Computational Linguistics

CSC 485/2501
Fall 2023

5

5. Chart parsing

Gerald Penn
Department of Computer Science, University of Toronto

Reading: Jurafsky & Martin: 13.3–4. Allen*: 3.4, 3.6.*
Bird et al: 8.4, online extras 8.2 to end of section “Chart Parsing in NLTK”.

Top-down parsing 1

- ***Top-down*** or ***rule-directed*** parsing:
  "Can I take these rules and match them to this input?"
  - Initial goal is an S.
  - Repeatedly look for rules that decompose /expand current goals and give new goals.
    E.g., goal of S may decompose to goals NP and VP.
  - Eventually get to goals that look at input.
    E.g., goal of NP may decompose to *det noun.*
  - Succeed iff entire input stream is accounted for as S.

Top-down parsing 2

- Example: A ***recursive descent parser***.
  ```
  >>> nltk.app.rdparser()
  ```
- Operations on ***leftmost frontier node***:
  - ***Expand*** it.
  - ***Match*** it to the next input word.

Bottom-up parsing 1

- ***Bottom-up*** or ***data-directed*** parsing:
  "Can I take this input and match it to these rules?"
  - Try to find rules that match a possible PoS of the input words …
  - … and then rules that match the constituents so formed.
  - Succeed iff the entire input is eventually matched to an S.

Bottom-up parsing 2

- Example: A ***shift–reduce parser***.
  `>>> nltk.app.srparser()`
- Operations:
  - ***Shift*** next input word onto stack.
  - Match the top *n* elements of stack to RHS of rule, ***reduce*** them to LHS.

Problems

- Neither top-down nor bottom-up search exploits useful idiosyncrasies that CFG rules, alone or together, often have.
- **Problems:**
  - Recomputation of constituents.
  - Recomputation of common prefixes.
- **Solution:** Keep track of:
  - Completed constituents.
  - Partial matches of rules.

Efficient parsing

- Want to avoid problems of blind search:
  - Avoid redoing analyses that are identical in more than one path of the search.
- Guide the analysis with both
  - the actual input
  - the expectations that follow from the choice of a grammar rule.
- Combine strengths of both top-down and bottom-up methods.

Efficient parsing

- **Want to avoid problems of blind search:**
    - **Avoid redoing analyses that are identical in more than one path of the search.**
- Guide the analysis with both
    - the actual input
    - the expectations that follow from the choice of a grammar rule.
- Combine strengths of both top-down and bottom-up methods.

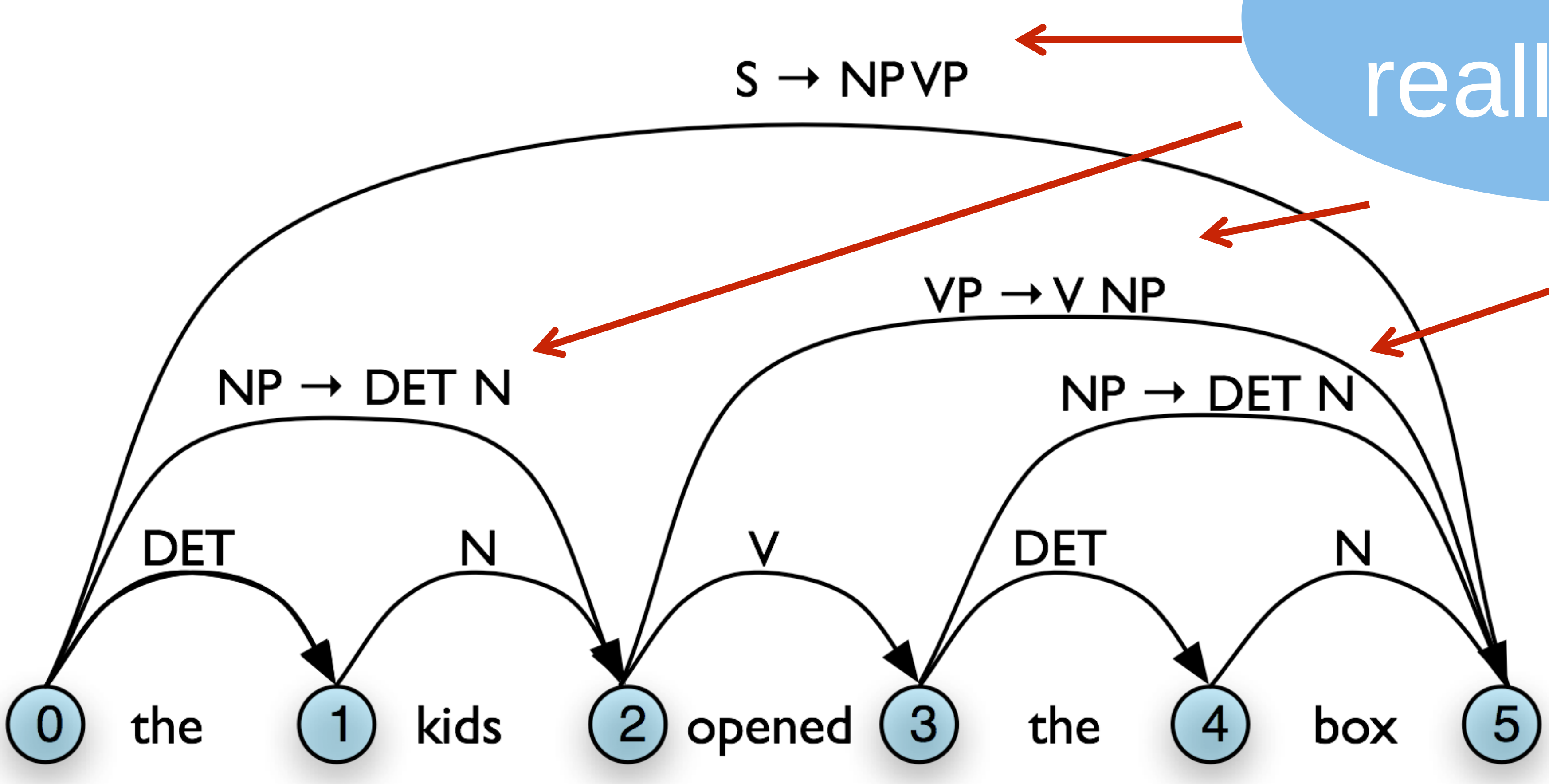
Chart parsing

- Main idea:
  - Use data structures to maintain information: a **chart** and an **agenda**
- **Agenda:**
  - List of constituents that need to be processed.
- **Chart:**
  - Records ("***memoizes***") work; obviates repetition.
  - Related ideas: Well-formed substring table (WFST); CKY parsing; Earley parsing; dynamic programming.

Charts 1

- Notation for positions in sentence from 0 to *n* (length of sentence):
- $_0$ *The* $_1$ *kids* $_2$ *opened* $_3$ *the* $_4$ *box* $_5$

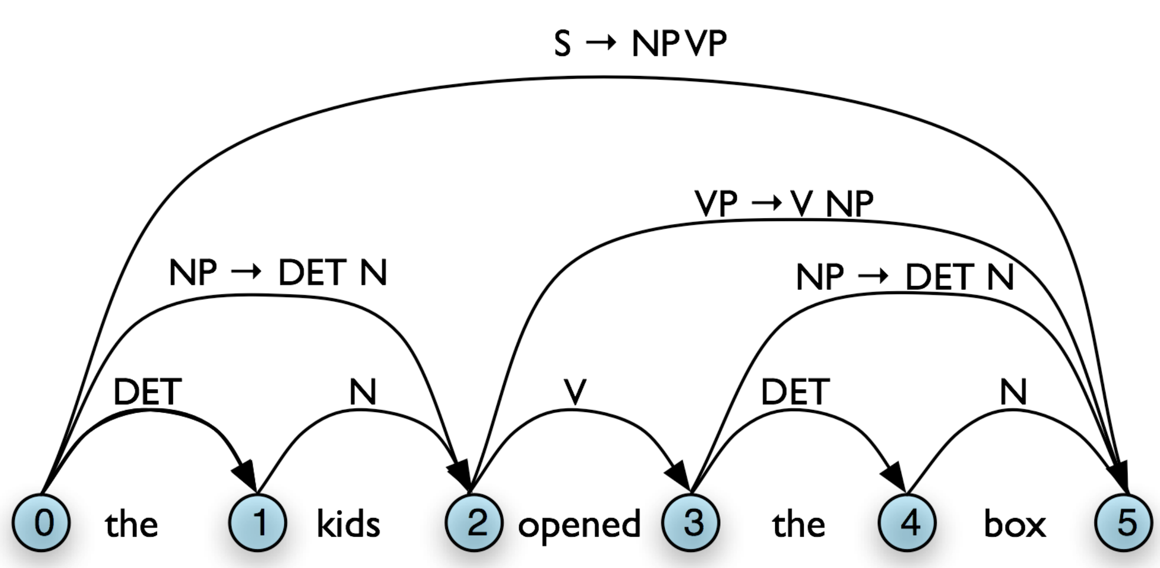
RHS not really there



From: Steven Bird, Ewan Klein, and Edward Loper, *Natural Language Processing in Python*, v. 9.5.3, July 2008. Used under Creative Commons licence.

Charts 2

- Contents of chart:
  1. Completed constituents (**inactive arcs**).
  - Representation: Labelled arc (edge) from one point in sentence to another (or same point).
  - Directed; always left-to-right (or to self).
  - Label is the *left nonterminal* of the grammar rule that derived it.

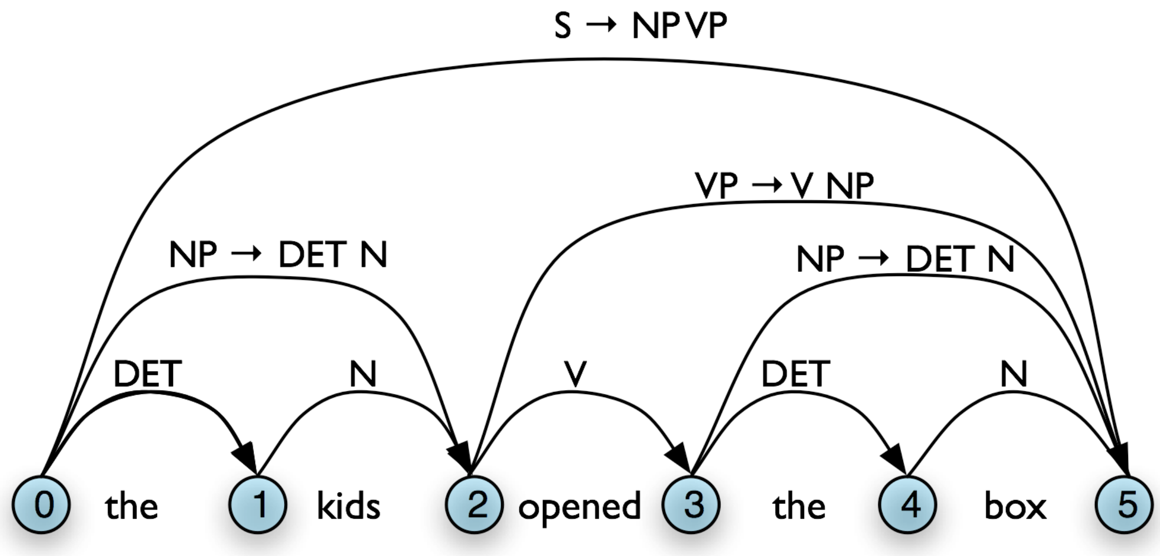


Charts 2

- Contents of chart:

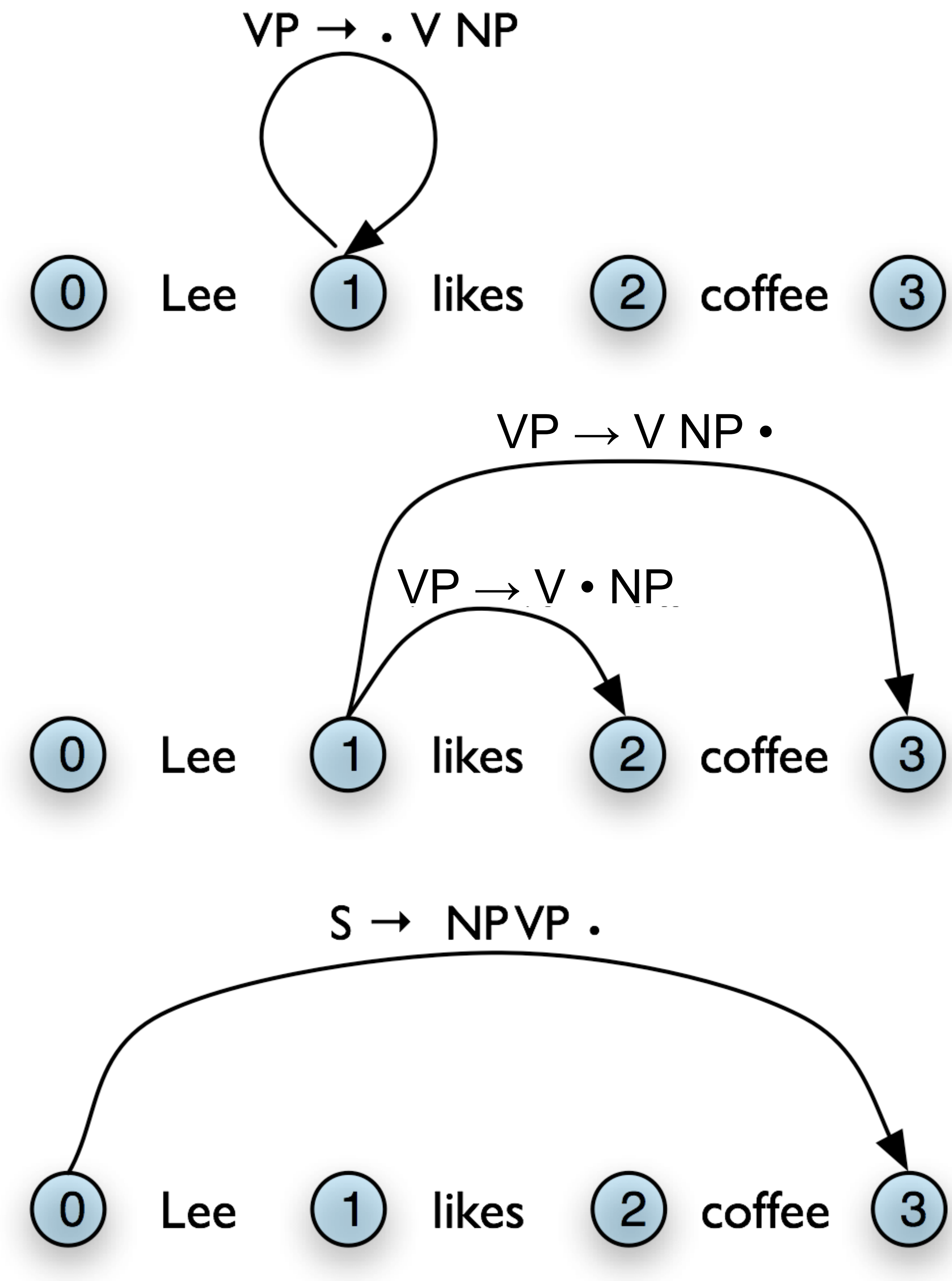
  2. Partially built constituents (also called **active arcs**).
     Can think of them as ***hypotheses***.

- Representation: Labelled arc (edge) from one point in sentence to another (or same point).
- Directed; always left-to-right (or to self).
- Label is grammar rule used for arc.



Notation for arc labels

- *Notation*: '•' means 'complete to here'.
  - $A \rightarrow X\ \ Y \bullet Z$
    'In parsing an A, we've so far seen an X and a Y, and our A will be complete once we've seen a Z.'
  - $A \rightarrow X\ \ Y\ \ Z \bullet$
    'We have seen an X, a Y, and a Z, and hence completed the parse of an A.'
  - $A \rightarrow \bullet X\ \ Y\ \ Z$
    'In parsing an A, so far we haven't seen anything.'


VP → • V NP
0 Lee 1 likes 2 coffee 3
VP → V NP •
VP → V • NP
0 Lee 1 likes 2 coffee 3
S → NP VP •
0 Lee 1 likes 2 coffee 3

Fundamental rule of chart parsing

- **Arc extension:**

Let X, Y, Z be sequences of symbols, where X and Y are possibly empty.

If the chart contains an active arc from *i* to *j* of the form

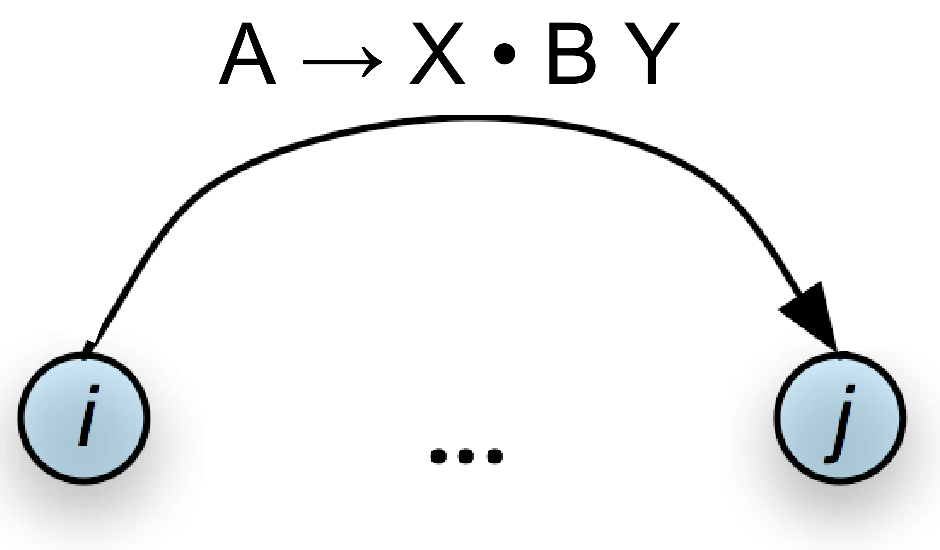
$$A \rightarrow X \bullet B\ Y$$

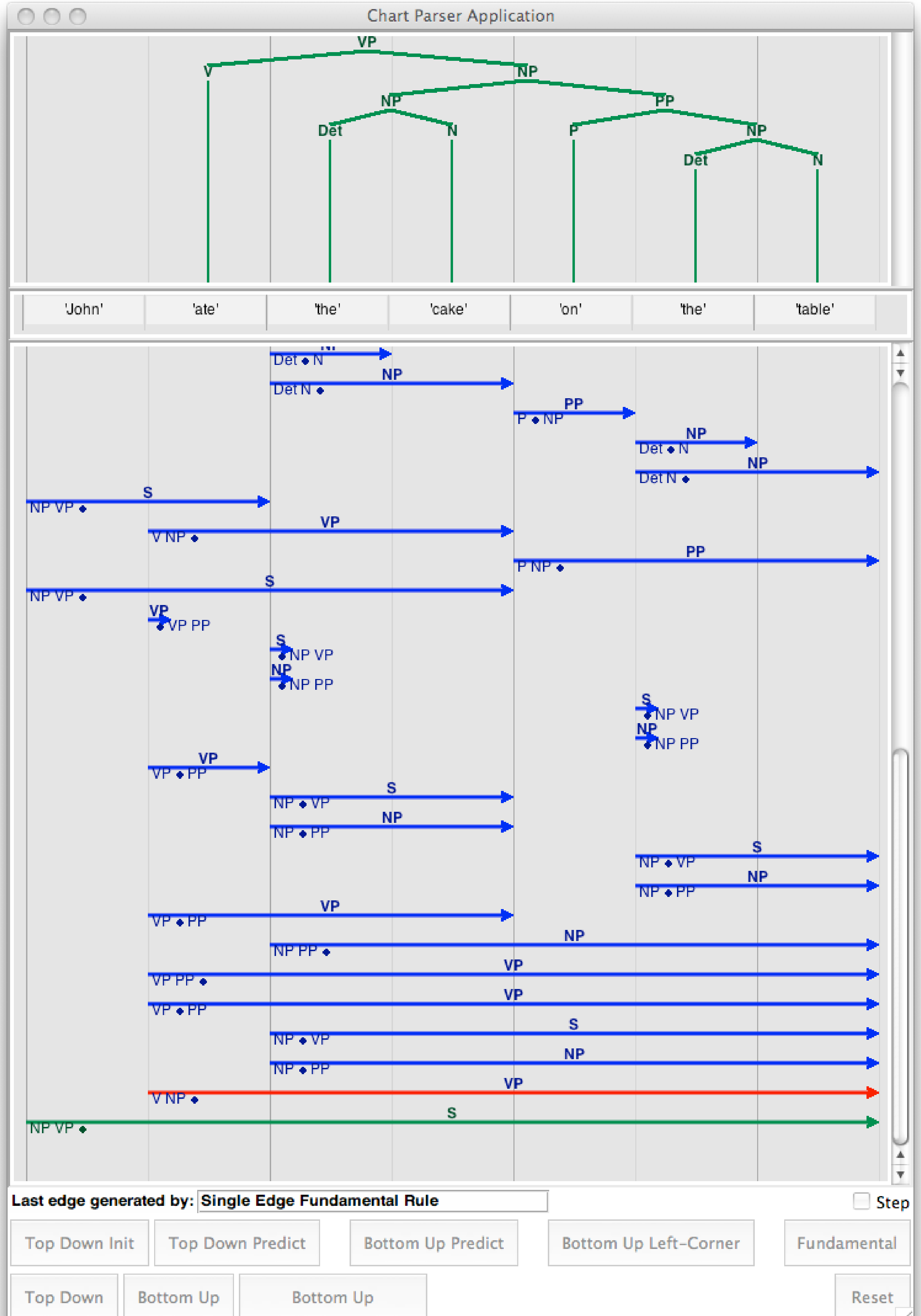
and a completed arc from *j* to *k* of the form

$B \rightarrow Z \bullet$ or $B \rightarrow \textit{word}$

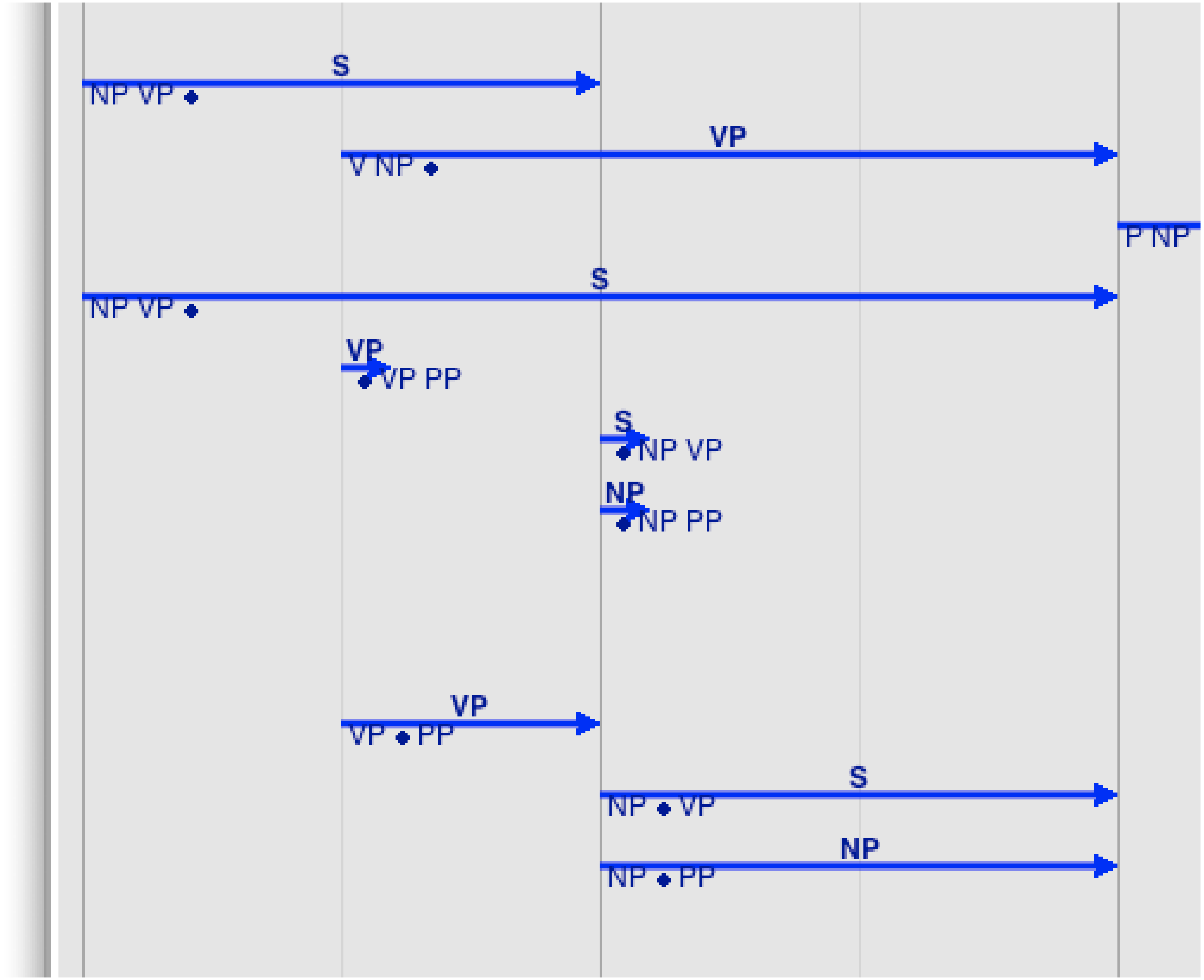
then add an arc from *i* to *k*

$$A \rightarrow X\ \ B \bullet Y$$


A → X • B Y
i
...
j



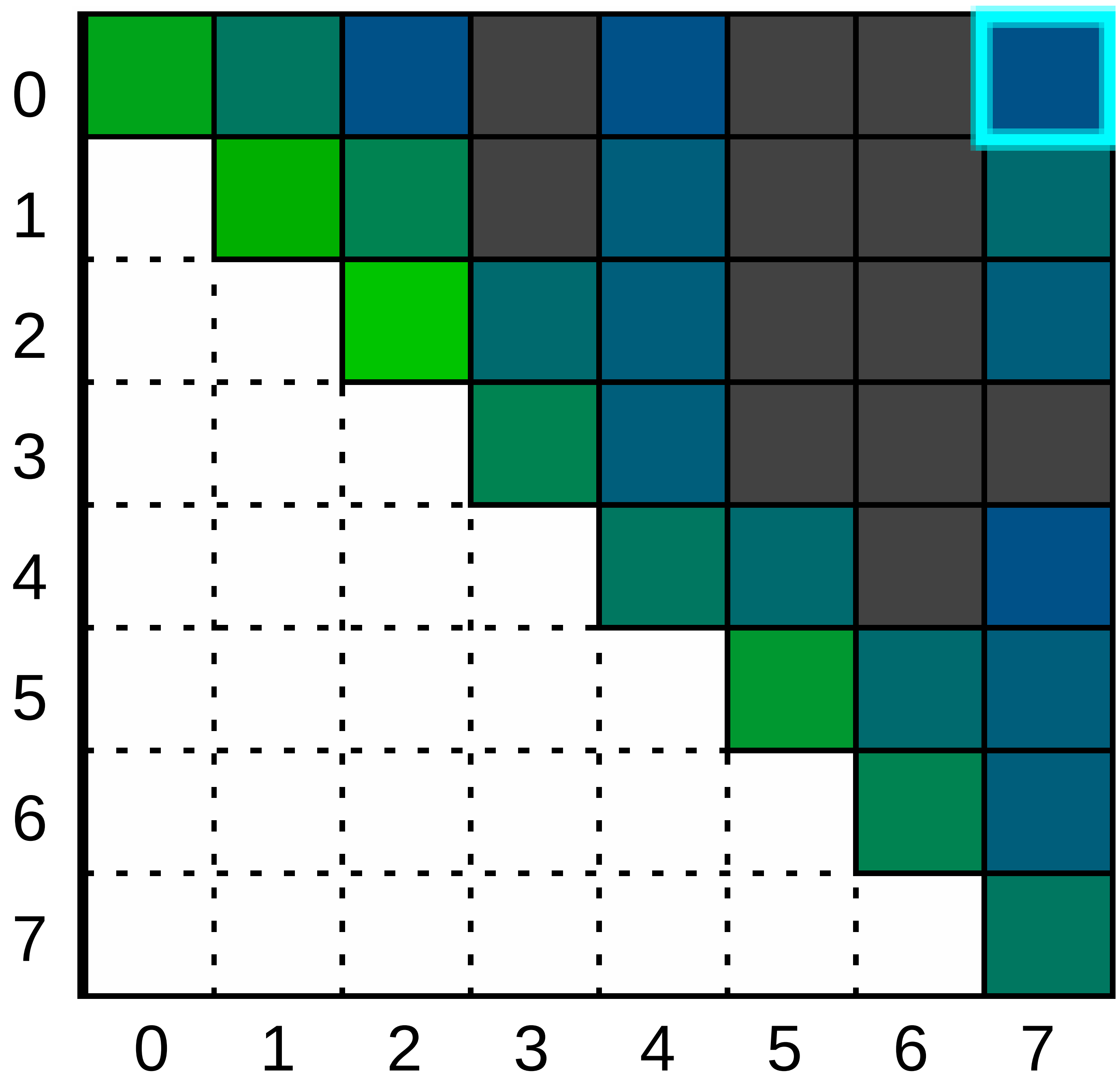
Part of a chart from the NLTK chart parser demo, `nltk.app.chartparser()`



Part of a chart from the NLTK chart parser demo, `nltk.app.chartparser()`

Charts 3

- An arc can connect any positions *i, j* ($0 \leq i \leq j \leq n$).
- Can have > 1 arc on any *i,j*…
- But only one label for any *i-j* arc!
- Can associate all arcs on positions *i,j* with cell *ij* of upper-triangular matrix.



Arcs in top right corner cell cover the whole sentence. Those for S are **parse edges**.

The matrix for a seven-word sentence from the NLTK chart parser demo
`nltk.app.chartparser()`

Bottom-up arc-addition rule

- **Arc addition** (or **prediction**):

If the chart contains an completed arc from *i* to *j* of the form

$A \rightarrow X \bullet$

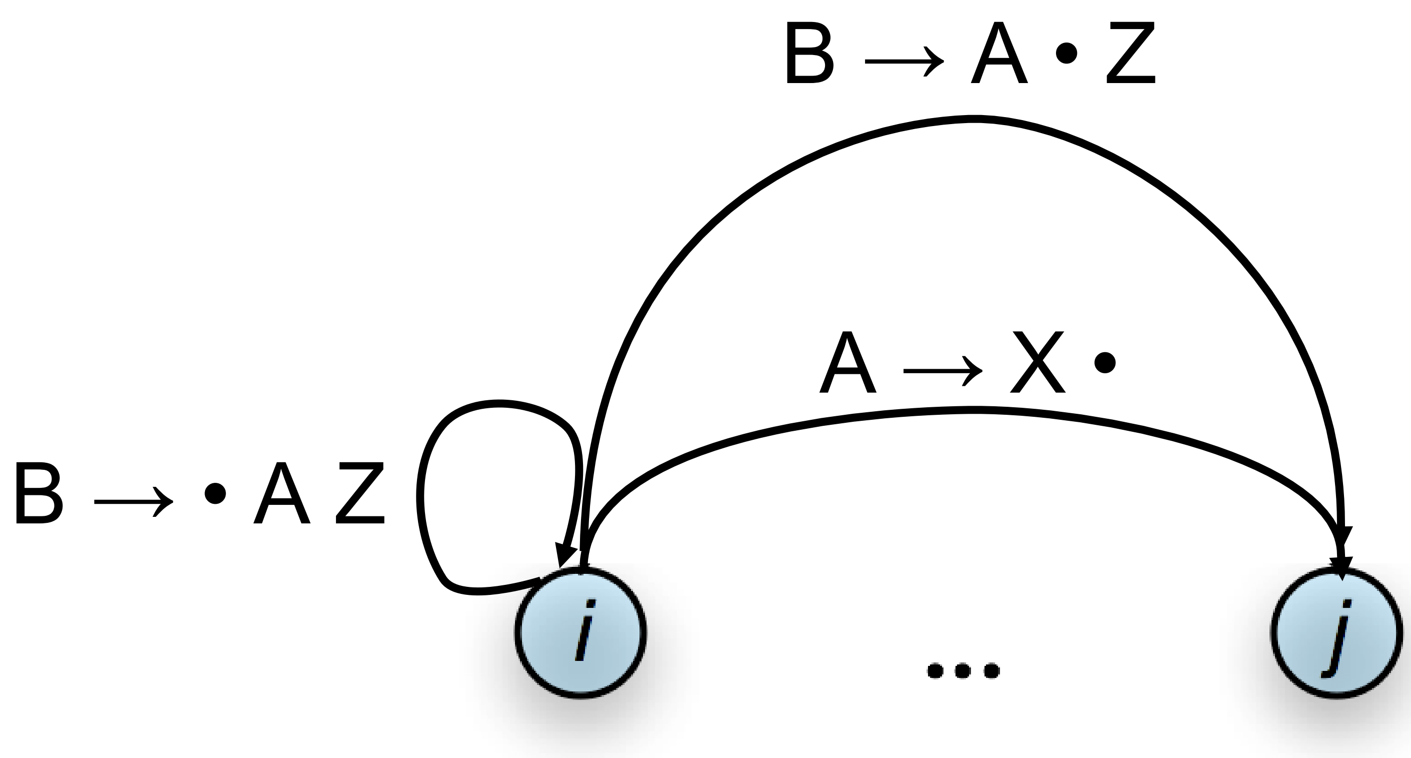
and the grammar contains a rule

$B \rightarrow A\ Z$

then add an arc from *i* to *i*

$B \rightarrow \bullet\ A\ Z$

or an arc $B \rightarrow A \bullet Z$ from *i* to *j*.



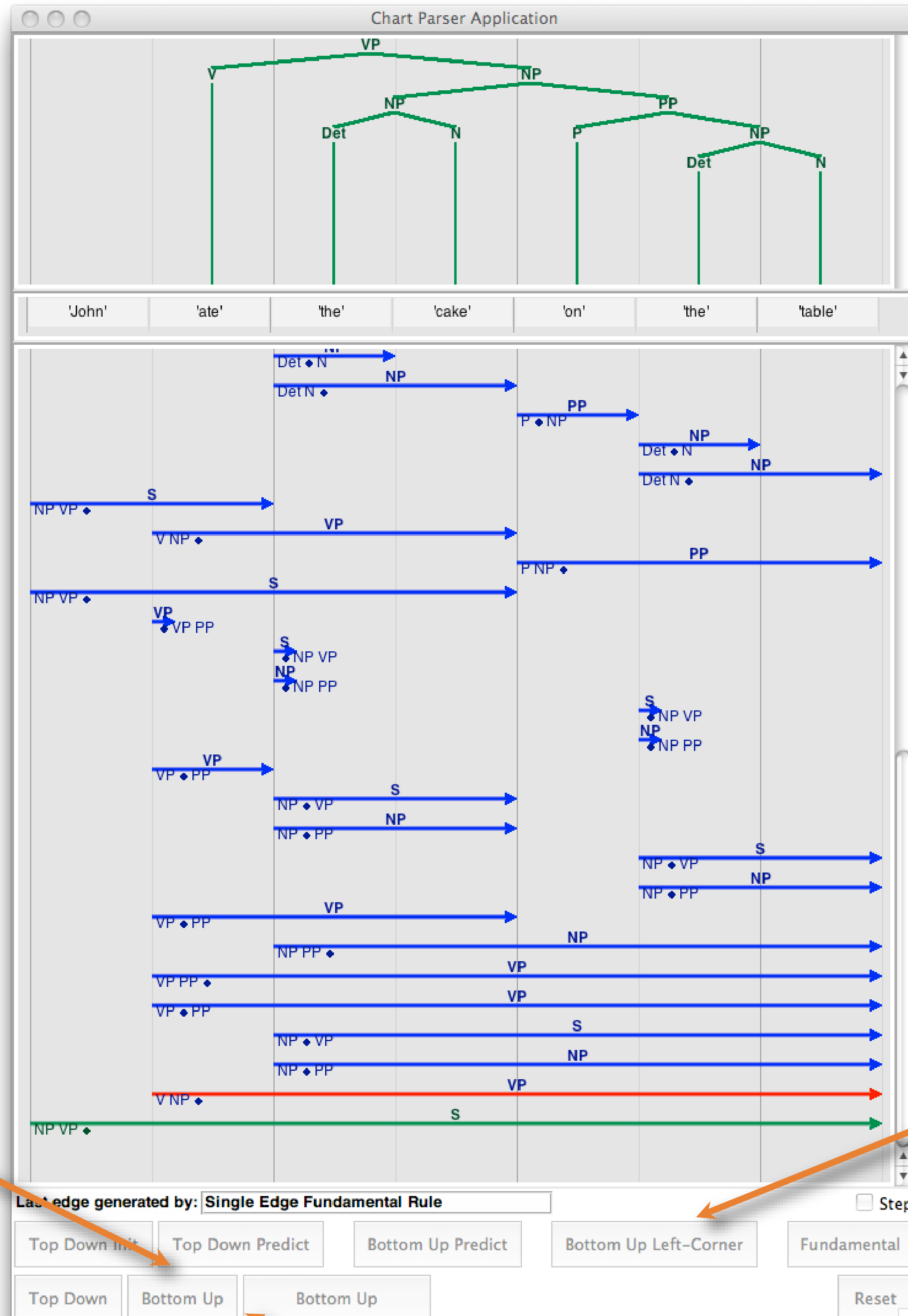
Adapted from: Steven Bird, Ewan Klein, and Edward Loper, *Natural Language Processing in Python*, v. 9.5.3, July 2008. Used under Creative Commons licence.

Bottom-up chart parsing BKL's view

- Initialize chart with each word in the input sentence (and, in effect, with their lexical categories).
- Loop until nothing more happens:
  - Apply the bottom-up prediction rule wherever you can.
  - Apply the fundamental rule wherever you can.
- Return the trees corresponding to the parse edges in the chart.

Implies that trees are built as parse progresses and are associated with each arc, or that each arc keeps pointers to the arcs of its constituents to allow post hoc reconstruction of trees.

>>> nltk.app.chartparser()



Top-down Init Rule

Top-down Predict Rule

Top-down Strategy

Bottom-up Strategy

Bottom-up Left-Corner Strategy

Bottom-up Predict Rule

Bottom-up Left-Corner Predict Rule

Fundamental Rule

Reset Parser

Observations

- Builds all constituents exactly once (almost – at least it won't add more than one inactive edge with the same label and *i-j*).
- Never re-computes the prefix of an RHS (of the same rule – it will if two rules share the same prefix).
- Exploits context-free nature of rules to reduce the search. How?

Controlling the process

- "Wherever you can": too uncontrolled.
  Try to avoid predictions and expansions that will lead nowhere.
- So use **agenda** — a list of completed arcs.
  - When an arc is completed, it is initially added to the agenda, not the chart.
  - Agenda rules decide which completed arc to move to the chart next.
  - *E.g.*, treat agenda as stack or as queue; or pick item that looks "most efficient" or "most likely"; or pick NPs first; or ….

Bottom-up chart parsing ~J&M's view

- Initialize agenda with the list of lexical categories of each word in the input sentence.
- Until agenda is empty, repeat:
  - Move next constituent *C* from agenda to chart.

  a. Find rules whose RHS starts with *C* and add corresponding active arcs to the chart.

  b. Find active arcs that continue with *C* and extend them; add the new active arcs to the chart.

  c. Find active arcs that have been completed; add their LHS as a new constituent to the agenda.

Bottom-up chart parsing algorithm 1

INITIALIZE:
set *Agenda* = list of all possible categories of each input word
(in order of input);
set *n* = length of input;
set *Chart* = ();

ITERATE:
loop
if *Agenda* = () **then**
if there is at least one *S* constituent from 0 to *n* **then**
return SUCCESS
else
return FAIL
end if
else …

Bottom-up chart parsing algorithm 2

Set $C_{i,j}$ = First(*Agenda*); /* *Remove first item from agenda.* */
/* *$C_{i,j}$ is a completed constituent of type C from position i to position j* */

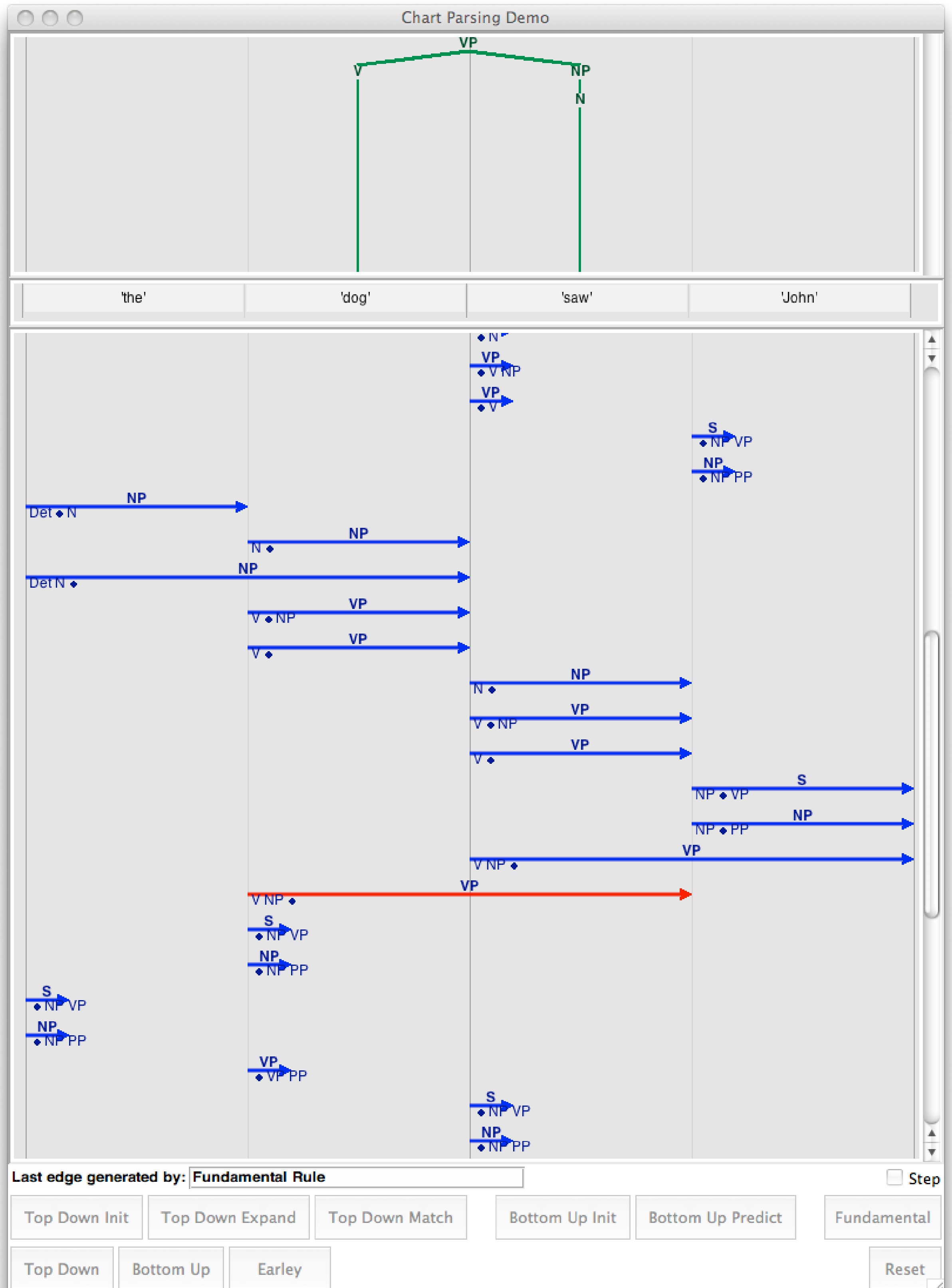
Add $C_{i,j}$ to *Chart*;

ARC UPDATE:
a. BOTTOM-UP ARC ADDITION (PREDICTION):
for each grammar rule X → C X1 … XN **do**
Add arc X → C • X1 … XN, from *i* to *j*, to *Chart;*
b. ARC EXTENSION (FUNDAMENTAL RULE):
for each arc X → X1 … • C … XN, from *k* to *i*, **do**
Add arc X → X1 … C • … XN, from *k* to *j*, to *Chart;*
c. ARC COMPLETION:
for each arc X → X1 … XN C • added in step (a) or step (b) **do**
Move completed constituent X to *Agenda*;
end if
end loop

Problem with bottom-up chart parsing

- Ignores useful top-down knowledge (rule contexts).

```
>>> nltk.app.chartparser()
```



Add ambiguity to lexicon:

N → *saw*

V → *dog*

NP → N

Parse bottom-up:

the dog saw John

Top-down chart parsing

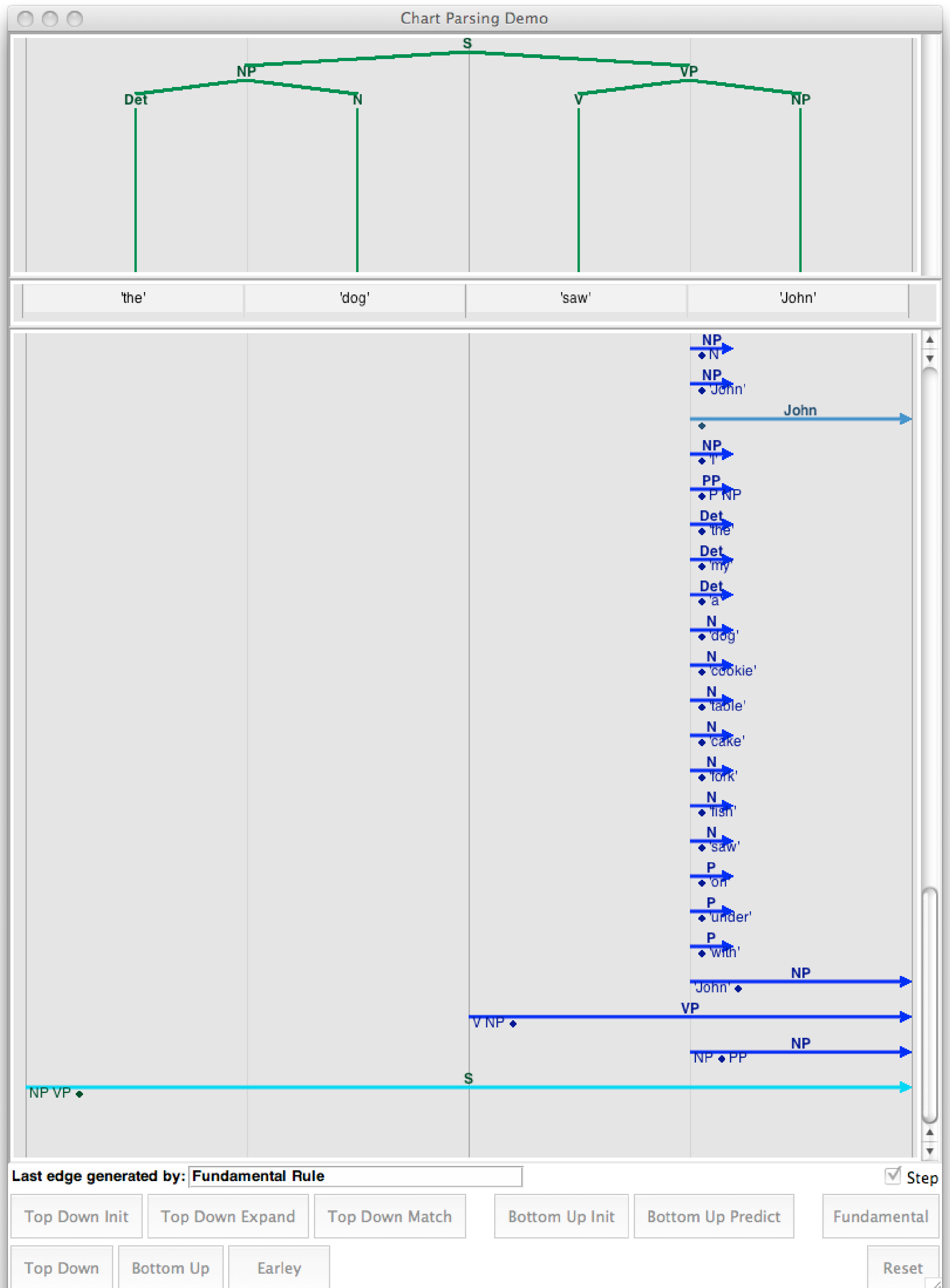
- Same as bottom-up, except new arcs are added to chart *only* if based on predictions from existing arcs.
- Initialize chart with unstarted active arcs for S.

$S \rightarrow \bullet X\ Y$

$S \rightarrow \bullet Z\ Q$

- Whenever an active arc is added, also add unstarted arcs for its next needed constituent.

```
>>> nltk.app.chartparser()
```



Add ambiguity to lexicon:

N → *saw*

V → *dog*

NP → N

Parse top-down:

the dog saw John

Top-down chart parsing algorithm 1

INITIALIZE:

set *Agenda* = list of all possible categories of each input word
(in order of input);
set *n* = length of input;
set *Chart* = ();
for each grammar rule S → X1 … XN **do**
    Add arc S → • X1 … XN to *Chart* at position 0;
    apply TOP-DOWN ARC ADDITION [step (a') below] to the new arc;
end for

ITERATE:

loop
    if *Agenda* = () **then**
        if there is at least one *S* constituent from 0 to *n* **then**
            return SUCCESS
        else
            return FAIL
        end if
    else …

Top-down chart parsing algorithm 2

Set $C_{i,j}$ = First(*Agenda*); */* Remove first item from agenda. */*

/ $C_{i,j}$ is a completed constituent of type C from position i to position j */*

Add $C_{i,j}$ to *Chart*;

ARC UPDATE:

b. ARC EXTENSION (FUNDAMENTAL RULE):

for each arc X → X1 … • C … XN, from *k* to *i*, **do**

Add arc X → X1 … C • … XN, from *k* to *j*, to *Chart;*

a’. TOP-DOWN ARC ADDITION (PREDICTION):

/ Recursive: until no new arcs can be added */*

for each arc X → X1 … • XL … XN, from *k* to *j*, added in step (b) or (a’), **do**

Add arc XL → • Y1 … YM, from *j* to *j*, to *Chart*;

c. ARC COMPLETION:

for each arc X → X1 … XN C • added in step (b) **do**

Move completed constituent X to *Agenda*;

end if

end loop

Notes on chart parsing

- Chart parsing separates:
  1. Policy for selecting constituent from agenda;
  2. Policy for adding new arcs to chart;
  3. Policy for initializing chart and agenda.
- “Top-down” and “bottom-up” now refer to arc-addition rule.
  - Initialization rule gives bottom-up aspect in either case.
- Polynomial algorithm (around $O(n^3)$), instead of exponential.