14주차 예비보고서

전공: 신문방송학과 학년: 4학년 학번: 20191150 이름: 전현길

1. FSM에 대해서 설명하시오. (예시 포함)

FSM(finite state machine; 유한상태기계)는 **유한한 상태(state)와, 이들 사이의 전환(transition)**을 다루는 수학적 모델이다. finite-state machine은 유한한 상태의 목록과 초기 상태, 각 전환을 일으키는 입력으로 정의된다.

이 때 FSM은 결정론적 FSM(deterministic FSM)과 비결정론적 FSM(non-deterministic FSM)으로 나뉜다. 이 때 결정론적 FSM은 각 전환이 이전 상태와 입력값 심볼을 통해 고유하게 결정되며, 상태 전환을 위해 입력이 필요한 FSM이다. 좀 더 간단히 설명하면, **현재 상태와 입력에 따른 다음 상태가 유일하고, 상태의 변화를 위해 입력이 반드시 필요**한 FSM이다. 비결정론적 FSM은 이러한 제약을 따르지 않는 FSM을 의미하므로, 넓은 의미로 모든 비결정론적 FSM은 결정론적 FSM에 포함된다. 단, 상황에 따라서 비결정론적 FSM과 결정론적 FSM이 아닌 모든 FSM을 가리키기도 한다.

FSM을 설계할 때는 한 번에 하나만의 상태를 가지고, 각 상태가 명확하게 정의되도록 설계해야 한다는 것이다. 상태가 중첩되거나 상태별로 어떤 동작을 하는지 명확하게 정의되지 않을 시 의도하지 않은 동작을 할 여지가 있다.

용어의 추상적인 정의에서 미루어 알 수 있듯이, FSM은 회로설계에 한정되어 사용되는 용어는 아니다. 소프트웨어의 디자인 패턴을 설명하는 데에 FSM이란 용어가 사용되기도 하는데, 이는 단순히 if, switch문 등으로 조건문을 작성해 동작을 정의하는 대신, 객체가 가질 수 있는 상태를 미리 정의해 두고 상태에 따라 특정한 동작을 수행하도록 구현하는 방식이다.

디자인 패턴으로서의 FSM은 일반적으로 객체지향 프로그래밍의 다형성을 활용해 구현된다. 구체적으로는 추상 클래스로 상태를 만들고, 자식 클래스 생성 시 상태를 상속받도록 해 상태에 따른 인터페이스를 다르게 정의하는 방식으로 구현된다. 아래는 FSM design pattern을 따라 객체를 구현한 Unity C# 코드의 예시이다.

Character FSM by Unity

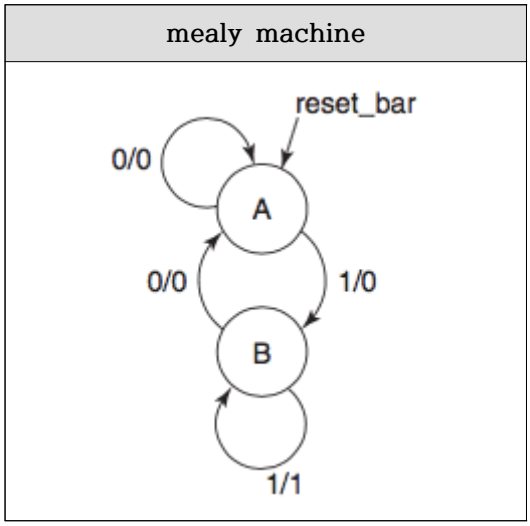
```
using UnityEngine;

public class CharacterFSM : MonoBehaviour
{
    private enum State { Idle, Move, Attack }
    private State currentState;
    private void Start() { currentState = State.Idle; }

    private void Update() {
        switch (currentState) {
            case State.Idle:
                if (Input.GetKeyDown(KeyCode.Space))
                    ChangeState(State.Attack);
                else if (Input.GetAxisRaw("Horizontal") != 0 || Input.GetAxisRaw("Vertical") != 0)
                    ChangeState(State.Move);
                break;
            case State.Move:
                MoveCharacter();
                if (Input.GetKeyUp(KeyCode.LeftArrow) || Input.GetKeyUp(KeyCode.RightArrow) ||
                    Input.GetKeyUp(KeyCode.UpArrow) || Input.GetKeyUp(KeyCode.DownArrow))
                    ChangeState(State.Idle);
                if (Input.GetKeyDown(KeyCode.Space))
                    ChangeState(State.Attack);
                break;
            case State.Attack:
                Attack();
                ChangeState(State.Idle);
                break;
        }
    }

    private void ChangeState(State newState) { currentState = newState; }
    private void MoveCharacter() { // 캐릭터 이동 로직 }
    private void Attack() { // 공격 로직 }
}
```

2. Mealy Machine에 대해 조사하시오.



mealy machine은 finite state machine의 한 종류로, 현재 상태에 더해 추가적인 입력에 따라 출력이 결정되는 FSM을 의미한다.

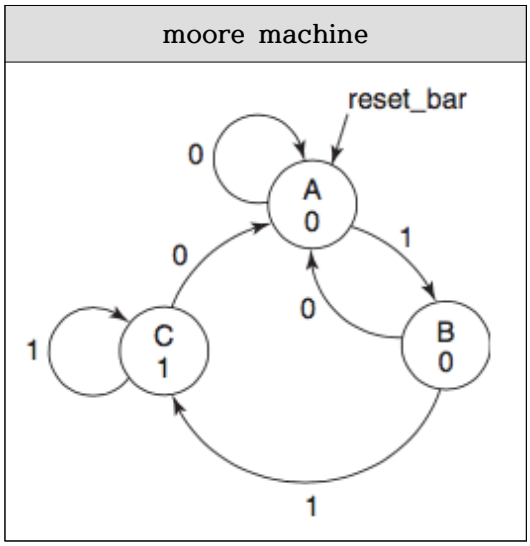
전형적인 state diagram에서 mealy machine을 표현하는 diagram에서 동그라미 안의 글자들은 현재 상태를 의미한다. 또 화살표에 할당되어 있는 글자들은 **해당 상태 변화를 일으키는 입력과, 현재 상태 + 해당 입력이 들어왔을 때의 출력**을 의미한다.

$$(S, S_0, \Sigma, \Lambda, T, G)$$

보다 학술적인 정의로는 위의 6-tuple을 통해 정의된다. S는 finite set of states, S_0은 initial state, Σ는 finite set of input alphapet, Λ는 finite set of output alphapet을 의미한다. 또, T는 transition function(전이 함수), G는 output function(출력 함수)를 의미한다.

이 때, 전이 함수는 현재 상태와 입력 알파벳에 따라 다음 상태로 이동하는 상태 간선 쌍을 의미하며, 출력 함수는 현재 상태와 입력 알파벳에 따른 출력값을 결정하는 순서쌍을 의미한다. s_i가 임의의 현재 상태, Σ_j 가 현재 입력값, s_{ij}가 대응되는 다음 상태라고 가정하면 전이 함수는 $((s_i, \Sigma_j), s_{ij})$ 간선 순서쌍으로 표현할 수 있고, 출력 함수는 o_j를 대응되는 출력값이라고 가정하면 $((s_i, \Sigma_j), o_{ij})$ 순서쌍으로 표현할 수 있다.

3. Moore machine 에 대해 조사하시오.



moore machine은 finite state machine의 한 종류로, 현재 상태만으로 출력이 결정되는 FSM을 의미한다.

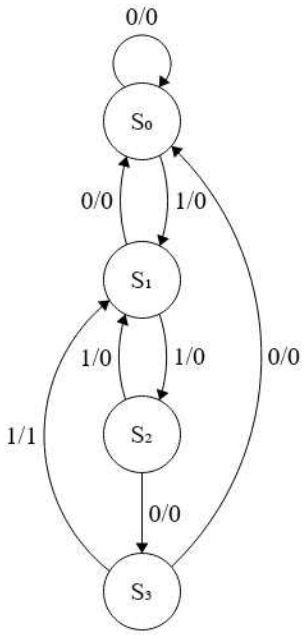
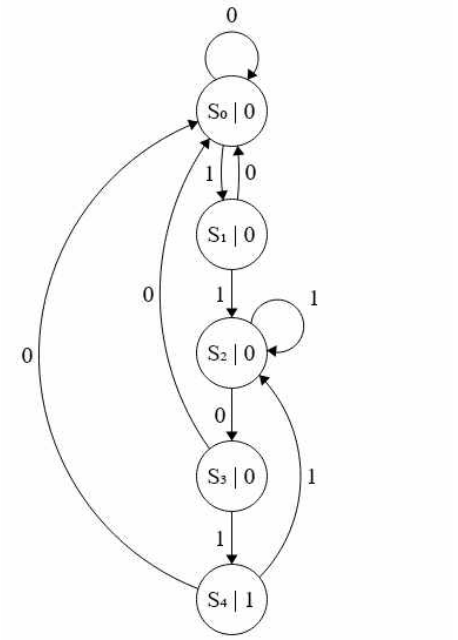
전형적인 state diagram에서 moore machine의 **동그라미 안의 글자들은 각각의 상태(=A, B, C)와 현재 상태에서의 출력(=0, 1)**을 의미한다. 또 **화살표에 할당되어 있는 글자들은 해당 상태 변화를 일으키는 입력**을 의미한다. 해당 moore machine은 입력이 1bit고, 상태가 3개 존재하므로 모든 입력에 대응되기 위해서 반드시 6개의 화살표가 존재해야 한다. (상태의 수(k) * 입력의 수(2^n) = 상태 변화의 수)

$$(S, s_0, \Sigma, O, \delta, G)$$

보다 학술적인 정의로 역시 6-tuple을 통해 정의한다. mealy machine에서 나온 것 외에 O, δ가 추가되었고, G가 mealy machine에서의 정의와 달라진 것을 제외하고 나머지는 mealy machine의 정의와 같다.

O는 mealy machine의 Λ의 의미와 정확히 동일하게, finite set of output alphapet을 뜻한다. δ는 전이 함수를 의미하며, mealy machine에서와 같이 $((s_i, \sum_j), s_{ij})$ 간선 순서쌍을 원소로 가진다. 출력 함수 G가 달라지는데, (s_i, o_i)를 원소로 갖는다. 이는 moore machine에서 출력값은 상태에 의해 유일하게 결정되기 때문이다.

4. sequence detector에 대해서 조사하시오.

1101 sequence detector mealy machine	1101 sequence detector moore machine
	

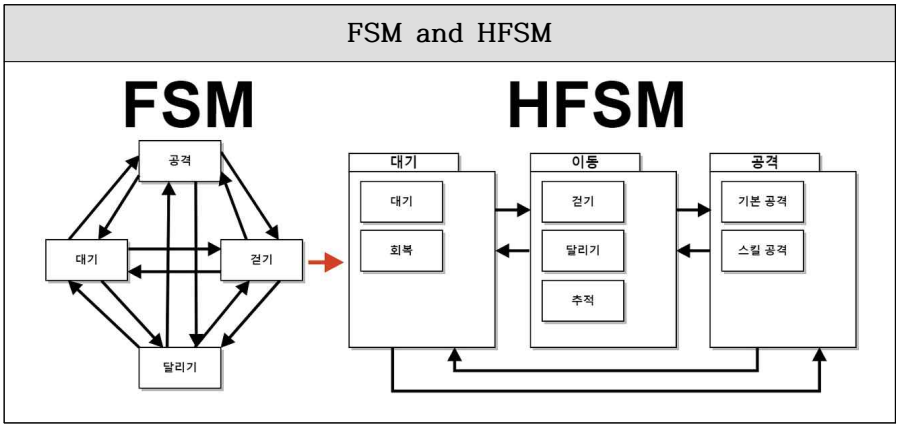
sequence detector는 연속적으로 입력되는 bit sequence에서 특정한 pattern의 bit sequence를 검출하는 순차 회로이다. moore machine, mealy machine으로 위와 같은 state diagram을 가지며, 이를 바탕으로 회로를 구현할 수 있다.

겹치는 것을 허용할지, 허용하지 않을지에 따라 구현 방식이 달라지는데, 회로를 구현하기 전 어떤 방식으로 구현할지 선택해야 한다. 예를 들면 문자열 1101$\overline{1101}$이 입력되었을 때 앞의 1101과 뒤의 1101에 중복되는 부분이 있는데, 중복되는 부분을 허용할 경우 2개의 1101 bit sequence를 검출하게 되고, 그렇지 않을 경우 1개의 bit sequence만 검출하게 된다. **위의 구현은 중복을 허용하는 방식으로 구현**한 1101 sequence detector이다.

moore machine, mealy machine diagram을 그리기 위해 https://madebyevan.com/fsm/을 사용했다.

(수정) mealy machine의 S_2에서 S_1으로 가는 화살표를 자폐선으로 바꾸어야 한다.

5. 기타 이론



FSM design pattern으로 객체를 구현할 경우 조건 분기를 통해 동작을 구현하는 것보다 가독성이 좋고 유지보수가 쉬우며, 간단한 코드를 작성할 수 있다. 단 FSM 역시 상태가 많아지면서 상태 전환의 가짓수가 많아질 경우 복잡해지게 되는데, 이를 해결하기 위해 HFSM(Hierachical FSM)이 도입되었다.

HFSM이란 다양한 상태를 병렬적으로 관리하는 대신, 각 상태를 추상화시킨 상위 상태를 구현해 상태를 계층적으로 관리하는 방식이다. 예를 들어 게임 캐릭터가 공격, 대기, 이동 상태를 갖는다면 공격 상태의 세부 상태로는 근거리 공격, 원거리 공격 등이 있을 수 있다. 이처럼 HFSM은 상위 상태에 따라 세부 상태를 추가로 구현하여 상태 전이를 계층적으로 관리한다.