Introduction to Algorithms and Data Structures

Lecture 21: Context-free languages and grammars

John Longley

School of Informatics
University of Edinburgh

3 February 2022

Algorithms and data structures in Language Processing

By 'Language Processing', we mean the processing of both

- Artificial (computer) languages: e.g. Java, Python, HTML, . . .
- Natural (human) languages: e.g. English, Greek, Japanese.

Despite major differences, there's a certain amount of theory common to both kinds: e.g. the theory of generative grammar. (Pioneered by Sanskrit scholar Pāṇini $\sim$ 500–400 BCE. Introduced into modern linguistics by Noam Chomsky around 1957.)

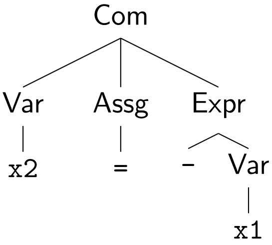
Pāṇini
Photo by Jameela P. – Own work

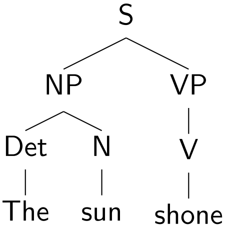

Noam Chomsky
Photo by Andrew Rusk from Toronto, Canada

We'll be looking at this theory (specifically context-free grammars) and some of the algorithms/data structures involved.

Common idea: **syntax trees**

A syntax tree displays the grammatical 'constituent structure' of a language text:



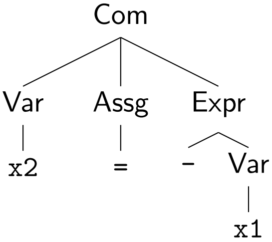


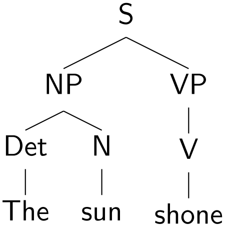
Constructing the tree is an important preliminary to many LP tasks.

A grammar for a language is a bunch of rules specifying what syntax trees are possible (and hence what strings are possible).

- This lecture: Defining languages via context-free grammars.
- Next lecture: How to find the syntax tree for a given program or sentence (parsing algorithms).

Those trees again





There are two kinds of symbols here:

- Symbols at the leaves are called terminals: these are the basic units from which sentences of the language are built.
- Symbols at internal nodes are called non-terminals: they don't themselves appear in sentences of the language, but name various kinds of 'sub-phrases'.

Context-free grammars: first example

Let's give a little grammar for arithmetic expressions, e.g.

$$6+7 \qquad 5*(x+3) \qquad x*((z*2)+y) \qquad 8 \qquad z$$

Terminals: $+, *, (,), x, y, z, 0, \ldots, 9$.

Non-terminals: Exp, Var, Num.

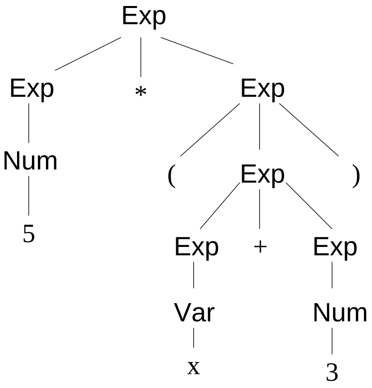
We designate the non-terminal Exp as the start symbol.

Rules:

$$\begin{array}{rcl}
\text{Exp} & \rightarrow & \text{Exp} \ + \ \text{Exp} \\
\text{Exp} & \rightarrow & \text{Exp} \ * \ \text{Exp} \\
\text{Exp} & \rightarrow & \text{Var} \mid \text{Num} \mid (\ \text{Exp}\ ) \\
\text{Var} & \rightarrow & x \mid y \mid z \\
\text{Num} & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid \\
 & & 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}$$

Generating a syntax tree from a grammar

Beginning with the start symbol, we can grow syntax trees by repeatedly expanding non-terminal symbols using these rules. E.g.:



$$
\begin{array}{rcl}
\mathsf{Exp} & \rightarrow & \mathsf{Exp} \ + \ \mathsf{Exp} \\
\mathsf{Exp} & \rightarrow & \mathsf{Exp} \ * \ \mathsf{Exp} \\
\mathsf{Exp} & \rightarrow & \mathsf{Var} \ \mid \ \mathsf{Num} \ \mid \ (\ \mathsf{Exp}\ ) \\
\mathsf{Var} & \rightarrow & x \ \mid \ y \ \mid \ z \\
\mathsf{Num} & \rightarrow & 0 \ \mid \ \cdots \ \mid \ 9
\end{array}
$$

This generates $5 * (x + 3)$ as a legal expression of the language.

The language defined by a grammar

We can generate infinitely many strings from this (finite) grammar!

The language defined by the grammar is (by definition) the set of all strings of terminals that can be obtained via such a tree.

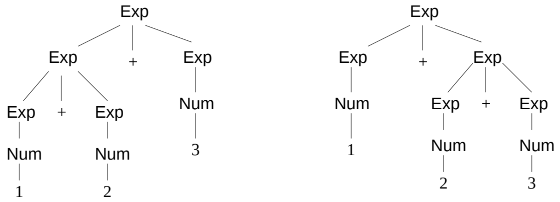
Or as a more 'machine-oriented' alternative, we can consider derivations involving sentential forms (i.e. strings of terminals and non-terminals reachable from the start symbol):

$$
\begin{array}{rcl}
\mathsf{Exp} & \Rightarrow & \mathsf{Exp} * \mathsf{Exp} \\
& \Rightarrow & \mathsf{Num} * \mathsf{Exp} \\
& \Rightarrow & \mathsf{Num} * (\mathsf{Exp}) \\
& \Rightarrow & \mathsf{Num} * (\mathsf{Exp} + \mathsf{Exp}) \\
& \Rightarrow & 5 * (\mathsf{Exp} + \mathsf{Exp}) \\
& \Rightarrow & 5 * (\mathsf{Exp} + \mathsf{Num}) \\
& \Rightarrow & 5 * (\mathsf{Var} + \mathsf{Exp}) \\
& \Rightarrow & 5 * (x + \mathsf{Exp}) \\
& \Rightarrow & 5 * (x + 3)
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{Exp} & \rightarrow & \mathsf{Exp} \ + \ \mathsf{Exp} \\
\mathsf{Exp} & \rightarrow & \mathsf{Exp} \ * \ \mathsf{Exp} \\
\mathsf{Exp} & \rightarrow & \mathsf{Var} \ \mid \ \mathsf{Num} \ \mid \ (\ \mathsf{Exp}\ ) \\
\mathsf{Var} & \rightarrow & x \ \mid \ y \ \mid \ z \\
\mathsf{Num} & \rightarrow & 0 \ \mid \ \cdots \ \mid \ 9
\end{array}
$$

Structural ambiguity

Note that strings such as 1+2+3 may be generated by more than one tree (structural ambiguity):



This might seem 'harmless' ... but what about 1+2*3 ?

- In computer languages, structural ambiguity is typically avoided by careful design of the grammar (e.g. enforcing that * takes precedence over +).
- In natural languages, structural ambiguity is a fact of life. E.g.

  I saw a man with a telescope.

Puzzle

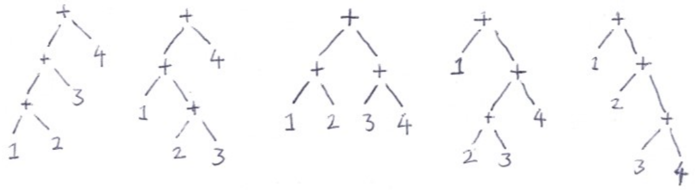
Grammar rules again:

$$
\begin{array}{rcl}
\text{Exp} & \rightarrow & \text{Exp} + \text{Exp} \\
\text{Exp} & \rightarrow & \text{Exp} * \text{Exp} \\
\text{Exp} & \rightarrow & \text{Var} \mid \text{Num} \mid (\text{Exp}) \\
\text{Var} & \rightarrow & x \mid y \mid z \\
\text{Num} & \rightarrow & 0 \mid \cdots \mid 9
\end{array}
$$

How many possible syntax trees are there for the string below?

$$1 + 2 + 3 + 4$$

Answer: **5**. Simplifying a bit, they have the following shapes:



Second example: comma-separated lists

Consider lists of (zero or more) alphabetic characters, separated by commas. E.g.:

$$\epsilon \qquad a \qquad e\textbf{,}d \qquad q\textbf{,}w\textbf{,}e\textbf{,}r\textbf{,}t\textbf{,}y$$

These can be generated by the following grammar.

Terminals:	$a, \ldots, z,$ **,**
Non-terminals:	List, Char, Tail
Start symbol:	List

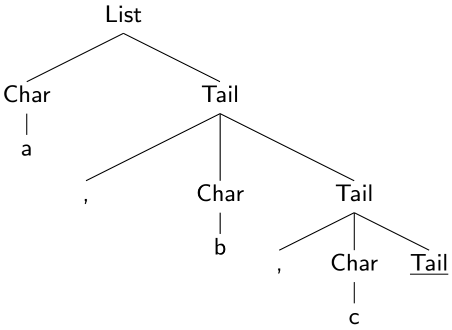
$$\begin{array}{rcl} \text{List} & \rightarrow & \epsilon \mid \text{Char Tail} \\ \text{Char} & \rightarrow & a \mid \cdots \mid z \\ \text{Tail} & \rightarrow & \epsilon \mid \textbf{,}\ \text{Char Tail} \end{array}$$

(Note the rules with empty right hand side, indicated by ϵ.)

Syntax trees for comma-separated lists

$$\begin{array}{rcl} \text{List} & \rightarrow & \epsilon \mid \text{Char Tail} \\ \text{Char} & \rightarrow & a \mid \cdots \mid z \\ \text{Tail} & \rightarrow & \epsilon \mid \textbf{,}\ \text{Char Tail} \end{array}$$

Here's the syntax tree for the list a, b, c:



(Note how we've indicated the application of an 'ϵ-rule'.)

Context-free grammars: formal definition

A context-free grammar (CFG) $\mathcal{G}$ consists of

- a finite set Σ of terminals,
- a finite set N of non-terminals, disjoint from Σ,
- a choice of start symbol $S \in N$,
- a finite set P of productions of the form $X \to \alpha$, where $X \in N$, $\alpha \in (\Sigma \cup N)^*$.

The language arising from a CFG

A sentential form is any sequence of terminals and nonterminals that can appear in a derivation starting from the start symbol.

Formal definition: The set of sentential forms derivable from $\mathcal{G}$ is the smallest set $\mathcal{S}(\mathcal{G}) \subseteq (N \cup \Sigma)^*$ such that

- $S \in \mathcal{S}(\mathcal{G})$
- if $\alpha X \beta \in \mathcal{S}(\mathcal{G})$ and $X \to \gamma \ \in P$, then $\alpha\gamma\beta \in \mathcal{S}(\mathcal{G})$.

The language associated with grammar is the set of sentential forms that contain only terminals.

Formal definition: The language associated with $\mathcal{G}$ is defined by $\mathbf{L}(\mathcal{G}) \ = \ \mathcal{S}(\mathcal{G}) \cap \Sigma^*$.

A language $L \subseteq \Sigma^*$ is defined to be context-free if there exists some CFG $\mathcal{G}$ such that $L = \mathbf{L}(\mathcal{G})$.

Assorted remarks

- $X \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$ is simply an abbreviation for a bunch of productions $X \to \alpha_1$, $X \to \alpha_2$, ..., $X \to \alpha_n$.
- These grammars are called context-free because a rule $X \to \alpha$ says that an X can *always* be expanded to α, no matter where the X occurs.
  This contrasts with context-sensitive rules, which might allow us to expand X only in certain contexts, e.g. $bXc \to b\alpha c$.
- Broad intuition: context-free languages allow nesting of phrase structures to arbitrary depth. E.g. brackets, begin-end blocks, for-loops, subordinate clauses in English, ...

A programming language example

Some context-free rules for a little programming language.

$$
\begin{array}{rcl}
\text{stmt} & \to & \text{if-stmt} \mid \text{while-stmt} \mid \text{begin-stmt} \mid \text{assg-stmt} \\
\text{if-stmt} & \to & \textbf{if } \text{bool-expr } \textbf{then } \text{stmt } \textbf{else } \text{stmt} \\
\text{while-stmt} & \to & \textbf{while } \text{bool-expr } \textbf{do } \text{stmt} \\
\text{begin-stmt} & \to & \textbf{begin } \text{stmt-list } \textbf{end} \\
\text{stmt-list} & \to & \text{stmt} \mid \text{stmt} \;;\; \text{stmt-list} \\
\text{assg-stmt} & \to & \text{var} := \text{arith-expr} \\
\text{bool-expr} & \to & \text{arith-expr compare-op arith-expr} \\
\text{compare-op} & \to & < \mid > \mid <= \mid >= \mid == \mid\; !=
\end{array}
$$

Also need rules for arith-expr and var (in style of earlier grammar).

Grammars like this (often with ::= or : in place of $\to$) are standard in computer language reference manuals. This notation is often called BNF (Backus-Naur Form).

A natural language example

Consider the following lexical classes ('parts of speech') in English:

N	nouns	(*alien, cat, dog, house, malt, owl, rat, table*)
Name	proper names	(*Jack, Susan*)
TrV	transitive verbs	(*admired, ate, built, chased, killed*)
LocV	locative verbs	(*is, lives, lay*)
Prep	prepositions	(*in, on, by, under*)
Det	determiners	(*the, my, some*)

Now consider the following productions (start symbol S):

$$
\begin{array}{rcl}
\mathrm{S} & \rightarrow & \mathrm{NP\ VP} \\
\mathrm{NP} & \rightarrow & \textit{this} \mid \mathrm{Name} \mid \mathrm{Det\ N} \mid \mathrm{Det\ N\ RelCl} \\
\mathrm{RelCl} & \rightarrow & \textit{that}\ \mathrm{VP} \mid \textit{that}\ \mathrm{NP\ TrV} \mid \mathrm{NP\ TrV} \mid \mathrm{NP\ LocV\ Prep} \\
\mathrm{VP} & \rightarrow & \textit{is}\ \mathrm{NP} \mid \mathrm{TrV\ NP} \mid \mathrm{LocV\ Prep\ NP}
\end{array}
$$

Natural language example in action

Even this modest bunch of rules can generate a rich multitude of English sentences, for example:

- *this is Jack*
- *some alien ate my owl*
- *Susan admired the dog that lay under my table*
- *this is the dog that chased the cat that killed the rat that ate the malt that lay in the house that Jack built*

Nesting in natural language

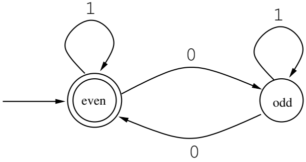
Excerpt from Jane Austen, *Mansfield Park*.

Whatever effect Sir Thomas's little harangue might really produce on Mr. Crawford, it raised some awkward sensations in two of the others, two of his most attentive listeners — Miss Crawford and Fanny. One of whom, having never before understood that Thornton was so soon and so completely to be his home, was pondering with downcast eyes on what it would be *not* to see Edmund every day; and the other, startled from the agreeable fancies she had been previously indulging on the strength of her brother's description , no longer able, in the picture she had been forming of a future Thornton, to shut out the church, sink the clergyman, and see only the respectable, elegant, modernized and occasional residence of a man of independent fortune, was considering Sir Thomas, with decided ill-will, as the destroyer of all this, and suffering the more from . . .

Non-examinable: Regular vs. context-free languages

A regular language is one that can be defined via a finite-state machine (with accepting states).
Or equivalently by a regular expression.

E.g. $\mathcal{L} = \{s \in \{0,1\}^* \mid s \text{ contains an even number of 0's}\}$.



$1^*(01^*01^*)^*$

- Every regular language is context-free (proof not too hard).
- But . . . not every context-free language is regular!
  (In fact, *most* CFLs of interest are non-regular.)

Example: a non-regular context-free language

Again non-examinable.

Consider $\mathcal{L} = \{ \ (^n\,)^n \ \mid \ n \in \mathbb{N} \ \}$. (E.g. $((())) \in \mathcal{L}$, but $((()))) \notin \mathcal{L}$.)

Context-free grammar: $S \to \ \epsilon \ \mid \ (S)$.

Claim: There's no deterministic FSM that accepts exactly $\mathcal{L}$.

Proof: Suppose there were one, with start state s_0.
Feed in (((..., and trace out the sequence of states visited:

$$s_0 \ \xrightarrow{(} \ s_1 \ \xrightarrow{(} \ s_2 \ \xrightarrow{(} \ \cdots$$

Keep going until some state appears twice: $s_i = s_j$ where $i \neq j$.
(Must happen eventually, as there are only finitely many states.)
Then starting from s_0, the strings $(^i)^i$ and $(^j)^i$ take us to the same state.
Impossible, since $(^i)^i$ should be accepted and $(^j)^i$ rejected.

A more general version of this idea is embodied by the infamous Pumping Lemma.

Reading/browsing

Reading on context-free grammars/languages (sadly not in CLRS):

- `https://en.wikipedia.org/wiki/Context-free_grammar`
  Aligns quite well with our treatment. Many examples.
- M. Sipser, *Introduction to the Theory of Computation* (3rd ed), Section 2.1. Online access via UoE library.
- `https://docs.python.org/3/reference/grammar.html`
- `https://docs.oracle.com/javase/specs/jls/se7/html/jls-18.html`
- Treebank Semantics Parsed Corpus (large browsable collection of syntax trees for a variety of English texts):
  `http://www.compling.jp/ajb129/tspc.html`

Next time: The parsing problem – given a program / expression / sentence, construct its syntax tree. Algorithms for this.

And finally . . .

A tribute to Chomsky's groundbreaking book,
Syntactic Structures (1957),
which introduced the theory of context-free grammars
and ushered in a new era of linguistics.

'The theory of *Syntactic S.*'