DSL Tutorial

November 27, 2023

1 Motivation

To understand technologies like DSLs (domain specific language) examples are the best way to explain different aspects, e.g. use cases, typical problems, limitations, etc. Instead of creating an artificial example, in general a real world instance offers an more plain view to a technology.

Because of this reason the IUPAC naming system as an example was chosen. Maybe you already heard this name from the school in the chemistry subject. For those, who did not have any idea about this naming system or for people without any chemical background knowledge, don't worry; *this tutorial focuses on the naming system itself* and has only low contact points to the chemical background.

The goal is to make a tutorial, that explains the creation of CFG (context-free grammar) and illustrate the differences of abstract syntax and concrete syntax. We do not expect, that you are a expert in this topic; but a basic knowledge should be available. Terms like „grammar“ or „production rule“ will not be explained.

2 History of the IUPAC naming convention

At be beginning some history. IUPAC is an abbreviation for *International Union of Pure and Applied Chemistry*. This union was founded in 1919 with the goal to simplify the global communication between chemists. Up to this year no global rules were available for naming chemical structures. Every country used his own system and this leaded to many misunderstandings; sometimes even in a single country. You can compare this situation like before the metric system was invented.

The IUPAC invented a systematic way to create a unique name for – in theory – every chemical structure. In most cases the literature uses the term „IUPAC name“ or „IUPAC nomenclature“ for a name in the IUPAC convention. In this tutorial both of this terms will be also used.

The rules of this naming convention can be formulated as context-free grammar. So it can be also understood as a kind of DSL. In the following sections basic rules of the naming convention will be introduced. And also the way to cast it to a context-free grammar.

3 Introduction to the IUPAC nomenclature

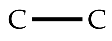
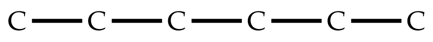
Carbon (C) has the unique skill to create long structures with branches and nesting, that are stable – in a chemical point of view. The possibilities are tremendous. Almost all other elements are not capable of creating larger structures. So the naming convention focuses on the carbon and the describing of that structures.

3.1 Nothing than straight chains

In a simplified manner you can compare a carbon atom with a building block: The simplest way is a straight chain of carbon atoms without any branches. The IUPAC nomenclature encodes the length of such a straight chain in specific terms like „Methan“, „Ethan“, etc. See the table 1 below for a length up to 10 carbon atoms.

Number of carbon atoms	**Encoded name**
1	Methan
2	Ethan
3	Propan
4	Butan
5	Pentan
6	Hexan
7	Heptan
8	Octan
9	Nonan
10	Decan

Table 1: List of the first 10 names for a straight chain

For example:
Ethan 
or
Hexan 

In a sense of a grammar: This term for the straight chain is the main symbol. We call this symbol c (for chain). S is our mandatory start symbol and thus it is a nonterminal symbol. For this tutorial T the set for terminal symbols; N for nonterminal and P for all production rules are used. In summary our first CFG grammar has the following definition:

$$\begin{aligned} T = &\quad \{c\} \\ N = &\quad \{S\} \\ P : & \\ S \rightarrow &\quad c \end{aligned}$$

Symbol	**Description**
c	Main chain

Figure 1: The CFG with only straight chains.

3.2 First order branches

A chain of carbon atoms can also contain branches. This branches could also be nested; for now we focus on branches without deeper nesting. To identify a branch we need the length of this branch and the position. The position will be formulated from the point of view of the main chain. The length of the branch will be encoded in terms. These terms are comparable with the terms for the main chain but with a postfix „yl“ instead of „an“. This are the first 5 terms for the encoded branch length:

Number of carbon atoms in branch	Encoded name
1	Methyl
2	Ethyl
3	Propyl
4	Butyl
5	Pentyl
...	...

Table 2: List of the first 5 names for a branch

An example for this:

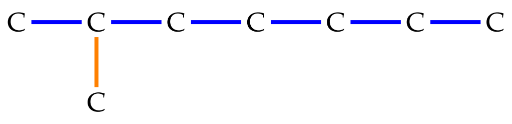


Figure 2: Example with one branch of the length 1

The orange one is the branch with the length of one C atom. So the name is *Methyl*. Any position information is written in front of the corresponding branch with a minus char between them. The full branch is therefore: *2-Methyl*. The main chain has the length 7 → *Heptan*.

So we can see, that a branch with the position information consists of three parts:

- Position $p \in \mathbb{N}$
- Minus sign m
- Term for the length of the branch b

Because every branch has this structure, we can introduce a nonterminal symbol to summarize all three terminals in this symbol B.

What about the order of branches and main chain? The IUPAC convention defines, that the term for the main chain will be always at the end of the name. This is good for the CFG definition, because we don't need to add production rules for both orders. With this and the other new information we have now following grammar (new elements are highlighted):

$$\begin{aligned} T &= \{c, p, m, b\} \\ N &= \{S, B, M\} \\ P &: \\ S &\rightarrow M \\ M &\rightarrow Bc \\ M &\rightarrow c \\ B &\rightarrow pmb \end{aligned}$$

Figure 3: CFG with exact one branch. $B \rightarrow pmb$ is an optional production rule.

Symbol	Description
c	Main chain
p	position
m	minus char
b	branch

It is important to notice, that an additional nonterminal (M) was added. Because a grammar can only contains exact one start variable, we need M as „middle variable“ to make words with and without branches possible. In summary: the B is – like a branch – an optional nonterminal. The whole name of the structure in figure 2 is: *2-MethylHeptan*[1]

3.3 Multiple first order branches

No one can forbid the carbon atoms to create multiple branches of the same length. Let add an another branch of the length one at the third position:

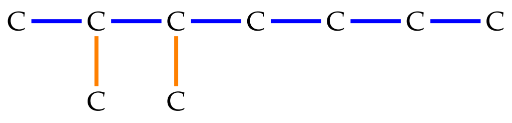


Figure 4: Example with two branches of the length 1

The intuitive way is to assume, that a second branch with the new position will be added to the begin of the name. It would be understandable; but it increases the name and in many molecules 5 to 10 branches and sometimes more are usual. For this reason there is a prefix to summarize branches of the same length. Table 3 shows the first 5 prefixes.

Branches with the same length	**Prefix**
1	Mono
2	Di
3	Tri
4	Tetra
5	Penta

Table 3: List of the first 5 prefixes to summarize branches with the same length

In normal cases the prefix „Mono“ is not used. So in the name *1-MethylHeptan* is no „Mono“ missing![2]

To identify also the second brach, we need the position of them. All position information will be written before the branch term and the prefix to summarize branches with the same length before the correspondig branch term: *2,3-DiMethylHeptan*. Usually the positions are sorted in ascending order. Between two numbers a comma is added to avoid ambiguous meanings if the positions larger than 9.

To formulate this in a grammar we have the situation, that position numbers can occur multiple times on the same branch term. So at least one production rule for branches needs to create a non-terminal symbol. We can isolate the parts for the position information from the symbol $B \rightarrow pmb$ in a new symbol C. p ans m will be moved to C, so that C is $C \rightarrow pm$. A second C, that creates itself, is necessary for multiple position information. This second C could be: $C \rightarrow poC$ (o represents here the comma char). *Or in more simple words: we need to define production rules, they can create itself to make loops possible.*

[1]It can produced with $S(M(B(pmb),c))$

[2]The nomenclature defines no specific rules for the usage. So for this tutorial „Mono“ will be not used.

With the prefix, that summarizes multiple branches (we call it d), we have a similar situation like with M. This prefix d is an optional symbol, so we need an additional production rule to achive this. An additional B seems to be a good choice.

$$\begin{aligned}
T = & \quad \{c, p, m, b, o, d\} \\
N = & \quad \{S, B, M, C\} \\
P : & \\
S \rightarrow & \quad M \\
M \rightarrow & \quad Bc \\
M \rightarrow & \quad c \\
B \rightarrow & \quad C \\
C \rightarrow & \quad pmb \\
C \rightarrow & \quad poC \\
C \rightarrow & \quad mdb
\end{aligned}$$

Symbol	Description
c	Main chain
p	position
m	minus char
b	branch
o	comma char
d	summary prefix

Figure 5: CFG with multiple branches and same length.
Several usage of $C \rightarrow poC$ creates successive positions.

The word *2,3-DiMethylHeptan* can be produced with: $S(M(C(po, C(po, C(mdb))), c))$

In theory we have the grammar, which fulfills the requirements. But look at the production: $S(M(B(C(mdb)), c))$ It creates the word **mdbc**. A word with a minus char, a prefix like „Tri“, a branch and the name for the main chain like „Hexan“. E.g.: *-TriMethylHexan*. Such a word makes no sense. So the grammer is too open. We describe more words than wanted. The reason for this is, that C contains multiple different productions; only with sense, if there are used in a specific order. This is a hint for missing nonterminal symbols. A generic C is also too simple.

One problem are the positon information. We have here two different scenarios:

- One singe position with a minus sign; E.g. *2-*
- Multiple position information separated with a comma and with a minus sign at the end. E.g. *2,3,4-*

We cannot gurantee with our current approach, that a summary prefix will be followed, if we have multiple position information. For example: *2,3,4-Methyl...* is in our grammer although the summary prefix *Tri* is missing. The reason is, that we here focused only on the position information and we have no way to influence the using of the summary prefix. The is an recurring problem with loops in a grammer when the number of used loops influence part of the words outside of the loop content. The figure 6 shows the situation with the current P.

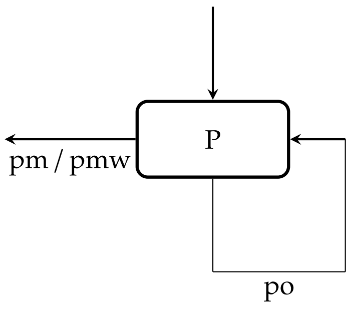


Figure 6: *P* can produce itself. *pmw* should only created, when at least one time *P* created itself.

P can create correct words. But it cannot determine, whether at least one time *P* created itself. So all words – with and without – summary prefix can be created. In a programming language we could use a flag or something similar. In a CFG this approach is not usable, because we don't have variables, that can be altered. The only possibility to solve this issue is to introduce a new nonterminal – here called P_2. For the sake of completeness we rename *P* to P_1.

P_2 can only be reached, if at least one times *po* was produced. So the usage of the nonterminal P_2 itself holds the information, that a summary prefix must be used. Figure 7 shows the modified visualization of *P*.

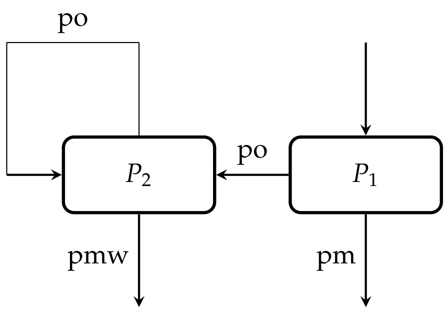


Figure 7: Modified *P*. The usage of P_2 gurantees, that at least one times *po* was created.

$$
\begin{aligned}
T =& \quad \{c, p, m, b, o, d\} \\
N =& \quad \{S, M, P_1, P_2\} \\
P :& \\
S \rightarrow& \quad M \\
M \rightarrow& \quad P_1 c \\
M \rightarrow& \quad c \\
P_1 \rightarrow& \quad pmb \\
P_1 \rightarrow& \quad poP_2 \\
P_2 \rightarrow& \quad poP_2 \\
P_2 \rightarrow& \quad pmdb
\end{aligned}
$$

Symbol	**Description**
c	Main chain
p	position
m	minus char
b	branch
o	comma char
d	summary prefix

Figure 8: Adjusted CFG with P_1 and P_2.

We see, that our grammar uses now complete new production rules. One noticeable change are the production rules P_1 and P_2 with the same result poP_2. This is an usual way to define loops in a CFG. With the altered CFG, the production of *2,3-DiMethylHeptan* is also modified: $S(M(P_1(po, P_2(pmdb)), c))$.

4 Abstract- and Concrete Syntax Trees

In the section 3 we used the tupel description of the CFG. In the tupel description every detail of the language is defined. And the production even of a simple word contains multiple nested parentheses. Already with a few production rules, the grammer becomes complex to handle and especially to debug (see chapter 3.3).