Use of the LR(0) automaton

The LR(0) automaton can be used for deriving a parsing table, which has a number of states equal to the states of the LR(0) automaton and the actions are dependent from the action of the automaton itself. The parsing table will have two different sections, one named ACTION and the other GOTO:

Parsing table

1. The ACTION table has a row for each state of the LR(0) automaton and a column for each terminal symbol. The value of ACTION[i,a] can have one of for forms:
   1. Shift j where j is a state (generally abbreviated as Sj).
   2. Reduce $A \rightarrow \beta$. The action of the parser reduces β to A in the stack (generally abbreviated as $R(A \rightarrow \beta)$)
   3. Accept
   4. Error
2. The GOTO table has a row for each state of the LR(0) automaton and a column for each nonterminal. The value of GOTO[I_i,A] = I_j if the GOTO function maps set of items accordingly on the LR(0) automaton

LR(0) table construction

LR(0) table

The LR(0) table is built according to the following rules, where "i" is the considered state and "a" a symbol in the input alphabet:

1. $\texttt{ACTION[}i,a\texttt{]} \leftarrow$ shift j
   if $[A \rightarrow \alpha \cdot a\beta]$ is in state i and $\texttt{GOTO(}i,a\texttt{)} = j$ – (generally represented as Sj)
2. $\texttt{ACTION[}i,*\texttt{]} \leftarrow$ reduce($A \rightarrow \beta$)
   if state i includes the item ($A \rightarrow \beta\cdot$) – (generally represented as R($A \rightarrow \beta$))
3. $\texttt{ACTION[}i,*\texttt{]} \leftarrow$ accept
   if the state includes the item $S' \rightarrow S\cdot$
4. $\texttt{ACTION[}i,*\texttt{]} \leftarrow$ error
   in all the other situations

LR(0) table construction

Consider the following grammars and sentences:
$S \rightarrow CC \quad C \rightarrow cC|d$ sentence: “ccd” and “ddd”

LR(0) table construction

Consider the following grammars and sentences:
$S \rightarrow aS|Ba \quad B \rightarrow Ba|b$ sentence: “aaba”

Use of the LR(0) automaton

Consider the string **id*id** and parse it

STACK	SYMBOLS	INPUT	ACTION
0	$	**id*id**$	$\cdots$

LR Parsing algorithm

General LR parsing program

The initial state of the parser is s_0 for the state and w (the whole string) on the input buffer.

Let a be the first symbol of w\$;
while true **do**
  let s be the state on top of the stack;
  if (ACTION[s,a] $=$ shift t) **then**
    push t onto the stack;
    let a be the next input symbol;
  else if (ACTION[s,a] $=$ reduce $A \to \beta$) **then**
    pop $|\beta|$ off the stack;
    let state t now be on top of the stack;
    push GOTO[t,A] onto the stack;
    output the production $A \to \beta$;
  else if (ACTION[s,a] $=$ accept) **then** break;
  else call error-recovery routine;
  end if
end while

SLR table construction

SLR(1) table

The LR(0) table is built according to the following rules, where "*i*" is the considered state and "*a*" a symbol in the input alphabet:

1. $\texttt{ACTION[}i,a\texttt{]} \leftarrow$ shift j
   if $[A \rightarrow \alpha \cdot a\beta]$ is in state i and $\texttt{GOTO(}i,a\texttt{)} = j$
2. $\texttt{ACTION[}i,a\texttt{]} \leftarrow$ reduce$(A \rightarrow \beta)$
   forall a in $\texttt{FOLLOW(}A\texttt{)}$ and if state i includes the item $(A \rightarrow \beta\cdot)$
3. $\texttt{ACTION[}i,\$\texttt{]} \leftarrow$ accept
   if the state includes the item $S' \rightarrow S\cdot$
4. $\texttt{ACTION[}i,*\texttt{]} \leftarrow$ error
   in all the other situations

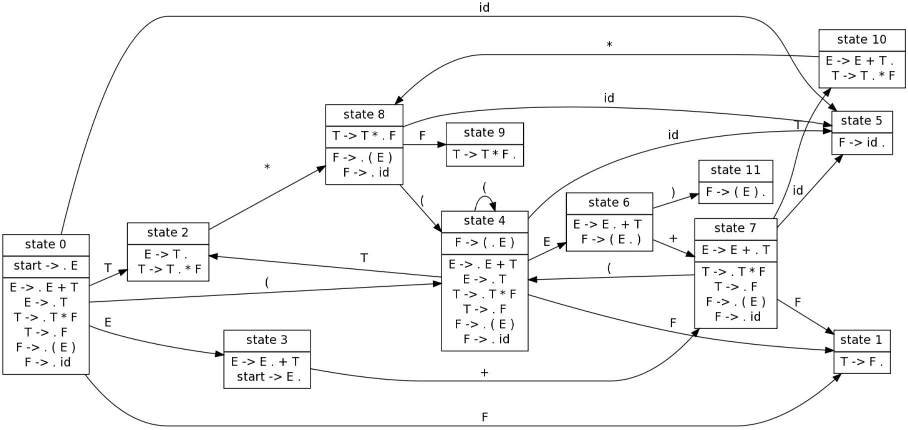
SLR table construction

Consider the following grammars and sentences:
$S \rightarrow aS|Ba \;\; B \rightarrow Ba|b$ sentence: “aaba”

LR(0) vs. SLR parsing

Consider the usual expression grammar:
$E' \rightarrow E \quad E \rightarrow E + T | T \quad T \rightarrow T * F | F \quad F \rightarrow (E) | id$
build LR(0) and SLR tables for the grammar, and then parse the sentence:

id∗id+id



http://smlweb.cpsc.ucalgary.ca/start.html

LL(1) vs. SLR(1)

Consider the following grammars:

- $S \rightarrow AaAb | BbBa \quad A \rightarrow \epsilon \quad B \rightarrow \epsilon$
- $S \rightarrow SA | A \quad A \rightarrow a$

Build parsing tables for LL(1) and SLR(1)

Towards more powerful parsers

Consider the following grammar and derive the SLR parsing table:
$S \rightarrow L = R | R \quad L \rightarrow *R | id \quad R \rightarrow L$