Parsing in Prolog

- Could write your own parser — Earley's algorithm, LR(k) shift-reduce (bottom-up) parser, LL(k) top-down parser, backtrack parser, etc.

- Write your grammar in logic and let Prolog be your backtrack parser.

This is what we will do.

Context-Free Grammars as Clauses

- Each terminal is just a constant.
- Each non-terminal is a binary predicate whose first argument will be input and second argument will be output.
- Each argument will be a string represented as a list of words.
- Predicate will succeed iff the corresponding non-terminal generates some prefix of the input string. Output string is the input string minus the prefix (the remainder).
**Simple Example**

\[ S \rightarrow \varepsilon \mid aSa \mid bSb \]

1. \( s(L, L) \).
2. \( s([a \mid T], X) \iff s(T, [a \mid X]) \).
3. \( s([b \mid T], X) \iff s(T, [b \mid X]) \).

\[
\begin{align*}
  s([a, b, b, a], []) & \iff s([a \mid T], X) \iff s(T, [a \mid X]) \\
  \text{\textcolor{blue}{\theta_1: \{T \rightarrow [b, b, a],}} & \text{\textcolor{blue}{X \rightarrow []\}}} \\
  s([b, b, a], [a]) & \iff s([b \mid T], X) \iff s(T, [b \mid X]) \\
  \text{\textcolor{blue}{\theta_2: \{T \rightarrow [b, b, a],}} & \text{\textcolor{blue}{X \rightarrow [a]\}}} \\
  s([b, a], [b, a]) & \iff s(L, L) \\
  \text{\textcolor{blue}{\theta_3: \{L \rightarrow [b, a]\}}} \\
\end{align*}
\]