1 Context free grammar parsing

1.1 Changelog

- 29.04.2021: Clarified definitions, extended examples, added split-head section

1.2 Context free grammar (CFG)

A *context free grammar* (CFG) is a 4-tuple (N, Σ, Π, S), where:

- N is a set of non-terminal symbols
- Σ is a set of terminal symbols
- Π is a set of production rules of the form $X \to \alpha$, where $X \in N$ and $\alpha \in (N \cup \Sigma)^*$ is a string of terminal and non terminal symbols
- $S \in N$ is the start symbol

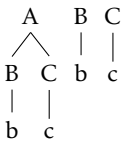
We write that $\alpha \Rightarrow \beta$ for some $\alpha, \beta \in (\Sigma \cup N)^*$ if β can be obtained from α through a single use of one of the production rules in Π. We also denote the transitive and reflexive closure of $\Rightarrow$ as $\Rightarrow^*$. For instance, given the following grammar:

- $N = \{A, B, C\}$, $S = A$
- $\Sigma = \{a, b, c\}$
- $\Pi = \{A \to AA, A \to BC, B \to b, C \to c\}$

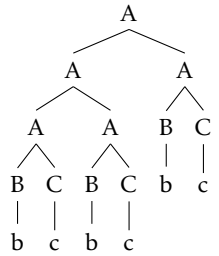
we can derive that:

- $ABC \Rightarrow AbC$, since B in ABC can be replaced by b following the rule $B \to b$
- $ABC \Rightarrow BCBC$, since A in ABC can be replaced by BC following the rule $A \to BC$
- $ABC \Rightarrow^* bcbc$, since $ABC \Rightarrow BCBC \Rightarrow bCBC \Rightarrow bcBC \Rightarrow bcbC \Rightarrow bcbc$

The last derivation can be also represented as a derivation forest, which abstracts over the exact order of applying the production rules:[1]



In CFG, a derivation tree (i.e. a derivation forest with single root) is the same as a derived (or parse) tree, i.e., the result of parsing a sentence with a CFG grammar. For instance, given the input sentence *bcbcbc* and the grammar defined above, a CFG parser could return the following derivation/derived/parse tree:



[1] For instance, the same forest could be also obtained with $ABC \Rightarrow AbC \Rightarrow Abc \Rightarrow BCbc \Rightarrow Bcbc \Rightarrow bcbc$.

1.3 Bilexical CFG

A *bilexical CFG* is a CFG (N, Σ, Π, S) in which:

- $N = \{Xt : t \in \Sigma\} \cup \{ROOT\}$
- $S = ROOT$
- Π consists of:
  - root dependencies $ROOT \rightarrow H$
  - left dependencies $H \rightarrow NH\ H$
  - right dependencies $H \rightarrow H\ NH$
  - and terminal dependencies $H \rightarrow h$

  where H and NH stand for non-terminals and h stands for a terminal symbol.

Here's an example of a bilexical CFG (taken from the lecture):

- $\Sigma = \{the, dog, barked, at, cat, .\}$
- Π :
  - $ROOT \rightarrow Xbarked$
  - $Xbarked \rightarrow Xdog\ Xbarked$
  - $Xbarked \rightarrow Xbarked\ Xat$
  - $Xbarked \rightarrow Xbarked\ X.$
  - $Xdog \rightarrow Xthe\ Xdog$
  - $Xat \rightarrow Xat\ Xcat$
  - $Xcat \rightarrow Xthe\ Xcat$
  - $Xthe \rightarrow the$
  - $Xdog \rightarrow dog$
  - $Xbarked \rightarrow barked$
  - $Xat \rightarrow at$
  - $Xcat \rightarrow cat$

The first rule ($ROOT \rightarrow Xbarked$) is a root dependency rule. In a bilexical CFG, there will be one root dependency rule $ROOT \rightarrow Xt$ per possible root t of a dependency tree.

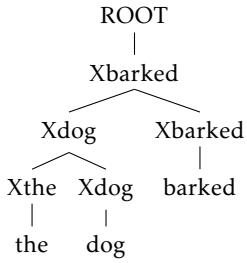
All binary rules (i.e. with two symbols in the RHS) are either left dependencies (e.g. $Xbarked \rightarrow Xdog\ Xbarked$), where the dependent (e.g. „dog") modifies the head (e.g. „barked") from the left, or right dependencies (e.g. $Xbarked \rightarrow Xbarked\ Xat$), where the dependent modifies the head from the right. Each binary rule thus captures a single dependent on either side of the head, and it is not possible to capture more complex constraints such as subcategorization requirements. Such fine-grained factorization can be however an advantage from the statistical parsing point of view.

Finally, for every non-terminal symbol Xt (with the exception of the $ROOT$) there is a terminal dependency rule $Xt \rightarrow t$, e.g. $Xat \rightarrow at$.

Here is an example of a derivation with the grammar given above:

$$ROOT \Rightarrow Xbarked \Rightarrow Xdog\ Xbarked \Rightarrow Xthe\ Xdog\ Xbarked$$
$$\Rightarrow the\ Xdog\ Xbarked \Rightarrow the\ dog\ Xbarked \Rightarrow the\ dog\ barked$$

which gives the following derivation tree:



A bilexical CFG is in the *Chomsky Normal Form*, with the exception of the unary root rules $ROOT \rightarrow H$. The CYK algorithm (or the parser itself) must be thus slightly modified to handle such rules.

1.4 CYK

We assume as given:

- a bilexical CFG $G = (N, \Sigma, \Pi, S)$
- an input sentence $w = w_1 w_2 \dots w_n \in \Sigma^*$ of length n

CYK is a chart parsing algorithm, in which an initially empty chart is progressively populated with *items* based on a set of *deduction rules* which specify how new items can be derived from the already existing items. Each item has the form $[A, i, j]$, where $A \in N$ and $1 \leq i \leq j \leq n$, and it states that $A \Rightarrow^* w_i \dots w_j$. The deduction rules underlying CYK are:

$$\text{Axiom}: \quad \frac{}{[A, i, i]} \quad A \to w_i \in \Pi$$

$$\text{Combine}: \quad \frac{[B, i, j] \quad [C, j+1, k]}{[A, i, k]} \quad A \to BC \in \Pi$$

We add one additional rule to handle the ROOT:

$$\text{Root}: \quad \frac{[A, 1, n]}{[ROOT, 1, n]} \quad ROOT \to A \in \Pi$$

1.4.1 Example

Let $G = (N, \Sigma, \Pi, S)$ be a bilexical CFG such that:

- $\Sigma = \{the, dog, barked, at, the, cat, .\}$
- Π consists of the following rules:
  - $ROOT \to Xbarked$
  - $Xbarked \to Xdog\ Xbarked$
  - $Xbarked \to Xbarked\ Xat$
  - $Xbarked \to Xbarked\ X.$
  - $Xdog \to Xthe\ Xdog$
  - $Xat \to Xat\ Xcat$
  - $Xcat \to Xthe\ Xcat$
  - $Xthe \to the$
  - $Xdog \to dog$
  - $Xbarked \to barked$
  - $Xat \to at$
  - $Xcat \to cat$

The tree in Fig. 1 can be then obtained with the following derivation:

1.	$[Xthe, 1, 1]$	axiom
2.	$[Xdog, 2, 2]$	axiom
3.	$[Xbarked, 3, 3]$	axiom
4.	$[Xat, 4, 4]$	axiom
5.	$[Xthe, 5, 5]$	axiom
6.	$[Xcat, 6, 6]$	axiom
7.	$[X., 7, 7]$	axiom
8.	$[Xdog, 1, 2]$	combine(1, 2)
9.	$[Xcat, 5, 6]$	combine(5, 6)
10.	$[Xat, 4, 6]$	combine(4, 9)
11.	$[Xbarked, 3, 6]$	combine(3, 10)
12.	$[Xbarked, 1, 6]$	combine(8, 11)
13.	$[Xbarked, 1, 7]$	combine(12, 7)
14.	$[ROOT, 1, 7]$	root(13)

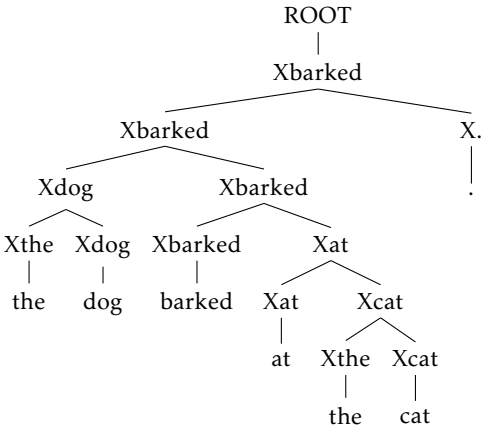


Figure 1

In line 1 of the derivation above, $[Xthe, 1, 1]$ is derived using the axiom deduction rule. The necessary condition is satisfied: $Xthe \rightarrow w_1 = Xthe \rightarrow the \in \Pi$. Similarly, the axiom rule is applied (in lines 2-7) to ,,replace" the remaining words with the corresponding non-terminals. In line 8, the combine rule is applied to $[Xthe, 1, 1]$ and $[Xdog, 2, 2]$ (hence ,,combine(1, 2)", since the two items are derived in lines 1 and 2, respectively) and outputs $[Xdog, 1, 2]$. Here as well, the necessary conditions are satisfied: the two items span adjacent intervals $(1, 1)$ and $(2, 2)$, and there is a production rule $Xdog \rightarrow Xthe\ Xdog$ in the grammar. The combine rule is similarly used in lines 9-13. Finally, the root deduction rule is used to add the *ROOT* node at the top of the tree.

Another tree can be obtained with the same grammar for the same sentence, as shown below. Both trees (in Fig. 1 and Fig. 2) however encode the same dependencies, hence this is an example of *spurious* ambiguity. It can be avoided using the *split-head* representation (see Sec. 2).

1.	[*Xthe*, 1, 1]	axiom
2.	[*Xdog*, 2, 2]	axiom
3.	[*Xbarked*, 3, 3]	axiom
4.	[*Xat*, 4, 4]	axiom
5.	[*Xthe*, 5, 5]	axiom
6.	[*Xcat*, 6, 6]	axiom
7.	[*X.*, 7, 7]	axiom
8.	[*Xdog*, 1, 2]	combine(1, 2)
9.	[*Xcat*, 5, 6]	combine(5, 6)
10.	[*Xat*, 4, 6]	combine(4, 9)
11.	[*Xbarked*, 1, 3]	combine(8, 3)
12.	[*Xbarked*, 1, 6]	combine(11, 10)
13.	[*Xbarked*, 1, 7]	combine(12, 7)
14.	[*ROOT*, 1, 7]	root(13)

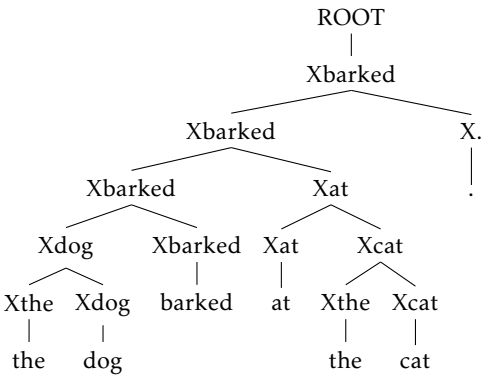


Figure 2

1.5 Time complexity analysis

The theoretical time complexity of CYK parsing for bilexical CFGs can be estimated based on the corresponding deduction rules, and more precisely on the number of possible instantiations of these rules. The ,,most costly" rule is *combine*, with two antecedent items, $[B, i, j]$ and $[C, j+1, k]$, and the consequent item $[A, i, k]$. In general, there are $\mathcal{O}(|N|^3)$ ways to instantiate A, B and C, but in bilexical CFG A must be equal to either B (right dependency) or C (left dependency), hence this is reduced to $\mathcal{O}(|N|^2)$. There are also $\mathcal{O}(n^3)$ ways to instantiate i, j, and k, and therefore the time complexity is $\mathcal{O}(|N|^2 \times n^3)$. Furthermore, N can be restricted to $\{w_1, \ldots, w_n\} \cup \{ROOT\}$ (and $|N| = n+1$), since B and C (and A) can be only instantiated to non-terminals related to the words present in the sentence (or ROOT). Therefore:

$$\begin{aligned}\mathcal{O}(|N|^2 \times n^3) &= \mathcal{O}((n+1)^2 \times n^3) = \mathcal{O}((n^2+2n+1) \times n^3) \\ &= \mathcal{O}(n^5 + 2n^4 + n^3) = \mathcal{O}(n^5) \qquad \text{(see this)}\end{aligned}$$

2 Split-head CFG

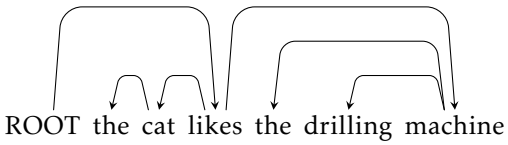
A *split-head CFG* is a CFG (N, Σ, Π, S) defined on top of the set of words W in which:

- $\Sigma = \{w^l : w \in W\} \cup \{w^r : w \in W\}$
- $N = \{Xw : w \in W\} \cup \{Lw : w \in W\} \cup \{Rw : w \in W\} \cup \{ROOT\}$
- $S = ROOT$
- Π is constructed as follows:

- Word $w \in W$ can be a root:
  $ROOT \rightarrow Xw$
- We collect left and right dependents separately:
  $Xw \rightarrow Lw\ Rw$
- Left ($v \leftarrow w$) and right ($w \rightarrow v$) dependencies:
  $Lw \rightarrow Xv\ Lw$
  $Rw \rightarrow Rw\ Xv$
- Two lexical rules per word:
  $Lw \rightarrow w^l$
  $Rw \rightarrow w^r$

2.1 Example

Consider the following dependency tree:



Task: Determine the (smallest) split-head CFG that allows to parse the sentence (and obtain the desired dependency tree) and show the corresponding constituency tree

Solution: The set of terminals is the set of words in the tree:

- $\Sigma = \{the, cat, likes, drilling, machine\}$

Next we determine the production rules. The root of the tree is *likes*, hence:

- $ROOT \rightarrow Xlikes$

The potential dependents of *likes* are *cat* on the left, *machine* on the right. In the split-head representation, we have two different non-terminals to collect the left dependencies (*Llikes*) and the right dependencies (*Rlikes*), and a rule to ,,split" *Xlikes* into the two:

- $Xlikes \rightarrow Llikes\ Rlikes$
- $Llikes \rightarrow Xcat\ Llikes$
- $Rlikes \rightarrow Rlikes\ Xmachine$

The dependents are collected the same way for all the other words with dependents:

- $Lcat \rightarrow Xthe\ Lcat$
- $Lmachine \rightarrow Xdrilling\ Lmachine$
- $Lmachine \rightarrow Xthe\ Lmachine$

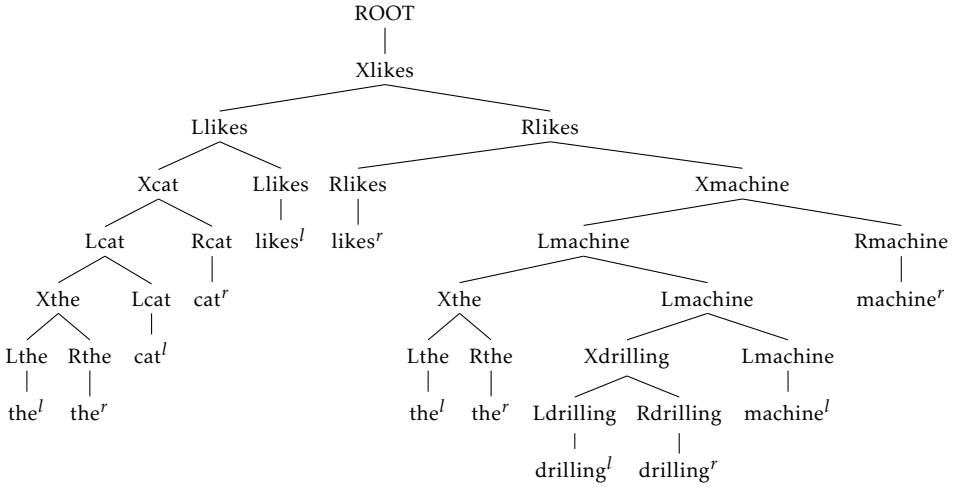
And, whether a word has dependents or not, we need the corresponding ,,splitting" rule:

- $Xthe \rightarrow Lthe\ Rthe$
- $Xcat \rightarrow Lcat\ Rcat$
- $Xdrilling \rightarrow Ldrilling\ Rdrilling$
- $Xmachine \rightarrow Lmachine\ Rmachine$

Finally we have to add the lexical rules (two per word):

- $Lthe \rightarrow the^l$
- $Rthe \rightarrow the^r$
- $Lcat \rightarrow cat^l$
- $Rcat \rightarrow cat^r$
- $Llikes \rightarrow likes^l$
- $Rlikes \rightarrow likes^r$
- $Ldrilling \rightarrow drilling^l$
- $Rdrilling \rightarrow drilling^r$
- $Lmachine \rightarrow machine^l$
- $Rmachine \rightarrow machine^r$

The corresponding constituency tree is:



Note that there is only one way to parse the underlying sentence with the grammar specified above. There is no spurious ambiguity (like in bilexical CFG, see Sec. 1.3).

2.2 Time complexity analysis

As in Sec. 1.5, we estimate the time complexity of CYK parsing for split-head CFGs based on the number of possible instantiations of the CYK deduction rules (when the grammar is a split-head CFG).

The improved complexity of CYK in this setting stems from the following invariants preserved by the deduction rules (within the context of a split-head CFG):

- **Invariant 1**: for each item of the form $[Lw_l, i, j]$, it holds that $j = 2l - 1$
- **Invariant 2**: for each item of the form $[Rw_l, i, j]$, it holds that $i = 2l$

Put differently, for any item of the form $[Lw_l, i, j]$ ($[Rw_l, i, j]$, respectively) the word w_l determines the end of the span $j = 2l-1$ (beginning of the span $i = 2l$). This is thanks to the fact that the left and right dependents are collected independently in the split-head representation.

We now focus on the combine deduction rule and consider the different forms of binary production rules in the split-head representation case by case:

1. Left dependencies:

$$\frac{[Xw_l, i, j] \quad [Lw_m, j+1, 2m-1]}{[Lw_m, i, 2m-1]} \quad Lw_m \rightarrow Xw_l Lw_m \in \Pi$$

   There are $\mathcal{O}(n^4)$ ways to instantiate i, j, l, and m, and there are no other free variables.

2. Right dependencies:

$$\frac{[Rw_l, 2l, j] \quad [Xw_m, j+1, k]}{[Rw_l, 2l, k]} \quad Rw_l \rightarrow Rw_l Xw_m \in \Pi$$

   There are $\mathcal{O}(n^4)$ ways to instantiate j, k, l, and m.

3. Split dependencies:

$$\frac{[Lw_l, i, 2l-1] \quad [Rw_l, 2l, k]}{[Xw_l, i, k]} \quad Xw_l \to Lw_l Rw_l \in \Pi$$

There are $\mathcal{O}(n^3)$ ways to instantiate i, k, and l.

Hence in total there are $\mathcal{O}(n^4) + \mathcal{O}(n^4) + \mathcal{O}(n^3) = \mathcal{O}(n^4)$ ways to instantiate the combine deduction rule.