We are not limited to recording complete productions (i.e., ones whose entire RHS have been recognized): We could also consider recording incomplete productions (i.e., ones for which there may so far be only partial evidence).

Incomplete productions (aka incomplete constituents) are effectively predictions about what might come next and what will be learned from finding it.

Incomplete constituents can be represented using an extended form of production rule called a dotted rule.

The dot indicates how much of the RHS has already been found. The rest is a prediction of what is to come.

Incomplete constituent is a useful concept: What else have you seen that indicates this?

The grammar rule $VP \rightarrow V \ NP$ yields the following dotted rules:

- $VP \rightarrow . \ V \ NP$ incomplete edge
- $VP \rightarrow V . \ NP$ incomplete edge
- $VP \rightarrow V \ NP .$ complete edge

Just as we can record in the chart the production rules for complete constituents, we can record the dotted rules for incomplete constituents, noting:

- the production rule used in the analysis, along with the start and end position of the material already found;
- the part of the rule already found (to the left of the dot), and the part still predicted to be found (to the right of the dot).

Thus the chart could record for the substring 

```
..5 on the 7 floor
```

indicating the word between 6 and 7 is spanned by Det, and an N is predicted next. If found, we get an NP starting at 6.
Dotted Rules

The extreme dotted rule has the dot at the left end of the RHS, meaning nothing on the RHS has yet been found, and everything is predicted.

Example of a chart labelled with dotted rules (in graph representation):

```
NP −→ Det N.
N −→ floor.
NP −→ Det N.
Det −→ the.
NP −→ Det N.
5 7 6 8
on the floor
the −→ .
on −→ .
Prep −→ on.
floor −→ .
```

Adding Edges

An active chart parser can add an edge for any of three reasons:

1. The input can license an edge. In particular, each word \( w_i \) in the input licenses the complete edge \([w_i \rightarrow \bullet, (i, i+1)]\);

2. The grammar can license an edge. In particular, each grammar production \( A \rightarrow \alpha \) licenses a self-loop edge \([A \rightarrow \bullet \alpha, (i, i)]\) for every \( i, 0 \leq i < n \), where \( n \) is the length of the string.

3. The contents of the current chart can license an edge.

It rarely makes sense to add all licensed edges to a chart, since many of them will not be used in any complete parse.

Fundamental Rule

The strategy of an active chart parser comprises the rules used to decide when to add an edge, along with a specification of when rules should be applied.

Every active chart parser uses what’s called the Fundamental Rule to combine an incomplete edge with a complete one.

If the chart contains the edges

\[
[A \rightarrow \alpha \bullet B \beta, (i, j)] \quad \text{and} \quad [B \rightarrow \gamma \bullet, (j, k)]
\]

then add the new edge \([A \rightarrow \alpha B \beta \bullet, (i, k)]\)

where \( \alpha, \beta, \) and \( \gamma \) are (possibly empty) sequences of terminals or non-terminals.

The dot moves one place right, and the new edge spans the other two.

Bottom-up Active Parsing

As with the CKY Parser, Bottom-up Active Parsing starts from the input string, identifying sequences of words and phrases that correspond to the RHS of a grammar production.

Bottom-up Initialization Rule: For every word \( w_i \), add the edge \([w_i \rightarrow \bullet, (i, i+1)]\);
**Bottom-up Active Parsing**

**Bottom-up Predict Rule:** If the chart contains the complete edge

$$[A \rightarrow \alpha \cdot, (i, j)]$$

and the grammar contains the production $$B \rightarrow A \beta$$

then add the self-loop edge $$[B \rightarrow \cdot A \beta, (i, i)]$$

---

**Top-down Active Parsing**

Like Bottom-up Active Parsing, TD Active Parsing uses an Initialization Rule and the Fundamental Rule. But it also has rules for making self-looping TD hypotheses (TD Expand) and for consuming words in the input (TD Match).

**Top-Down Initialization Rule:** For every grammar production of the form:

$$S \rightarrow \alpha$$

add the self-loop edge $$[S \rightarrow \cdot \alpha, (0, 0)]$$
Top-down Active Parsing

Top-Down Match Rule consumes a word from the input string: If the chart contains the incomplete edge

\[ A \rightarrow \alpha \cdot w_j \beta , (i, j) \]

where \( w_j \) is the \( j \)th word of the string, add a new edge \( [w_j \rightarrow \cdot , (j, j+1)] \)

Top-down Active Parsing Strategy

- Create an empty chart spanning the sentence.
- Apply the Top-Down Initialization Rule.
- Until no more edges are added:
  - Apply the Top-Down Expand Rule everywhere it applies.
  - Apply the Top-Down Match Rule everywhere it applies.
  - Apply the Fundamental Rule everywhere it applies.
- Return all of the parse trees corresponding to the parse edges in the chart.

Earley Algorithm

The Earley algorithm is a bottom-up chart parser with top-down prediction.

- **Predictor:** TD Initialization Rule + TD Expansion Rule
- **Scanner:** TD/BU Match Rule
- **Completer:** Fundamental Rule

Unlike the BU Active Chart Parser, edges are added in strictly left-to-right order. If \( A \rightarrow X \cdot B, [i,j] \) is added before \( C \rightarrow Y \cdot D, [i',j'] \) then \( j \leq j' \).

Koehn ANLP Lecture 13 19 October 2010
Earley Algorithm

Scanner Rule uses parts of speech rather than words: If the chart contains the incomplete edge

$$[A \rightarrow \alpha \cdot P \beta, (i, j)]$$

and $w_j$ is the $j^{th}$ word of the input string and $P$ is a valid part of speech for $w_j$, then add the new complete edges:

$$[P \rightarrow w_j \cdot, (j, j + 1)]$$
$$[w_j \rightarrow \cdot, (j, j + 1)]$$

Visualizing the Chart

Grammatical rules  | Lexical rules
--- | ---
$S \rightarrow NP \ VP$ | Det $\rightarrow$ a | the (determiner)
$NP \rightarrow$ Det Nom | N $\rightarrow$ fish | frogs | soup (noun)
$NP \rightarrow$ Nom | Prep $\rightarrow$ in | for (preposition)
Nom $\rightarrow$ N SRel | TV $\rightarrow$ saw | ate (transitive verb)
Nom $\rightarrow$ N | IV $\rightarrow$ fish | swim (intransitive verb)
VP $\rightarrow$ TV NP | Relpro $\rightarrow$ that (relative pronoun)
VP $\rightarrow$ IV PP
VP $\rightarrow$ IV
PP $\rightarrow$ Prep NP
SRel $\rightarrow$ Relpro VP

Step 0

<table>
<thead>
<tr>
<th>$j = 0$</th>
<th>label</th>
<th>begin</th>
<th>end</th>
<th>reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\gamma \rightarrow . S$</td>
<td>0</td>
<td>0</td>
<td>TD initialise</td>
</tr>
<tr>
<td>1</td>
<td>$S \rightarrow . NP \ VP$</td>
<td>0</td>
<td>0</td>
<td>TD expand</td>
</tr>
<tr>
<td>2</td>
<td>$NP \rightarrow . Det \ Nom$</td>
<td>0</td>
<td>0</td>
<td>predict from 1</td>
</tr>
<tr>
<td>3</td>
<td>$NP \rightarrow . Nom$</td>
<td>0</td>
<td>0</td>
<td>predict from 1</td>
</tr>
<tr>
<td>4</td>
<td>$Nom \rightarrow . N \ SRel$</td>
<td>0</td>
<td>0</td>
<td>predict from 3</td>
</tr>
<tr>
<td>5</td>
<td>$Nom \rightarrow . N$</td>
<td>0</td>
<td>0</td>
<td>predict from 3</td>
</tr>
</tbody>
</table>
### Step 1

<table>
<thead>
<tr>
<th>j = 1</th>
<th>label</th>
<th>begin</th>
<th>end</th>
<th>reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>N → fish .</td>
<td>0</td>
<td>1</td>
<td>scan</td>
</tr>
<tr>
<td>7</td>
<td>IV → fish .</td>
<td>0</td>
<td>1</td>
<td>scan</td>
</tr>
<tr>
<td>8</td>
<td>Nom → N . SRel</td>
<td>0</td>
<td>1</td>
<td>complete from 4 using 6</td>
</tr>
<tr>
<td>9</td>
<td>Nom → N .</td>
<td>0</td>
<td>1</td>
<td>complete from 5 using 6</td>
</tr>
<tr>
<td>10</td>
<td>NP → Nom .</td>
<td>0</td>
<td>1</td>
<td>complete from 3 using 9</td>
</tr>
<tr>
<td>11</td>
<td>SRel → . Relpro VP</td>
<td>1</td>
<td>1</td>
<td>predict from 8</td>
</tr>
<tr>
<td>12</td>
<td>S → NP . VP</td>
<td>0</td>
<td>1</td>
<td>complete from 1 using 10</td>
</tr>
<tr>
<td>13</td>
<td>VP → . TV NP</td>
<td>1</td>
<td>1</td>
<td>predict from 12</td>
</tr>
<tr>
<td>14</td>
<td>VP → . IV</td>
<td>1</td>
<td>1</td>
<td>predict from 12</td>
</tr>
<tr>
<td>15</td>
<td>VP → . IV PP</td>
<td>1</td>
<td>1</td>
<td>predict from 12</td>
</tr>
</tbody>
</table>

### Step 2

<table>
<thead>
<tr>
<th>j = 2</th>
<th>label</th>
<th>begin</th>
<th>end</th>
<th>reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>IV → swim .</td>
<td>1</td>
<td>2</td>
<td>scan</td>
</tr>
<tr>
<td>17</td>
<td>VP → IV .</td>
<td>1</td>
<td>2</td>
<td>complete from 14 using 16</td>
</tr>
<tr>
<td>18</td>
<td>S → NP VP .</td>
<td>0</td>
<td>2</td>
<td>complete from 12 using 17</td>
</tr>
<tr>
<td>19</td>
<td>VP → IV . PP</td>
<td>1</td>
<td>2</td>
<td>complete from 15 using 16</td>
</tr>
<tr>
<td>20</td>
<td>PP → . Prep NP</td>
<td>2</td>
<td>2</td>
<td>predict from 19</td>
</tr>
</tbody>
</table>

### Step 3

<table>
<thead>
<tr>
<th>j = 3</th>
<th>label</th>
<th>begin</th>
<th>end</th>
<th>reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Prep → in .</td>
<td>2</td>
<td>3</td>
<td>scan</td>
</tr>
<tr>
<td>22</td>
<td>PP → Prep . NP</td>
<td>2</td>
<td>3</td>
<td>complete from 20 using 21</td>
</tr>
<tr>
<td>23</td>
<td>NP → . Det Nom</td>
<td>3</td>
<td>3</td>
<td>predict from 22</td>
</tr>
<tr>
<td>24</td>
<td>NP → . Nom</td>
<td>3</td>
<td>3</td>
<td>predict from 22</td>
</tr>
<tr>
<td>25</td>
<td>Nom → . N SRel</td>
<td>3</td>
<td>3</td>
<td>predict from 24</td>
</tr>
<tr>
<td>26</td>
<td>Nom → . N</td>
<td>3</td>
<td>3</td>
<td>predict from 24</td>
</tr>
</tbody>
</table>

### Step 4

<table>
<thead>
<tr>
<th>j = 4</th>
<th>label</th>
<th>begin</th>
<th>end</th>
<th>reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Det → the .</td>
<td>3</td>
<td>4</td>
<td>scan</td>
</tr>
<tr>
<td>27</td>
<td>NP → Det . Nom</td>
<td>3</td>
<td>4</td>
<td>complete from 23 using 26</td>
</tr>
<tr>
<td>28</td>
<td>Nom → . N SRel</td>
<td>4</td>
<td>4</td>
<td>predict from 27</td>
</tr>
<tr>
<td>29</td>
<td>Nom → . N</td>
<td>4</td>
<td>4</td>
<td>predict from 27</td>
</tr>
</tbody>
</table>
Step 5

<table>
<thead>
<tr>
<th>j = 5</th>
<th>label</th>
<th>begin</th>
<th>end</th>
<th>reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>N → soup .</td>
<td>4</td>
<td>5</td>
<td>scan</td>
</tr>
<tr>
<td>31</td>
<td>Nom → N . SRel</td>
<td>4</td>
<td>5</td>
<td>complete from 28 using 30</td>
</tr>
<tr>
<td>32</td>
<td>SRel → . Relpro NP</td>
<td>5</td>
<td>5</td>
<td>predict from 31</td>
</tr>
<tr>
<td>33</td>
<td>Nom → N .</td>
<td>4</td>
<td>5</td>
<td>complete from 29 using 30</td>
</tr>
<tr>
<td>34</td>
<td>NP → Det Nom .</td>
<td>3</td>
<td>5</td>
<td>complete from 27 using 33</td>
</tr>
<tr>
<td>35</td>
<td>PP → Prep NP .</td>
<td>2</td>
<td>5</td>
<td>complete from 22 using 34</td>
</tr>
<tr>
<td>36</td>
<td>VP → IV PP .</td>
<td>1</td>
<td>5</td>
<td>complete from 19 using 35</td>
</tr>
<tr>
<td>37</td>
<td>S → NP VP .</td>
<td>0</td>
<td>5</td>
<td>complete from 12 using 36</td>
</tr>
</tbody>
</table>

Too Many Grammar Rules!

- We have a lot of rules, such as
  - NP → Nom
  - NP → Det Nom
  - NP → Det JJ Nom
  - NP → Det JJ CC JJ Nom
  - NP → Det CARD Nom
  - NP → CARD Nom
  - NP → CARD JJ Nom
  - NP → ... many more ...

- Top-Down Initialization Rules requires us to use all of them
- Madness!

Compact Data Structure

- Prefix Tree

Use Compact Data Structure

Instead of storing many rules that start the same way
- NP → Det . Nom
- NP → Det . JJ Nom
- NP → Det . JJ CC JJ Nom
- NP → Det . CARD Nom

Store pointer to prefix tree

We always just need to look ahead to the next symbol in the prefix tree