Lecture 8: More DCGs

• Theory
  – Examine two important capabilities offered by DCG notation:
    • Extra arguments
    • Extra tests
  – Discuss the status and limitations of DCGs

• Exercises
  – Exercises of LPN: 8.1, 8.2
  – Practical session
Extra arguments

• In the previous lecture we introduced basic DCG notation
• But DCGs offer more than we have seen so far
  – DCGs allow us to specify **extra arguments**
  – These extra arguments can be used for many purposes
Extending the grammar

• This is the simple grammar from the previous lecture

• Suppose we also want to deal with sentences containing pronouns such as

  she shoots him

  and

  he shoots her

• What do we need to do?

s --> np, vp.
np --> det, n.
vp --> v, np.
vp --> v.
det --> [the].
det --> [a].
n --> [woman].
n --> [man].
v --> [shoots].
Extending the grammar

- Add rules for pronouns
- Add a rule saying that noun phrases can be pronouns

- Is this new DCG any good?
- What is the problem?

```
s  -->  np, vp.
np  -->  det, n.
np  -->  pro.
vp  -->  v, np.
vp  -->  v.
det  -->  [the].
det  -->  [a].
n  -->  [woman].
n  -->  [man].
v  -->  [shoots].
pro  -->  [he].
pro  -->  [she].
pro  -->  [him].
pro  -->  [her].
```
Some examples of grammatical strings accepted by this DCG

?- s([she,shoots,him],[ ]).
yes

?- s([a,woman,shoots,him],[ ]).
yes

s --> np, vp.
np --> det, n.
np --> pro.
vp --> v, np.
vp --> v.
det --> [the].
det --> [a].
n --> [woman].
n --> [man].
v --> [shoots].
pro --> [he].
pro --> [she].
pro --> [him].
pro --> [her].
Some examples of ungrammatical strings accepted by this DCG

?- s([a,woman,shoots,he],[ ]).
yes
?- s([her,shoots,a,man],[ ]).
yes
s([her,shoots,she],[ ]).  
yes

s --> np, vp.
np --> det, n.
np --> pro.
vp --> v, np.
vp --> v.
det --> [the].
det --> [a].
n --> [woman].
n --> [man].
v --> [shoots].
pro --> [he].
pro --> [she].
pro --> [him].
pro --> [her].
What is going wrong?

- The DCG ignores some basic facts about English
  - *she* and *he* are **subject pronouns** and cannot be used in object position
  - *her* and *him* are **object pronouns** and cannot be used in subject position
- It is obvious what we need to do: extend the DCG with information about subject and object
- How do we do this?
A naïve way...

s --> np_subject, vp.
np_subject --> det, n.  np_object --> det, n.
np_subject --> pro_subject.  np_object --> pro_object.
vp --> v, np_object.
vp --> v.
det --> [the].
det --> [a].
n --> [woman].
n --> [man].
v --> [shoots].
pro_subject --> [he].
pro_subject --> [she].
pro_object --> [him].
pro_object --> [her].
Nice way using extra arguments

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].
v --> [shoots].
pro(subject) --> [he].
pro(subject) --> [she].
pro(object) --> [him].
pro(object) --> [her].
This works...

\[
\begin{align*}
\text{s} & \rightarrow \text{np(subject)}, \text{vp}. \\
\text{np}(\_\_\_\_\_) & \rightarrow \text{det, n.} \\
\text{np}(X) & \rightarrow \text{pro}(X). \\
\text{vp} & \rightarrow \text{v, np(object)}. \\
\text{vp} & \rightarrow \text{v}. \\
\text{det} & \rightarrow [\text{the}]. \\
\text{det} & \rightarrow [\text{a}]. \\
\text{n} & \rightarrow [\text{woman}]. \\
\text{n} & \rightarrow [\text{man}]. \\
\text{v} & \rightarrow [\text{shoots}]. \\
\text{pro(subject)} & \rightarrow [\text{he}]. \\
\text{pro(subject)} & \rightarrow [\text{she}]. \\
\text{pro(object)} & \rightarrow [\text{him}]. \\
\text{pro(object)} & \rightarrow [\text{her}].
\end{align*}
\]

?- s([she,shoots,him],[ ]).
   yes
?- s([she,shoots,he],[ ]).
   no
?-
What is really going on?

- Recall that the rule:
  \[ s \rightarrow \text{np, vp}. \]
  is really syntactic sugar for:
  \[ s(A,B) :\text{- np}(A,C), \text{ vp}(C,B). \]
What is really going on?

• Recall that the rule:
  \[ s \rightarrow \text{np,vp}. \]
  is really syntactic sugar for:
  \[ s(A,B) :\text{np}(A,C), \text{vp}(C,B). \]

• The rule
  \[ s \rightarrow \text{np}(\text{subject}),\text{vp}. \]
  translates into:
  \[ s(A,B) :\text{np}(\text{subject},A,C), \text{vp}(C,B). \]
## Listing noun phrases

<table>
<thead>
<tr>
<th>s</th>
<th>np(subject), vp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>np( _)</td>
<td>det, n.</td>
</tr>
<tr>
<td>np(X)</td>
<td>pro(X).</td>
</tr>
<tr>
<td>vp</td>
<td>v, np(object).</td>
</tr>
<tr>
<td>vp</td>
<td>v.</td>
</tr>
<tr>
<td>det</td>
<td>[the].</td>
</tr>
<tr>
<td>det</td>
<td>[a].</td>
</tr>
<tr>
<td>n</td>
<td>[woman].</td>
</tr>
<tr>
<td>n</td>
<td>[man].</td>
</tr>
<tr>
<td>v</td>
<td>[shoots].</td>
</tr>
<tr>
<td>pro(subject)</td>
<td>[he].</td>
</tr>
<tr>
<td>pro(subject)</td>
<td>[she].</td>
</tr>
<tr>
<td>pro(object)</td>
<td>[him].</td>
</tr>
<tr>
<td>pro(object)</td>
<td>[her].</td>
</tr>
</tbody>
</table>

?- np(Type, NP, [ ]).
Type = _
NP = [the, woman];

Type = _
NP = [the, man];

Type = _
NP = [a, woman];

Type = _
NP = [a, man];

Type = subject
NP = [he]
Building parse trees

- The programs we have discussed so far have been able to recognise grammatical structure of sentences.
- But we would also like to have a program that gives us an analysis of their structure.
- In particular we would like to see the trees the grammar assigns to sentences.
Parse tree example

```
s
  vp
    np
      det  n  v  det  n
      the  woman  shoots  a  man
```
Parse tree in Prolog

\[
\text{s(np(det(the),n(woman))), vp(v(shoots),np(det(a),n(man)))))}
\]
DCG that builds parse tree

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].
v → [shoots].
pro(subject) → [he].
pro(subject) → [she].
pro(object) → [him].
pro(object) → [her].
DCG that builds parse tree

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

s(s(NP,VP)) --> np(subject,NP), vp(VP).
np(_,np(Det,N)) --> det(Det), n(N).
np(X,np(Pro)) --> pro(X,Pro).
vp(vp(V,NP)) --> v(V), np(object,NP).
vp(vp(V)) --> v(V)).
det(det(the)) --> [the].
det(det(a)) --> [a].
n(n(woman)) --> [woman].
n(n(man)) --> [man].
v(v(shoots)) --> [shoots].
pro(subject,pro(he)) --> [he].
pro(subject,pro(she)) --> [she].
pro(object,pro(him)) --> [him].
pro(object,pro(her)) --> [her].
Generating parse trees

\[
\begin{align*}
\text{s}(\text{s}(\text{NP},\text{VP})) & \rightarrow \text{np}(\text{subject},\text{NP}), \text{vp}(\text{VP}). \\
\text{np}(\_,\text{np}(\text{Det},\text{N})) & \rightarrow \text{det}(\text{Det}), \text{n}(\text{N}). \\
\text{np}(\text{X},\text{np}(\text{Pro})) & \rightarrow \text{pro}(\text{X},\text{Pro}). \\
\text{vp}(\text{vp}(\text{V},\text{NP})) & \rightarrow \text{v}(\text{V}), \text{np}(\text{object},\text{NP}). \\
\text{vp}(\text{vp}(\text{V})) & \rightarrow \text{v}(\text{V}). \\
\text{det}(\text{det}(\text{the})) & \rightarrow \lbrack \text{the} \rbrack.
\end{align*}
\]

?- s(T,[he,shoots],[]).
T = s(np(pro(he)),vp(v(shoots)))
yes
Generating parse trees

?- s(Tree,S,[ ]).

s(s(NP,VP)) --> np(subject,NP), vp(VP).
np(_,np(Det,N)) --> det(Det), n(N).
np(X,np(Pro)) --> pro(X,Pro).
vp(vp(V,NP)) --> v(V), np(object,NP).
vp(vp(V)) --> v(V)).
det(det(the)) --> [the].
det(det(a)) --> [a].
n(n(woman)) --> [woman].
n(n(man)) --> [man].
v(v(shoots)) --> [shoots].
pro(subject,pro(he)) --> [he].
pro(subject,pro(she)) --> [she].
pro(object,pro(him)) --> [him].
pro(object,pro(her)) --> [her].
Beyond context free languages

• In the previous lecture we presented DCGs as a useful tool for working with context free grammars
• However, DCGs can deal with a lot more than just context free grammars
• The extra arguments gives us the tools for coping with any computable language
• We will illustrate this by looking at the formal language $a^n b^n c^n \{\varepsilon\}$
An example

• The language \( a^n b^n c^n \setminus \{\varepsilon\} \) consists of strings such as abc, aabbcc, aaabbbccc, aaaabbbbbc, and so on.

• This language is not context free – it is impossible to write a context free grammar that produces exactly these strings.

• But it is very easy to write a DCG that does this.
DCG for $a^n b^n c^n \setminus \{\varepsilon\}$

s(Count) --> as(Count), bc(Count), cs(Count).

as(0) --> [].
as(succ(Count)) --> [a], as(Count).

bs(0) --> [].
bs(succ(Count)) --> [b], bs(Count).

cs(0) --> [].
cs(succ(Count)) --> [c], cs(Count).
Exercises

- LPN 8.1
Extra goals

- Any DCG rule is really syntactic structure for ordinary Prolog rule
- So it is not really surprising we can also call any Prolog predicate from the right-hand side of a DCG rule
- This is done by using curly brackets { }
Example: DCG for $a^n b^n c^n \{ \varepsilon \}$

\[
\begin{align*}
s(\text{Count}) & \rightarrow \text{as}(\text{Count}), \text{bc}(\text{Count}), \text{cs}(\text{Count}). \\
\text{as}(0) & \rightarrow []. \\
\text{as}(\text{NewCnt}) & \rightarrow [a], \text{as}(\text{Cnt}), \{\text{NewCnt} \text{ is } \text{Cnt} + 1\}. \\
\text{bs}(0) & \rightarrow []. \\
\text{bs}(\text{NewCnt}) & \rightarrow [b], \text{bs}(\text{Cnt}), \{\text{NewCnt} \text{ is } \text{Cnt} + 1\}. \\
\text{cs}(0) & \rightarrow []. \\
\text{cs}(\text{NewCnt}) & \rightarrow [c], \text{cs}(\text{Cnt}), \{\text{NewCnt} \text{ is } \text{Cnt} + 1\}.
\end{align*}
\]
Separating rules and lexicon

• One classic application of the extra goals of DCGs in computational linguistics is separating the grammar rules from the lexicon

• What does this mean?
  – Eliminate all mention of individual words in the DCG
  – Record all information about individual words in a separate lexicon
The basic grammar

s --> np, vp.

np --> det, n.

vp --> v, np.

vp --> v.

det --> [the].

det --> [a].

n --> [woman].

n --> [man].

v --> [shoots].
The modular grammar

s --> np, vp.
np --> det, n.
vp --> v, np.
vp --> v.
det --> [the].
det --> [a].
n --> [woman].
n --> [man].
v --> [shoots].

s --> np, vp.
np --> det, n.
vp --> v, np.
vp --> v.
det --> [Word], \{lex(Word,det)\}.
n --> [Word], \{lex(Word,n)\}.
v --> [Word], \{lex(Word,v)\}.

lex(lex(the, det)).
lex(lex(a, det)).
lex(lex(woman, n)).
lex(lex(man, n)).
lex(lex(shoots, v)).
Concluding Remarks

• DCGs are a simple tool for encoding context-free grammars
• But in fact DCGs are a full-fledged programming language and can be used for many different purposes
• For linguistic purposes, DCG have drawbacks
  – Left-recursive rules
  – DCGs are interpreted top-down
• DCGs are no longer state-of-the-art, but they remain a useful tool
Next lecture

• A closer look at terms
  – Introduce the identity predicate
  – Take a closer look at term structure
  – Introduce pre-defined Prolog predicates that test whether a given term is of a certain type
  – Show how to define new operators in Prolog