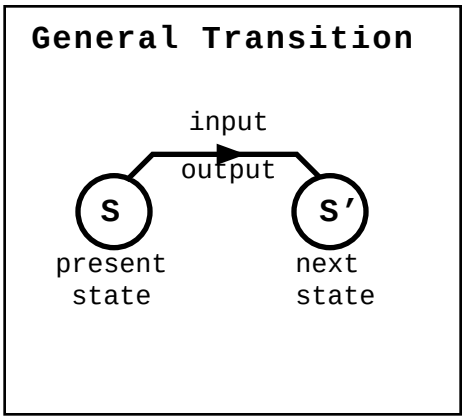
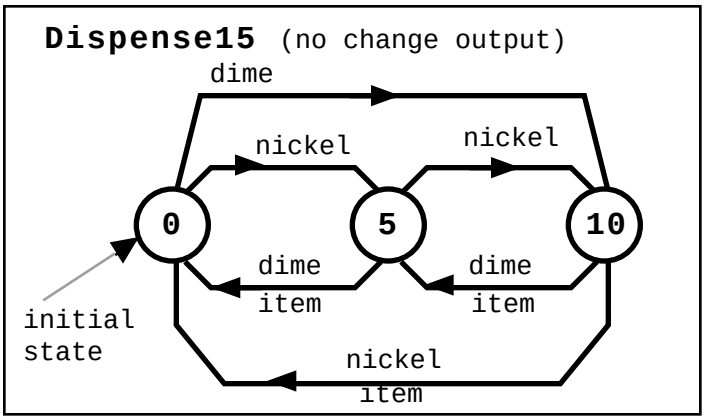
SOP: State-Oriented Programming

State-oriented programming is a programming paradigm which emphasizes the state (data values) and the transitions (or changes) between states. It is most useful when inputs come in a sequence, such as a string of characters, or a stream of coins.

State diagrams (sometimes called finite state machines, sequential circuits, or regular automata) are useful to describe such systems. State diagrams consist of circles or rounded boxes representing states and arrows indicating the transitions (changes) between states, as shown in the figure. Every transition arrow from one state S to the next state S', must have an input, and often has a corresponding output. The input and output are separated; sometimes by a slash, or other times the input is above the arrow and the output is below it.





Dispense15 of the given figure shows the behavior of an algorithm which accepts sequences of nickels (5 cent pieces) and dimes (10 cent pieces), and outputs an item when the accumulated sum reaches 15 cents. The states in this case represent the amounts of money (0, 5, and 10 cents) accumulated at any instant of time. The initial state, shown by a dotted arrow, is state 0. Then when a nickel is input, the state changes to 5 (without putting out an item). When the state is 10, and a nickel is input then an item is output and the next state is 0.

Transitions, or state changes, caused by input sequences of coins can be shown by a trace as given below. Notice that there is no change output by this machine, so that when the state is 10 and a dime is input, then an item is output and the next state is 5 (crediting the extra 5 cents to the next transaction). That is not very friendly! An alternate dispensing machine, which outputs change, is shown next.