Course on Artificial Intelligence and Intelligent Systems

Natural language analysis with DCG and CHR:

Examples and exercises

Henning Christiansen
Roskilde University, Computer Science Dept.

PRELIMINARY VERSION; EXERCISES TO BE ADDED BY 7 OCT

This note gives some examples on the recommended literature, and gives some examples of grammars and exercises. Chapters 7 and 8, minus 7.2.3 and 8.1.3 of the text [1], gives an introduction to Definite Clause Grammars; we will not repeat the basics here.

This note adds a few comments, examples and exercises.

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 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 they are 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.

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 a 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 A first DCG

The following program is copied from [1] and shows a DCG without extra arguments. Available on file `dcg1`.

```
np  --> det,n.

vp  --> v,np.
vp  --> v.
```

```
det --> [the].
det --> [a].

n   --> [woman].
n   --> [man].

v   --> [shoots].
```

3 Adding extra arguments

The following program is copied from [1] and shows a DCG with extra arguments in order do describe how `he-him` and `she-her` should be applied correctly. Available on file `dcg2`.

```
s      --> np(subject),vp.

np(_) --> det,n.
np(X) --> pro(X).

vp     --> v,np(object).
vp     --> v.

det    --> [the].
det    --> [a].

n      --> [woman].
n      --> [man].

pro(subject) --> [he].
pro(subject) --> [she].
pro(object)  --> [him].
pro(object)  --> [her].

v --> [shoots].
```

Exercise 1

The grammar should be extended so that it includes plural-singular distinction. The grammar above has only singular, so to make it interesting, add the following:

- nouns `men`, `women`,
- pronouns `they, them`,

- verb `shoot`.

You will need to add an extra argument to some of the nonterminals to hold the number which should be one of `sing`, `plu` (for singular and plural).

Write the grammar in such a way that it only accepts grammatical correct sentences such as "the women shoot the men", "she shoots the man", but not "she shoot the man".

Implement it on the computer and test it.

Exercise 2

Extend the grammar so that it will accept sentences such as the following.

> *He shoots her with a water pistol.*
>
> *A woman shoots him with a gun.*

The linguists call such addition *"with a water pistol"* a prepositional phrase, where *"with"* is the preposition.

It is suggested that you write the grammar so that you can accept any `np` after `with`. So, for example, the following sentence will be syntactically legal although semantically problematic.

> *The water pistol shoots the gun with a woman.*

4 Adding extra conditions

The following grammar describes the command language for a simplistic robot that can walk along a straight line. The grammar uses the curly-bracket notation in order to assign a "meaning" to a command sequence, which is the final position of the robot (assuming that it starts at position `0`).

Curly brackets are useful for conditions that cannot be expressed in an easy way using unification of arguments only.

The grammar is available at file `trip`. Notice that there are two nonterminals called `trip`, but they differ in the number of arguments.

```
trip(To) --> trip(0,To).

trip(Here,Here) --> [].

trip(From,To) -->
   step(HowMuch),
   {NewFrom is From + HowMuch},
   trip(NewFrom,To).
```

```
step(1) --> [forward].

step(-1) --> [back].

step(0) --> [think].
```

The following shows a query and answer for this grammar.

```
?- phrase(trip(N), [forward,forward,think,back,think,forward,forward]).
N = 3
```

Exercise 3

Calculate by hand how the Prolog system got to this answer in the example above.

Exercise 4

The robot in the example above is now modified so that it walks in a 2D space. Extend the command language in the previous example with commands `up` and `down`, and (using two arguments, one for x and one for y coordinates) modify the grammar so that it determines the final position in the 2D space.

5 Discourse analysis as abduction with CHR

As we have seen, CHR adds a global store to Prolog's execution state that we can use for abductive reasoning. Recalling that a DCG is just a special kind of Prolog program, we can apply the same principle for natural language analysis. We consider the text to be analyzed as observable and the explanations produced as an interpretation of the text. Consider an over-simplistic example.

```
:- chr_constraint problem/0.
sentence --> [help], {problem}.
```

The rule reads, that `problem` must be true for `help` to be produced as an acceptable sentence. The other way round, we can express the same by saying that the sentence `help` has an explanation, namely `problem` (and in this very little grammar, this is the only explanation).

A *discourse* means a longer sequence of sentences that is considered as a whole, and an interpretation is then a representation of the knowledge contained in that discourse. Discourse analysis differs from single-sentence analysis since the interpretation of one sentence may depend on the interpretation of previous sentences; we may also refer to this interpretation as the *context* for the discourse. As we will see, CHR is well suited for describingsuch context dependencies.

The formulation of discourse interpretation as abduction was expressed in [6], and the application of CHR for this purpose was described by [5, 2, 3].

Example

We show first a simple abductive DCG in which uses CHR to collect the meaning of the different sentences, but not yet with any CHR rules. It is available at the course website as a file name `discourse1`.

```
:- use_module(library(chr)).

:- chr_constraint at/2, see/2.

story --> [] ; s, ['.'], story.

s --> np(X), [sees], np(Y), {see(X,Y)}.
s --> np(X), [is,at], np(E), {at(E,X)}.
s --> np(X), [is,on,vacation], {at(vacation,X)}.

np(peter)    --> [peter].
np(mary)     --> [mary].
np(jane)     --> [jane].
np(ruc)  --> [ruc].
np(kiis) --> [the,kiis,course].
```

Exercise 5

Consider the following queries to the grammar shown above above.

```
phrase(story, [peter,is,at,ruc,'.']).

phrase(story, [mary,is,at,the,kiis,course,'.']).

phrase(story, [peter,sees,mary,'.']).

phrase(story, [peter,sees,mary,'.', peter,sees,jane,'.',
               peter,is,at,ruc,'.', mary,is,at,the,kiis,course, '.',
               jane,is,on,vacation,'.']).
```

Predict the answers you would expect for each of them, and explain what these answers mean intuitively. Then test the queries on the computer.

Exercise 6

MODIFY THE EXAMPLE!!!

Example with context dependent interpretation

The following abducible DCH extends the previous one with CHR rules that describe context dependencies.[1] It is available at the course website as a file name `discourse1`.

```
:- use_module(library(chr)).

:- chr_constraint at/2, in/2, see/2, skypes/2.

at(ruc,X) ==> in(roskilde,X).

in(Loc1,X) \ in(Loc2,X) <=> Loc1=Loc2.

at(kiis,X) ==> at(ruc,X).

at(vacation,X) ==> in(Loc,X), diff(Loc,roskilde).

see(X,Y) ==> true | (in(L,X), in(L,Y)
       ; in(Lx,X), in(Ly,Y), diff(Lx,Ly), skypes(X,Y)).

story --> [] ; s, ['.'], story.
s --> np(X), [sees], np(Y), {see(X,Y)}.
s --> np(X), [is,at], np(E), {at(E,X)}.
s --> np(X), [is,on,vacation], {at(vacation,X)}.

np(peter)    --> [peter].
np(mary)     --> [mary].
np(jane)     --> [jane].
np(ruc)  --> [ruc].
np(kiis) --> [the,kiis,course].
```

Exercise 7

Explain the intuitive meaning of each of the CHR rules in the grammar above. Consider again the queries of exercise 5, but now in relation to the

[1] The predicate `diff/2` is an alternative version of SICStus Prolog's built-in `dif/2`; it basically does the same thing, but when used together with other CHR rules, in gives a nicer looking output. It can be implemented as follows; the `?=` predicates is a special sicstus device.

```
:- chr_constraint diff/2.
diff(X,X) <=> fail.
diff(A,B) \ diff(A,B) <=> true.
diff(A,B) \ diff(B,A) <=> true.
diff(A,B) <=> ?=(A,B) | true.
```

new grammar; explain what do they mean intuitively and test them on the computer.

Exercise 8

MODIFY THE EXAMPLE!!!

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.

[4] Henning Christiansen. User's guide to the HYPROLOG system: A logic programming language with assumptions and abduction. `http://www.ruc.dk/~henning/hyprolog/HyprologUsersGuide.html`, 2005.

[5] Henning Christiansen and Veronica Dahl. Assumptions and abduction in Prolog. In Elvira Albert, Michael Hanus, Petra Hofstedt, and Peter Van Roy, editors, *3rd International Workshop on Multiparadigm Constraint Programming Languages, MultiCPL'04; At the 20th International Conference on Logic Programming, ICLP'04 Saint-Malo, France, 6-10 September, 2004*, pages 87–101, 2004.

[6] Jerry R. Hobbs, Mark E. Stickel, Douglas E. Appelt, and Paul A. Martin. Interpretation as abduction. *Artificial Intelligence*, 63(1-2):69–142, 1993.