Handout 20: GDev and PCFGs

Grammar Development

1. The grammar developer

a. It is an extended version of the HW 8 GDev class. Launching it:

```
$ python -m gdev g20
G>
```

b. Start from a regression set of labeled sentences (`g20.sents`)

```
the dog barks
*the dogs barks
...
```

c. We also require a grammar: `g20.fcfg`

2. Commands

a. Listing the sentences:

```
G> ss
 =>  0 OK the dog barks
     1  * the dogs barks
...
    82  * you give the present to
```

b. The current sentence

```
G> s
  0 OK the dog barks
```

c. Parsing it

```
Number of trees: 1

Tree 0
(Root[]
  (S[]
    (NP[f='sg']
      (Det[] the)
      (N2[f='sg'] (N1[f='sg'] (NC[f='sg'] (N[f='sg'] dog)))))
    (VP[f='sg'] (V[f='sg', s=0, -t] barks))))
```

d. Next sentence

```
G> n
  1  * the dogs barks
```

e. Going backwards

```
G> b
  0 OK the dog barks
```

f. Jumping to a particular sentence

```
G> 11
 11 OK cats bark
```

3. Finding errors

a. Sentences where parser prediction differs from sentence label

```
G> e
 => 11  * (OK) cats bark
    20  * (OK) Alice thinks that the white rabbit is in a hurry
...
```

b. First label is prediction, parenthesized is truth

4. Scoring

a. Computing:

```
G> sc
Accuracy:    0.6904761904761905
Sensitivity: 0.46808510638297873
Specificity: 0.972972972972973
```

b. Accuracy: proportion correct

c. Errors of **sensitivity**: good sentence, but the grammar fails to parse it. Add rules.

d. Errors of **specificity**: bad sentence, but the grammar accepts it. Add constraints (features).

5. Fixing errors

a. If it parses, but shouldn't, examine the parse tree

b. If it doesn't parse, look at the chart

```
G> c
cats
    N[f='pl'] -> 'cats'
    NC[f='pl'] -> N[f='pl']
    N1[f='pl'] -> NC[f='pl']
    N2[f='pl'] -> N1[f='pl']
bark
    V[f='pl', s=0, -t] -> 'bark'
    V[f='base', s=0, -t] -> 'bark'
    VP[f='base'] -> V[f='base', s=0, -t]
    VP[f='pl'] -> V[f='pl', s=0, -t]
```

c. After editing grammar or sents, reload:

```
G> r
```

6. Examining vocabulary

a. Finding words that need to be added to the grammar:

```
G> unk
I
did
do
...
```

b. Finding words that are not tested in any of the sentences:

```
G> unt
Frodo
Gertrude
...
```

c. Listing parts of speech:

```
G> pos
Adj
Aux
Det
...
```

7. Parsing detects errors of omission; generation, errors of commission.

a. Generating a sentence

```
>>> g.gen()
G> gen
  0 Gertrude thinks about these aardvarks
  1 will the first one brighter wine know
...
  9 Alice knew
```

b. If it is not clear how the string was generated, look at the tree:

```
G> t9
Number of trees: 10.  Tree 9:
(Root[]
  (S[]
    (NP[f='sg'] (PropN[] Alice))
    (VP[f='sg'] (V[f='sg', s=0, -t] knew))))
```

c. Interesting sentences may be added to the regression set. Keep a good sentence:

```
G> k0
```

d. Keep a sentence, but label it as bad:

```
G> *1
```

Probabilistic Parsing

8. Ambiguity

```
>>> p = load_parser('g1.cfg')
>>> trees = list(p.parse('Mary walked the cat in the park'.split()))
>>> print(trees[0])
(S
  (NP (Name Mary))
  (VP
    (V walked)
    (NP (Det the) (N cat))
    (PP (P in) (NP (Det the) (N park)))))
>>> print(trees[1])
(S
  (NP (Name Mary))
  (VP
    (V walked)
    (NP (Det the) (N cat) (PP (P in) (NP (Det the) (N park))))))
```

9. Weighted PCFG

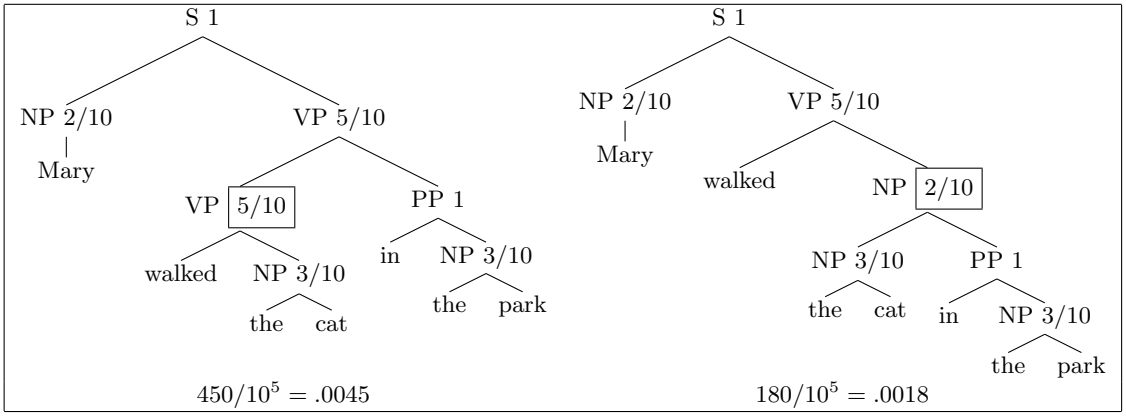
a. Example—put it in the file `g2.pcfg`:

```
% start S
S -> NP VP          [1]
NP -> 'Mary'        [.2]
NP -> 'the' 'cat'   [.3]
NP -> 'the' 'park'  [.3]
NP -> NP PP         [.2]
PP -> 'in' NP       [1]
VP -> 'walked' NP   [.5]
VP -> VP PP         [.5]
```

b. Parse

```
>>> p = load_parser('g2.pcfg')
>>> trees = list(p.parse('Mary walked the cat in the park'.split()))
>>> print(trees[0])
(S
  (NP Mary)
  (VP (VP walked (NP the cat)) (PP in (NP the park)))) (p=0.0045)
>>> print(trees[1])
(S
  (NP Mary)
  (VP walked (NP (NP the cat) (PP in (NP the park))))) (p=0.0018)
```

10. Compute the probabilities of the trees



11. Suppose we want the other parse on top

a. Change the VP weights (`g2-alt.pcfg`):

```
VP -> 'walked' NP  [.9]
VP -> VP PP        [.1]
```

b. Now:

```
>>> p = load_parser('g2-alt.pcfg')
>>> trees = list(p.parse('Mary walked the cat in the park'.split()))
>>> print(trees[0])
(S
  (NP Mary)
  (VP walked (NP (NP the cat) (PP in (NP the park))))) (p=0.00324)
>>> print(trees[1])
(S
  (NP Mary)
  (VP (VP walked (NP the cat)) (PP in (NP the park)))) (p=0.00162)
```

12. Estimate a grammar from the following treebank: