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
  - he shoots her
- What do we need to do?

\[
\begin{align*}
s & \rightarrow \text{np, vp.} \\
\text{np} & \rightarrow \text{det, n.} \\
\text{vp} & \rightarrow \text{v, np.} \\
\text{vp} & \rightarrow \text{v.} \\
\text{det} & \rightarrow \text{[the].} \\
\text{det} & \rightarrow \text{[a].} \\
\text{n} & \rightarrow \text{[woman].} \\
\text{n} & \rightarrow \text{[man].} \\
\text{v} & \rightarrow \text{[shoots].}
\end{align*}
\]
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?
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

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

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

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

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

  \[
  s(A,B) :\text{- np(\text{subject},A,C), \text{ vp}(C,B).}\]
Listing noun phrases

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].

?- 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

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

s \rightarrow \text{np(subject), 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}].
DCG that builds parse tree

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

<table>
<thead>
<tr>
<th>s(s(NP,VP)) --&gt; np(subject,NP), vp(VP).</th>
</tr>
</thead>
<tbody>
<tr>
<td>np(__,np(Det,N)) --&gt; det(Det), n(N).</td>
</tr>
<tr>
<td>np(X,np(Pro)) --&gt; pro(X,Pro).</td>
</tr>
<tr>
<td>vp(vp(V,NP)) --&gt; v(V), np(object,NP).</td>
</tr>
<tr>
<td>vp(vp(V)) --&gt; v(V)).</td>
</tr>
<tr>
<td>det(det(the)) --&gt; [the].</td>
</tr>
<tr>
<td>det(det(a)) --&gt; [a].</td>
</tr>
<tr>
<td>n(n(woman)) --&gt; [woman].</td>
</tr>
<tr>
<td>n(n(man)) --&gt; [man].</td>
</tr>
<tr>
<td>v(v(shoots)) --&gt; [shoots].</td>
</tr>
<tr>
<td>pro(subject,pro(he)) --&gt; [he].</td>
</tr>
<tr>
<td>pro(subject,pro(she)) --&gt; [she].</td>
</tr>
<tr>
<td>pro(object,pro(him)) --&gt; [him].</td>
</tr>
<tr>
<td>pro(object,pro(her)) --&gt; [her].</td>
</tr>
</tbody>
</table>
Generating parse trees

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

?- 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 $a^n b^n c^n \{\epsilon\}$ consists of strings such as abc, aabbcc, aaabbbcccc, aaaaabbbccccc, 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 $a^n b^n c^n \{\varepsilon\}$

\[
\begin{align*}
\text{s(Count)} & \rightarrow \text{as(Count), bs(Count), cs(Count)}.
\text{as}(0) & \rightarrow []. \\
\text{as}(\text{succ(Count)}) & \rightarrow [a], \text{as(Count)}.
\text{bs}(0) & \rightarrow []. \\
\text{bs}(\text{succ(Count)}) & \rightarrow [b], \text{bs(Count)}.
\text{cs}(0) & \rightarrow []. \\
\text{cs}(\text{succ(Count)}) & \rightarrow [c], \text{cs(Count)}.
\end{align*}
\]
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 $a^nb^nc^n\{\epsilon\}$

\[
s(\text{Count}) \rightarrow \text{as}(\text{Count}), \text{bs}(\text{Count}), \text{cs}(\text{Count}).
\]

\[
\text{as}(0) \rightarrow [].
\]

\[
\text{as}(\text{NewCnt}) \rightarrow [a], \text{as}(\text{Cnt}), \{\text{NewCnt is Cnt + 1}\}.
\]

\[
\text{bs}(0) \rightarrow [].
\]

\[
\text{bs}(\text{NewCnt}) \rightarrow [b], \text{bs}(\text{Cnt}), \{\text{NewCnt is Cnt + 1}\}.
\]

\[
\text{cs}(0) \rightarrow [].
\]

\[
\text{cs}(\text{NewCnt}) \rightarrow [c], \text{cs}(\text{Cnt}), \{\text{NewCnt is Cnt + 1}\}.
\]
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

<table>
<thead>
<tr>
<th>s</th>
<th>np, vp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>np</td>
<td>det, n.</td>
</tr>
<tr>
<td>vp</td>
<td>v, np.</td>
</tr>
<tr>
<td></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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>np, vp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>np</td>
<td>det, n.</td>
</tr>
<tr>
<td>vp</td>
<td>v, np.</td>
</tr>
<tr>
<td></td>
<td>v.</td>
</tr>
<tr>
<td>det</td>
<td>[Word], {lex(Word,det)}.</td>
</tr>
<tr>
<td>n</td>
<td>[Word], {lex(Word,n)}.</td>
</tr>
<tr>
<td>v</td>
<td>[Word], {lex(Word,v)}.</td>
</tr>
</tbody>
</table>

+ lext(the, det).
  lext(a, det).
  lext(woman, n).
  lext(man, n).
  lext(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 not allowed
  – DCGs are interpreted top-down
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