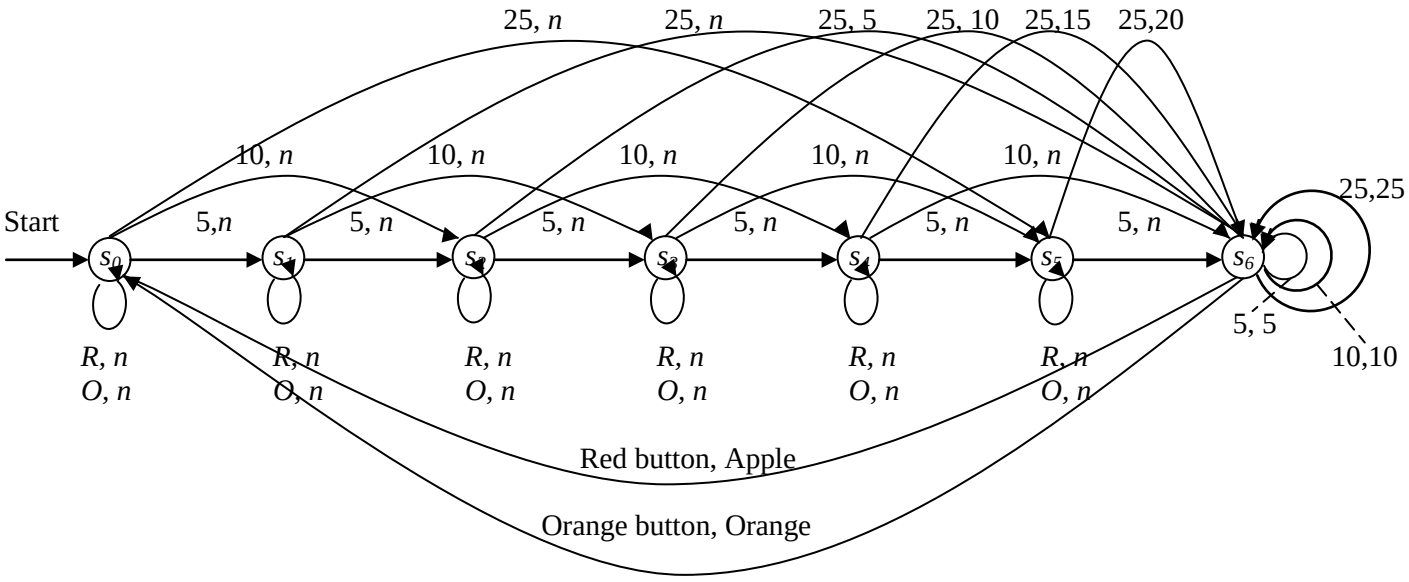
1. Introduction
   1.1. Finite State Machines (FSM) are similar to Finite State Automata (FSA) except an FSM prints out output using an output alphabet
2. Finite State Machines
   2.1. A "finite state machine" *M* consists of six parts. The first four are identical to a finite state automata's. The accepting states of FSAs are replaced with an output function and output symbols.
      2.1.1. A finite set Σ of input symbols
      2.1.2. A finite set *S* of "internal" states.
      2.1.3. An initial state s_0, with $s_0 \in S$.
      2.1.4. A next-state function, *f* from $S \times \Sigma$ into *S*.
      2.1.5. A finite set *Z* of output symbols.
      2.1.6. An output function *g* from $S \times \Sigma$ into *Z*.
      2.1.7. Providing all six parts defines an FSM.
   2.2. State Table and State Diagram of a Finite State Machine
      2.2.1. There are two ways of representing an FSM: a state diagram and a state table. To demonstrate both we model a vending machine that accepts nickels, dimes, and quarters. To keep things simple, we will assume that an orange or an apple costs 30 cents. When at least 30 cents has been deposited the machine returns the excess, and then the customer can push an orange button to receive an orange, or a red button to receive an apple.
      2.2.2. A state diagram is a labeled directed graph that represents the states of the FSM and both the next-state and output functions. The labels are pairs of a next-state input, and an output function output.



      2.2.3. A state table combines the next-state function and the output function into a single table. Here, the *n* in the output function entries indicates no output.

	Next State Function					**Output Function**				
	Input					**Input**				
State	**5**	**10**	**25**	**Orange button**	**Red button**	**5**	**10**	**25**	**Orange button**	**Red button**
s_0	s_1	s_2	s_5	s_0	s_0	*n*	*n*	*n*	*n*	*n*
s_1	s_2	s_3	s_6	s_1	s_1	*n*	*n*	*n*	*n*	*n*
s_2	s_3	s_4	s_6	s_2	s_2	*n*	*n*	5	*n*	*n*
s_3	s_4	s_5	s_6	s_3	s_3	*n*	*n*	10	*n*	*n*
s_4	s_5	s_6	s_6	s_4	s_4	*n*	*n*	15	*n*	*n*
s_5	s_6	s_6	s_6	s_5	s_5	*n*	5	20	*n*	*n*
s_6	s_6	s_6	s_6	s_0	s_0	5	10	25	Orange	Apple

2.3. Input and Output Tapes

2.3.1. A sequence of input symbols of an FSM can be thought of as an input tape, and the output from such a sequence can be thought of as an output tape.

2.3.2. In general, given a input sequence, $a_1, a_2, a_3, \ldots, a_n$, then the series of states would be $s_i = f(s_{i-1}, a_i)$, and the output sequence would be $z_i = g(s_{i-1}, a_i)$

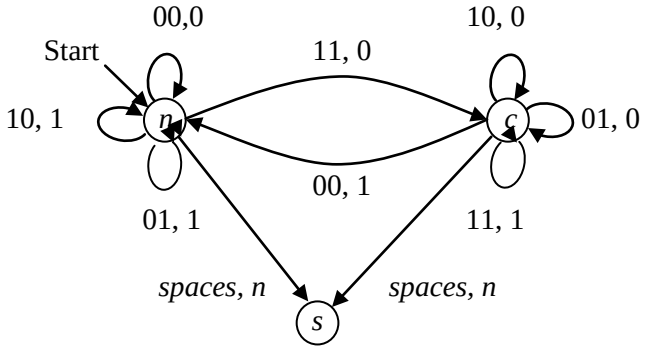
2.3.3. Given the following state table, what would the output tape be if the input tape was 0110001001110?

	f		***g***	
	Input		**Input**	
State	**0**	**1**	**0**	**1**
s_0	s_1	s_0	1	0
s_1	s_3	s_0	1	1
s_2	s_1	s_2	0	1
s_3	s_2	s_1	0	0

2.4. Binary Addition

2.4.1. A FSM can be designed to add two binary numbers as long as they are the same length. We can overcome difference in lengths by adding 0's to the front of the shorter number.

2.4.2. There are three states: *n* = no carry from previous addition, *c* = carry from previous addition, and *s* = stop state which is entered when a space character is encountered. The *n* state is the initial state. There are two input symbols for each input, one from each number.



3. Godel Numbers (skipped)

4. Turing Machines

4.1. A "Turing machine" is a device that manipulates symbols on a strip of tape according to a table of rules. Despite its simplicity, a Turing machine can be adapted to simulate the logic of any computer algorithm, and is particularly useful in explaining the functions of a CPU inside a computer.

4.2. Basic Definitions

4.2.1. A Turing machine involves three disjoint sets:

4.2.1.1. A finite "tape" set $A = \{a_1, a_2, \ldots, a_m\} \cup \{B\}$, where "B" is the blank symbol.

4.2.1.2. A finite "state" set $S = \{s_1, s_2, \ldots, s_n\} \cup \{s_0\} \cup \{s_H, s_Y, s_N\}$, where s_0 is the initial state, s_H is the halting state, s_Y is the accepting state, and s_N is the non-accepting state.

4.2.1.3. A "direction" set $d = \{L, R\}$, where L = left, and R = right.

4.2.2. An "expression" is a finite (possibly empty) sequence of elements from $A \cup S \cup d$.

4.2.3. A "tape expression" is an expression using only elements from the tape set *A*.

4.2.4. A Turing machine is able to do the following three things simultaneously:

4.2.4.1. Erase the scanned symbol a_k and writes in its place a tape symbol a_p, where we permit $a_k = a_p$.

4.2.4.2. Change its internal state s_i to s_j, where we permit $s_i = s_j$.

4.2.4.3. Move one square to the left or one square to the right.

4.2.5. A quintuple *q* is a five-letter expression of the form: $q = (s_i, a_k, a_p, s_j, \begin{Bmatrix} L \\ R \end{Bmatrix})$.

4.2.6. A "Turing Machine," is a finite set of quintuples such that:

4.2.6.1. No two quintuples begin with the same first two letters. This guarantees that the machine cannot do more than one thing at any given step.

4.2.6.2. No quintuple begins with s_H, s_Y, or s_N. This guarantees that the machine halts in state s_H, s_Y, or s_N.

4.2.7. A “picture” P is an expression $P = Ts_ia_kV$ where T and V are tape expressions (possibly empty), and s_i is a state, and a_k is a tape symbol.

4.2.8. Let P_1 and P_2 be pictures. We write $P_1 \rightarrow P_2$ if one of the following holds, were a, b, c are tape letters and T and V are tape expressions (possibly empty).

4.2.8.1. $P_1 = Ts_iacV$, $P_2 = Tbs_jcV$, and M contains the quintuple $q = s_iabs_jR$.

4.2.8.2. $P_1 = Tcs_iaV$, $P_2 = Ts_jcbV$, and M contains the quintuple $q = s_iabs_jL$.

4.2.8.3. $P_1 = Ts_ia$, $P_2 = Tbs_jB$, and M contains the quintuple $q = s_iabs_jR$.

4.2.8.4. $P_1 = s_iaV$, $P_2 = s_jBbV$, and M contains the quintuple $q = s_iabs_jL$.

4.3. Computing with a Turing Machine (skipped)

4.4. Turing Machines with Input (skipped)

4.5. Grammars and Turing Machines (skipped)

5. Computable Functions (skipped)