Class 11: SLR Parsing

SI 413 - Programming Languages and Implementation

Dr. Daniel S. Roche

United States Naval Academy

Fall 2011

Simple grammar from last lecture

$$
\begin{aligned}
S &\rightarrow E \\
E &\rightarrow E + T \\
E &\rightarrow T \\
T &\rightarrow \texttt{n}
\end{aligned}
$$

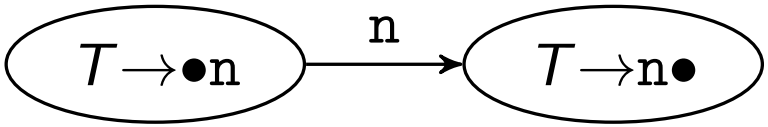
LR items:

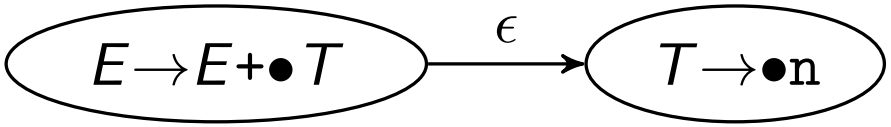
$$
\begin{aligned}
S &\rightarrow \bullet\, E & E &\rightarrow E + T \,\bullet \\
S &\rightarrow E \,\bullet & E &\rightarrow \bullet\, T \\
E &\rightarrow \bullet\, E + T & E &\rightarrow T \,\bullet \\
E &\rightarrow E \,\bullet + T & T &\rightarrow \bullet\, \texttt{n} \\
E &\rightarrow E + \bullet\, T & T &\rightarrow \texttt{n} \,\bullet
\end{aligned}
$$

Pieces of the CFSM

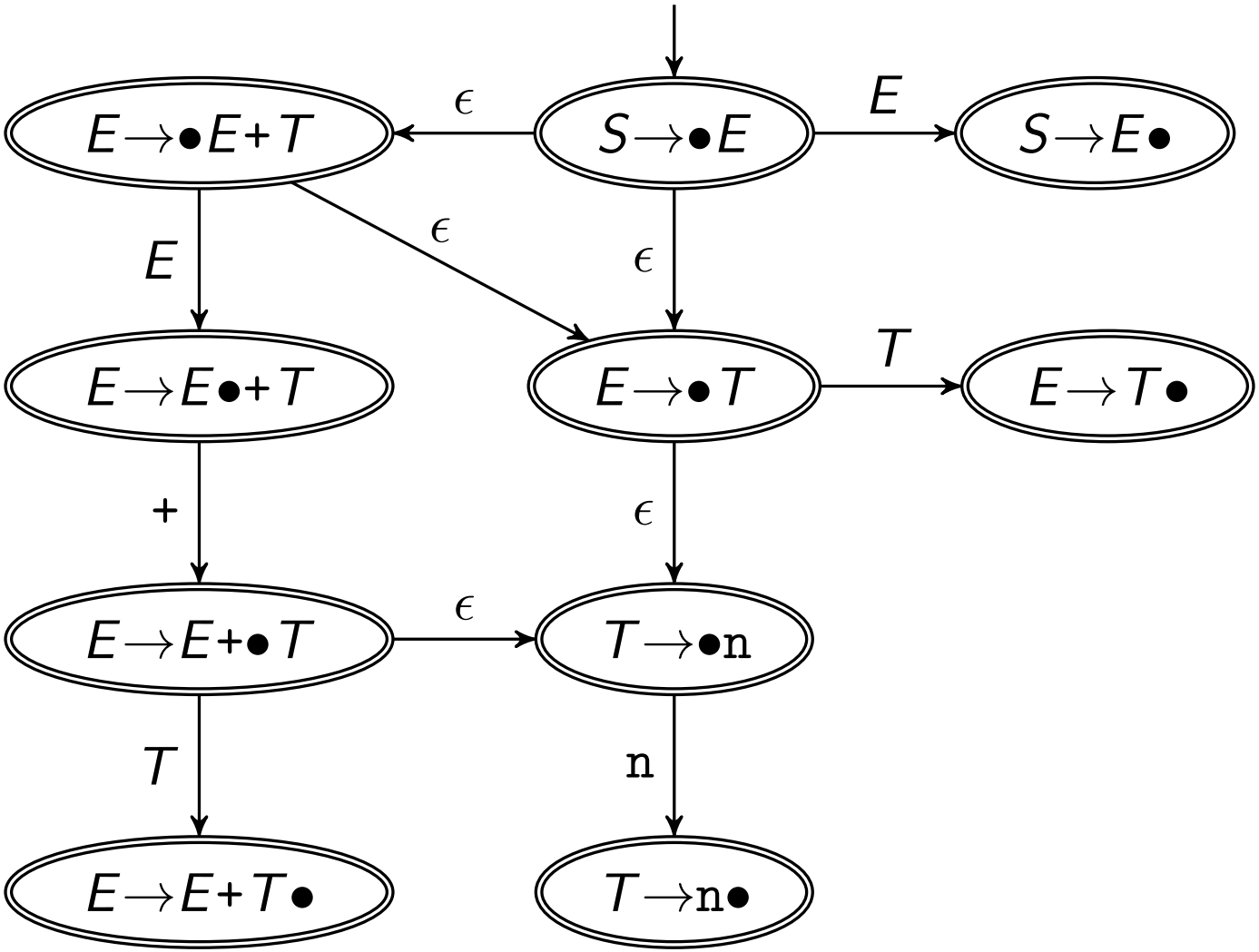
The CSFM (Characteristic Finite State Machine) is a FA representing the *transitions* between the LR item "states".

There are two types of transitions:

- **Shift**: consume a terminal *or non-terminal* symbol and move the • to the right by one.
  Example: 

- **Closure**: If the • is to the left of a non-terminal, we have an ϵ-transition to any production of that non-terminal with the • all the way to the left.
  Example: 

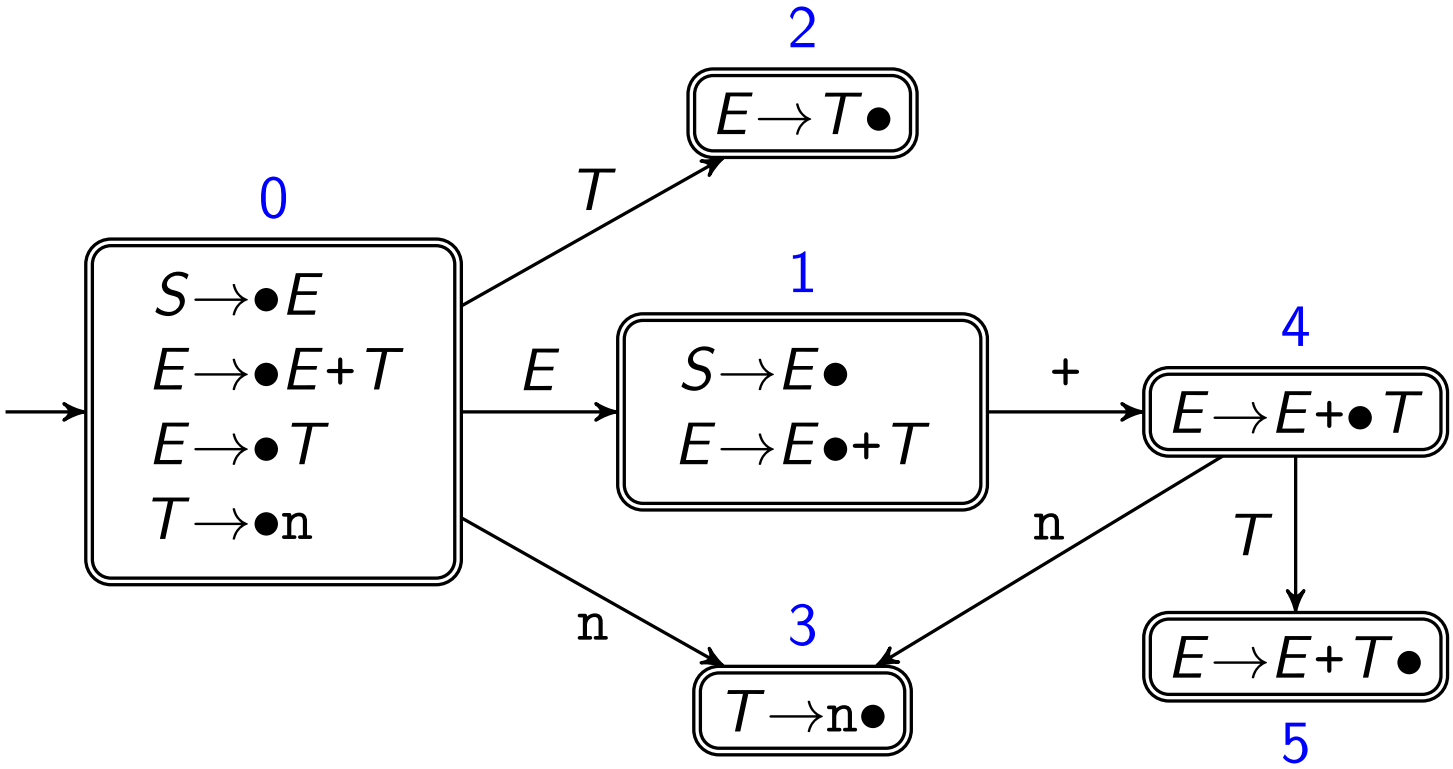
Nondeterministic CFSM for example grammar



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.

SLR

Parsing this way using a (deterministic) CFSM is called *SLR Parsing*.

Following an edge in the CFSM means shifting;
coming to a rule that ends in • means reducing.

SLR(k) means SLR with k tokens of look-ahead.
The previous grammar was SLR(0); i.e., no look-ahead required.

When might we need look-ahead?

Example Grammar 2

Consider the following grammar:

$$
\begin{aligned}
S &\rightarrow W\ W \\
W &\rightarrow \texttt{a} \\
W &\rightarrow \texttt{ab}
\end{aligned}
$$

Draw the CSFM for this grammar.
What is the problem?

Example Grammar 2

Consider the following grammar:

$$
\begin{aligned}
S &\rightarrow W\ W \\
W &\rightarrow \mathtt{a} \\
W &\rightarrow \mathtt{ab}
\end{aligned}
$$

Draw the CSFM for this grammar.
What is the problem?

The state that looks like $\boxed{\begin{array}{l} W \rightarrow \mathtt{a}\bullet \\ W \rightarrow \mathtt{a}\bullet\mathtt{b} \end{array}}$

has a *shift-reduce conflict*.

Example Grammar 3

Consider the following grammar:

$$
\begin{aligned}
S &\rightarrow W\ \texttt{b} \\
W &\rightarrow \texttt{a} \\
W &\rightarrow X\ \texttt{a} \\
X &\rightarrow \texttt{a}
\end{aligned}
$$

Draw the CSFM for this grammar.
What is the problem?

Example Grammar 3

Consider the following grammar:

$$
\begin{aligned}
S &\rightarrow W\ \texttt{b} \\
W &\rightarrow \texttt{a} \\
W &\rightarrow X\ \texttt{a} \\
X &\rightarrow \texttt{a}
\end{aligned}
$$

Draw the CSFM for this grammar.
What is the problem?

The state that looks like $\boxed{\begin{array}{c} W \rightarrow \texttt{a}\bullet \\ X \rightarrow \texttt{a}\bullet \end{array}}$

has a *reduce-reduce conflict*.

SLR(1)

SLR(1) parsers handle conflicts by using one token of look-ahead:

- If the next token is an outgoing edge label of that state, shift and move on.
- If the next token is in the *follow set* of a non-terminal *that we can reduce to*, then do that reduction.

Of course, there may still be conflicts, in which case the grammar is not SLR(1). More look-ahead may be needed.