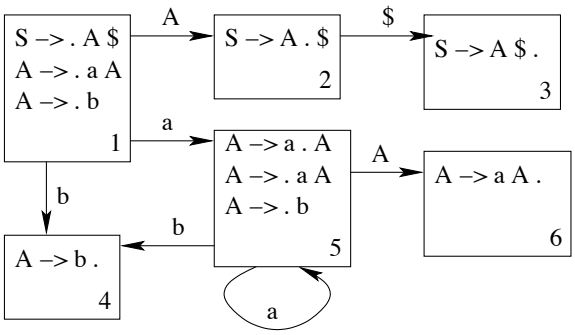
Simple LR(0) Parsing Example

Consider the grammar G:

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

The strings $b\ \$$, $a\ b\ \$$, $a\ a\ b\ \$$ and so forth are members of L(G).

Let us look at the parse of $a\ a\ b\ \$$ by an LR(0) parser for this grammar. The parser can be represented as a Characteristic Finite State Machine (CFSM). We are not concerned at this time how the CFSM is created. The CFSM for the grammar G is:



The parser begins in state **1**, and is attempting to match tokens to a right hand side of $E\ \$$. The parser will maintain two stacks

1. A symbol stack containing terminals shifted during the parse that are part of the right hand side of a not yet recognized prodction, and non-terminals that are part of the right hand side of a not yet recognized production;
2. A stack of states.

These stacks and the current token are initially:

consumed symbols	ϵ
current symbol	a
symbol stack	ϵ
state stack	1

The current symbol is an 'a', a terminal, and so it is *shift*ed onto the stack. Thus the parser is attempting to recognize the production $S \rightarrow A$ \$, which in turn requires recognizing an A production. Because the current symbol is 'a', we shift to state **5**, which is recognizing $A \rightarrow a\ A$.

consumed symbols	a
current symbol	a
symbol stack	a
state stack	1 5

The current symbol is now 'a' (the second 'a' in the input string being parsed.) We are in state **5**, parsing $A \rightarrow a\ A$. To complete parsing of $A \rightarrow a\ A$ the A on the right hand side of the production must be recognized. This can either be done by recognizing $A \rightarrow a\ A$ or $A \rightarrow b$. Since the current input is an 'a', the production that will yield a A is $A \rightarrow a\ A$, and we do a shift into state **5**. The stacks contents are now:

consumed symbols	aa
current symbol	b
symbol stack	aa
state stack	1 5 5

As before, In state **5** we again need to recognize the A on the right hand side of $A \rightarrow a\ A$, and again can do it by either recognizing the right hand side of $A \rightarrow a\ A$ or of $A \rightarrow b$. The current symbol is a 'b', and so the right hand side of $A \rightarrow b$ is the candidate to be recognized. The parser performs a shift into state **4**, and the resulting stack is:

consumed symbols	aab
current symbol	\$
symbol stack	aab
state stack	1 5 5 4

The state **4** corresponds to the entire right hand side of the production $A \rightarrow b$, and thus it has been recognized. The parser now needs to return to the state where it began to recognize the right hand side of $A \rightarrow b$. This it does by performing a *reduce*. Let *len* be the length of the production right hand side. Then going back *len* states will get us to the state where the parser began pushing the right hand side of the production onto the stack, that is the state where the parser began recognizing the production $A \rightarrow b$. The parser thus pops *len* symbols and states off the symbol and state stacks and pushes the production left hand side (A) onto the stack, giving:

consumed symbols	aab
current symbol	\$
symbol stack	aaA
state stack	1 5 5

The state at the top of the state stack is **5**. Going to state **5**, there is a transition on A to state **6**. Pushing this transition onto the stack (this is still part of the reduce, and not a shift) the parser ends up with stacks that look like:

consumed symbols	aab
current symbol	\$
symbol stack	aaA
state stack	1 5 5 6

At state **6** the parser has recognized another production right hand side, this time the production $A \rightarrow a\ A$. Thus another reduce will be performed. The length of this right hand side is 2, so two symbols and two states are popped off the respective stack, and the production left hand side A is pushed onto the symbol stack, yielding:

consumed symbols	aab
current symbol	\$
symbol stack	aA
state stack	1 5

State **5** is on top of the stack, and at state **5** with symbol A, the parser goes to state **6**. The stack at the end of the reduce is:

consumed symbols	aab
current symbol	\$
symbol stack	aA
state stack	1 5 6

The parser is again at stack **6**, and will again perform a reduction. Again popping two symbols and two states off the stack yields and pushing the production left hand side onto the stack yields:

consumed symbols	aab
current symbol	\$
symbol stack	A
state stack	1

State **1** will goto state **2** when an A is seen, finishing the reduce. The stacks are now:

consumed symbols	aab
current symbol	\$
symbol stack	A
state stack	1 2

In state **2**, when a '\$' is seen, a shift into state **3** is performed. This yields the stack contents:

consumed symbols	aab\$
current symbol	ϵ
symbol stack	A\$
state stack	1 2 3

State **3** recognizes the goal production. No reduce is performed. Instead the parser is said to recognize the string as a sentence in the language.