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

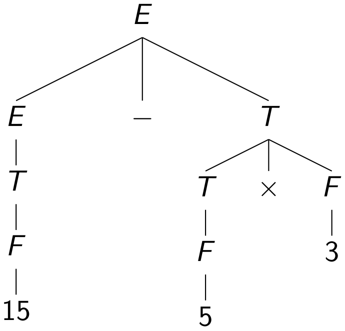
- Given a grammar G on some alphabet Σ...
- The **parsing problem** for G:

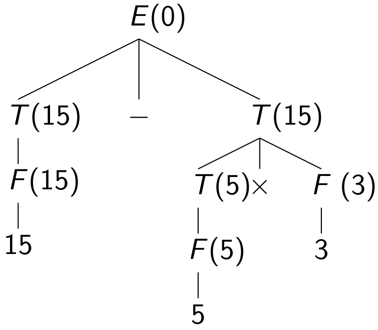
  Given some $w \in \Sigma^\star$,
  what are the derivations (if any) of w in G?

- w is the **query**.
- If G is a CF grammar, parsing w is equivalent to finding the constituent trees (if any) of w.

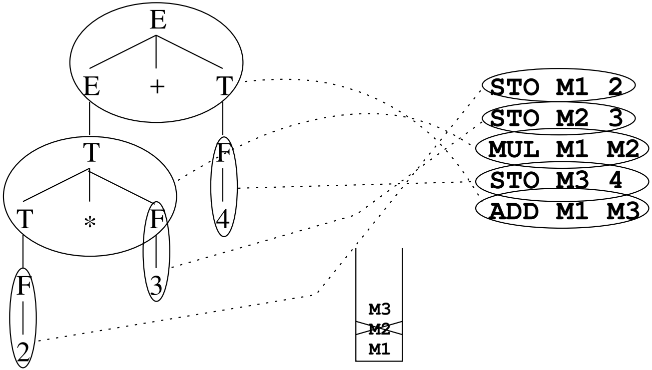
Syntactic structure can be useful

$15 - 5 \times 3 = ?0$





Syntactic structure can be useful



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

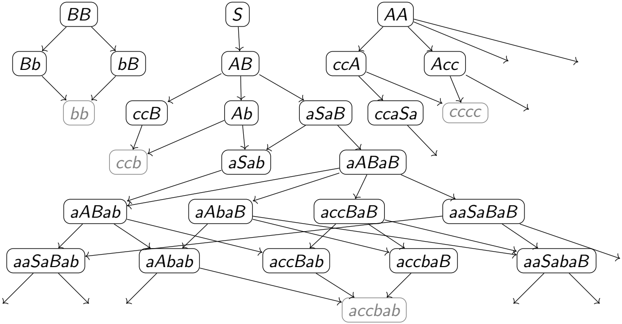
Derivation graph of a grammar

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

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

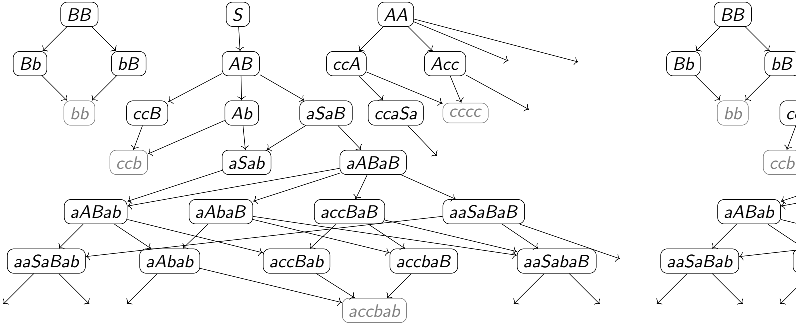
Example:

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



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

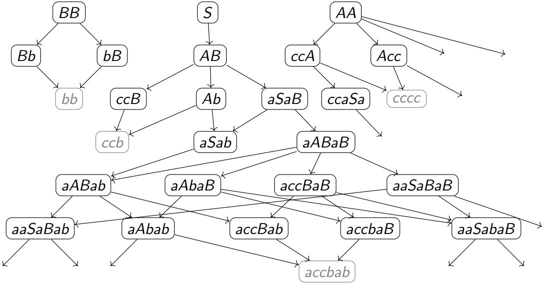
- Two possible strategies.
  - **Top-down**: start from the axiom S.
  - **Bottom-up**: start from the query w.
- It is also possible to mix top-down and bottom-up approaches.

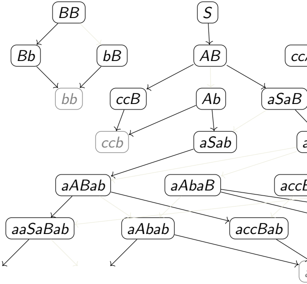
Left derivation

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

Parsing can focus on left derivations

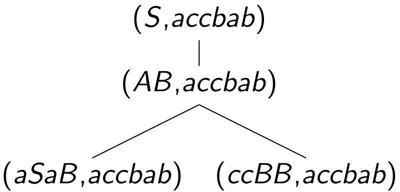
- Without loss of generality, parsing can be defined as a search for left derivation(s) [or right].





Top-down approaches: example (I)

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

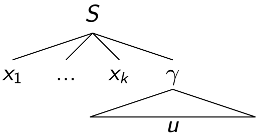


Pruning the graph with the prefix property

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

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

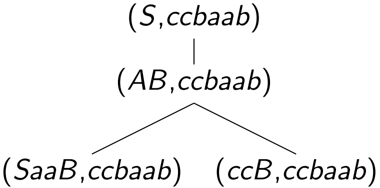
$$\exists u \in \Sigma^\star \text{ s.t. } w = x_1 \dots x_k u.$$



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?

$$\begin{array}{rcl} S & \to & AB \\ A & \to & aa \\ B & \to & b \end{array}$$

(S, aab)

$$\begin{array}{rcl} S & \to & aB \\ & | & B \\ B & \to & bB \\ & | & \epsilon \end{array}$$

$(S, aaabb)$

$$\begin{array}{rcl} S & \to & aSa \\ & | & aSb \\ & | & \epsilon \end{array}$$

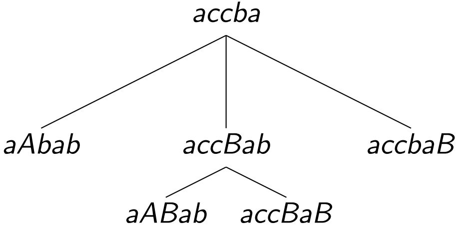
$(S, abbbba)$

- **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?"

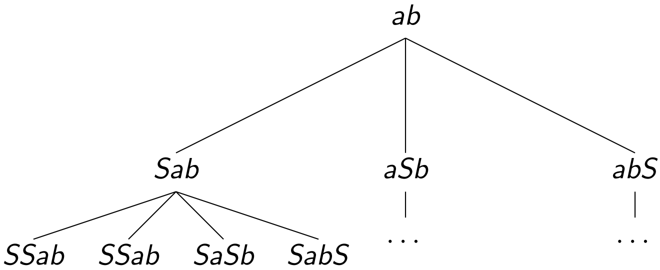
$$
\begin{array}{lcl}
S & \rightarrow & A\ B \\
A & \rightarrow & Saa \\
 & | & cc \\
B & \rightarrow & b
\end{array}
$$



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

Bottom-up approaches: example (II)

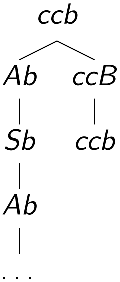
$$S \rightarrow aSb \mid \epsilon$$



The query *ab* could have been produced from any $S^i a S^j b S^k$.

Bottom-up approaches: example (III)

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



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 $S \to \alpha S \beta$.

- With a clean grammar, the *Shift-Reduce algorithm* ($\to$ next slide) cannot fall into an infinite loop.

Shift-Reduce: a classic bottom-up algorithm

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

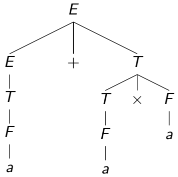
- **stack** (initially empty);
- **buffer** (initially contains the query).

Transition system:

- **shift**: the next letter in the buffer is transferred to the stack (only if the buffer is non-empty);
- **reduce**($A \to \alpha$): pop α and push A on the stack (only if $A \to \alpha \in G$ and α is on top of the stack);
- **accept**: success (only when the buffer is empty and the stack contains only S);
- **reject**: failure (only when no other action is possible).

Example

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



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

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

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

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

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

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

→ Deterministic parsing is not available for natural languages.

Chart parsing is a possible answer to ambiguity

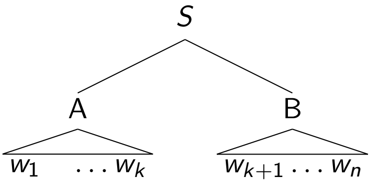
- Idea:
    - decompose the analysis of the query w into the independent analyses of all spans of w,
    - the result of these subanalyses are then combined to provide analyses of the whole w.
- A data structure stores all partial analyses so that the analysis of each span is done only once.
  $\rightarrow$ dynamic programming

Two well-known chart parsing algorithms:

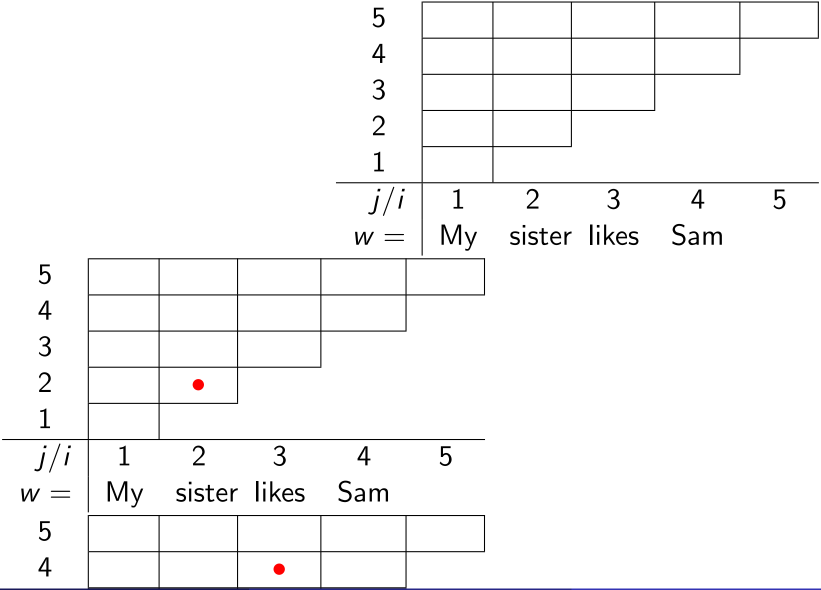
- CYK: bottom-up algorithm, works with CFGs in Chomsky Normal Form.
- Earley: mostly top-down, works with all CFGs.

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



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	

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

Constituent chart: a complete example

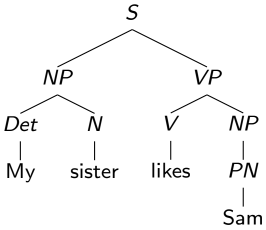
S	$\rightarrow$	$NP\ VP$
NP	$\rightarrow$	$Det\ N$
NP	$\rightarrow$	PN
VP	$\rightarrow$	$V\ NP$
VP	$\rightarrow$	V
Det	$\rightarrow$	my | the
N	$\rightarrow$	sister | moon
V	$\rightarrow$	likes | knows
PN	$\rightarrow$	Sam | Joan

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 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 one of the following forms:

- (**binary rule**) $A \to BC$, with $A, B, C \in N$,
- (**lexical rule**) $A \to a$, with $a \in \Sigma$ and $A \in N$.

- Any CFG is weakly equivalent to some CFG in CNF.

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	

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

5				PN	$\emptyset$
4			V	$\emptyset$	

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

Filling the chart with a CNF grammar (II)

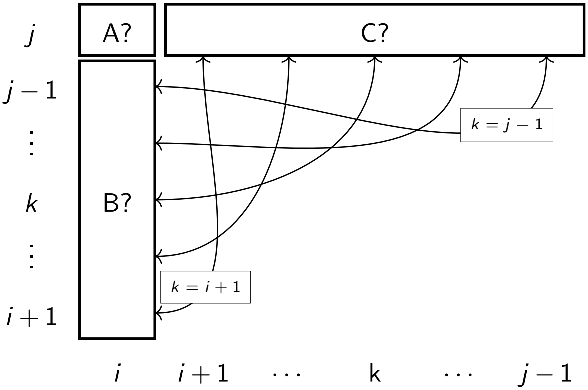
5				PN	$\emptyset$
4			V	$\emptyset$	
3		N	$\emptyset$		
2	Det	$\emptyset$			
1	$\emptyset$				
j/i	1	2	3	4	5
$w =$	My	sister	likes	Sam	

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	

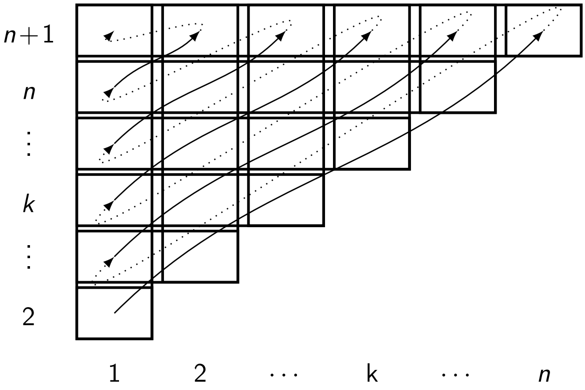
5			VP	PN	$\emptyset$
4			V	$\emptyset$	

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

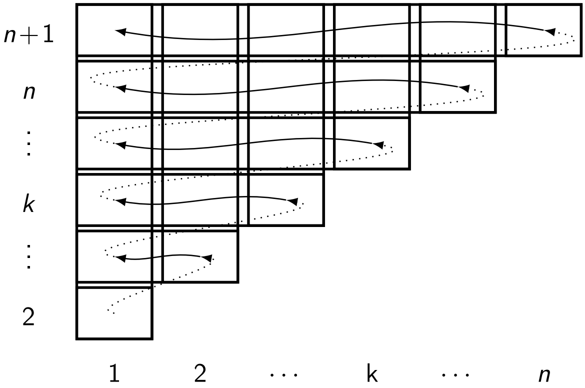
Filling the chart: general case



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

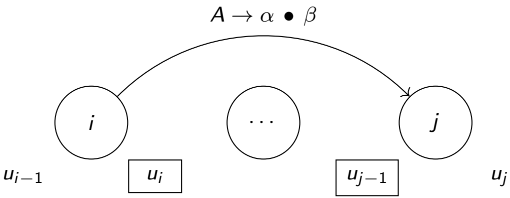
- Time complexity: $O(n^3)$
- Additional information can be stored for decoding the chart into trees.
- Efficient algorithm but requires transformation into CNF.
- Can be adapted for CCG, TAG, probabilistic CFG...

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**: $(A \to \alpha \bullet \beta) \in T[i,j]$ iff $\alpha \overset{\star}{\Rightarrow} w_{i:j-1}$.
- Successful analysis: $\exists \alpha,\ (S \to \alpha \bullet) \in T[1, |w|+1]$.

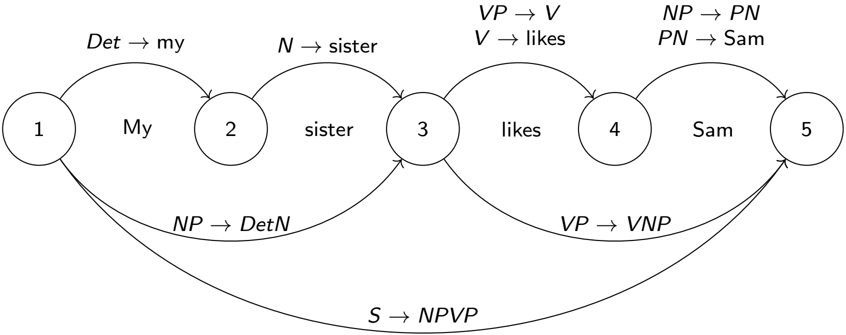
Earley items

- 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).

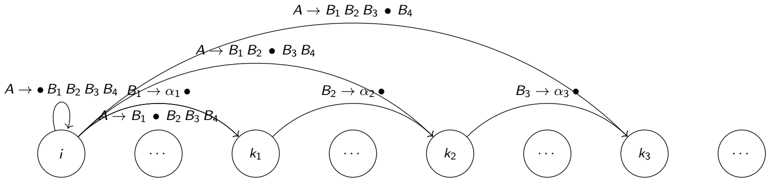
Another view on the CYK chart



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.
- 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.

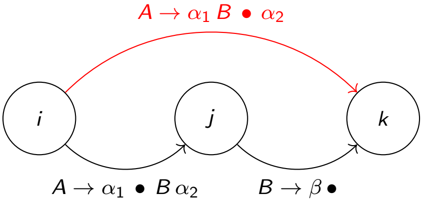
Vocabulary

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

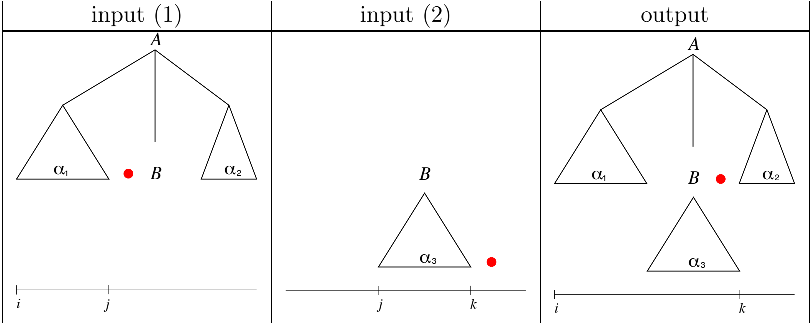
Fundamental operation

comp ("complete")

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



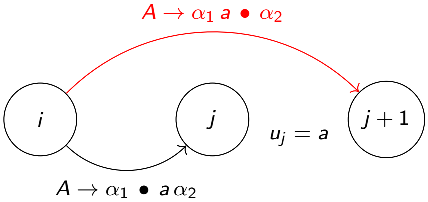
Another view on comp



Other essential operation

scan

- 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.
- How/when are initial items introduced?
- $\rightarrow$ 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

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

First strategy

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

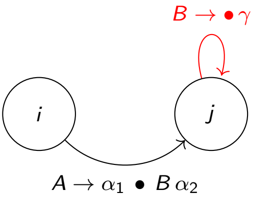
Better version

- The original Earley algorithm.
- The only items introduced initially are the ones of the shape $(S \rightarrow \bullet \alpha, 1, 1)$.
- 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.
- $\rightarrow$ bottom-up parsing with top-down information.

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 → α) ∈ P do T[1].add((S → •α, 1));
    // 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 β_1 ∈ N then                            // pred?
                foreach (β_1 → γ) ∈ P do
                    T[j].add((β_1 → •γ, j));
            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;
```

Better version

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

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

Better version

Sabine	1	(S → • NP VP, 1), (NP → • PN, 1) · · · (PN → • Sabine, 1) · · ·
saw	2	(PN → Sabine •, 1), (NP → PN •, 1), (S → NP • VP, 1), (VP → • V, 2), (VP → • V NP, 2), (V → • saw, 2) · · ·
a	3	(V → saw •, 2), (VP → V •, 2), (VP → V • NP, 2), (S → NP VP •, 1), (NP → • DET N, 3) · · · (DET → • a, 3) · · ·
truck	4	(DET → a •, 3), (NP → DET • N, 3), (N → • truck, 4) · · ·
	5	(N → truck •, 4), (NP → DET N •, 3), (VP → V NP •, 2) (S → NP VP •, 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 system.
- Some (formal) languages have grammars that can be parsed deterministically.
- This is not possible with intrinsically ambiguous languages, such as natural languages.
- Chart-parsing methods (e.g. CYK, Earley) have $O(n^3)$ worst-case time complexity, even with ambiguous grammars.