1908. Reprinted in *Plain Language: Principles and Practice*, Erwin Steinberg (Ed.). Detroit: Wayne State University Press (1991).