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

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

LR items:

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

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: \[
  \begin{array}{ccc}
  T & \rightarrow & \bullet n \\
  T & \rightarrow & n \bullet
  \end{array}
  \]

- **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: \[
  \begin{array}{ccc}
  E & \rightarrow & E \bullet T \\
  T & \rightarrow & \bullet n
  \end{array}
  \]
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:

\[ S \rightarrow W \ W \]
\[ W \rightarrow a \]
\[ W \rightarrow ab \]

Draw the CSFM for this grammar.

What is the problem?

The state that looks like
\[ W \rightarrow a \]
\[ W \rightarrow a \]

has a shift-reduce conflict.

Example Grammar 3

Consider the following grammar:

\[ S \rightarrow W \ b \]
\[ W \rightarrow a \]
\[ W \rightarrow X \ a \]
\[ X \rightarrow a \]

Draw the CSFM for this grammar.

What is the problem?
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.