CS164 - LR Parsing Table Example (Weimer)

Consider the grammar:

$$\begin{aligned} A &\rightarrow A + B \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

Some strings in this grammar are: a, $a+b$, $a+b+b$, etc. This grammar is left-recursive. Let's build the LR parsing table through closures and state diagrams. First we add the extra production:

$$S \rightarrow A$$

Just to make it clear where things start. Now it's time to start building the DFA, state by state. The first state is related to the production $S \rightarrow A$. We'll discover how many states there are later. Recall that an LR(1) item looks like this: $[A \rightarrow \alpha \bullet \beta, t]$ where $A \rightarrow \alpha\beta$ is a production in the grammar, t is a terminal (called the lookahead) and the dot conceptually marks where we are (we have already seen α, we're looking for β).

State 0:

We start with: $[S \rightarrow \bullet A, \$]$. Now we have to take the closure. To do that, we look to see if there is a $\bullet$ before a non-terminal. In fact, there is a $\bullet$ before an A. So we look at all of the productions in the grammar that start with $A \rightarrow$ and we bring them in to the closure. That adds $[A \rightarrow \bullet A + B, \$]$ and $[A \rightarrow \bullet a, \$]$. Are we done yet? Not necessarily: we brought in another item that has a $\bullet$ before a non-terminal. So we need to take the closure again. This time, since the production near the dot looks like $\bullet A+$ we'll use $+$ as the lookahead. So once again we bring in another item for all of the productions that start with $A \rightarrow$. That add $[A \rightarrow \bullet A + B, +]$ and $[A \rightarrow \bullet a, +]$. We've brought in another item with a $\bullet$ before a non-terminal, but taking the closure again would not yield any new items (try it!).

Final tally: $[S \rightarrow \bullet A, \$]$, $[A \rightarrow \bullet A + B, \$]$, $[A \rightarrow \bullet a, \$]$, $[A \rightarrow \bullet A + B, +]$, $[A \rightarrow \bullet a, +]$.

Now we want to find transitions out of this state. There is a transition out of this state for every symbol X after a $\bullet$ in one of the items. Looking at our five items, we find a $\bullet$ before an a and an A. So there are two transitions out of **State 0**. Let's say that on A we transition to **State 1** and that on a we transition to **State 2**. You may find it helpful to draw out a diagram on a piece of scratch paper. [Note that I could have said that A goes to State 2 and a goes to State 1 and the DFA would have been the same up to relabeling.]

State 1:

We need to figure out which items appear in **State 1**. We go here by coming from **State 0** and transitioning on A. So we look at all of those items in **State 0** with a $\bullet$ before an A:

State 0 items with $\bullet A$: $[S \rightarrow \bullet A, \$]$, $[A \rightarrow \bullet A + B, \$]$, $[A \rightarrow \bullet A + B, +]$.

In each of those items, we push the $\bullet$ past the A to get **State 1** items.

State 1 items: $[S \rightarrow A\bullet, \$]$, $[A \rightarrow A \bullet + B, \$]$, $[A \rightarrow A \bullet + B, +]$.

Now we take the closure. Since there are no items with a $\bullet$ in front of a non-terminal, we are done (huzzah!).

Where do we go from here? One of our items ($[S \rightarrow A\bullet, \$]$) has the dot way at the end of the production. So it's a possible reduction. We mark that in **State 1** we reduce $S \rightarrow A$ on \$. There are also transitions out of this state for every symbol after the dot in other items. Here we only see the $+$ symbol after a dot, so there is only one transition out of here. Let's say that on $+$ we transition to **State 3**.

State 2:

How did we get here? We came from **State 0** on a. So we start with all of the **State 0** items that have a dot before an a:

State 0 items: $[A \rightarrow \bullet a, \$]$, $[A \rightarrow \bullet a, +]$.

And then we push the dot past the a.

State 2 items: $[A \rightarrow a\bullet, \$]$, $[A \rightarrow a\bullet, +]$.

Now we take the closure by looking for places where there is a dot before a non-terminal. Huzzah, there are none. So we're done with the closure. In this state, all of our items have the dot at the end of the production. So we only have reductions here. We mark that on + or \$, reduce $A \to a$. There are no transitions out of this state because there are no items with a dot before another symbol.

State 3:

We are in this state if we started in **State 1** and saw a +. So we take all of the items from **State 1** with a dot before a +.

Select **State 1** items: $[A \to A \bullet + B, \$]$, $[A \to A \bullet + B, +]$.

Now we push the $\bullet$ past the + to get the **State 3** items.

State 3 items: $[A \to A + \bullet B, \$]$, $[A \to A + \bullet B, +]$.

Now we have to take the closure. Uh-oh! This time there is a dot before a non-terminal. Two dots before non-terminals, in fact. Let's bring in the closure of item $[A \to A + \bullet B, \$]$ first. We need to bring in all of the productions of the form $B \to$ *something*. The lookahead token will remain \$. So that brings in the item $[B \to \bullet b, \$]$. Now let's bring in the closure of the item $[A \to A + \bullet B, +]$. This time the lookahead token will be +. This brings in the item $[B \to \bullet b, +]$. Now we're done taking the closure (because there are no new productions with a dot before a non-terminal).

State 3 final tally: $[A \to A + \bullet B, \$]$, $[A \to A + \bullet B, +]$, $[B \to \bullet b, \$]$, $[B \to \bullet b, +]$.

What can we do in **State 3**? There are no dots at the end of a production, so there are no reductions here. But there are dots before symbols. We have one transition out for each symbol after a dot. In the items we see that there is a $\bullet$ before B and b. So we'll have two transitions out. Let's say that on B we transition to **State 4** and on b we transition to **State 5**.

State 4.

We got here from **State 3** on B. So we start with all the **State 3** items that have a dot before a B.

Select **State 3** items: $[A \to A + \bullet B, \$]$, $[A \to A + \bullet B, +]$.

Now we push the $\bullet$ past the B.

State 4 items: $[A \to A + B \bullet, \$]$, $[A \to A + B \bullet, +]$.

Now we take the closure. There are no dots before non-terminals. Huzzah! So we've got all of the items for **State 4**. What can we do here? Well, there are dots at the end of productions, so reductions are possible. In particular, on \$ or + we reduce $A \to A + B$. There are no items with dots in front of symbols, so no transitions are possible.

State 5.

We got here from **State 3** on b. So take all of the **State 3** items that have $\bullet b$ in them:

Select **State 3** items: $[B \to \bullet b, \$]$, $[B \to \bullet b, +]$.

Now we push the dot past the b to get **State 5** items.

State 5 items: $[B \to b \bullet, \$]$, $[B \to b \bullet, +]$.

What can we do here? Reductions! We have dots at the ends of productions, so on \$ or + we reduce $B \to b$. There are no dots in front of symbols, so there are no transitions out of here. So we're done with this state.

And that's it. No more states to deal with. If you've been following along, you've drawn a pretty picture of the annotated DFA. It has six states (numbered **0** through **5**) and five transitions. Let's make the LR action/goto table for it. That's a table where the states are the rows and the terminals/non-terminals are the columns:

	a	b	$+$	\$	A	B
0						
1						
2						
3						
4						
5						

Let's fill in the row for **State 0** together. Looking back at our notes above, we see that on A we transition to **State 1** and on a we transition to **State 2**. Transitions on terminals like a are called "shifts" and transitions on non-terminals like A are marked as "gotos".

	a	b	$+$	\$	A	B
0	shift 2				goto 1	

All of the other entries are blank. So, for example, if we are parsing and we are in **State 0** and we see a $+$, it's a parse error.

Now let's fill in the row for **State 1**. Checking our notes we see that on \$ we reduce $S \to A$ and $+$ we transition to 3. Once again, a transition on a terminal like $+$ is a "shift".

	a	b	$+$	\$	A	B
1			shift 3	reduce $S \to A$		

Now that we've seen all of the possible cell entries (shift, reduce and goto), let's fill in the rest of the table all at once. If you're following along at home, fill out the blank table on the previous page without looking at the answer here and then check to see if you got it right.

	a	b	$+$	\$	A	B
0	shift 2				goto 1	
1			shift 3	reduce $S \to A$		
2			reduce $A \to a$	reduce $A \to a$		
3		shift 5				goto 4
4			reduce $A \to A + B$	reduce $A \to A + B$		
5			reduce $B \to b$	reduce $B \to b$		

Great! That table worked out perfectly. Sometimes you end up with a cell with two entries. For example, for some other grammar maybe in State 7 when you see a $*$ you want to shift to 8 and reduce $Q \to E/T$. That's called a shift/reduce conflict. There are also reduce/reduce conflicts. This table does not have any of those, so the grammar is LR(1). Convenient, since this is supposed to be an LR(1) example. :-)

Now that we have the table, let's practice parsing something. How about the string $a + b + b$? Take a moment to verify that it's in the grammar (does anyone actually do that when the book or notes asks them to? I thought not).

When we say "parsing" here, we're trying to get a derivation for $S \to^* a + b + b$. LR parsing will give us the reverse of a right-most derivation by reading the input left to right. Since this grammar is unambiguous (take a moment to verify that!) the right-most derivation is the same as the left-most derivation (it's the only derivation).

When we simulate parsing, we have to keep track of the Stack and the remaining input. We can use the the DFA or the Table to figure out what state we are in given the Stack. Each time we perform a reduction we get part of the derivation.

We start in **State 0** with nothing on the Stack. The input is $a + b + b$. We see that on a we shift to **State 2**. Shift means "take the first part of the input and put it on the stack".

So now we're in **State 2** with a on the Stack. The input is $+b + b$. We see that on $+$ we reduce $A \to a$. Reduce $X \to Y$ means "take Y off the top of the stack and put X there instead". So we take a off the top of the stack and put A there instead. But what state are we in now? One way to find out is to run the DFA on the Stack, starting in **State 0**. We see that on A we go to **State 1**. We could also (equivalently, since they hold the same information) use the Table and the Stack. Starting in **State 0** on the Table, we see that on A we goto **1**. So any way you slice it we're in State 1.

So now we're in **State 1** with A on the Stack. The input is $+b + b$. We see that on $+$ we shift to **State 3**. So now the Stack has $A+$ (where the top of the Stack is on the right) and the input has $b + b$.

So now we're in **State 3** with $A+$ on the Stack and $b + b$ left in the input. We look up $(3, b)$ in the table and find that we should shift to **State 5**. So we take the b from the input and put it on the Stack, leaving us with $+b$ in the input and $A + b$ on the Stack.

Now we're in **State 5** with $A + b$ on the Stack and $+b$ left in the input. Looking in the table we see that on $+$ we reduce $B \to b$. So we take the b off the top of the stack and replace it with a B, giving us $A + B$. The input is unchanged. Where are we now? We have to feed the Stack to the DFA (or the Table) to find out. Starting in **State 0**, on A we goto **State 1** and then on $+$ we goto **State 3** and then on B we goto **State 4**. So we're in **State 4**.

Now we're in **State 4** and we have $A + B$ on the Stack and $+b$ left in the input. On $+$ we see that we reduce $A \to A + B$, so we take $A + B$ off the Stack and put an A there. So now the Stack has just A. The input is unchanged. What state are we in? Run the DFA (or the Table) on the Stack starting from **State 0**. On A we goto **State 1**, so that's it.

Here we are in **State 1** with A on the Stack and $+b$ left in the input. The action is "shift 3", so we consume the $+$ and put it on the Stack, giving us $A+$ on the Stack and b left in the input.

So now we're in **State 3** with $A+$ on the Stack and b in the input. On b we shift to **State 5**. So now we have $A + b$ on the Stack and \$ left in the input. We better not do any more shifts, because we don't have any input left to consume!

Now we're in **State 5** with $A + b$ on the Stack and \$ in the input. **State 5** on \$ reduces $B \to b$, so we take the b off the top of the stack and replace it with B. That gives us $A + B$. What state are we in? Run the DFA (or the Table) on the Stack. Starting in **State 0**, on A we goto **State 1** and then on $+$ we goto **State 3** and then on B we goto **State 4**. So **State 4** it is.

Now we're in **State 4** with $A + B$ on the Stack and \$ in the input. We reduce $A \to A + B$, so now the Stack is A and we should go to **State 1**. (Why? For a hint, look at what we did in **State 4** last time.)

Now we're in **State 1** with A on the Stack and \$ in the input. So we reduce $S \to A$. So now the stack has S and the input is \$. Since S is what we wanted to end up with and we covered all the input, by convention we accept. We could also have had an explicit "accept" somewhere in our Table, but that's not critically important.

I hope you had fun with LR(1) parsing!