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?

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

```plaintext
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
Some examples of ungrammatical strings accepted by this DCG

?- s([a,woman,shoots,he],[ ]). yes
?- s([her,shoots,a,man],[ ]). yes
s([her,shoots,he],[ ]). 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].
det --> [a].                       n --> [woman].
n --> [man].                        n --> [man].
v --> [shoots].                    v --> [shoots].
pro_subject --> [he].              pro_subject --> [he].
pro_subject --> [she].             pro_object --> [him].
pro_object --> [her].              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...

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

```
[the woman] shoots [a man]
```

```
the          woman        shoots       a       man
  det        n                v                det       n

s                 vp
  np                np
```
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

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

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

?- 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(Det,N)) --> det(Det), n(N).
np(Pro)) --> pro(X,Pro).
v(V,NP)) --> v(V), np(object,NP).
v(V)) --> v(V).
det(the)) --> [the].
det(a)) --> [a].
pro(subject,pro(woman)) --> [woman].
pro(subject,pro(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 $a^n b^n c^n \setminus \{\varepsilon\}$
An example

• The language \(a^n b^n c^n \setminus \{\varepsilon\}\) consists of strings such as abc, aabbcc, aaabbbcccc, aaaaabbbcccccc, 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 $a^n b^n c^n \setminus \{\varepsilon\}$

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

\[
as(0) \rightarrow [].
\]

\[
as(\text{succ}(\text{Count})) \rightarrow [a], \text{as}(\text{Count}).
\]

\[
bs(0) \rightarrow [].
\]

\[
bs(\text{succ}(\text{Count})) \rightarrow [b], \text{bs}(\text{Count}).
\]

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

\[
\text{cs}(\text{succ}(\text{Count})) \rightarrow [c], \text{cs}(\text{Count}).
\]
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 $a^n b^n c^n\{\varepsilon\}$

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

$as$(0) --> [].
$as$(NewCnt) --> [a], $as$(Cnt), \{NewCnt is Cnt + 1\}.

$bs$(0) --> [].
$bs$(NewCnt) --> [b], $bs$(Cnt), \{NewCnt is Cnt + 1\}.

$cs$(0) --> [].
$cs$(NewCnt) --> [c], $cs$(Cnt), \{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

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

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

\[
\begin{align*}
\text{s} & \rightarrow \text{np, vp} \\
\text{np} & \rightarrow \text{det, n} \\
\text{vp} & \rightarrow \text{v, np} \\
\text{vp} & \rightarrow \text{v} \\
\text{det} & \rightarrow \text{[Word], \{\text{lex}\text{(Word, det)}\}} \\
\text{n} & \rightarrow \text{[Word], \{\text{lex}\text{(Word, n)}\}} \\
\text{v} & \rightarrow \text{[Word], \{\text{lex}\text{(Word, v)}\}}
\end{align*}
\]
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