The English grammar

An English grammar for chartparse.

This grammar was originally written by Steve Isard at the University of Sussex. The vocabulary is designed to amuse undergraduate Experimental Psychology students, hence the references to pigeons and cages.

The grammar is almost entirely Steve’s original. The only changes are a few words, proper names, and the production:

NP -> det Nn PP

which was changed to

NP -> NP PP

The intent is to demonstrate ambiguous grouping of modifiers.

As in the original LIB CHART [1], features on the categories are ignored. There are three features used case, num and tr. They could reasonably be handled in this file, via compilation to a plain CFG, since their purpose is only to enforce agreement.

2.1 References

The original LIB CHART [RI]
class english.Grammar(grammar, lexicon, state=None)
   Class for creating grammars from text strings.

   Parameters grammar: string
       the grammar rules, lines of the form lhs -> rhs (rhs)*

   lexicon: string
       the words, lines of the form word category+

Examples

>>> g = Grammar(RULES, WORDS)
>>> g.grammar[0]
    Rule(lhs='S', rhs=['Np', 'Vp'])

Methods

    make_rule(lhs)

class english.Rule
   One production of a context-free grammar.
Examples

```python
>>> r = Rule('s',('np','vp'))
```

Attributes

<table>
<thead>
<tr>
<th>lhs: string</th>
<th>The left hand side of the rule.</th>
</tr>
</thead>
<tbody>
<tr>
<td>rhs: list [string]</td>
<td>The right hand side of the rule.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>count(...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>index((value, [start, ...)) Raises ValueError if the value is not present.</td>
</tr>
</tbody>
</table>
3.1 Lattice parsing

Chart parsing with word lattice input.

Chart parsing is based on two key ideas

• Collapsing together derivations that can be shown to have a common fate.

• Building data structures that are indexed by their start and end points.

3.1.1 Standard Chart Parsing

The standard chart parser works with string positions as exemplified below:

0 show 1 me 2 a 3 movie 4 where 5 the 6 director 7 is 8 clint 9 eastwood 10

The chart is seeded with entries such as Item("Nn","show",0,1), Item("V","show",0,1) and Item("Det","the",5,6). These entries are all of length 1. Lexical ambiguity shows up as multiple items spanning the same set of words but giving them different labels, as happens here for the span 0:1

The parser then uses grammar rules to combine items and generate longer items such as Item("Np",5,7). Two items can be combined if the left hand item ends in the same place that the right hand item starts, and the grammar licenses the combination. If, at the end of the process, an item has been built that spans the string from beginning to end and has a suitable label (e.g. sentence, imperative, question, whatever ...), the input has been fully analyzed.

Depending on the details of the implementation, there may also be items that represent partial constituents such as Edge("SImp",0,2,"[Np]"). This one says that if material to the right of the span 0:2 can be made into an Np we will have an imperative sentence.

The chart enforces a no-duplicates condition. When the same item can be made more than one way, it is stored in the chart only once, and a separate data structure is updated to keep track of the alternatives. Two items are equivalent if they span the same words and have the same label. For simple grammars, this is enough to enforce the principle of common fate. If features are used, a little more care is needed, but the essential principle is unchanged: two alternative derivations are collapsed together when it has been shown that subsequent parsing actions will affect them in exactly the same way.
3.1.2 Lattice Chart Parsing

A chart parser can be adapted to work with a word lattice, where the identity of the words is uncertain. Suppose that an ASR system has tried to identify the example sentence, and is unsure about the words “director” and “eastwood”. It thinks that “director” might have been either (“direct” “or”) or (“dye” “rector”) and that “eastwood” might have been (“is”, “would”), (“is”, “wood”), or (“east”, “wood”). A real lattice might have more ambiguity than this, we are keeping it short for readability. Note that (“east”, “would”) is not a possibility.

This uncertainty can be represented by a finite-state machine, as follows:

Arcs:

```
0 show 1  
1 me  2   
2 a    3  
3 movie 4  
4 where 5  
5 the 6   
6 director 7  
6 direct 6.1 
6 dye 6.2   
6.1 or 7   
6.2 rector 7 
7 is 8     
8 clint 9  
9 eastwood 10  
9 is 9.1   
9 east 9.1 
9 is 9.2 
9.1 wood 10 
9.2 would 10
```

For convenience, the arcs can be renumbered with consecutive integers:

```
0 show 1  
1 me  2   
2 a    3  
3 movie 4  
4 where 5  
5 the 6   
6 director 9  
6 direct 7  
6 dye 8   
7 or 9   
8 rector 9  
9 is 10 
10 clint 11 
11 eastwood 14 
11 is 12 
11 east 12 
11 is 13 
12 wood 14 
13 would 14
```

Once this is done, the chart can be seeded in the same way as before, except that the numbers now represent states of the finite-state machine, rather than string positions.

The process that builds combinations from the initial seeds can now be left unchanged. Items can combine if the start state of one is the end state of the other, and the grammar licenses that combination. Because of the renumbering,
words that are on incompatible paths pass through different intermediate states, therefore never have the opportunity to combine.

The termination condition, also changes slightly: we now say that an analysis is complete if an item is built whose start point is a start state of the finite-state machine and whose end point is an accepting state of the machine.

```python
class lattice.DemoLatticeWords(arcs=[(0, 'show', 1), (1, 'me', 2), (2, 'a', 3), (3, 'movie', 4), (4, 'where', 5), (5, 'the', 6), (6, 'director', 9), (6, 'direct', 7), (6, 'dye', 8), (7, 'or', 9), (8, 'rector', 9), (9, 'is', 10), (10, 'clint', 11), (11, 'eastwood', 14), (11, 'is', 12), (11, 'east', 12), (11, 'is', 13), (12, 'wood', 14), (13, 'would', 14)])
```

Run a chart from a lattice rather than a linear set of words.

```python
>>> import chart
>>> chart.parse(demo_arcs, topcat='SImp', sep='_', input_source=DemoLatticeWords)
[(0, 'show', 1), (1, 'me', 2), (2, 'a', 3), (3, 'movie', 4), (4, 'where', 5), (5, 'the', 6), (6, 'director', 9), (6, 'direct', 7), (6, 'dye', 8), (7, 'or', 9), (8, 'rector', 9), (9, 'is', 10), (10, 'clint', 11), (11, 'eastwood', 14), (11, 'is', 12), (11, 'east', 12), (11, 'is', 13), (12, 'wood', 14), (13, 'would', 14)]
```

### Attributes

- **final_state** The final state should be in final position.

### Methods

- **arcs()**

### final_state

The final state should be in final position.
CHAPTER 4

Indices and tables

- genindex
- modindex
- search
English, 3

Lattice, 7
Index

D
DemoLatticeWords (class in lattice), 9

E
english (module), 3

F
final_state (lattice.DemoLatticeWords attribute), 9

G
Grammar (class in english), 4

L
lattice (module), 7

R
Rule (class in english), 4