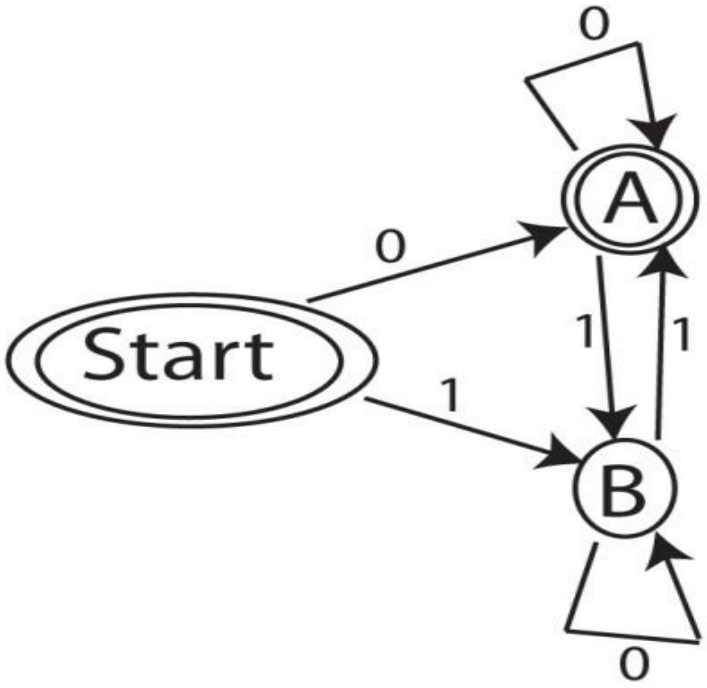
Table-Driven Scanners

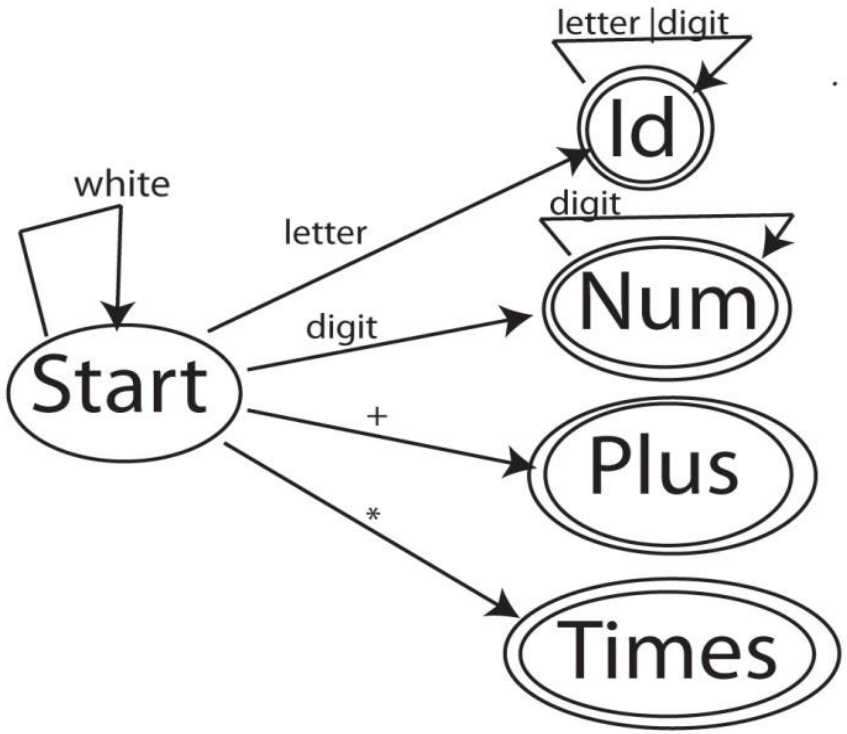
The idea of a table-driven scanner is simple -- we have a table that says what to do if we are in one particular state and see a particular input character. This is basically the transition table for the DFA. Building the table can be a pain, but then your scanner is a simple loop. The table size can be significant (my Scanner for BPL has 35 tokens, so the DFA needs at least 35 states; there are at least 128 possible input characters, so you need a table with about 5,000 entries for BPL) but even for a full implementation of C or Java it wouldn't be too big to reside in memory.

Here is a simple example: a table that represents a DFA:



	Char		
State	0	1	End-Of-String
Start	A	B	Accept
A	A	B	Accept
B	B	A	Reject

Things get a little more complex if you have an actual scanner, which has to find token boundaries and emit tokens. Consider the following DFA, that represents a scanner that finds Id, Number, + and * tokens:



We could represent this with the following table:

	Char					
State	letter	digit	+	*	white	eof
Start	Id	Num	Plus	Times	Start	T_ERR
Id	Id	Id	T_ID	T_ID	T_ID	T_ID
Num	T_NUM	NUM	T_NUM	T_NUM	T_NUM	T_NUM
Plus	T_PLUS	T_PLUS	T_PLUS	T_PLUS	T_PLUS	T_PLUS
Times	T_TIMES	T_TIMES	T_TIMES	T_TIMES	T_TIMES	T_TIMES

Here a table entry Id means to go to the Id state and the next character of the input, while an entry such as T_ID means to issue a T_ID token and to not consume the current input character.

Some people separate this into two tables: a Transition table for states, and an Action table that issues tokens and consumes input.

Should you write a table-driven scanner for BPL? I wouldn't. Scanners are not very interesting and you just want to get yours done; it seems to me that code is easier to debug than a table. But if you are so inclined, go for it. To write such a scanner you will need some way to generate the table; don't try to do it by hand.