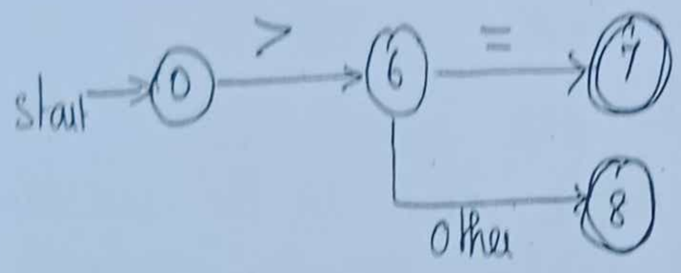
Transition Diagrams:

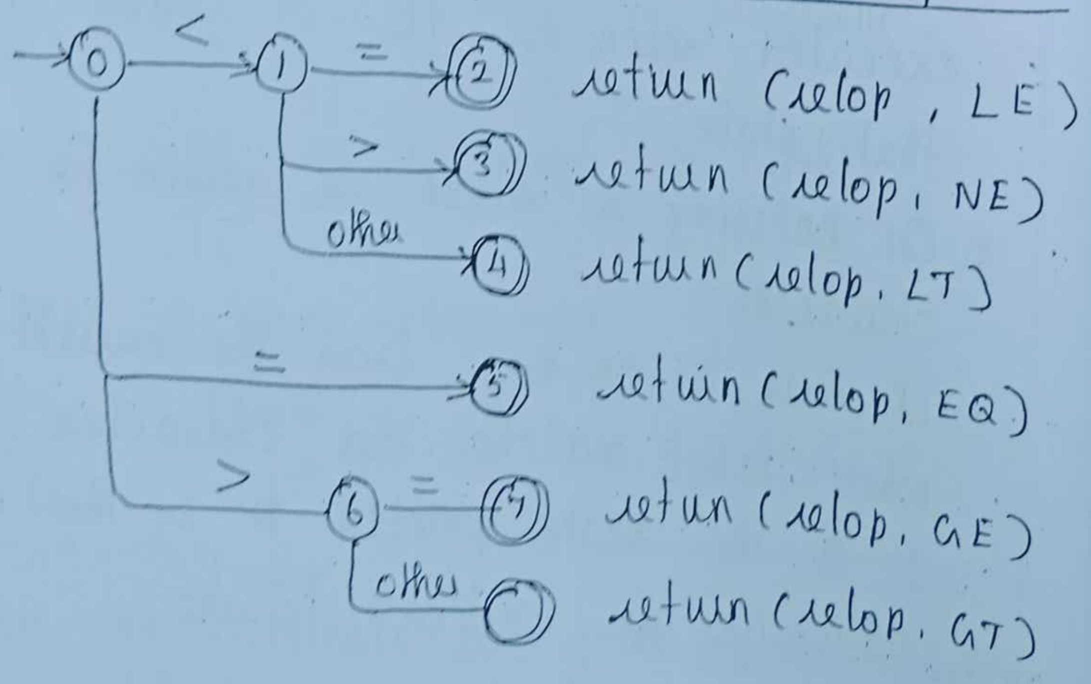
- As an intermediate step in the construction of a lexical analyzer, we first produce a stylized flowchart called a transition diagram
- This transitions diagram are deterministic.
- One state is labeled as the start state; it is the initial state of the transition diagram where control resides when we begin to recognize a token.
- Certain states may have actions that are executed when the flow-of control reaches that state.
- On entering a state we reach the next input character.
- If there is an edge from the current state whose label matches this character, we then go to the state pointed to by the edge.
- Otherwise we indicate failure.

Transition diagram for >=

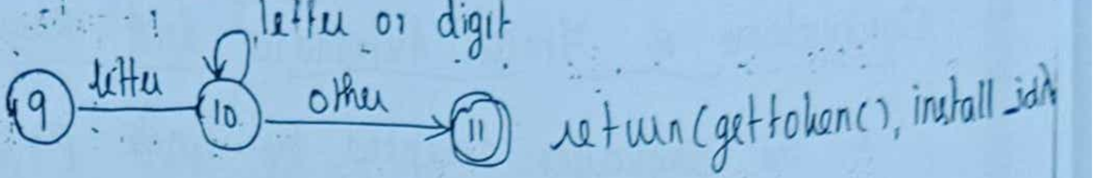


- Its start state is 0. In state 0, we read the next input character. The edge labeled > from state 0 is to be followed to state 6. if this input character is >.
- Otherwise we have failed to recognize either > or >=.

A transition diagram for relational operator:



Transition diagram for identifiers and keywords



- Since keywords are sequence of letters, they are exceptions to the rule that a sequence of letters and digits starting with a letter is an identifier.

- When the accepting state is reached, we execute some code to determine if the lexeme leading to the accepting state is a keyword or an identifier.

- The return statement next to the accepting state uses

  gettoken() → to optain the token.

  install-id() → to obtain the attribute value to be returned.

- The symbol table is examined and if the lexeme is found there marked as a keyword, install-id() returns 0.

- If the lexeme is found and is a program variable, install-id() returns a pointer to the symbol table entry.

- If the lexeme is not found in the symbol table it is installed as a variable and a pointer to the newly created entry is returned.