Lecture 20: Finishing up the LR(0) method

Jeffrey Shallit
School of Computer Science
University of Waterloo
shallit@uwaterloo.ca
https://cs.uwaterloo.ca/~shallit/
Knuth DFA

\[ S \rightarrow T \]
\[ T \rightarrow aTa \mid bTb \mid c \]
LR(0) parsing

How do we parse using the Knuth DFA?

- An LR(0) parser is a DPDA that generates a rightmost derivation.
- The stack of the DPDA holds a viable prefix of a right sentential form \( \alpha \), including all variables of \( \alpha \).
- Actually, the stack holds this viable prefix together with states of the Knuth DFA interspersed between symbols of the viable prefix: a kluge that allows us to easily “back up” in the Knuth DFA.
- The remainder of \( \alpha \) appears as the unread input.
Initially the LR(0) parser is in a configuration \((q, w, q_0)\) where \(q_0\) now means the initial state of the corresponding Knuth DFA for the grammar.

Our description only mentions a single state \(q\), but allows the PDA to pop multiple symbols in a single step. Any implementation by a normal PDA would require extra states to handle these pops.

At each step, the parser has two choices:

(i) to **shift** a symbol from the input to the stack, updating the state of the Knuth DFA, or

(ii) to **“reduce”**, or pop \(2|\alpha|\) symbols from the stack, where \(A \rightarrow \alpha \bullet\) is a complete item on top of the stack, and then push \(A\) and the appropriate state of the Knuth DFA back on top of the stack.

For this reason, LR parsers are sometimes called *shift-reduce* parsers.
**Definition.** A grammar \( G = (V, \Sigma, P, S) \) is LR(0) if each of the following conditions hold:

(a) \( G \) has no useless symbols;

(b) The start symbol \( S \) does not appear on the right-hand side of any production;

(c) For all viable prefixes \( \gamma \), if \( A \to \alpha \bullet \) is a complete item valid for \( \gamma \), then no other complete item nor any item with a terminal immediately to the right of the \( \bullet \) is valid for \( \gamma \).

Condition (a) was needed to prove that the Knuth DFA works. Condition (b) is a technical condition that allows us to know when the derivation is complete. Condition (c) says there is never a reduce-reduce conflict, or a shift-reduce conflict.
How do we tell if a given grammar is LR(0)?

Just check the conditions.

Conditions (a) and (b) are easy to check.

For condition (c), just build the Knuth DFA and see if there is a state having reduce-reduce conflicts or shift-reduce conflicts.

If the grammar isn’t LR(0), this can be demonstrated by finding a conflict, or building only enough of the NFA-\(\epsilon\) or DFA to exhibit it.
An LR(0) parser behaves as follows: a typical configuration before a move looks like

\[(q, a_t a_{t+1} \cdots a_n, q_k X_k q_{k-1} X_{k-1} \cdots q_1 X_1 q_0)\]

where

- \(x = a_1 \cdots a_n\) is the input;
- \(X_1 \cdots X_k a_t a_{t+1} \cdots a_n\) is the current right sentential form; and
- \(q_j = d(q_{j-1}, X_j), 1 \leq j \leq k\), where \(d\) is the transition function of the Knuth DFA.
If $q_k$ contains a complete item of the form $A \rightarrow \alpha\bullet$, then
\[
\alpha = X_{i+1} \cdots X_k \text{ for some } i \geq 0,
\]
and the new configuration is
\[
(q, a_t a_{t+1} \cdots a_n, q'A q_i X_i q_{i-1} \cdots q_1 X_1 q_0)
\]
where $q' = d(q_i, A)$.

Otherwise, the new configuration is
\[
(q, a_{t+1} \cdots a_n, q' a_t q_k X_k q_{k-1} X_{k-1} \cdots q_1 X_1 q_0)
\]
where $q' = d(q_k, a_t)$. We accept, by emptying the stack, if there is a complete item $S \rightarrow \alpha\bullet$ on top of the stack.
Theorem. Let $G$ be an LR(0) grammar, let $x \in L(G)$, and let $\alpha \neq S$ be an rsf appearing in a derivation of $x$, that is, suppose $S \Longrightarrow^* \alpha \Longrightarrow^* x$ by a rightmost derivation. Then there is a unique rsf $\beta$ such that $S \Longrightarrow^* \beta \Longrightarrow \alpha \Longrightarrow^* x$. 
Proof. Suppose the rsf is $\alpha = X_1X_2\cdots X_ky$, $y \in \Sigma^*$ and one rightmost derivation is

$$S \Rightarrow^* X_1X_2\cdots X_jAy \Rightarrow \alpha = X_1X_2\cdots X_ky \Rightarrow^* x,$$

using the production $A \rightarrow X_{j+1}\cdots X_k$. Suppose there is another possible rsf previous to $\alpha$, and consider the corresponding right end of the handle in $\alpha$. There are three possibilities:

(i) the handle ends to the right of $X_k$ (and hence the end is inside $y$);

(ii) the handle ends at $X_k$;

(iii) the handle ends at $X_t$ for some $t < k$. 
Consider $s = d(q_0, X_1 X_2 \cdots X_k)$ in the Knuth DFA for $G$.

Then $s$ contains a complete item, namely $A \rightarrow X_{j+1} \cdots X_k \bullet$.

But by the LR(0) rules, this means that $s$ contains no other complete items (ruling out case (ii)) and contains no items with a terminal immediately to the right of the dot (ruling out case (i)).
Finally, we have to rule out case (iii).

To do so, suppose there is a rightmost derivation

\[ X_1 X_2 \cdots X_r B X_{t+1} \cdots X_k y \Rightarrow X_1 X_2 \cdots X_k y \]

using a production \( B \rightarrow X_{r+1} \cdots X_t \).

Since the derivation is rightmost,

\[ \text{each of } X_{t+1}, \ldots, X_k \text{ is a terminal.} \quad (1) \]

Now complete item \( B \rightarrow X_{r+1} \cdots X_t \bullet \) is valid for viable prefix \( X_1 \cdots X_t \), but then, since \( X_1 X_2 \cdots X_k \) is also a viable prefix, there must be some other item valid for \( X_1 \cdots X_t \). And by (1), this item must have a terminal to the right of the dot or be complete. But this would violate the LR(0) rules, a contradiction.
An example of parsing via LR(0)

\[ S \rightarrow T \]
\[ T \rightarrow aTa \mid bTb \mid c \]

<table>
<thead>
<tr>
<th>unread input</th>
<th>stack contents</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>abcba</td>
<td>0</td>
<td>shift</td>
</tr>
<tr>
<td>bcba</td>
<td>2a0</td>
<td>shift</td>
</tr>
<tr>
<td>cba</td>
<td>3b2a0</td>
<td>shift</td>
</tr>
<tr>
<td>ba</td>
<td>4c3b2a0</td>
<td>reduce</td>
</tr>
<tr>
<td>ba</td>
<td>6T3b2a0</td>
<td>shift</td>
</tr>
<tr>
<td>a</td>
<td>8b6T3b2a0</td>
<td>reduce</td>
</tr>
<tr>
<td>a</td>
<td>5T2a0</td>
<td>shift</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>7a5T2a0</td>
<td>reduce</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>1T0</td>
<td>reduce</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>-</td>
<td>accept</td>
</tr>
</tbody>
</table>
Corollary. If $G$ is $LR(0)$, then it is unambiguous.

Proof. For every right sentential form in the derivation of $w \in L(G)$ there is only one previous right sentential form.
Theorem. Let $M$ be the DPDA specified above, based on the LR(0) grammar $G$. Then $L(G) = L_e(M)$, where by $L_e$ we mean acceptance by empty stack.

Proof. First we prove that $L_e(M) \subseteq L(G)$.

Suppose that $x \in L_e(M)$. We will prove that $x \in L(G)$ by producing a rightmost derivation of $x$.

We define $\alpha_i$, the right sentential form represented by the configuration of the DPDA $M$ at step $i$, to be the string $X_1X_2\cdots X_ky$ if the DPDA at step $i$ has configuration $(q, y, s_kX_ks_{k-1}\cdots s_1X_1s_0)$.
Wrapping it all up

We will prove the following two assertions by induction on \( i \). Let \( \alpha_{-1} = \alpha_0 \).

1. \( d(q_0, X_1X_2 \cdots X_j) = s_j \) for all \( j, 0 \leq j \leq k \), where \( d \) is the transition function of the associated Knuth DFA; and

2. Either \( \alpha_i \implies \alpha_{i-1} \) or \( \alpha_i = \alpha_{i-1} \).

For \( i = 0 \), both (1) and (2) are true. (1) is true since the initial configuration of the DPDA is \((q, x, q_0)\), and \( d(q_0, \epsilon) = q_0 \). (2) is true since \( \alpha_0 = \alpha_{-1} = x \).

Now assume the assertions are true for steps \( < i \); we prove them for \( i \).

Suppose the configuration of the DPDA before step \( i \) is \((q, y, s_kX_ks_{k-1} \cdots s_1X_1s_0)\). At step \( i \) the DPDA either reduces or shifts.
(A) Reduce move: If the DPDA makes a reduce move, we know that $s_k$ contains a complete item $A \rightarrow \gamma \bullet$.

By induction $s_k = d(q_0, X_1 X_2 \cdots X_k)$.

Since $d$ is the transition function for the Knuth automaton, we know that $A \rightarrow \gamma \bullet$ is valid for viable prefix $X_1 X_2 \cdots X_k$.

In other words, there exists a rightmost derivation

$$S \Rightarrow^* \beta A z \Rightarrow \beta \gamma z$$

where $\beta \gamma = X_1 X_2 \cdots X_k$. 
Wrapping it all up

It follows that $\gamma$ is a suffix of $X_1X_2\cdots X_k$ and hence when we pop $2|\gamma|$ symbols from the stack we are left with

$$s_jX_1s_{j-1}\cdots s_1X_1s_0$$

for some $j$, with $X_1\cdots X_j = \beta$, $X_{j+1}\cdots X_k = \gamma$.

Then we push $A$ and $d(s_j, A)$ maintaining the invariant (1).

On the other hand, the invariant (2) is preserved because we have

$$\alpha_i = X_1X_2\cdots X_jAy \implies X_1X_2\cdots X_j\gamma y = X_1X_2\cdots X_ky = \alpha_{i-1}.$$
(B) Shift move: If the DPDA makes a shift move, the invariant (1) is trivially preserved, and (2) is preserved because \( \alpha_j = \alpha_{j-1} \).

Finally, since \( x \in L_e(M) \), we know the DPDA eventually empties its stack and accepts its input.

This can only occur if at some step, say step \( n \), the configuration is \( (q, \epsilon, s_kX ks_{k-1} \cdots s_1X_1s_0) \) and \( s_k \) contains a complete item of the form \( S \rightarrow \gamma \bullet \).
If this is the case, by the reasoning above, there is a rightmost derivation 

\[ S \Rightarrow^* \beta Sz \Rightarrow \beta \gamma z, \]

with \( X_1X_2 \cdots X_k = \beta \gamma. \)

However, since \( S \) does not appear on the right-hand side of any production, we must have \( S = \beta Sz \), so it follows that \( \beta = \epsilon \) and \( z = \epsilon. \)

Hence \( X_1X_2 \cdots X_k = \gamma. \)

Now define \( \alpha_{n+1} = S. \) Then we have \( \alpha_i \Rightarrow^* \alpha_{i-1} \) for \( 1 \leq i \leq n + 1. \)

Since \( \alpha_0 = x, \) this gives a derivation of \( x \) in \( G. \)
Wrapping it all up

Now let us show that $L(G) \subseteq L_e(M)$. Let $x \in L(G)$. As we have seen there is only one rightmost derivation $S \Rightarrow^* x$.

Suppose this derivation is of length $n$ and

$$S = \alpha_n \Rightarrow \alpha_{n-1} \Rightarrow \cdots \Rightarrow \alpha_0 = x.$$ 

We want to argue that $M$, when given $x$, eventually pops its stack and halts.

To do so, we need to create a measure of “progress” towards an accepting computation.
Wrapping it all up

Suppose $C = (q, y, s_k X_{k-1} \cdots s_1 X_1 s_0)$ is a configuration of $M$ on input $x$. If $X_1 X_2 \cdots X_k y = \alpha_i$, then we define the weight of $C$ to be $n - i + |y|$. We then argue that each move of the DPDA is correct and reduces the weight of its configuration.

Initially the configuration is $(q, x, q_0)$, with weight $n + |x|$. At each step $M$ either reduces or shifts. Consider what happens at step $i$.

If $M$ reduces, then there must be a complete item on top of the stack. By above, there is only one handle in $X_1 \cdots X_k y$ and it must be $X_{j+1} \cdots X_k$ with corresponding production $A \to X_{j+1} \cdots X_k$. The machine now performs a reduce move, and the corresponding weight decreases by 1.
Wrapping it all up

If $M$ shifts, then there is no complete item on top of the stack. We now argue that shifting is the right thing to do.

Suppose there were a complete item $A \rightarrow \gamma$ buried in the stack. Then this complete item would have been added at some point.

Consider the very next step. Since a complete item is on top, we would do a reduce move, popping $2|\gamma|$ symbols from the stack, so if $\gamma \neq \epsilon$, this complete item gets popped from the stack and cannot be buried.

If $\gamma = \epsilon$, then $A$ and $d(s_k, A)$ is put on top of $A \rightarrow \bullet$ in the stack.
Wrapping it all up

If in any future step $X_1X_2\cdots X_k$ has not risen to the top of the stack, then there will be a variable on top of $X_1X_2\cdots X_k$.

But then $\epsilon$ (via $A \to \epsilon$) cannot be the handle of any right sentential form $X_1X_2\cdots X_k\epsilon\beta$, where $\beta$ contains a variable, because then $X_1X_2\cdots X_kA\beta \Rightarrow X_1X_2\cdots X_k\beta$ would not be a rightmost derivation.

Thus the handle must include some symbols of the input, and shifting is the right thing to do. This reduces the weight of the configuration by 1.

Eventually the weight of the configuration becomes 0. At this point we have $i = n$ and $y = \epsilon$, and the DPDA pops its stack, accepting. Thus $x \in L_e(M)$. We’re done!
Unfortunately, LR(0) grammars are not strong enough for the syntax of most programming languages.

It turns out that LR(0) grammars define precisely the DCFL’s with the prefix property: if $x \in L$, then no proper prefix of $x$ is in $L$.

This is pretty restrictive. But you can always turn a DCFL $L$ into one with the prefix property by considering $L\$$, where $\$$ is a special new symbol that signals the end of the string.

A generalization of LR(0), called LR($k$), allows the parser to look at the next $k$ symbols of the input. Provided $k \geq 1$, these grammars can specify all DCFL’s.