Bottom-up Parsing

A bottom-up (LR) parser reads tokens from left to right and maintains a stack of terminal and non-terminal symbols. At each step it does one of two things:

- **Shift**: Read in the next token and push it onto the stack
- **Reduce**: Recognize that the top of the stack is the r.h.s. of a production rule, and replace that r.h.s. by the l.h.s., which will be a non-terminal symbol.

The question is how to build an LR parser that applies these rules systematically, deterministically, and of course quickly.

Simple grammar for LR parsing

Consider the following example grammar:

\[
S \rightarrow E \\
E \rightarrow E + T \\
E \rightarrow T \\
T \rightarrow n
\]

Examine a bottom-up parse for the string \(n + n\).

How can we model the “state” of the parser?
Parser states

At any point during parsing, we are trying to expand one or more production rules.

The state of a given (potential) expansion is represented by an “LR item”.

For our example grammar we have the following LR items:

\[
\begin{align*}
S & \rightarrow \bullet E \\
E & \rightarrow E + T \\
S & \rightarrow E \\
E & \rightarrow E + T \\
E & \rightarrow E + T \\
E & \rightarrow E + T \\
E & \rightarrow E + T \\
E & \rightarrow E + T \\
E & \rightarrow E + T \\
S & \rightarrow E \\
E & \rightarrow E \\
\end{align*}
\]

The \( \bullet \) represents “where we are” in expanding that production.

Pieces of the CFSM

The CSFM (Characteristic Finite State Machine) is a FA representing the transitions between the LR item “states”.

There are two types of transitions:

- **Shift**: consume a terminal or non-terminal symbol and move the \( \bullet \) to the right by one.
  
  Example:
  
  \[
  T \rightarrow \bullet n \\
  T \rightarrow n \bullet
  \]

- **Closure**: If the \( \bullet \) is to the left of a non-terminal, we have an \( \epsilon \)-transition to any production of that non-terminal with the \( \bullet \) all the way to the left.
  
  Example:
  
  \[
  E \rightarrow E + T \\
  \epsilon \\
  T \rightarrow \bullet n
  \]

Nondeterministic CFSM for example grammar

A diagram illustrating the nondeterministic CFSM for the example grammar is shown above.
CFSM Properties

- Observe that every state is accepting.
- This is an NDFA that accepts valid stack contents.
- The “trap states” correspond to a reduce operation:
  Replace r.h.s. on stack with the l.h.s. non-terminal.
- We can simulate an LR parse by following the CFSM on the current stack symbols AND un-parsed tokens, then starting over after every reduce operation changes the stack.
- We can turn this into a DFA just by combining states.

Deterministic CFSM for example grammar

- Every state is labelled with a number.
- Labels are pushed on the stack along with symbols.
- After a reduce, go back to the state label left at the top of the stack.