1. Dynamic Programming

It seems like we should be able to avoid the kind of repeated reparsing a simple recursive descent parser must often do.

A CFG parser, that is, a context-free parser, should be able to avoid re-analyzing sub-strings

- because the analysis of any sub-string is independent of the rest of the parse.

The parser’s exploration of its search space can exploit this independence

- if the parser uses dynamic programming.

Dynamic programming is the basis for all chart parsing algorithms.

2. Parsing as dynamic programming

Given a problem, dynamic programming systematically fills a table of solutions to sub-problems

- A process sometimes called memoisation.

Once solutions to all sub-problems have been accumulated

- DP solves the overall problem by composing them.

For parsing, sub-problems are analyses of sub-strings

- which can be memoised
- in a chart.
• also know as a **well-formed substring table**, WFST

Each entry in the chart or WFST corresponds to either:

• a complete **constituent** (sub-tree), indexed by the start and end of the sub-string that it covers;
• or a **hypothesis** about what complete constituent might be found, indexed by the start and end of any constituent sub-strings found so far

### 3. Depicting a WFST/Chart

A well-formed substring table (aka chart) can be depicted as either a **matrix** or a **graph**

- Both contain the same information

When a **WFST** (aka **chart**) is depicted as a matrix:

- Rows and columns of the matrix correspond to the start and end positions of a span of items from the input
  - That is, starting **right before** the first word, ending **right after** the final one
- A cell in the matrix corresponds to the sub-string that starts at the row index and ends at the column index
- A cell can contain
  - information about the **type** of constituent (or constituents) that span(s) the substring
  - pointers to its sub-constituents
  - If the constituent is incomplete, **predictions** about what constituents might follow the substring

### 4. Depicting a WFST as a matrix

Here's a sample matrix, part-way through a parse

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Prep</td>
<td></td>
<td>PP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Det</td>
<td>NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

0 See 1 with 2 a 3 telescope 4 in 5 hand 6
We can read this as saying:

- There is a **PP** from 1 to 4
  - *Because* there is a **Prep** from 1 to 2
  - *and* an **NP** from 2 to 4

5. Depicting a WFST as a graph

A sample graph, for the same situation mid-parse

- Here, **nodes** (or **vertices**) represent positions in the text string, starting **before** the first word, ending **after** the final word.
- **arcs** (or **edges**) connect vertices at the start and the end of a span to represent a particular substring
  - Edges can be labelled with the same information as in a cell in the matrix representation

6. Algorithms for chart parsing

Important examples of parser types which use a WFST include:

- The CKY algorithm, which memoises only complete constituents
- Three algorithm families that involve memoisation of both complete *and* incomplete constituents
  - Incomplete constituents can be understood as **predictions**
    - **bottom-up chart parsers**
      - May include top-down filtering
    - **top-down chart parsers**
      - May include bottom-up filtering
    - **the Earley algorithm**

7. CKY Algorithm

CKY (Cocke, Kasami, Younger) is an algorithm for recognising constituents and recording them in the chart (WFST).

CKY was originally defined for Chomsky Normal Form
We can enter constituent $A$ in cell $(i,j)$ iff either

- there is a rule $A \rightarrow b$ and
  - $b$ is found in cell $(i,j)$
- or if there is a rule $A \rightarrow B \ C$ and
  - $B$ is found in cell $(i,k)$
  - $C$ is found in cell $(k,j)$

Proceeding systematically, CKY guarantees that the parser only looks for rules that use a constituent from $i$ to $j$ after it has processed all the constituents that end at $i$

- This guarantees that every possible constituent will be found

Note that this last remark illustrates the fundamental weakness of blind bottom-up parsing:

- Large numbers of constituents will be found which do not participate in the ultimately spanning 'correct' analyses.

8. Visualising the chart: YACFG

<table>
<thead>
<tr>
<th>Grammatical rules</th>
<th>Lexical rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>Det $\rightarrow$ a</td>
</tr>
<tr>
<td>$NP \rightarrow Det \ Nom$</td>
<td>$N \rightarrow$ fish</td>
</tr>
<tr>
<td>$NP \rightarrow Nom$</td>
<td>Prep $\rightarrow$ in</td>
</tr>
<tr>
<td>Nom $\rightarrow N$ SRel</td>
<td>TV $\rightarrow$ saw</td>
</tr>
<tr>
<td>Nom $\rightarrow N$</td>
<td>IV $\rightarrow$ fish</td>
</tr>
<tr>
<td>VP $\rightarrow TV$ NP</td>
<td>Relpro $\rightarrow$ that (relative pronoun)</td>
</tr>
<tr>
<td>VP $\rightarrow IV$ PP</td>
<td></td>
</tr>
<tr>
<td>VP $\rightarrow IV$</td>
<td></td>
</tr>
<tr>
<td>PP $\rightarrow$ Prep NP</td>
<td></td>
</tr>
<tr>
<td>SRel $\rightarrow$ Relpro VP</td>
<td></td>
</tr>
</tbody>
</table>

Nom: nominal (the part of the NP after the determiner, if any)

SRel: subject relative clause, as in the frogs that ate fish.

9. Visualising the chart: the setup

Just the empty matrix
10. Visualising the Chart (0,1)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Unary branching rules: det → the

the frogs ate fish

the frogs ate fish
11. Visualising the Chart (1,2)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>n</td>
<td>nom</td>
<td>np</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the  frogs  ate  fish

Unary branching rules: $N \rightarrow$ frogs, $Nom \rightarrow N$, $NP \rightarrow Nom$

12. Visualising the Chart (2,3)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>n</td>
<td>nom</td>
<td>np</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>tv</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the  frogs  ate  fish
Unary branching rules: $tv \rightarrow ate$

13. Visualising the Chart (3,4)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>n, nom, np</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>tv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>n, nom, np, iv, vp</td>
<td></td>
</tr>
</tbody>
</table>

the frogs ate fish

Unary branching rules: $N \rightarrow frogs$, $Nom \rightarrow N$, $NP \rightarrow Nom$, $iv \rightarrow fish$, $vp \rightarrow iv$
14. Visualising the Chart (0,2)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>det</td>
<td>np</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>nom</td>
<td>np</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>tv</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>nom</td>
<td>np</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>iv</td>
<td>vp</td>
</tr>
</tbody>
</table>

the  frogs  ate  fish

Binary branching rule: NP → Det Nom

• (0,1) & (1,2) ⇒ (0,2)
15. Visualising the Chart (1,3)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>det</td>
<td>np</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>n</td>
<td>nom</td>
<td>np</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>tv</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>n</td>
</tr>
</tbody>
</table>

the frogs ate fish

- (1,2) & (2,3) ⊢

16. Visualising the Chart (2,4)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>det</td>
<td>np</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>n</td>
<td>nom</td>
<td>np</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>tv</td>
<td></td>
<td>vp</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>n</td>
</tr>
</tbody>
</table>

the frogs ate fish

Binary branching rule: VP → TV NP
17. Visualising the Chart (1,4)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>det</td>
<td>np</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>nm</td>
<td>np</td>
<td>s</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>tv</td>
<td>vp</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>nm</td>
<td>np</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>iv</td>
<td>vp</td>
</tr>
</tbody>
</table>

- \((1,2) \& (2,4) \Rightarrow (1,4)\)
- \((1,3) \& (3,4) \not\Rightarrow\)

Binary branching rule: \(S \rightarrow NP \ VP\)

- \((1,2) \& (2,4) \Rightarrow (1,4)\)
- \((1,3) \& (3,4) \not\Rightarrow\)
18. Visualising the Chart (0,4)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>det</td>
<td>np</td>
<td></td>
<td>s</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>n</td>
<td>nom</td>
<td>np</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>tv</td>
<td></td>
<td>vp</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>n</td>
<td>nom</td>
</tr>
</tbody>
</table>

the frogs ate fish

Binary branching rule: \( S \rightarrow NP \ VP \)

- \((0,1) \& (1,4) \nRightarrow \)
- \((0,2) \& (2,4) \Rightarrow (0,4) \)
- \((0,3) \& (3,4) \nRightarrow \)

19. From CKY Recogniser to CKY Parser

We cannot tell from the CKY chart as specified, the syntactic analysis of the input string.

We just have a chart recogniser, a way of determining whether a string belongs to the language generated by the grammar.

Changing this to a parser requires recording which existing constituents were combined to make each new constituent. This requires another field to record the one or more ways in which a constituent spanning \((i,j)\) can be made from constituents spanning \((i,k)\) and \((k,j)\).

20. 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:

\[ VP \rightarrow V \ NP \]
The chart will show us both ways to get the VP from an ambiguous input:

21. 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

22. 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.

23. 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.

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

24. 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

25. 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

**26. 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

**Bottom-up LIFO Chart parse example**

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) $D \rightarrow \text{ the}</td>
<td>\ldots$</td>
</tr>
<tr>
<td>B) $N \rightarrow \text{ dog}</td>
<td>\ldots$</td>
</tr>
<tr>
<td>C) $NP \rightarrow D N$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Agenda</th>
<th>Chart</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $L$</td>
<td>1 dog</td>
</tr>
<tr>
<td>2 $L$</td>
<td>0 the</td>
</tr>
</tbody>
</table>
Bottom-up LIFO Chart parse example

Grammar

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

... 

Input

"the dog"

Agenda

<table>
<thead>
<tr>
<th>Level</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 dog 2</td>
</tr>
<tr>
<td>2</td>
<td>0 the 1</td>
</tr>
</tbody>
</table>

Chart

```
  0                  1
  |                  |
  | the             |
  |                 |
  | 0 → 1           |
```

A new inactive edge in the chart: we check the grammar
Active meets inactive for the first time -- the fundamental rule is tried

And succeeds
New inactive edge: we check the grammar
Active met inactive again also, but *didn't* match

**Bottom-up LIFO Chart parse example**

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) $D \rightarrow \text{the} \mid ...$</td>
<td>&quot;the dog&quot;</td>
</tr>
<tr>
<td>B) $N \rightarrow \text{dog} \mid ...$</td>
<td></td>
</tr>
<tr>
<td>C) $NP \rightarrow D N$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Agenda</th>
<th>Chart</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $L$ 1 dog 2</td>
<td><img src="chart1.png" alt="Chart Diagram" /></td>
</tr>
<tr>
<td>2 $L$ 0 the 1</td>
<td></td>
</tr>
<tr>
<td>3 $BU(2,A)$ 0 $D \rightarrow \cdot \text{the} 0$</td>
<td></td>
</tr>
<tr>
<td>4 $FR(3,2)$ 0 $D [\text{the}] 1$</td>
<td></td>
</tr>
<tr>
<td>5 $BU(4)$ 0 $NP \rightarrow \cdot D N 0$</td>
<td></td>
</tr>
</tbody>
</table>

New active means *two* instances of the fundamental rule to check
One wins, giving us our first partial constituent

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) \( D \rightarrow \text{the} \mid \ldots \)
B) \( N \rightarrow \text{dog} \mid \ldots \)
C) \( NP \rightarrow D \cdot N \)

Input

"the dog"

Agenda

1 L 1 dog
2 L 0 the
3 BU(2,A) 0 D \rightarrow \text{the} 0
4 FR(3,2) 0 D [the] 1
5 BU(4) 0 NP \rightarrow D \cdot N 0
6 FR(5,4) 0 NP \rightarrow D \cdot N 1
7 BU(1,B) 1 N \rightarrow \text{dog} 1

Chart

Bottom-up LIFO Chart parse example

Grammar

A) \( D \rightarrow \text{the} \mid \ldots \)
B) \( N \rightarrow \text{dog} \mid \ldots \)
C) \( NP \rightarrow D \cdot N \)

Input

"the dog"

Agenda

1 L 1 dog
2 L 0 the
3 BU(2,A) 0 D \rightarrow \text{the} 0
4 FR(3,2) 0 D [the] 1
5 BU(4) 0 NP \rightarrow D \cdot N 0
6 FR(5,4) 0 NP \rightarrow D \cdot N 1
7 BU(1,B) 1 N \rightarrow \text{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 L  0 the
3 BU(2,A)  0 D→ · the
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 L  0 the
3 BU(2,A)  0 D→ · the
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
Chart parsing and left recursion

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

- **NP → NP PP**
- **NP → Det Nom**
- **Det → '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