An unrestricted grammar is a 4-tuple \( G = (V, \Sigma, S, P) \), where \( V \) and \( \Sigma \) are disjoint sets of variables and terminals respectively. \( S \in V \) is called the start symbol and \( P \) is a set of production rules of the form \( \alpha \rightarrow \beta \).

A Context-Sensitive Grammar (CSG) is an unrestricted grammar in which every production is of the form \( \alpha \rightarrow \beta \) and \( |\beta| \geq |\alpha| \), i.e. no production rule is length-decreasing.
An Example CSG

An example CSG is:

\[
\begin{align*}
S & \rightarrow aBCT | aBC \\
T & \rightarrow ABCT | ABC \\
BA & \rightarrow AB \\
CA & \rightarrow AC \\
CB & \rightarrow BC \\
aA & \rightarrow aa \\
aB & \rightarrow ab \\
bB & \rightarrow bb \\
bC & \rightarrow bc \\
cC & \rightarrow cc
\end{align*}
\]

How a variable is derived depends on the context!

An Example CSG (2)

Here are some derivations from this CSG.

\[
\begin{align*}
S & \Rightarrow aBC \\
& \Rightarrow abC \\
& \Rightarrow abc \\
& \Rightarrow aBCT \\
& \Rightarrow aBCABC \\
& \Rightarrow aBACBC \\
& \Rightarrow aABCBC \\
& \Rightarrow aABBCC \\
& \Rightarrow aaBBCC \\
& \Rightarrow aabBCC \\
& \Rightarrow aabbCC \\
& \Rightarrow aabbcC \\
& \Rightarrow aabbcc
\end{align*}
\]

\[L(G) = \{a^n b^n c^n | n \geq 1\}\]

We have already shown that \(A^n B^n C^n\) is not a context-free language using the CFG pumping lemma. But it is a context-sensitive language.

CSLs are not a generalisation of CFGs as CSLs cannot have any \(\Lambda\)-productions.
A linear bounded automaton (LBA) is a finite state machine with a finite length data store called a tape. The tape consists of a sequence of cells, where each cell can store a symbol from the machine’s alphabet. Symbols can be written or read from any position on this tape and therefore the LBA has a read-write head that can be moved left or right one cell.

The tape is used both to store the input and any ongoing calculations. There are 2 special symbols, [ and ], that are used to mark the finite bounds of the tape. The read-write head cannot move beyond either of these symbols and it cannot overwrite these symbols.

At each step the LBA read the symbol under the read-write head, replaces the by another symbol (could be the same symbol) and then perform one of four possible actions \( A \in \{ Y, N, L, R \} \), where:

\begin{itemize}
  \item \( Y \) denotes “Yes”, accept the input string
  \item \( N \) denotes “No”, reject the input string
  \item \( L \) denotes “Left”, move the read-write head one cell to the left
  \item \( R \) denotes “Right”, move the read-write head one cell to the right
\end{itemize}
Formally, a linear bounded automaton is a 5-tuple $M = \{Q, \Sigma, \Gamma, q_0, \delta\}$ where:

- $Q$ is a finite set of states;
- $\Sigma$ is a finite alphabet (input symbols);
- $\Gamma$ is a finite alphabet (store symbols);
- $q_0 \in Q$ is the initial state; and
- $\delta : Q \times (\Gamma \cup \{[1]\}) \rightarrow Q \times (\Gamma \cup \{[1]\}) \times A$, is the transition function.

If $((q, \sigma), (q', \psi, A)) \in \delta$ then when in state $q$ with $\sigma$ at the current read-write head position, $M$ will replace $\sigma$ by $\psi$ and perform action $A$ and enter state $q'$.

$M$ accepts $w \in \Sigma^*$ iff it starts with configuration $(q_0, [w])$ and the action $Y$ is taken.

An LBA to accept $AnBnCn = \{a^n b^n c^n | n \geq 0\}$ is:

- $Q = \{s, t, u, v, w\}$
- $\Sigma = \{a, b, c\}$
- $\Gamma = \{a, b, c, x\}$
- $q_0 = s$
- $\delta = \{((s, [1]), (t, [1], R)), ((t, [1]), (t, ], Y)), ((t, x), (t, x, R)), ((t, a), (u, x, r)), ((u, a), (u, a, R)), ((u, x), (u, x, R)), ((u, b), (v, x, R)), ((v, b), (v, b, R)), ((v, x), (v, x, R)), ((v, c), (w, x, L)), ((w, c), (w, c, L)), ((w, b), (w, b, L)), ((w, a), (w, a, L)), ((w, x), (w, x, L)), ((w, [1]), (t, [1], R))\}$
Linear Bounded Automaton Example (2)

Or as a transition diagram;

where $\sigma/\psi/A$ denotes reading symbol $\sigma$, writing symbol $\psi$ and performing action $A$.

Linear Bounded Automaton Example (3)

Intuitively the previous LBA behaves as follows.

- The LBA performs multiple passes over the input string.
- On each pass (from left to right) starting at the start symbol $[$ in state $t$, the LBA converts the first $a$ into an $x$, and then the first $b$ into an $x$ and finally the first $c$ into an $x$.
- After converting a $c$ into an $x$ (after matching it with an $a$ and $b$) the LBA moves right to left until it reaches the start symbol $[$ and goes into state $t$.
- If the LBA in state $t$ only encounters $x$ symbols from the start symbol $[$ to the end symbol $]$, then the LBA performs a $Y$ action.
- the LBA gets stuck in either state $u$, $v$ or $w$ if there is not a “matching” $a$, $b$ or $c$ symbol respectively.

Can you design a better version of this LBA that actually rejects string that are not in the language $AnBnCn$?