Natural language syntax: parsing and complexity

Timothée Bernard and Pascal Amsili

Université Paris Cité, Université Sorbonne Nouvelle
timothee.bernard@u-paris.fr, pascal.amsili@ens.fr

Ljubljana, Slovenia – August 7-11, 2023
ESSLLI foundational course in Language and Computation

Overview of the course

- Day 1: Formal languages and syntactic complexity.
- Day 2: The complexity of natural language.
- Day 3: Historic algorithms for parsing.
- Day 4: Modern approaches to parsing.
- Day 5: Neural networks and error propagation.

Day 3

Recap from Day 2

- There are theoretical and practical reasons for determining where NL is in the Chomsky-Schützenberger hierarchy.
- center-embedding (very common) $\rightarrow$ NL is not regular
- cross-serial dependencies (less common) $\rightarrow$ NL is not context-free
- Good candidates: TAG/CCG and MCFG/LCFRS/MG.
- It can make sense to use much more powerful formalisms (e.g. HPSG).

Today's contents

- Grammar-based constituency parsing algorithms with no machine learning. ($\rightarrow$ ML in Days 4-5)
- Top-down and bottom-up naive algorithms.
- The Shift-Reduce algorithm.
- Chart parsing:
  - CYK,
  - Earley.

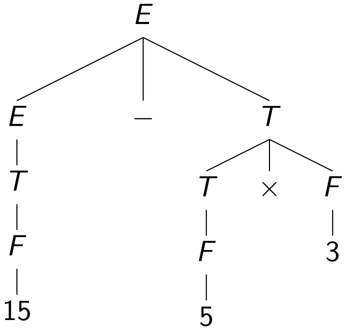
Reminder about the parsing problem

- Given a grammar G on some alphabet Σ. . .
- The **parsing problem** for G:

  Given some $w \in \Sigma^\star$,
  what are the derivations (if any) of w in G?
- If G is a CF grammar, parsing w is equivalent to finding the constituent trees (if any) of w.

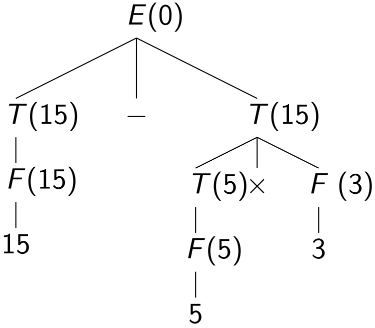
Syntactic structure can be useful

$15 - 5 \times 3 = ?$

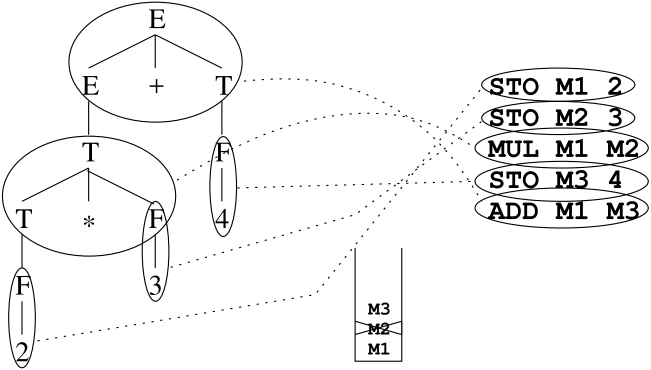


Syntactic structure can be useful

$15 - 5 \times 3 = 0$



Syntactic structure can be useful



Unless stated otherwise, we work with CF grammars from now on.

Derivation graph of a grammar

Given a grammar, one can build the directed graph such that

- nodes are words of $(\Sigma \cup N)^\star$,
- edges correspond to rewriting.

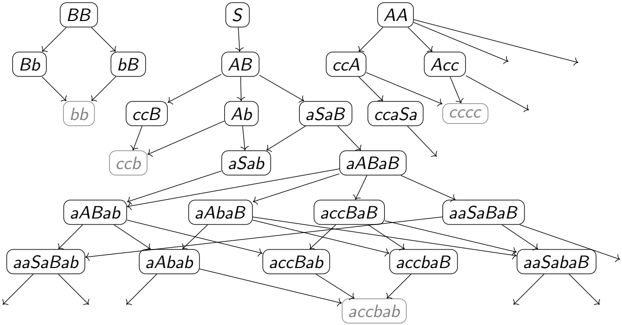
Derivation graph of a grammar

Given a grammar, one can build the directed graph such that

- nodes are words of $(\Sigma \cup N)^\star$,
- edges correspond to rewriting.

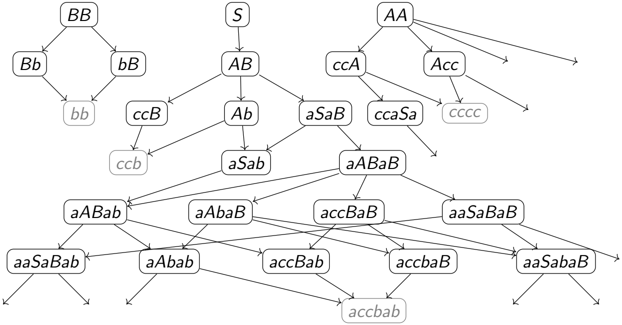
Example:

$$
\begin{array}{lcl}
S & \rightarrow & A\ B \\
A & \rightarrow & cc \\
 & | & aSa \\
B & \rightarrow & b
\end{array}
$$



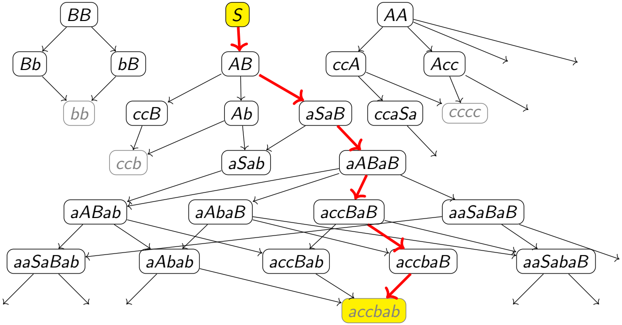
Parsing is a search in the derivation graph

Parsing $w \in \Sigma^*$:



Parsing is a search in the derivation graph

Parsing $w \in \Sigma^*$: finding a path in the graph going from S to w.



General parsing strategies

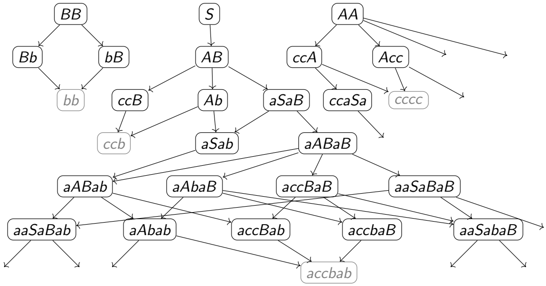
- Two possible strategies:
  - start from the axiom S (**top-down**),
  - start from the query w (**bottom-up**).
- It is also possible to mix top-down and bottom-up approaches.

Left derivation

- **Left derivation**: always rewrite the leftmost non-terminal symbol. Examples:
  - $S \Rightarrow AB \Rightarrow ccB \Rightarrow ccb$
  - $S \Rightarrow AB \Rightarrow Ab \Rightarrow ccb$
- In a CFG, every syntactic structure is associated with a single left derivation, and vice versa.

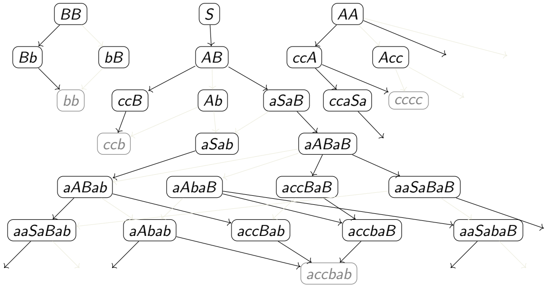
Parsing can focus on left derivations

- Without loss of generality, parsing can be defined as a search for left derivation(s) [or right].



Parsing can focus on left derivations

- Without loss of generality, parsing can be defined as a search for left derivation(s) [or right].



Top-down approaches: example (I)

$$
\begin{array}{lcl}
S & \rightarrow & A\ B \\
A & \rightarrow & cc \\
 & \mid & aSa \\
B & \rightarrow & b
\end{array}
$$

$$(S, accbab)$$

$$(AB, accbab)$$

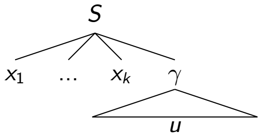
$$(aAaB, accbab) \qquad (ccBB, accbab)$$

Pruning the graph with the prefix property

Suppose that $S \overset{\star}{\Rightarrow} x_1 \ldots x_k \gamma$ with $x_1 \ldots x_k \in \Sigma^\star$, and $\gamma \in (\Sigma \cup N)^\star$.

Any word w s.t. $S \overset{\star}{\Rightarrow} x_1 \ldots x_k \gamma \overset{\star}{\Rightarrow} w$ has $x_1 \ldots x_k$ as a prefix:

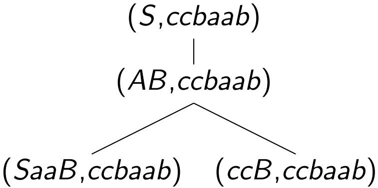
$$\exists u \in \Sigma^\star \text{ s.t. } w = x_1 \ldots x_k u.$$



A top-down derivation can be stopped as soon as it contains a non-empty prefix of letters that does not match the query.

Top-down approaches: example (II)

$$
\begin{array}{lcl}
S & \rightarrow & A\ B \\
A & \rightarrow & Saa \\
  & | & cc \\
B & \rightarrow & b
\end{array}
$$



Top-down parsing

- If the grammar is not left-recursive, we will stop at some point.
- Any CFG G_1 is *weakly equivalent* to some non-left-recursive CFG G_2 (i.e. s.t. $\mathcal{L}(G_1) = \mathcal{L}(G_2)$).
- But the worst-case time complexity of naive top-down parsing is still very high.

More efficient algorithms?

$$\begin{array}{lcl} S & \rightarrow & AB \\ A & \rightarrow & aa \\ B & \rightarrow & b \end{array}$$

(S, aab)

$$\begin{array}{lcl} S & \rightarrow & aB \\ & \mid & B \\ B & \rightarrow & bB \\ & \mid & \epsilon \end{array}$$

$(S, aaabb)$

$$\begin{array}{lcl} S & \rightarrow & aSa \\ & \mid & aSb \\ & \mid & \epsilon \end{array}$$

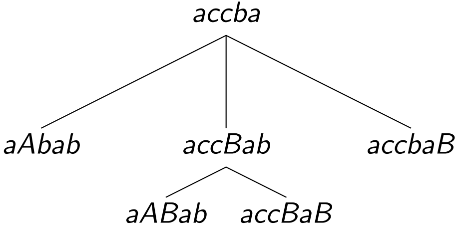
$(S, abbbba)$

- **Predictive parsing**: use the next letters in the query to better select the rewriting rules.
- For some grammars, $\exists k \in \mathbb{N}$ s.t. when considering the next k letters in the query, the choice is always reduced to a single rule: deterministic parsing.
  $\rightarrow$ Grammars of this sort are called $LL(k)$.
- But most CF languages have no $LL(k)$ grammar (whatever k).

Bottom-up approaches: example (I)

At each step:
"Which parts of *w* are identical to the right-hand side of a rule?"

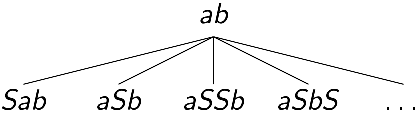
$$
\begin{aligned}
S &\rightarrow A\ B \\
A &\rightarrow Saa \\
&\ \ \mid\ \ cc \\
B &\rightarrow b
\end{aligned}
$$



A parsing is a a sequence of *reductions* ("rewriting inversions")

Bottom-up approaches: example (II)

$$S \rightarrow aSb \mid \epsilon$$



Bottom-up approaches: example (II)

$$S \rightarrow aSb \mid \epsilon$$

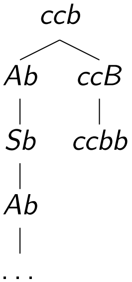
$$ab$$

$$Sab \quad aSb \quad aSSb \quad aSbS \quad \ldots$$

The query ab could have been produced from any $S^i aS^j bS^k$.

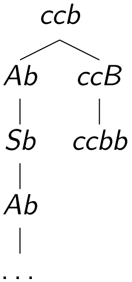
Bottom-up approaches: example (III)

$$
\begin{array}{lcl}
S & \rightarrow & A \mid A\,B \\
A & \rightarrow & cc \mid S \\
B & \rightarrow & b
\end{array}
$$



Bottom-up approaches: example (III)

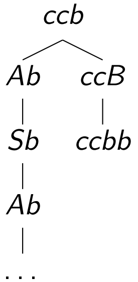
$$
\begin{array}{lcl}
S & \rightarrow & A \mid A\,B \\
A & \rightarrow & cc \mid S \\
B & \rightarrow & b
\end{array}
$$



What about *ccbb* ?

Bottom-up approaches: example (III)

$$\begin{array}{lcl} S & \rightarrow & A \mid A\,B \\ A & \rightarrow & cc \mid S \\ B & \rightarrow & b \end{array}$$



What about *ccbb* ?

Singleton rules (i.e. of the form $A \rightarrow B$ with $A, B \in N$) which may create cycles in the grammar: $X \overset{+}{\Rightarrow} X$.

Clean grammars

- Every context-free grammar is weakly equivalent to a “clean” context-free grammar:
  - No singleton rule (therefore, no cycle).
  - No ϵ-rule,
    except a rule $S \to \epsilon$ if $\epsilon \in \mathcal{L}(G)$, and no rule such that $S \to \alpha S \beta$.

- The *Shift-Reduce algorithm* ($\to$ next slide) works with clean grammars.

Shift-Reduce: a classic bottom-up algorithm

Data structures (store strings of terminal/non-terminal symbols):

- **stack** (initially empty);
- **buffer** (initially contains the query).

Transition system:

- **shift**: the first letter of the buffer is transferred to the stack;
- **reduce**: if rule $A \to \alpha$ is in the grammar and α is on top of the stack: pop α and push A on the stack;
- **accept**: success of the parsing, when the buffer is empty and the stack comprises only the axiom;
- **reject**: failure occurs when no other action is possible.

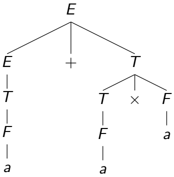
Example

stack	buffer	action
ϵ	$a + a \times a$	

$$
\begin{array}{lcl}
E & \rightarrow & E + T \\
E & \rightarrow & T \\
T & \rightarrow & T \times F \\
T & \rightarrow & F \\
F & \rightarrow & a
\end{array}
$$

Example

$$
\begin{array}{lcl}
E & \rightarrow & E + T \\
E & \rightarrow & T \\
T & \rightarrow & T \times F \\
T & \rightarrow & F \\
F & \rightarrow & a
\end{array}
$$



stack	buffer	action
ϵ	$a + a \times a$	shift
a	$+ a \times a$	reduce $(F \rightarrow a)$
F	$+ a \times a$	reduce $(T \rightarrow F)$
T	$+ a \times a$	reduce $(E \rightarrow T)$
E	$+ a \times a$	shift
$E+$	$a \times a$	shift
$E + a$	$\times a$	reduce $(F \rightarrow a)$
$E + F$	$\times a$	reduce $(T \rightarrow F)$
$E + T$	$\times a$	shift
$E + T \times$	a	shift
$E + T \times a$	ϵ	reduce $(F \rightarrow a)$
$E + T \times F$	ϵ	reduce $(T \rightarrow T \times F)$
$E + T$	ϵ	reduce $(E \rightarrow E + T)$
E	ϵ	accept

$$E \rightarrow E{+}T \rightarrow E{+}T{\times}F \rightarrow E{+}T{\times}a \rightarrow E{+}F{\times}a \rightarrow E{+}a{\times}a \rightarrow T{+}a{\times}a \rightarrow a{+}a{\times}a$$

Sources of non-determinism

$$(E + a, \times a)$$

$$(E + F, \ \times a) \qquad (E + a \times, \ a)$$

Sources of non-determinism

$$(E + a, \times a)$$

$$(E + F, \ \times a) \qquad (E + a \times, \ a)$$

$$(E + T \times F, \epsilon)$$

$$(E + F \times T, \epsilon) \qquad (E + T, \epsilon)$$

Sources of non-determinism

$(E + a, \times a)$

$(E + F, \ \times a) \quad (E + a \times, \ a)$

$(E + T \times F, \epsilon)$

$(E + F \times T, \epsilon) \quad (E + T, \epsilon)$

- Choice of the suffix of the stack to reduce.
- Choice of the rule to reduce with.
- Choice between shift and reduce.

A possibly efficient parsing algorithm

A **deterministic shift-reduce parser** can determine at each step and in constant time, based on the content of the stack and the content of the buffer, which action to perform.

For some CFGs, this is possible.
$LR(k)$ grammars, for the main parts of many programming languages.
For most CFGs, this is impossible.

Natural language syntax has lots of ambiguities

PP attachment, modifier scope, etc.

(1) a. Bob saw a passer-by with his telescope.
b. Wild cats and dogs chase rats.
c. The men and women from Tirol...

→ Deterministic parsing is not available for natural languages.

Natural language syntax has lots of ambiguities

PP attachment, modifier scope, etc.

(1) a. Bob saw [a passer-by [with his telescope]].
b. Wild cats and dogs chase rats.
c. The men and women from Tirol...

$\rightarrow$ Deterministic parsing is not available for natural languages.

Natural language syntax has lots of ambiguities

PP attachment, modifier scope, etc.

(1) a. Bob saw [a passer-by] [with his telescope].
b. Wild cats and dogs chase rats.
c. The men and women from Tirol...

→ Deterministic parsing is not available for natural languages.

Natural language syntax has lots of ambiguities

PP attachment, modifier scope, etc.

(1) a. Bob saw [a passer-by] [with his telescope].
b. [[Wild cats] and dogs] chase rats.
c. The men and women from Tirol...

→ Deterministic parsing is not available for natural languages.

Natural language syntax has lots of ambiguities

PP attachment, modifier scope, etc.

(1) a. Bob saw [a passer-by] [with his telescope].
    b. [Wild [cats and dogs]] chase rats.
    c. The men and women from Tirol...

→ Deterministic parsing is not available for natural languages.

Natural language syntax has lots of ambiguities

PP attachment, modifier scope, etc.

(1) a. Bob saw [a passer-by] [with his telescope].
b. [Wild [cats and dogs]] chase rats.
c. [[The [men and women]] from Tirol]...

$\rightarrow$ Deterministic parsing is not available for natural languages.

Natural language syntax has lots of ambiguities

PP attachment, modifier scope, etc.

(1) a. Bob saw [a passer-by] [with his telescope].
    b. [Wild [cats and dogs]] chase rats.
    c. [The men] and [women from Tirol]...

$\rightarrow$ Deterministic parsing is not available for natural languages.

Natural language syntax has lots of ambiguities

PP attachment, modifier scope, etc.

(1) a. Bob saw [a passer-by] [with his telescope].
b. [Wild [cats and dogs]] chase rats.
c. [[The men] and [women] from Tirol]...

→ Deterministic parsing is not available for natural languages.

Chart parsing is a possible answer to ambiguity

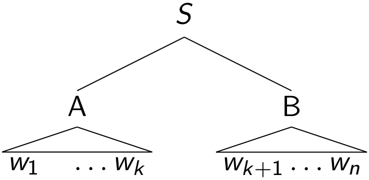
- Idea: decompose the analysis of a word w into the independent analyses of the spans of w.
- The result of these subanalyses are then combined to provide analyses of the whole w.
- Based on a data structure that stores all partial possible parses:
  - computations are done only once,
    $\rightarrow$ dynamic programming
  - multiple analyses can be handled.

Two well-known chart parsing algorithms:

- CYK: a bottom-up algorithm that works with grammars in Chomsky Normal Form.
- Earley: a mostly top-down algorithm that works with any CFG.

Factoring the computation

- Given a CFG G and a query w...
- Suppose $S \overset{\star}{\Rightarrow} AB$.
- To answer the question whether $AB \overset{\star}{\Rightarrow} w$,
- we may look for a $k \in [1, n]$ (with $n = |w|$) such that:
- $A \overset{\star}{\Rightarrow} w_{1:k}$ and $B \overset{\star}{\Rightarrow} w_{k+1:n}$.

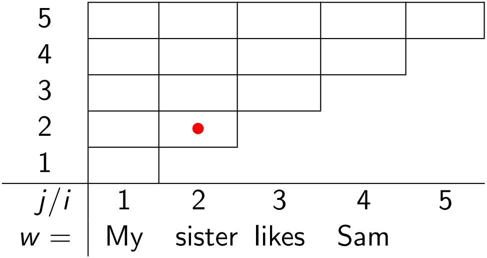


Constituent chart: cells correspond to spans

5					
4					
3					
2					
1					
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

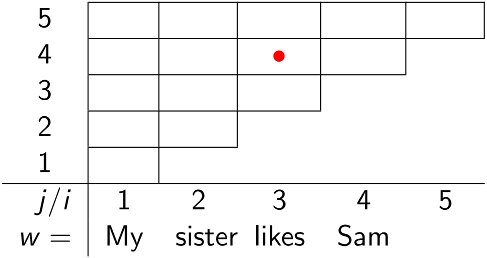
Convention: $T[i,j] \sim$ span $w_i \dots w_{j-1}$.

Constituent chart: cells correspond to spans



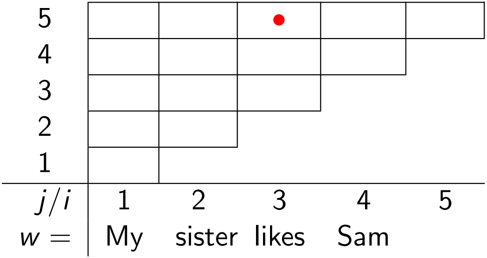
Convention: $T[i, j] \sim$ span $w_i \dots w_{j-1}$.
$T[2, 2] \rightsquigarrow \epsilon$

Constituent chart: cells correspond to spans



Convention: $T[i, j] \sim$ span $w_i \ldots w_{j-1}$.
$T[3, 4] \rightsquigarrow$ likes

Constituent chart: cells correspond to spans



Convention: $T[i,j] \sim$ span $w_i \dots w_{j-1}$.
$T[3,5] \rightsquigarrow$ likes Sam

Constituent chart: cells correspond to spans

5					
4					
3					
2					
1					
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

Convention: $T[i, j] \sim$ span $w_i \ldots w_{j-1}$.
$T[1, 5] \rightsquigarrow w$

Constituent chart: information stored in the cells

- CYK: Non-terminal symbols are stored in the chart.
- Earley: ...

5					
4					
3					
2					
1					
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

Convention: $T[i,j] = \{A \in N \mid A \overset{\star}{\Rightarrow} w_{i:j-1}\}$

Constituent chart: information stored in the cells

- CYK: Non-terminal symbols are stored in the chart.
- Earley: ...

5					
4					
3		N			
2					
1					
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

Convention: $T[i,j] = \{A \in N \mid A \overset{\star}{\Rightarrow} w_{i:j-1}\}$

$N \rightarrow$ sister $\in G$

Constituent chart: information stored in the cells

- CYK: Non-terminal symbols are stored in the chart.
- Earley: ...

5					
4					
3		N			
2	Det				
1					
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

Convention: $T[i,j] = \{A \in N \mid A \overset{\star}{\Rightarrow} w_{i:j-1}\}$

$N \to$ sister $\in G$

$Det \to$ my $\in G$

Constituent chart: information stored in the cells

- CYK: Non-terminal symbols are stored in the chart.
- Earley: . . .

5					
4					
3	NP	N			
2	Det				
1					
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

Convention: $T[i,j] = \{A \in N \mid A \stackrel{\star}{\Rightarrow} w_{i:j-1}\}$

$N \rightarrow$ sister $\in G$

$Det \rightarrow$ my $\in G$

$NP \rightarrow Det\ N \in G$

Constituent chart: a complete example

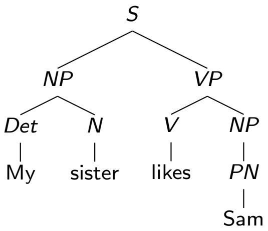
$$
\begin{array}{lcl}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Det\ N \\
NP & \rightarrow & PN \\
VP & \rightarrow & V\ NP \\
VP & \rightarrow & V \\
Det & \rightarrow & \text{my} \mid \text{the} \\
N & \rightarrow & \text{sister} \mid \text{moon} \\
V & \rightarrow & \text{likes} \mid \text{knows} \\
PN & \rightarrow & \text{Sam} \mid \text{Joan}
\end{array}
$$

5	S	$\emptyset$	VP	$\{PN, NP\}$	$\emptyset$
4	S	$\emptyset$	$\{V, VP\}$	$\emptyset$	
3	NP	N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

- There can be multiple non-terminal symbols in a cell.
- $w \in \Sigma^* \in \mathcal{L}(G)$ iff $S \in T[1, |w| + 1]$

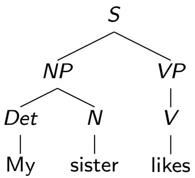
Decoding of the constituent chart

5	S	$\emptyset$	VP	$\{PN, NP\}$	$\emptyset$
4	S	$\emptyset$	$\{V, VP\}$	$\emptyset$	
3	NP	N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	



Decoding of the constituent chart

5	S	$\emptyset$	VP	$\{PN, NP\}$	$\emptyset$
4	S	$\emptyset$	$\{V, VP\}$	$\emptyset$	
3	NP	N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	



Chomsky Normal Form

A grammar is said to be in Chomsky Normal Form (CNF) iff all its rules are of the following form:

- $A \rightarrow BC$, with $A, B, C \in N$,
- or $A \rightarrow a$, with $a \in \Sigma$ and $A \in N$.

- Any CF grammar is weakly equivalent to a CNF grammar.
- Grammars in CNF have no ϵ-rule (apart maybe from a non recursive axiom).
- Grammars in CNF have no non productive cycle.

Filling the chart with a CNF grammar (I)

5					
4					
3					
2					
1					
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

$$
\begin{array}{lcl}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Det\ N \\
VP & \rightarrow & V\ PN \\
Det & \rightarrow & \text{my} \\
N & \rightarrow & \text{sister} \\
V & \rightarrow & \text{likes} \\
PN & \rightarrow & \text{Sam}
\end{array}
$$

Filling the chart with a CNF grammar (I)

5					$\emptyset$
4				$\emptyset$	
3			$\emptyset$		
2		$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

$$
\begin{array}{lcl}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Det\ N \\
VP & \rightarrow & V\ PN \\
Det & \rightarrow & \text{my} \\
N & \rightarrow & \text{sister} \\
V & \rightarrow & \text{likes} \\
PN & \rightarrow & \text{Sam}
\end{array}
$$

- Spans of length 0 (diagonal) are never generated in a CNF: $\forall i \in [1, n], T[i, i] = \emptyset$.

Filling the chart with a CNF grammar (I)

5				PN	$\emptyset$
4			V	$\emptyset$	
3		N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

S	$\rightarrow$	$NP\ VP$
NP	$\rightarrow$	$Det\ N$
VP	$\rightarrow$	$V\ PN$
Det	$\rightarrow$	my
N	$\rightarrow$	sister
V	$\rightarrow$	likes
PN	$\rightarrow$	Sam

- Spans of length 0 (diagonal) are never generated in a CNF: $\forall i \in [1, n], T[i, i] = \emptyset$.
- Spans of length 1 are generated by lexical rules: $\forall i \in [1, n], T[i, i + 1] = \{A \in N \mid A \rightarrow w_i \in P\}$.

Filling the chart with a CNF grammar (II)

5				PN	$\emptyset$
4			V	$\emptyset$	
3		N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

$$
\begin{array}{lcl}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Det\ N \\
VP & \rightarrow & V\ PN \\
Det & \rightarrow & \text{my} \\
N & \rightarrow & \text{sister} \\
V & \rightarrow & \text{likes} \\
PN & \rightarrow & \text{Sam}
\end{array}
$$

- Spans of length ≥ 2 :

$$
\begin{aligned}
T[i,j] = \quad & \{A \in N \mid \exists k \in [i+1, j-1], \\
& \exists B \in T[i,k], \\
& \exists C \in T[k,j], \\
& A \rightarrow BC \in P\}
\end{aligned}
$$

Filling the chart with a CNF grammar (II)

5				PN	$\emptyset$
4			V	$\emptyset$	
3	NP	N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

S	$\rightarrow$	$NP\ VP$
NP	$\rightarrow$	$Det\ N$
VP	$\rightarrow$	$V\ PN$
Det	$\rightarrow$	my
N	$\rightarrow$	sister
V	$\rightarrow$	likes
PN	$\rightarrow$	Sam

- Spans of length ≥ 2 :

$$\begin{aligned} T[i,j] = \quad & \{A \in N \mid \exists k \in [i+1, j-1], \\ & \exists B \in T[i,k], \\ & \exists C \in T[k,j], \\ & A \rightarrow BC \in P\} \end{aligned}$$

Filling the chart with a CNF grammar (II)

5			VP	PN	$\emptyset$
4			V	$\emptyset$	
3	NP	N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

$$
\begin{array}{lcl}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Det\ N \\
VP & \rightarrow & V\ PN \\
Det & \rightarrow & \text{my} \\
N & \rightarrow & \text{sister} \\
V & \rightarrow & \text{likes} \\
PN & \rightarrow & \text{Sam}
\end{array}
$$

- Spans of length ≥ 2 :

$$
\begin{aligned}
T[i,j] = \quad & \{A \in N \mid \exists k \in [i+1, j-1], \\
& \exists B \in T[i,k], \\
& \exists C \in T[k,j], \\
& A \rightarrow BC \in P\}
\end{aligned}
$$

Filling the chart with a CNF grammar (II)

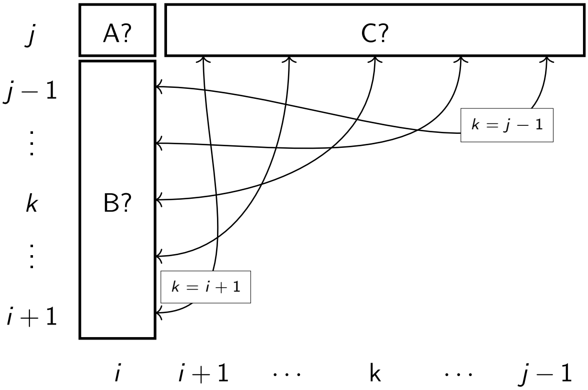
5	S		VP	PN	$\emptyset$
4			V	$\emptyset$	
3	NP	N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

$$
\begin{array}{lcl}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Det\ N \\
VP & \rightarrow & V\ PN \\
Det & \rightarrow & \text{my} \\
N & \rightarrow & \text{sister} \\
V & \rightarrow & \text{likes} \\
PN & \rightarrow & \text{Sam}
\end{array}
$$

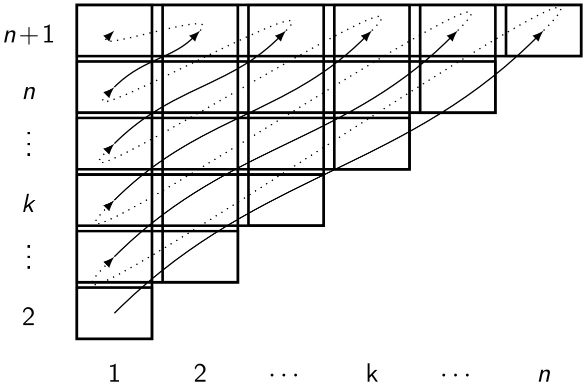
- Spans of length ≥ 2 :

$$
\begin{aligned}
T[i,j] = \quad & \{A \in N \mid \exists k \in [i+1, j-1], \\
& \exists B \in T[i,k], \\
& \exists C \in T[k,j], \\
& A \rightarrow BC \in P\}
\end{aligned}
$$

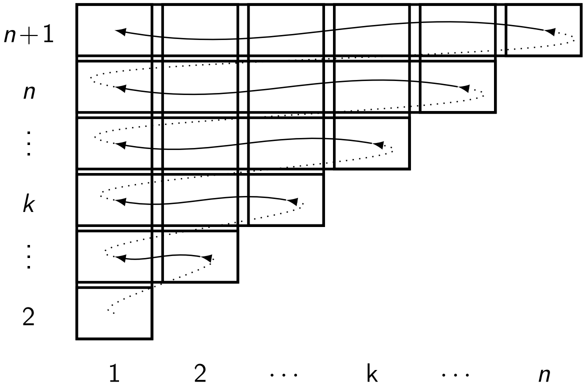
Filling the chart: general case



Diagonal strategy



Line strategy



CYK: Algorithm

```
// Input:   u ∈ Σ*
// Output:  the constituent chart of u
Function CYK-diagonal(u)
    T := empty chart(u);
    // First diagonal (unary cases)
    for i := 1 to |u| do
        foreach (A → u_i) ∈ P do
            T[i, i+1].add(A);
    // Other diagonals (binary cases)
    for l := 2 to |u| do                          // loop on the length of the span
        for i := 1 to |u| + 1 − l do              // loop on the beginning of the span
            j := i + l;                           // end of the span
            for k := i + 1 to j − 1 do            // loop on the splitting point
                foreach (A → B C) ∈ P do
                    if B ∈ T[i, k] and C ∈ T[k, j] then
                        T[i, j].add(A);
    return T;
```

CYK Summary

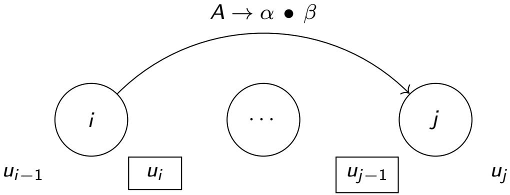
- Time complexity: $O(n^3)$
- Additional information can be stored for decoding the chart into trees.
- Efficient algorithm but requires transformation into CNF.
- Can be adapted for CCG, TAG, probabilistic CFG...

Earley Algorithm

- Works with any CFG (no transformation required).
- For CYK, it was possible to store non-terminals in the chart ($A \in T[i,j]$ iff $A \overset{\star}{\Rightarrow} w_{i:j-1}$).
- For Earley parsing, the chart will contain **dotted rules**: $(A \rightarrow \alpha \bullet \beta) \in T[i,j]$ iff $\alpha \overset{\star}{\Rightarrow} w_{i:j-1}$.
- Successful analysis: $\exists\alpha,\ (S \rightarrow \alpha \bullet) \in T[1, |w|+1]$.

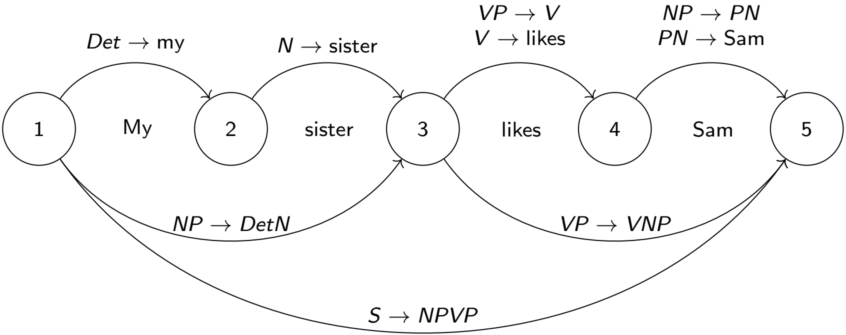
Earley items

- The information that $(A \to \alpha \bullet \beta) \in T[i,j]$
  - is an **(Earley) item**
  - and is written "$(A \to \alpha \bullet \beta, i, j)$".
- Graphical representation:



- Interpretation:
  - one is trying to recognise A starting from u_i;
  - so far, one has recognised α up to u_{j-1} (included).

Another view on the chart



5	S	$\emptyset$	VP	$\{PN, NP\}$	$\emptyset$
4	S	$\emptyset$	$\{V, VP\}$	$\emptyset$	
3	NP	N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

Use of dotted rules

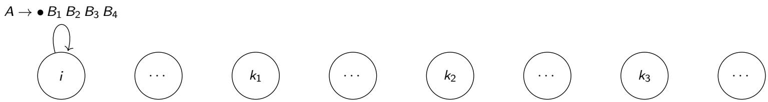
- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.

Use of dotted rules

- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.
- Example:

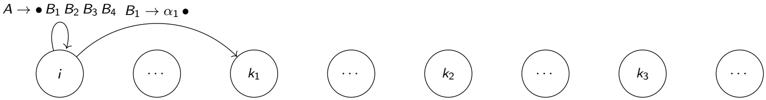
Use of dotted rules

- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.
- Example:



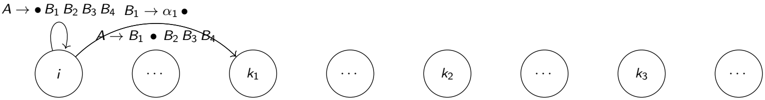
Use of dotted rules

- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.
- Example:



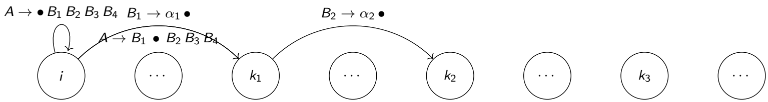
Use of dotted rules

- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.
- Example:



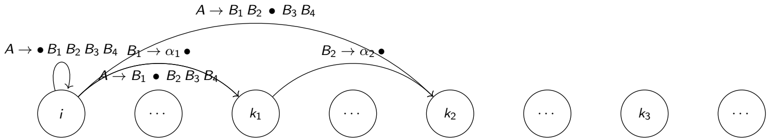
Use of dotted rules

- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.
- Example:



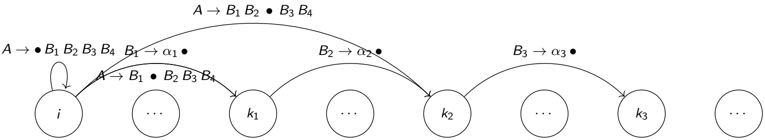
Use of dotted rules

- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.
- Example:



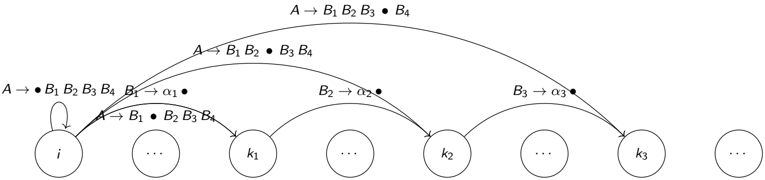
Use of dotted rules

- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.
- Example:



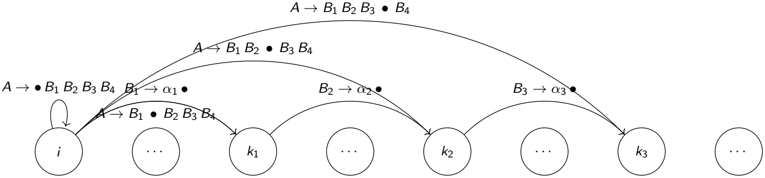
Use of dotted rules

- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.
- Example:



Use of dotted rules

- The use of dotted rules makes it relatively easy to generalise the idea behind CYK to non-binary rules.
- Example:



- An Earley item can be interpreted as an hypothesis: $(A \to \alpha \bullet \beta, i, j)$ indicates that one is trying to recognise A starting from i.

Vocabulary

- Inactive item: $(A \to \alpha \bullet, i, j)$.
- Active item: item that is not inactive.
- Initial item: $(A \to \bullet \alpha, i, j)$.

Fundamental operation

`comp` ("complete")

Fundamental operation

`comp` ("complete")

- Input: $(A \to \alpha_1 \bullet B\,\alpha_2, i, j)$ and $(B \to \beta\,\bullet, j, k)$

Fundamental operation

comp ("complete")

- Input: $(A \to \alpha_1 \bullet B\, \alpha_2, i, j)$ and $(B \to \beta \bullet, j, k)$
- Output: $(A \to \alpha_1\, B \bullet \alpha_2, i, k)$

Fundamental operation

comp ("complete")

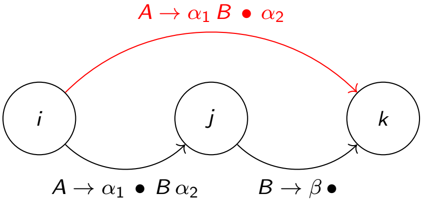
- Input: $(A \to \alpha_1 \bullet B\,\alpha_2, i, j)$ and $(B \to \beta\,\bullet, j, k)$
- Output: $(A \to \alpha_1\, B \bullet \alpha_2, i, k)$

$$\begin{tikzcd}
i \arrow[r, bend right, "A \to \alpha_1 \bullet B\,\alpha_2"'] & j \arrow[r, bend right, "B \to \beta\,\bullet"'] & k
\end{tikzcd}$$

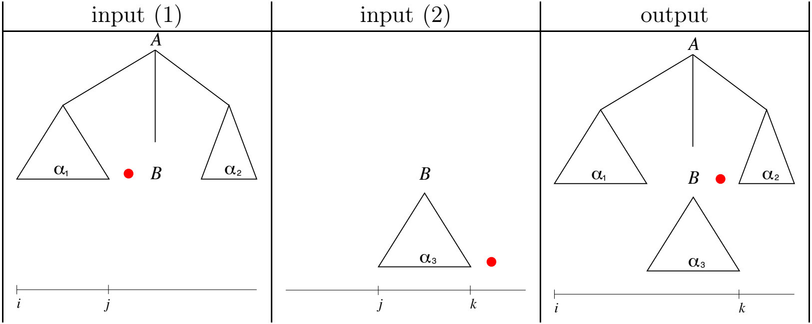
Fundamental operation

comp ("complete")

- Input: $(A \to \alpha_1 \bullet B\,\alpha_2, i, j)$ and $(B \to \beta\bullet, j, k)$
- Output: $(A \to \alpha_1\, B \bullet \alpha_2, i, k)$



Another view on comp



Other essential operation

scan

Other essential operation

scan

- Input: $(A \to \alpha_1 \bullet a\,\alpha_2, i, j)$ provided that $u_j = a$

Other essential operation

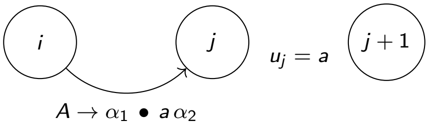
scan

- Input: $(A \to \alpha_1 \bullet a\,\alpha_2, i, j)$ provided that $u_j = a$
- Output: $(A \to \alpha_1\, a \bullet \alpha_2, i, j+1)$

Other essential operation

scan

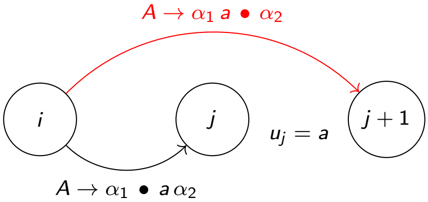
- Input: $(A \to \alpha_1 \bullet a\,\alpha_2, i, j)$ provided that $u_j = a$
- Output: $(A \to \alpha_1\, a \bullet \alpha_2, i, j+1)$



Other essential operation

scan

- Input: $(A \to \alpha_1 \bullet a\,\alpha_2, i, j)$ provided that $u_j = a$
- Output: $(A \to \alpha_1\, a \bullet \alpha_2, i, j+1)$



- `comp` and `scan` “advance” existing items.

- `comp` and `scan` “advance” existing items.
- How/when are initial items introduced?

- comp and scan "advance" existing items.
- How/when are initial items introduced?
- → Several versions (i.e. strategies) of the algorithm.

First strategy

- The chart is initialised with all possible initial items (i.e. $(A \to \bullet\, \alpha, i, i)$).
- $\to$ bottom-up parsing (not unlike CYK).

First strategy

Algorithm 1: Simple Earley analysis

Function `earley-simple(u)`

```
// Initialisation
T := empty chart(u);
for j := 1 to |u| + 1 do
    T[j] := ordered_set();
    foreach (A → α) ∈ P do T[j].add((A → •α, j));
// Main loop
for j := 1 to |u| + 1 do
    k := 0;
    while k < len(T[j]) do
        (A → α • β, i) := T[j][k];
        if β = ε then                                   // comp?
            k' := 0;
            while k' < len(T[i]) do
                (A' → α' • β', i') := T[i][k'];
                if β'_1 = A then
                    T[j].add((A' → α' β'_1 • β'_{2:|β'|}, i'));
                k' += 1;
        else if j < |u| + 1 then                        // scan?
            if β_1 = u_j then
                T[j + 1].add((A → α β_1 • β_{2:|β|}, i)) ;
        k += 1;
return T;
```

First strategy

- Let's analyse *Sabine saw a truck* with a grammar such that

$$P = \left\{ \begin{array}{l} \mathrm{S} \rightarrow \mathrm{NP}\,\mathrm{VP}, \\ \mathrm{NP} \rightarrow \mathrm{DET}\,\mathrm{N} \mid \mathrm{PN}, \\ \mathrm{VP} \rightarrow \mathrm{V} \mid \mathrm{V}\,\mathrm{NP}, \\ \mathrm{DET} \rightarrow \mathit{the} \mid a(n), \\ \mathrm{N} \rightarrow \mathit{truck} \mid \mathit{experiment}, \\ \mathrm{PN} \rightarrow \mathit{Sabine} \mid \mathit{Fred} \mid \mathit{Jamy}, \\ \mathrm{V} \rightarrow \mathit{saw} \mid \mathit{prepared} \end{array} \right\}.$$

First strategy

Sabine	1	$\cdots$ (PN $\rightarrow \bullet$ Sabine, 1), (NP $\rightarrow \bullet$ PN, 1), (S $\rightarrow \bullet$ NP VP, 1) $\cdots$
saw	2	$\cdots$ (V $\rightarrow \bullet$ saw, 2), (VP $\rightarrow \bullet$ V, 2), (VP $\rightarrow \bullet$ V NP, 2) $\cdots$ (PN $\rightarrow$ Sabine $\bullet$, 1), (NP $\rightarrow$ PN $\bullet$, 1), (S $\rightarrow$ NP $\bullet$ VP, 1)
a	3	$\cdots$ (DET $\rightarrow \bullet$ a, 3), (NP $\rightarrow \bullet$ DET N, 3) $\cdots$ (V $\rightarrow$ saw $\bullet$, 2), (VP $\rightarrow$ V $\bullet$, 2), (VP $\rightarrow$ V $\bullet$ NP, 2), (S $\rightarrow$ NP VP $\bullet$, 1)
truck	4	$\cdots$ (N $\rightarrow \bullet$ truck, 4) $\cdots$ (DET $\rightarrow$ a $\bullet$, 3), (NP $\rightarrow$ DET $\bullet$ N, 3)
	5	$\cdots$ (N $\rightarrow$ truck $\bullet$, 4), (NP $\rightarrow$ DET N $\bullet$, 3), (VP $\rightarrow$ V NP $\bullet$, 2), (S $\rightarrow$ NP VP $\bullet$, 1)

Table: Chart built during the analysis of *Sabine saw a truck*. Items introduced during the initialisation are shown in black (only the useful ones are shown). Items introduced by `scan` are shown in green. Items introduced by `comp` are shown in blue.

Better version

- The original Earley algorithm.
- The only items introduced initially are the ones of the shape $(S \to \bullet \alpha, 1, 1)$.
- A new operation, `pred` (*predict*), is used to introduce additional initial items.
- `pred` is used to introduce an initial item only if this item may be used to advance an item already introduced.
- $\to$ bottom-up parsing with top-down information.

Better version

New operation: `pred`

Better version

New operation: `pred`

- Input: $(A \to \alpha_1 \bullet B\,\alpha_2, i, j)$ where $B \in N$

Better version

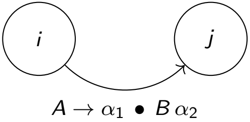
New operation: `pred`

- Input: $(A \to \alpha_1 \bullet B\,\alpha_2, i, j)$ where $B \in N$
- Output: $(B \to \bullet\gamma, j, j)$ for all $(B \to \gamma) \in P$

Better version

New operation: `pred`

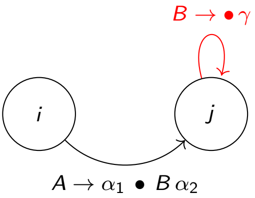
- Input: $(A \to \alpha_1 \bullet B\,\alpha_2, i, j)$ where $B \in N$
- Output: $(B \to \bullet\gamma, j, j)$ for all $(B \to \gamma) \in P$



Better version

New operation: `pred`

- Input: $(A \to \alpha_1 \bullet B\,\alpha_2, i, j)$ where $B \in N$
- Output: $(B \to \bullet\gamma, j, j)$ for all $(B \to \gamma) \in P$



Better version

Algorithm 2: Earley analysis

Function earley(u)
    // Initialisation
    T := empty chart(u);
    for $j := 1$ **to** $|u| + 1$ **do** $T[j]$:= ordered_set();
    foreach $(S \to \alpha) \in P$ **do** $T[1]$.add$((S \to \bullet\alpha, 1))$;
    // Main loop
    for $j := 1$ **to** $|u| + 1$ **do**
        $k := 0$;
        while $k < len(T[j])$ **do**
            $(A \to \alpha \bullet \beta, i) :=\in T[j][k]$;
            if $\beta = \epsilon$ **then**   // comp?
                $k' := 0$;
                while $k' < len(T[i])$ **do**
                    $(A' \to \alpha' \bullet \beta', i') := T[i][k']$;
                    if $\beta'_1 = A$ **then**
                        $T[j]$.add$((A' \to \alpha'\, \beta'_1 \bullet \beta'_{2:|\beta'|}, i'))$;
                    $k' \mathrel{+}= 1$;
            else if $\beta_1 \in N$ **then**   // pred?
                foreach $(\beta_1 \to \gamma) \in P$ **do**
                    $T[j]$.add$((\beta_1 \to \bullet\gamma, j))$;
            else if $j < |u| + 1$ **then**   // scan?
                if $\beta_1 = u_j$ **then**
                    $T[j+1]$.add$((A \to \alpha\, \beta_1 \bullet \beta_{2:|\beta|}, i))$;
            $k \mathrel{+}= 1$;
    return T;

Better version

- Let's analyse *Sabine saw a truck* with a grammar such that

$$P = \left\{ \begin{array}{l} \mathsf{S} \rightarrow \mathsf{NP}\,\mathsf{VP}, \\ \mathsf{NP} \rightarrow \mathsf{DET}\,\mathsf{N} \mid \mathsf{PN}, \\ \mathsf{VP} \rightarrow \mathsf{V} \mid \mathsf{V}\,\mathsf{NP}, \\ \mathsf{DET} \rightarrow \textit{the} \mid \textit{a}(\textit{n}), \\ \mathsf{N} \rightarrow \textit{truck} \mid \textit{experiment}, \\ \mathsf{PN} \rightarrow \textit{Sabine} \mid \textit{Fred} \mid \textit{Jamy}, \\ \mathsf{V} \rightarrow \textit{saw} \mid \textit{prepared} \end{array} \right\}.$$

Better version

Sabine	1	$(\mathsf{S} \to \bullet\, \mathsf{NP}\,\mathsf{VP}, 1)$, $(\mathsf{NP} \to \bullet\, \mathsf{PN}, 1) \cdots (\mathsf{PN} \to \bullet\, \mathsf{Sabine}, 1) \cdots$
saw	2	$(\mathsf{PN} \to \mathsf{Sabine}\, \bullet, 1)$, $(\mathsf{NP} \to \mathsf{PN}\, \bullet, 1)$, $(\mathsf{S} \to \mathsf{NP}\, \bullet\, \mathsf{VP}, 1)$, $(\mathsf{VP} \to \bullet\, \mathsf{V}, 2)$, $(\mathsf{VP} \to \bullet\, \mathsf{V}\,\mathsf{NP}, 2)$, $(\mathsf{V} \to \bullet\, \mathsf{saw}, 2) \cdots$
a	3	$(\mathsf{V} \to \mathsf{saw}\, \bullet, 2)$, $(\mathsf{VP} \to \mathsf{V}\, \bullet, 2)$, $(\mathsf{VP} \to \mathsf{V}\, \bullet\, \mathsf{NP}, 2)$, $(\mathsf{S} \to \mathsf{NP}\,\mathsf{VP}\, \bullet, 1)$, $(\mathsf{NP} \to \bullet\, \mathsf{DET}\,\mathsf{N}, 3) \cdots (\mathsf{DET} \to \bullet\, \mathsf{a}, 3) \cdots$
truck	4	$(\mathsf{DET} \to \mathsf{a}\, \bullet, 3)$, $(\mathsf{NP} \to \mathsf{DET}\, \bullet\, \mathsf{N}, 3)$, $(\mathsf{N} \to \bullet\, \mathsf{truck}, 4) \cdots$
	5	$(\mathsf{N} \to \mathsf{truck}\, \bullet, 4)$, $(\mathsf{NP} \to \mathsf{DET}\,\mathsf{N}\, \bullet, 3)$, $(\mathsf{VP} \to \mathsf{V}\,\mathsf{NP}\, \bullet, 2)$ $(\mathsf{S} \to \mathsf{NP}\,\mathsf{VP}\, \bullet, 1)$

Table: Chart built during the analysis of *Sabine saw a truck*. Items introduced during the initialisation are shown in black. Items introduced by `scan` are shown in green. Items introduced by `comp` are shown in blue. Items introduced by `pred` are shown in red (only the useful ones are shown).

- For both versions, the worst-case time complexity is $O(n^3)$ (where n is the length of the query).
- Concerning the version with `pred`:
  - The worst-case time complexity is lowered to $O(n^2)$ for unambiguous grammars.
  - In practice (for usual grammars and inputs), the observed run-time is often better than $O(n^3)$.

Day 3: Summary

- Top-down parsing: rewrite the axiom into the query.
- Bottom-up parsing: "unwrite" the query into the axiom.
- Shift-Reduce is a bottom-up transition parser.
- Some (formal) languages have non-ambiguous grammars that can be parsed deterministically.
- This is not possible with natural languages.
- Chart-parsing methods (e.g. CYK, Earley) are more appropriate for NL, with $O(n^3)$ time complexity.