CS 406: More Powerful LR Parsers

Stefan D. Bruda

Winter 2016

LR(0) Automaton and Shift-Reduce Decisions

- In state 2 we generally reduce, but we can also shift whenever the next input token is $*$
  - If we look at the two items in State 2 then it is immediate we should shift (since no other rule will eat up $*$ next)
  - Such a decision can be made algorithmically by looking one token ahead

	+	$*$	(	)	id	$	⟨E'⟩	⟨E⟩	⟨T⟩	⟨F⟩
0			4		5			1	2	3
1	1, 6	1	1	1	1	1, accept	1	1	1	1
2	3	3, 7	3	3	3	3	3	3	3	3
3	5	5	5	5	5	5	5	5	5	5
4			4		5			8	2	
5	7	7	7	7	7	7	7	7	7	7
6				4	5				9	3
7			4		5					10
8	6			11						
9	2	2, 7	2	2	2	2	2	2	2	2
10	4	4	4	4	4	4	4	4	4	4
11	6	6	6	6	6	6	6	6	6	6

⟨E'⟩	::=	⟨E⟩	(1)
⟨E⟩	::=	⟨E⟩ + ⟨T⟩	(2)
⟨E⟩	::=	⟨T⟩	(3)
⟨T⟩	::=	⟨T⟩ $*$ ⟨F⟩	(4)
⟨T⟩	::=	⟨F⟩	(5)
⟨F⟩	::=	(⟨E⟩)	(6)
⟨F⟩	::=	id	(7)

SLR PARSING

- We assume an augmented grammar with the added rule ⟨S'⟩ ::= ⟨S⟩ (where ⟨S⟩ is the original axiom)
- We begin with the *LR*(0) items and automaton
- The decision on when to reduce to ⟨A⟩ are taken based on the set FOLLOW(⟨A⟩)
- The table is constructed using the following algorithm:
  1. Construct the *LR*(0) automaton with states $I_0, \ldots, I_n$
  2. Line i of the table is constructed starting from I_i, $1 \leq i \leq n$ as follows:
     1. If ⟨A⟩ ::= $\alpha \bullet a\beta \in I_i$, $a \in \Sigma$, and GOTO$(I_i, a) = I_j$ then $Action[i, a] = \boxed{j}$ (shift j)
     2. If ⟨A⟩ ::= $\alpha\bullet \in I_i$ then for all $a \in$ FOLLOW(⟨A⟩) set $Action[i, a]$ = reduce ⟨A⟩ ::= α
     3. If ⟨S'⟩ ::= ⟨S⟩$\bullet \in I_i$ then $Action[i, \$]$ = accept
     4. $Action[i,$ ⟨A⟩$]$ for ⟨A⟩ $\in N$ are computed as before (based on the automaton)
- We thus obtain a *SLR* parsing table and thus an *SLR* parser
  - Technically, this is a *SLR*(1) parsing table/parser
  - If any conflicting actions result from this algorithm then the grammar is not *SLR*(1)

State	+	*	(	)	id	$	⟨E'⟩	⟨E⟩	⟨T⟩	⟨F⟩
0			4		5			1	2	3
1	6					accept				
2	3	7		3		3				
3	5	5		5		5				
4			4		5			8	2	
5	7	7		7		7				
6			4		5				9	3
7			4		5					10
8	6			11						
9	2	7		2		2				
10	4	4		4		4				
11	6	6		6		6				

- The $LR(0)$ automaton characterizes the strings that can appear on the stack of a shift-reduce parser
- If the stack content is α and the rest of the input is x then a sequence of reductions will take αx to $\langle S \rangle$
- However, not all the prefixes can appear on the stack, since the parser must not shift past a handle
  - Example: $\langle E \rangle \overset{R}{\Rightarrow}^* \langle F \rangle * \text{id} \overset{R}{\Rightarrow}^* (\langle E \rangle) * \text{id}$
  - At various times the stack will hold the prefixes (, ($\langle E \rangle$, and ($\langle E \rangle$), but will never hold ($\langle E \rangle$) $*$ since ($\langle E \rangle$) is already a handle which must to be reduced to $\langle F \rangle$ before shifting $*$
  - We say that ($\langle E \rangle$) $*$ is not a viable prefix (but the others above are)
- The $LR(0)$ automaton recognizes viable prefixes
  - Item $\langle A \rangle ::= \beta_1 \bullet \beta_2$ is valid for a viable prefix $\alpha\beta_1$ if there exists a derivation $\langle S \rangle \overset{R}{\Rightarrow}^* \alpha \langle A \rangle w \overset{R}{\Rightarrow} \alpha\beta_1\beta_2 w$
  - That $\langle A \rangle ::= \beta_1 \bullet \beta_2$ is valid for the prefix $\alpha\beta_1$ tells us a lot about whether to reduce or to shift
    - If $\beta_1 \neq \varepsilon$ then this suggests that we do not yet have a handle on the stack, so we'd better shift
    - If $\beta_1 = \varepsilon$ then we do have a handle on the stack and so we can reduce

- The construction of a *SLR*(1) table may fail because the FOLLOW information is computed considering all the rules in the grammar
  - Sometimes this casts a larger net than necessary; consider:

$\langle S' \rangle$	$::=$	$\langle S \rangle$ \$	(1)
$\langle S \rangle$	$::=$	$\langle A \rangle\ \langle B \rangle$	(2)
	$\mid$	$a\ c$	(3)
	$\mid$	$x\ \langle A \rangle\ c$	(4)
$\langle A \rangle$	$::=$	a	(5)
$\langle B \rangle$	$::=$	b	(6)
	$\mid$	ε	(7)

I_0	GoTo
$\langle S' \rangle ::= \bullet \langle S \rangle$ \$	4
$\langle S \rangle ::= \bullet \langle A \rangle\ \langle B \rangle$	2
$\langle S \rangle ::= \bullet a\ c$	3
$\langle S \rangle ::= \bullet x\ \langle A \rangle\ c$	1
$\langle A \rangle ::= \bullet a$	3

I_3	GoTo
$\langle S \rangle ::= a \bullet c$	6
$\langle A \rangle ::= a \bullet$	

LIMITS TO *SLR* TABLE CONSTRUCTION

- The construction of a *SLR*(1) table may fail because the FOLLOW information is computed considering all the rules in the grammar
  - Sometimes this casts a larger net than necessary; consider:

$\langle S' \rangle$	$::=$	$\langle S \rangle$ \$	(1)
$\langle S \rangle$	$::=$	$\langle A \rangle \ \langle B \rangle$	(2)
	$\mid$	$a\ c$	(3)
	$\mid$	$x\ \langle A \rangle\ c$	(4)
$\langle A \rangle$	$::=$	a	(5)
$\langle B \rangle$	$::=$	b	(6)
	$\mid$	ε	(7)

I_0	GoTo
$\langle S' \rangle ::= \bullet \langle S \rangle$ \$	4
$\langle S \rangle ::= \bullet \langle A \rangle \ \langle B \rangle$	2
$\langle S \rangle ::= \bullet a\ c$	3
$\langle S \rangle ::= \bullet x\ \langle A \rangle\ c$	1
$\langle A \rangle ::= \bullet a$	3

I_3	GoTo
$\langle S \rangle ::= a \bullet c$	6
$\langle A \rangle ::= a \bullet$	

  - FOLLOW($\langle A \rangle$) includes b (because rule 2) and also c (because of rule 4)
  - If we reduce $\langle A \rangle$ in state I_3 then we would eventually need to apply rule 2, yet this inclusion is caused by the fact that $c \in$ FOLLOW($\langle A \rangle$), which happens because of rule 4
  - If we "split" $\langle A \rangle$ into two nonterminals, one used in rule 2 and the other in rule (4) then the grammar becomes *SLR*(1)!

LALR(1) Table Construction

- *LALR* (lookahead LR) offers a more precise decision on which tokens can follow a nonterminal
  - Based on the same *LR*(0) automaton
  - So the *LALR*(1) table has the same number of rows as the *SLR* table
  - Most popular given the good balance of power and efficiency
- The table is constructed using the following algorithm:
  - Given an augmented grammar with the added rule ⟨S'⟩ ::= ⟨S⟩ (where ⟨S⟩ is the original axiom):

1. Construct the *LR*(0) automaton with states $I_0, \ldots, I_n$
2. Line i of the table is constructed starting from I_i, $1 \leq i \leq n$ as follows:
   1. If $\langle A \rangle ::= \alpha \bullet a\beta \in I_i$, $a \in \Sigma$, and $\textsc{GoTo}(I_i, a) = I_j$ then $Action[i, a] = \boxed{j}$ (shift j)
   2. If $\langle A \rangle ::= \alpha\bullet \in I_i$ then for all $a \in ItemFollow((s, \langle A \rangle ::= \alpha\bullet))$ set $Action[i, a] =$ reduce $\langle A \rangle ::= \alpha$
   3. If $\langle S' \rangle ::= \langle S \rangle\bullet \in I_i$ then $Action[i, \$] =$ accept
   4. $Action[i, \langle A \rangle]$ for $\langle A \rangle \in N$ are computed as before (based on the automaton)

LALR(1) Propagation Graph

function BuildGraph()**:**

```
foreach state s do
    foreach item ∈ s do
        v ← Graph.AddVertex((s,item))
        ItemFollow(w) ← ∅
foreach rule ⟨S'⟩ ::= w do
    ItemFollow((start,⟨S'⟩ ::= •w)) ← {$}
foreach state s do
    foreach item ⟨A⟩ ::= α • ⟨B⟩γ ∈ s do
        v ← Graph.FindVertex((s, ⟨A⟩ ::= α • ⟨B⟩γ))
        Graph.AddEdge(v, (Action[s, ⟨B⟩], ⟨A⟩ ::= α⟨B⟩ • γ))
        foreach w ∈ Graph.FindVertex(s, ⟨B⟩ ::= •δ) do
            ItemFollow(w) ← ItemFollow(w) ∪ First(γ)
            if γ ⇒* x implies x = ε then
                Graph.AddEdge(v,w)
```

- While creating vertices we add to *ItemFollow* whatever actual tokens follow the nonterminal in discussion in the rules themselves (First(γ))
- Edges account for those cases in which whatever follows the nonterminal in the rule rewrites to ε

LALR(1) PROPAGATION GRAPH (CONT'D)

```
function EVALGRAPH():
    repeat
        changed ← False
        foreach all edges (v, w) in Graph do
            old ← ItemFollow(w)
            ItemFollow(w) ← ItemFollow(w) ∪ ItemFollow(v)
            if ItemFollow(w) ≠ old then changed ← True
    until not changed:
function LALRLOOKAHEAD():
    BUILDGRAPH()
    EVALGRAPH()
```

- Recall that the edges are created when whatever follows the nonterminal ⟨B⟩ in the rule ⟨A⟩ $::= \ \alpha \bullet$ ⟨B⟩γ rewrites to ε
- In such a case whatever follows ⟨A⟩ must also follow ⟨B⟩

- Lookahead is either generated (when FIRST(γ) $\neq \emptyset$) or propagated (when $\gamma \Rightarrow^* \varepsilon$)
- There is no guarantee for the running time of EVALGRAPH since multiple passes may be necessary
    - In practice however the algorithm converges quickly