Natural language analysis with DCG; extended with abduction and assumptions

Comments on the recommended reading, KIIS course, autumn 2005

Henning Christiansen
Roskilde University, Computer Science Dept.
Denmark

1 Definite clause grammars

Most Prolog systems include the popular Definite clause grammar notation, colloquially DCG, which makes it very easy to implement simple language analyzers. DCGs have been used for fragments of natural language as well as programming languages, mostly for analysis and translation, but it is interesting to notice that that they can be used also for *generation* of text. However, in the course we are mostly interested in analysis.

You should be aware that DCGs only take care of part of what normally is expected for a language processing system. There is no straightforward way to have a text file analyzed; if that is what you want, you need to do a lot of programming yourself. A DCG works on *Prolog lists* which in principle can contain arbitrary elements, but in typical applications constant symbols (in Prolog terminology: atoms). For example, if you have written a little DCG for simple sentences, having a nonterminal called `sentence`, the following example query may show how you can activate analysis.

```
?- phrase(sentence,[logic,programming,is,fun]).
yes
```

(Here we used the SICStus convention using the `phrase` predicate; some systems may do it in another way.)

Most textbooks on Prolog has a chapter on DCG, and in this course we use chapters 7 and 8 of [1] which is available online. It is suggested that you skip section 7.2.3 and 8.1.3.

Comments on the text

Many introductions to DCG, including chapter 7 of [1], tend to explain DCGs in terms of their implementation which may not be that smart from

a pedagogical point of view. A better way (in the present writers view) will be to explain DCGs in terms of example grammars and then leaving the implementation principles to an appendix.

In [1], the authors start explaining that you can do language analysis by means of the `append` predicate, then they explain you that this is not a good idea. Next they explain s technique called difference lists and conclude that this is a better way to implement language analysis (difference lists *are* smart, but it may be difficult for the novice to see this). Finally, they show you the grammar notation. At the KIIS course, we are interested in the applications of DCGs and very little in the implementation. However, [1] is fairly easy to read, so it is suggested that you read the chapters 7 and 8 from the beginning. In fact, the only things you really need to care about the concerning the implementation of DCGs are the following:

- A top-down, left-to-right parsing strategy is used.
- Backtracking is applied in order to find the rules that should be applied.
- In case of an ambiguous grammar, you'll get the alternative analyses by backtracking (e.g., typing semicolon).
- If your grammar rules are left-recursive, it will most likely run into infinite loops.

2 Discourse analysis as abduction

We refer here to two recent research papers [3, 2] which may be a bit difficult to read, and you'll most like not understand everything.

A programming language called HYPROLOG is introduced in [2] which is based on CHR and Prolog. It includes abduction implemented in the way we have seen already in the course, and it includes also a related notion of assumptions which are useful for language analysis; small examples and references are given in the paper. Most importantly, the paper indicates how abduction and assumptions can be applied together with the DCG notation.

Although [2] claims that HYPROLOG is available on the web, this is not exactly the case at the moment of writing. However, if you use only the abduction part, you can implement it as we have seen earlier in the course: just declare the abducibles as constraints of CHR,

The second paper, [3], entitled "Meaning in Context" gives a theoretical account on this way of using abduction for language analysis. It may be a bit tough to read if you are not used to this sort of theoretical papers, but try anyhow — it's not that complicated after all. This paper refers to a language called A^2LP, which is a forerunner of HYPROLOG (basically the same!).

References

[1] Patrick Blackburn, Johan Bos, and Kristina Striegnitz. Learn Prolog Now!, 2005. Online document, `http://www.coli.uni-saarland.de/~kris/learn-prolog-now/`; also available as pdf, `http://www.coli.uni-saarland.de/~kris/learn-prolog-now/html/prolog-notes.pdf`

[2] H. Christiansen and V. Dahl. HYPROLOG: a new approach to logic programming with assumptions and abduction. In Maurizio Gabbrielli and Gopal Gupta, editors, *Proceedings of Twenty First International Conference on Logic Programming (ICLP 2005)*, Lecture Notes in Computer Science, 2005. To appear.

[3] H. Christiansen and V. Dahl. Meaning in Context. In Anind Dey, Boicho Kokinov, David Leake, and Roy Turner, editors, *Proceedings of Fifth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT-05)*, volume 3554 of *Lecture Notes in Artificial Intelligence*, pages 97–111, 2005.