COMP2048

Mid-Term exam

Student number: 45004830

Name: Bosheng Zhang

1. Consider a finite alphabet {a, b} then set of all finite strings A* = {ε, a, b, aa, ab, ba, bb, ...}
   (Note that A∗ is an infinite set, but it only includes finite strings)
   A FSM can represent a set of states as a string, an alphabet as a string, the transition function as a string (finite table), the initial state and set of accept states can also be encoded as a string. By picking all strings from A* and specify the FSA (bijective mapping between natural numbers and FSM) repeating the following steps:
   1. begin with i=1, n=1
   2. generate the next binary string Si in lexicographic order
   3.1. if Si is a valid encoding of an FSM then output Si as the n-th FSM and set n=n+1;
   3.2. if Si is not a valid encoding of an FSM then ignore it
   4. set i=i+1 and go to step 2

   by this way each natural number (n=1,2,3,4,...) corresponds to a FSA, and each FSA has a corresponding n, since we scan all possible binary strings. So, the number of FSA is countable.

   A language L is a subset (possible infinite) of A*: L⊆A*. So the set of all languages is exactly the power set of A*: 2^{A^*}={{ϵ}, {a},{b},{a,b},...} But the power set of a countably infinite set is uncountable. Prove by contradiction: suppose that the languages are countable, then we can place entire languages in a table in which each (infinite) row i include the elements of the language Li and the columns include the string over the alphabet {a,b} ((i,j)=1 if and only if string j is in Li):

	a	b	aa	ab	ba	bb	⋯
L1	[0]	1	0	1	0	1	⋯
L2	1	[0]	1	0	1	0	⋯
L3	0	0	[0]	0	0	0	⋯
⋯	⋯						⋯

Now select the diagonal bits (i , i) and flip them to produce a new language:

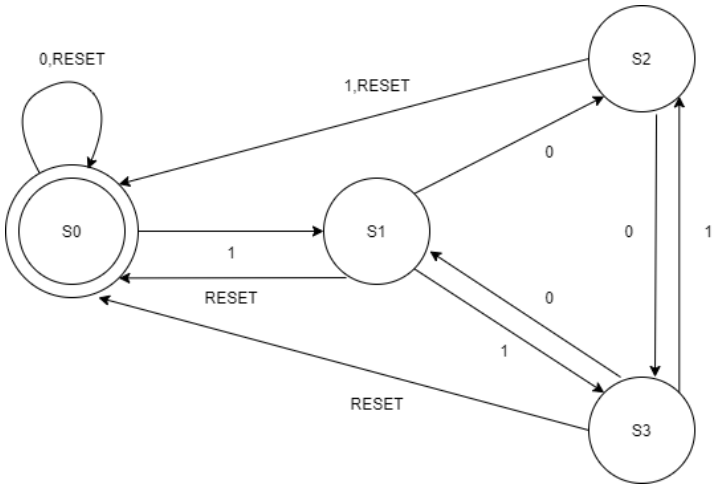
	a	b	aa	ab	ba	bb	⋯
L_{new}	[1]	1	0	1	0	1	⋯

The new language L_{new} = {a, b, ab, bb} build from inverting diagonal bits is distinct from every language Li: a ∉ L1, b ∉ L2, ab ∉ L3, … This implies that the above table is not an enumeration of all languages, a contradiction, so no such table exists. Therefore, the set of all possible languages is uncountably infinite.

2. The FSM is aiming to decide whether the string accept or not by comparing certain character (e.g. begin with 1 and end with 1)
   The FSM accepts strings which (begin with 1 and end with 1) or (end with 0 or RESET).
   strings which (begin with 1 and end with 1) e.g. 100011
   strings which (end with 0) e.g. 0
   strings which (end with RESET) e.g. 10001101RESET
   Transition table:

	S0	S1	S2	S3	S4
RESET	S0	S0	S0	S0	S0
0	S0	S2	S4	S1	S3
1	S1	S3	S0	S2	S4

   A close guess alternative FSM accepts strings which (begin with 1 and end with 1) or (end with 0 or RESET):

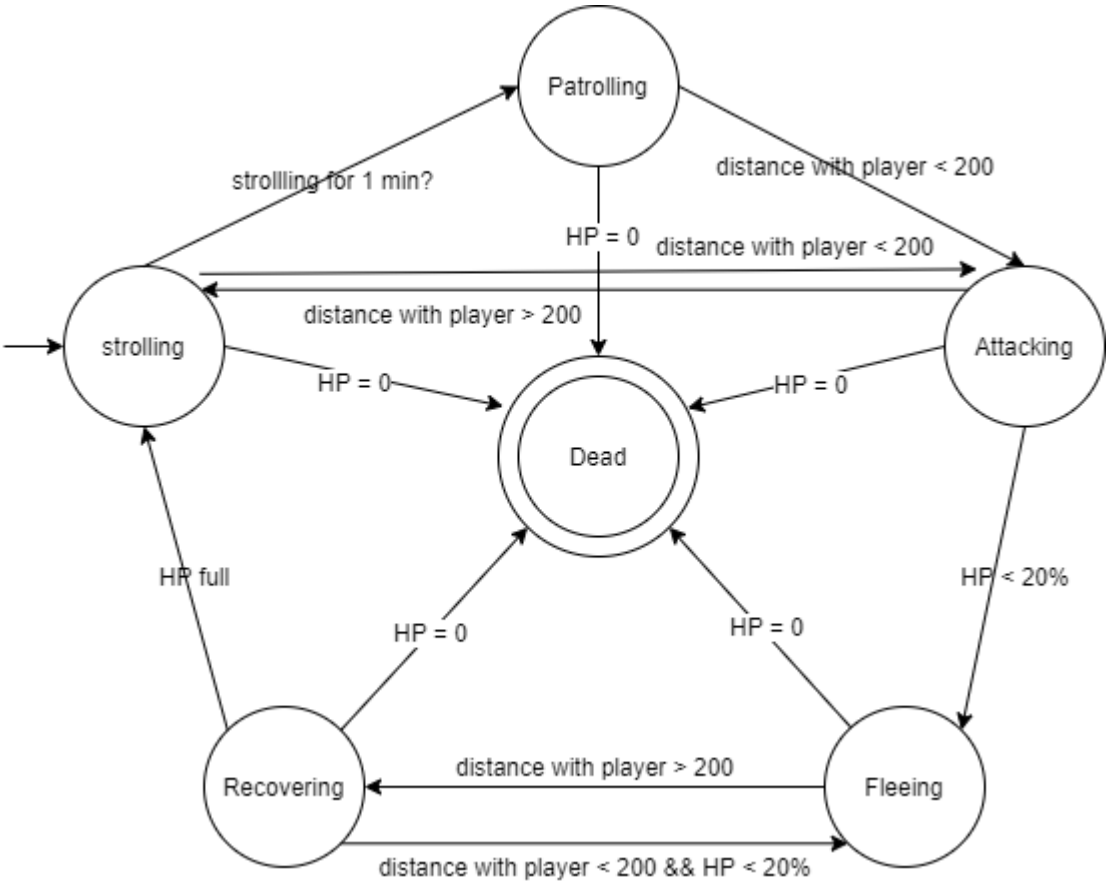


   But not with same functionality. e.g. the five state FSM can accept string 100011, but the above FSM do not accept.
   I believe it is impossible to make that equivalent FSM with less states, because there is no equivalent states to reduce, and the two loops (four one and four zero) can not be achieve by using four states.

3. A finite state machine can be determined as a 5 tuple $(Q,\{A\}_{input},\delta,q0,q_{accpet})$, basically how FSM works can be said as this reads the input string from the left to right in a one pass; compare to TM, the read/write head permits it to back along the input and mark on the tape. Unlike TM, FSM has one and only one choice of state to move to during reading the string. The language that a finite state machine may accept are languages generated by regular grammar. Conversely, by writing and erasing symbols on the tape, a TM can carry out the productions of an unrestricted grammar, hence making its total configurations arbitrary large.
   A Turing machine is a 7 tuple $T = (Q,\{A\}_{input},\{A\}_{Tape},\delta,q0,q_{accept},q_{reject})$. For theoretical reasons, we may consider the tape to be infinite in length, but in FSMs the input and output is always finite. TM also has an initial state q0 like an FSM, but only has single accept and reject states, q_{accept} and q_{reject} respectively, the immediate effect of q_{accept} and q_{reject} allow us to complete the computation instantly without having to process the entire input string. An example of such a computation that halts immediately is T that checks the string includes at least a single 0, once a 0 is met, regardless the size of the input, further computation is not required. In contrast, FSM must process the whole string to decide whether accept the string or not.
   Modern-day computers don't have unlimited storage, but the Turing machine does. Thus, technically, modern computers are closer to finite state machines, but the current state of an 8 GB ram machine will be 2^3936*2^2^36, which is technically huge, but again the I/O is not specified in the Turing machine. Input is whatever is inserted in the tape and output is whatever is written on the tape when the machine halts or stops, in by contrast modern-day computer accept I/O all the while during execution.

4. FSM AI system have six states: Strolling, Patrolling, Attacking, Fleeing, Recovering and dead. We will implicitly assume that the enemy has some HP statistic.
   A player might start in Strolling State and perform strolling action in the game.
   By allowing the player to change states conditionally, the NPC is based on the following rules:
   - If strolling for one minute already, switch to patrol;
   - If the distance between another player is less than 200m, to attack the player;
   - If the distance between another player is greater than 200m, to stroll;
   - If the player's HP is lower than 20%, to flee and if the distance between another player is greater than 200m, to recover;
   - If the player recovers to full HP, go stroll;
   - If the player's HP reaches zero, it dies.

   Transition diagram as below:



   This captures the essence of the player's decision based on their situation. If the condition next to the arrow is true, then each arrow shows the transition between states.
   This system through the lens of sense / thinking / acting. The senses are embodied in the data used by the transition logic. This thinking is reflected in the transition of each state. Acting is accomplished through actions that are periodically taken on transitions within or between states.
   The main limitation of this system is its poor scalability. As the number of states and transitions increases, so does the complexity of the document. Simply put, it is difficult to arrange large state machine diagrams to make them readable and understandable.
   The capability that AI lacks is that it cannot walk and recover at the same time; NPC can only be in one state at a time.

Unlike existing systems, by using a Turing machine, we can control state using a stack. The top of the stack contains the active state. Transitions are handled by pushing or popping states from the stack. It can pop itself off the stack and push into another state, which means a full transition (as the FSM does). It can pop itself from the stack, which means that the current state is complete and the next state in the stack should be active. Also, it can push a new state, which means that the current active state will change for a while, but when it pops off the stack, the previous active state will take over again. Turing machine knows what state it is in, but also the memory of what state it was in, it helps the AI player to go back to the previous state.