Accelerated Natural Language Processing 2016

Lecture 15: Probabilistic CF-PSGs, best-first parsing

Henry S. Thompson
20 October 2016

1. The *active* chart

The Earley algorithm (see Jurafsky and Martin) was the first one to put *hypotheses*, that is, partial constituents, into the chart

We'll look at chart parsing

- Originally described by Martin Kay
- It's the most general and can be configured to emulate the others
- And lots more besides

It distinguishes between

inactive edges
: which represent complete constituents, including pointers to the edges representing their children

active edges
: which represent *incomplete* constituents, the labels required to complete them, and their substructure so far, if any

2. Chart parsing basics

Two basic components:

- The **chart**
- The **agenda**

The chart is composed of **edges** and **vertices**

In its simplest form, the agenda is just a LIFO or FIFO queue of edges

Edges come off the agenda and into the chart one at a time

- And the addition of an edge to the chart may create new edges, which go back on the agenda
- The parse is over when the agenda is empty

3. Chart parsing: More on edges

Active edges consist of a **dotted rule**, a sequence of descendants and a start and endpoint

- A dotted rule is a CF-PSG rule with a dot, indicating progress towards satisfaction

```
S → NP   VP
```

```
VP →   Vt NP
```

Inactive edges consist of a label, a sequence of descendants and a start and end point

4. Chart parsing: where do edges come from?

The **fundamental rule** of chart parsing:

- When an active and inactive edge *meet* for the first time, try to *satisfy* the active edge with the inactive edge

An active edge meets an inactive one if the active edge's endpoint is the same as the inactive edge's start point

An inactive edge satisfies an active edge if its label matches the next symbol after the dot in the active edge's dotted rule

If you win, add a new edge

- from the start of the active edge
- to the end of the inactive edge
- If that was the last thing the active edge needed the result is inactive
- Otherwise it's active, with the dot moved along one

5. Chart parsing basics, cont'd

Also, we have **rule invocation strategies**:

top-down
: When an active edge is added to the chart, add new empty (looping) active edges at its end for each rule in the grammar which expands the symbol after the dot
    - That is, edges which might result in what is needed next

bottom-up
: When an inactive edge is added to the chart, add empty active edges at its start for every rule in the grammar whose first symbol on the RHS matches the label of the edge
    - That is, edges which might make use of what was just found
    - This is where the name **left-corner** comes in

For bottom-up parsing, just adding the lexical edges will get us started

For top-down parsing, we need to add active edges for **S** at the beginning

LIFO vs. FIFO queuing determine depth- vs. breadth-first search

6. Chart parsing worked example

We can work through an example of parsing a trivial phrase ("the dog") with a trivial grammar, using a bottom-up, depth-first approach

Depth-first means we do LIFO queuing

Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

Input

"the dog"

Agenda

1 L $_{1}\text{dog}_{2}$

2 L $_{0}\text{the}_{1}$

Chart

$0 \xrightarrow{\text{the}} 1$

A new *inactive* edge in the chart: we check the grammar

Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

Input

"the dog"

Agenda

1 L ${}_{1}$ dog ${}_{2}$

2 L ${}_{0}$ the ${}_{1}$

3 BU(2,A) ${}_{0}$ **D→ · the** ${}_{0}$

Chart

Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

Input

"the dog"

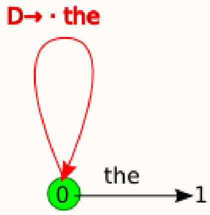
Agenda

1 L ${}_{1}$ dog ${}_{2}$

2 L ${}_{0}$ the ${}_{1}$

3 BU(2,A) ${}_{0}$ D→ · the ${}_{0}$

Chart



Active meets inactive for the first time -- the fundamental rule is tried

And succeeds

Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

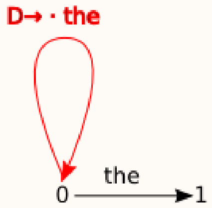
Input

"the dog"

Agenda

1 L $_1$ dog $_2$
2 L $_0$ the $_1$
3 BU(2,A) $_0$ D→ · the $_0$
4 FR(3,2) $_0$ **D [the]** $_1$

Chart



Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

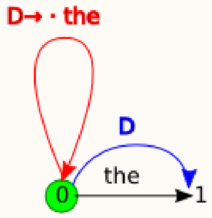
Input

"the dog"

Agenda

1 L $_1$ dog $_2$
2 L $_0$ the $_1$
3 BU(2,A) $_0$ D→ · the $_0$
4 FR(3,2) $_0$ D [the] $_1$

Chart



New inactive edge: we check the grammar

Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

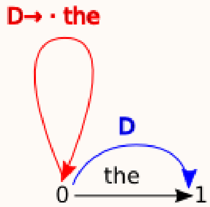
Input

"the dog"

Agenda

1 L	$_1$ dog $_2$
2 L	$_0$ the $_1$
3 BU(2,A)	$_0$ D→ · the $_0$
4 FR(3,2)	$_0$ D [the] $_1$
5 BU(4)	**$_0$ NP→ · D N $_0$**

Chart



Active met inactive again also, but *didn't* match

Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

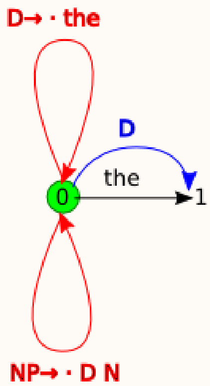
Input

"the dog"

Agenda

1 L	$_1$ dog $_2$
2 L	$_0$ the $_1$
3 BU(2,A)	$_0$ D→ · the $_0$
4 FR(3,2)	$_0$ D [the] $_1$
5 BU(4)	$_0$ NP→ · D N $_0$

Chart



New active means *two* instances of the fundamental rule to check

Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

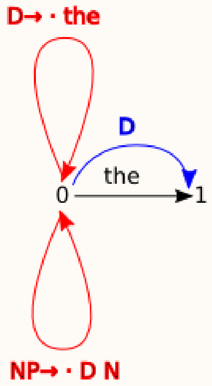
Input

"the dog"

Agenda

1 L	$_1$ dog $_2$
2 L	$_0$ the $_1$
3 BU(2,A)	$_0$ D→ · the $_0$
4 FR(3,2)	$_0$ D [the] $_1$
5 BU(4)	$_0$ NP→ · D N $_0$
6 FR(5,4)	**$_0$ NP→ D · N $_1$**

Chart



One wins, giving us our first partial constituent

Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

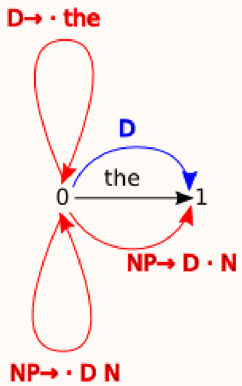
Input

"the dog"

Agenda

1 L	$_1$ dog $_2$
2 L	$_0$ the $_1$
3 BU(2,A)	$_0$ D→ · the $_0$
4 FR(3,2)	$_0$ D [the] $_1$
5 BU(4)	$_0$ NP→ · D N $_0$
6 FR(5,4)	$_0$ NP→ D · N $_1$

Chart



Finally we go back to the first entry in the agenda

Bottom-up LIFO Chart parse example

Grammar

A) D → the | …
B) N → dog | …
C) NP → D N
…

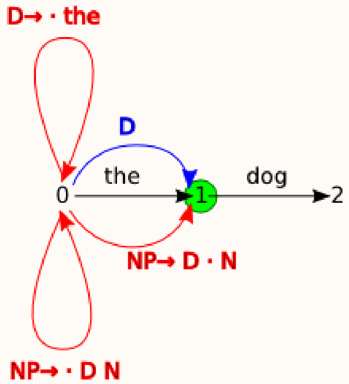
Input

"the dog"

Agenda

1	L	$_1$ dog $_2$
2	L	$_0$ the $_1$
3	BU(2,A)	$_0$ D→ · the $_0$
4	FR(3,2)	$_0$ D [the] $_1$
5	BU(4)	$_0$ NP→ · D N $_0$
6	FR(5,4)	$_0$ NP→ D · N $_1$

Chart



Bottom-up LIFO Chart parse example

Grammar

A) D → the | …
B) N → dog | …
C) NP → D N
…

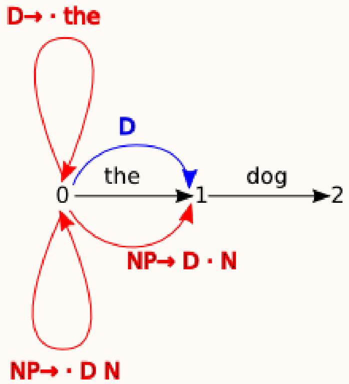
Input

"the dog"

Agenda

1	L	$_1$ dog $_2$
2	L	$_0$ the $_1$
3	BU(2,A)	$_0$ D→ · the $_0$
4	FR(3,2)	$_0$ D [the] $_1$
5	BU(4)	$_0$ NP→ · D N $_0$
6	FR(5,4)	$_0$ NP→ D · N $_1$
7	BU(1,B)	$_1$ N→ · dog $_1$

Chart



Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

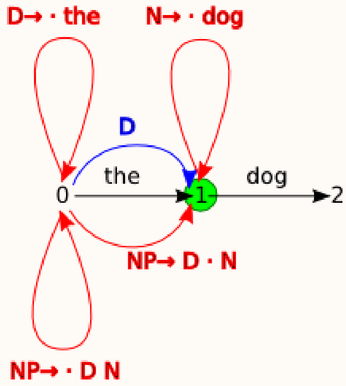
Input

"the dog"

Agenda

1 L	$_1$ dog $_2$
2 L	$_0$ the $_1$
3 BU(2,A)	$_0$ D→ · the $_0$
4 FR(3,2)	$_0$ D [the] $_1$
5 BU(4)	$_0$ NP→ · D N $_0$
6 FR(5,4)	$_0$ NP→ D · N $_1$
7 BU(1,B)	$_1$ N→ · dog $_1$

Chart



Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

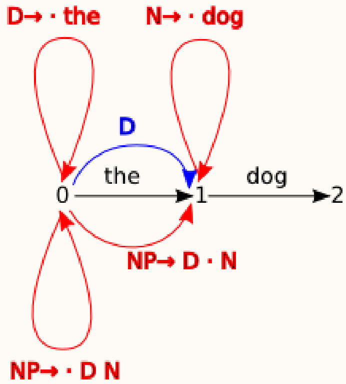
Input

"the dog"

Agenda

1 L	$_1$ dog $_2$
2 L	$_0$ the $_1$
3 BU(2,A)	$_0$ D→ · the $_0$
4 FR(3,2)	$_0$ D [the] $_1$
5 BU(4)	$_0$ NP→ · D N $_0$
6 FR(5,4)	$_0$ NP→ D · N $_1$
7 BU(1,B)	$_1$ N→ · dog $_1$
8 FR(7,1)	$_1$ **N [dog]** $_2$

Chart



Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

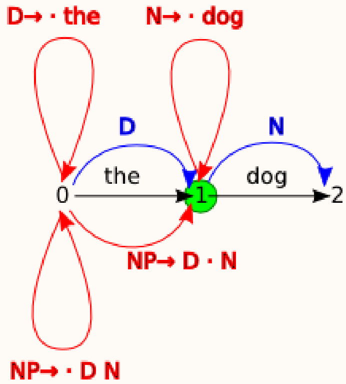
Input

"the dog"

Agenda

1	L	$_1$ dog $_2$
2	L	$_0$ the $_1$
3	BU(2,A)	$_0$ D→ · the $_0$
4	FR(3,2)	$_0$ D [the] $_1$
5	BU(4)	$_0$ NP→ · D N $_0$
6	FR(5,4)	$_0$ NP→ D · N $_1$
7	BU(1,B)	$_1$ N→ · dog $_1$
8	FR(7,1)	$_1$ N [dog] $_2$

Chart



Bottom-up LIFO Chart parse example

Grammar

A) D → the | ...
B) N → dog | ...
C) NP → D N
...

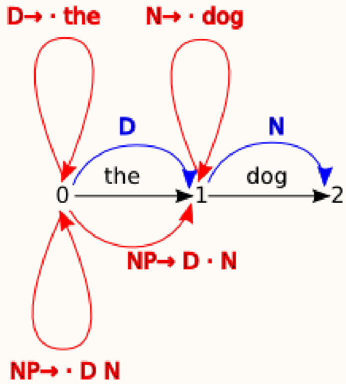
Input

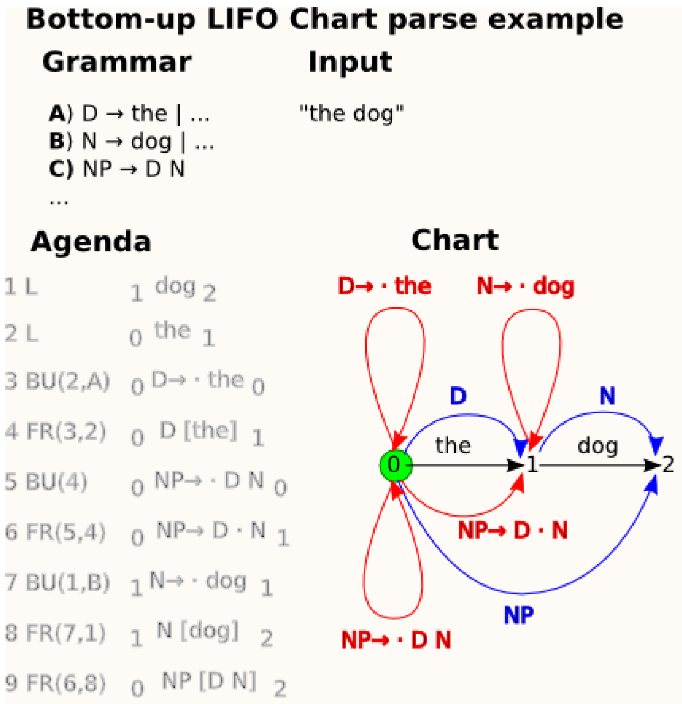
"the dog"

Agenda

1	L	$_1$ dog $_2$
2	L	$_0$ the $_1$
3	BU(2,A)	$_0$ D→ · the $_0$
4	FR(3,2)	$_0$ D [the] $_1$
5	BU(4)	$_0$ NP→ · D N $_0$
6	FR(5,4)	$_0$ NP→ D · N $_1$
7	BU(1,B)	$_1$ N→ · dog $_1$
8	FR(7,1)	$_1$ N [dog] $_2$
9	FR(6,8)	**$_0$ NP [D N] $_2$**

Chart





7. Chart entries: reconstructing rules

Including subconstituent pointers allows us to reconstruct the rule that enabled a cell entry

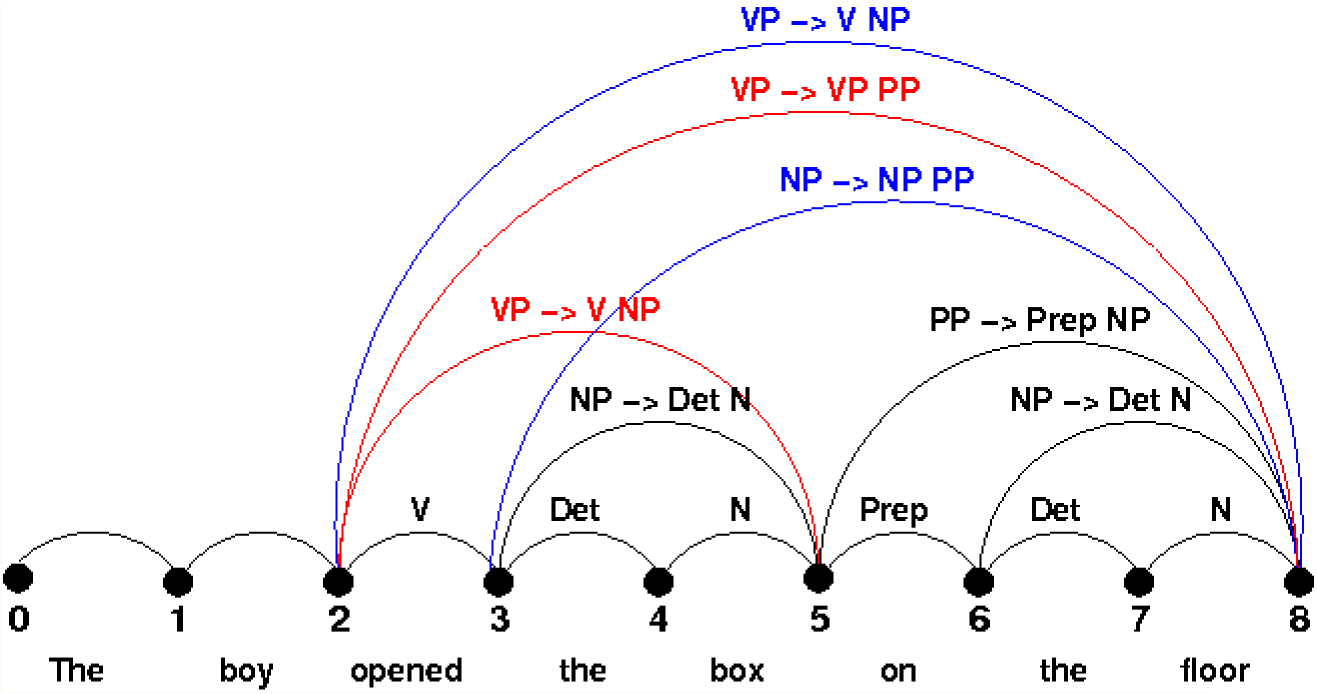
- Or *rules*

So if we have two VP rules:

```
VP → V NP
```

```
VP → VP PP
```

The chart will show us *both* ways to get the VP from an ambiguous input:



8. Chart parsing and left recursion

Recursive-descent parsers have a problem with **left-recursive** rules:

`NP → NP PP`

`NP → Det Nom` `Det → NP 's`

Because the chart records hypotheses as well as results

- It's easy to avoid *redundant* hypotheses
- By not adding (empty, active) edges to the chart if an identical one is already there

9. Ambiguity is still the problem

Features help express the grammar neatly, but they don't change the ambiguity problem

Chart parsing gives us flexibility, but that doesn't solve the ambiguity problem

Examples of ambiguity:

global

PP attachment

“One morning I shot an elephant in my pyjamas ...” (Groucho Marx)

gerundive VP attachment

We saw penguins flying over Antarctica

coordination

hot tea and coffee

vs

empty bottles and fag-ends

10. Compositional semantics, again

The significance of attachment ambiguity is clear when we look at semantics

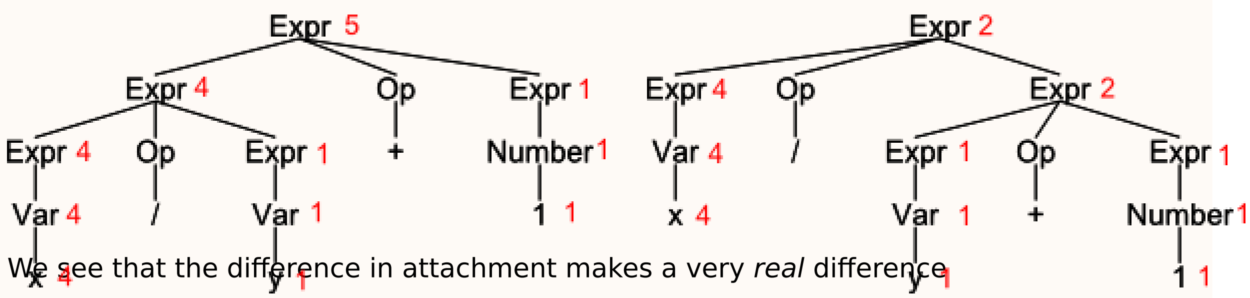
Back to our simple grammar for Python arithmetic expressions:

```
Expr → Expr Op Expr | Var | Number
Op → '+' | '-' | '*' | '/'
```

Assuming some more work to **tokenise** the input, this will give us two analyses for the string `"x/y+1"`:

If we associate the appropriate computation with each production

- (skipping a lot of details here, some of which will be covered in a few weeks)
- And assuming that `x` is `4` and `y` is `1`



We see that the difference in attachment makes a very *real* difference

We call that kind of semantic computation a **compositional** one

"The meaning of the whole [constituent] is a function of the meaning of the parts [children]"

11. Compositional semantics, cont'd

How does this relate to natural language?

- Many approaches to language *understanding* proceed in a similar way
  - In what's called a **rule-to-rule** way
  - Associating a (compositional) *semantic* rule with each *syntactic rule*

So, for example, for the kind of PP attachment ambiguity we keep encountering

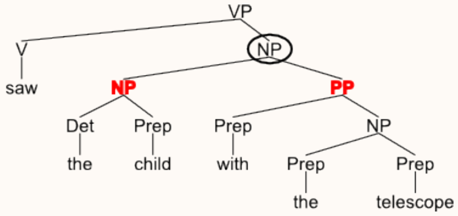
- Where the PP is attached determines where its semantic contribution is made

In other words, with respect to "saw the child with the telescope"

VP → V NP, NP → NP PP

The semantics of the PP contributes to the semantics of the (higher) NP

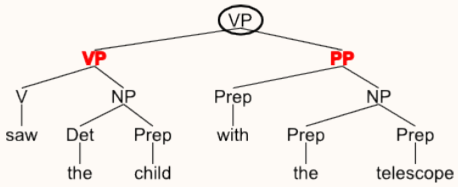
- The child has the telescope



VP → VP PP

The semantics of the PP contributes to the semantics of the (higher) VP

- The seeing was done with the telescope



12. Back to ambiguity

We need a way to choose the best analysis from among many

- *Very* many
- The average sentence in the Wall Street Journal dataset we used in lab on Tuesday is just under *26* words long
- The exponential consequence of multiple local ambiguities given a broad-coverage CF-PSG
- Will be 1000s, if not 100s of 1000s, of parses

And we need a sound basis for ranking these

A gold standard provides at least partial solutions. . .

13. Treebanks: big investment, big reward

Mitch Marcus at the Univ. of Pennsylvania took this seriously

- As did the US DARPA (Defense Advanced Research Projects Agency)
  - In the context of their evaluation-led research funding approach

The Penn Treebank project was launched in 1989

- Over a number of phases has annotated *many* datasets, including
  - the Brown Corpus, WSJ data, air-travel request data, phone conversation data
- and has spawned many follow-ons for other languages

14. Treebank examples and evolution

The first release contained 'skeletal' parses, that is, simple syntactic trees

- With a few quirks based on a choice of underlying grammatical theory

- Now somewhat dated, this mostly involves the use of what are called **traces** to indicate 'missing' material
- For example in relative clauses and some complement clauses
    - "I liked the show that we watched * last night"
    - "Robin found it difficult to * lift the boxes"

Here's an example:

```
(S
  (NP
    (ADJP Battle-tested Japanese industrial)
     managers)
  here always
  (VP buck up
      (NP nervous newcomers)
      (PP with
         (NP the tale
             (PP of
                 (NP the
                    (ADJP first
                          (PP of
                          (NP their countrymen)))
                    (S (NP *)
                       to
                       (VP visit
                          (NP Mexico))))))))
```

This annotation was produced by manually correcting the output of a simple chunking parser, then removing the POS-tags and some NP-internal structure

15. Penn Treebank, mark two

The second release added much more information, including some predicate-argument indications:

```
(S (NP-SBJ (NP Battle-tested Japanese industrial managers)
             (ADVP-LOC here))
     (ADVP-TMP always)
     (VP buck
         (PRT up)
         (NP nervous newcomers)
         (PP-CLR with
                 (NP (NP the tale)
                     (PP of
                         (NP (NP the
                                 first)
                             (PP of
                                 (NP their countrymen))
                             (SBAR (WHNP-1 0)
                                   (S (NP-SBJ *T*-1)
                                      (VP to
                                          (VP visit
                                              (NP Mexico))))))))))
```

16. Deriving a grammar from a treebank

Trivial, really

For every node in every tree with non-lexical children

- Add a rule to the grammar (or increment the count of an existing rule)
- **NodeLab** → **Child1Lab Child2Lab** . . .

So, for the tree in the previous slide, we'd get e.g.

NP-SBJ → NP ADVP-LOC
NP → NP PP
NP → NP PP SBAR

The result is guaranteed to give at least one parse to every sentence in the Treebank

17. Simple Probabilistic CF-PSGs

Given a treebank, we can easily compute a simple kind of probabilistic grammar

- Either directly with respect to treebank-derived rules
- Or by mapping from treebank statistics to some other ruleset

For a treebank-derived ruleset, the maximum likelihood probability estimate is simple:

$$\boxed{P(\mathrm{NT} \rightarrow C_1 C_2 \ldots C_n \mid \mathrm{NT}) = \frac{\mathrm{count}(\mathrm{NT} \rightarrow C_1 C_2 \ldots C_n)}{\mathrm{count}(\mathrm{NT})}}$$

So, for example, given that in the NLTK treebank subset, there are 52041 subtrees labelled **S**

And in 29201 of these there are exactly two children of **S**, labelled **NP-SBJ** followed by **VP**, we get

$$\boxed{P(S \rightarrow \mathrm{NP\text{-}SBJ}\ \mathrm{VP} \mid S) = \frac{29201}{52041} = 0.56}$$

The probability of a whole parse tree is then just the product of the probabilities of every non-terminal node in that tree

- Or we can use the sum of the costs for simpler calculations
  - Recall that **costs** are negative log probabilities
  - We sum them, instead of multiplying
  - And prefer lower costs (== higher probabilities)

It's not hard to modify a chart parser to use probabilities to actually guide parsing

- So that you get the most likely parse first
- By maintaining the agenda in sorted order
- And always taking the most probable agenda entry to put in the chart next

18. Evaluation against a treebank

No-one's grammatical theory/detailed grammar is going to give exactly the same results as a treebank

- Unless it's *derived* automatically *from* the treebank

So how do you evaluate a parser against a treebank?

It turns out just looking at major constituent boundaries is surprisingly good

- Basically, because most parsers are pretty *bad*

The same (D)ARPA push towards evaluation-driven funding drove the development of the **PARSEVAL** metric

PARSEVAL just compares bracketings, without regard to labels

- In its simplest form, it just counts parenthesis-crossing
    - (A (B C)) vs. ((A B) C)
    - Fewer is better
- And constituent recall
    - (A B C) vs. (A (B C))
    - More is better

Each of these is effectively penalising an attachment mistake

19. PARSEVAL examples

```
(saw (a child with a telescope))
```

will be penalised wrt

```
(saw (a child) (with a telescope))
```

- by the second rule above, because of the mismatch in number of sub-constituents

```
((hot tea) and coffee)
```

will be penalised wrt

```
(hot (tea and coffee))
```

- because of the crossing parentheses