CODIFICAÇÃO DE MÁQUINA DE ESTADOS FINITOS

A maior parte desse material foi tirada do artigo "*Using finite state machines to design software*", disponível no site https://www.embedded.com/design/prototyping-and-development/4026990/Using-finite-state-machines-to-design-software-item-1

Desenvolvimento de Software

- Quando criando um software novo, muitos programadores preferem iniciar a codificação sem qualquer planejamento formal prévio
- Estão convencidos que não existe um método de projeto apropriado para seu projeto
- O argumento é que qualquer planejamento formal desperdiça muito tempo sem melhoria significativa da qualidade do produto final

Desenvolvimento de Software

- Engenharia de software e qualidade de software **são muitas vezes necessárias**
- Seriam bem-vindos métodos de projeto **simples de usar** e que ajude a criar **código melhor**, além de gerar a **documentação**

Máquina de estados finitos

- Uma **máquina de estados finitos** pode ser utilizada no planejamento e implementação de projetos de software
- O tipo de software que é melhor aplicada ao modelo MEF são aqueles que têm distintos "modos" ou os que possuem "controle intensivo", ou seja, aqueles que possuem uma estrutura lógica complexa

Definição Informal

- MEF pode ser informalmente definida como qualquer dispositivo (eletrônico, mecânico, ou só conceitual) que aceita um número finito de entradas e pode produzir um número finito de saídas

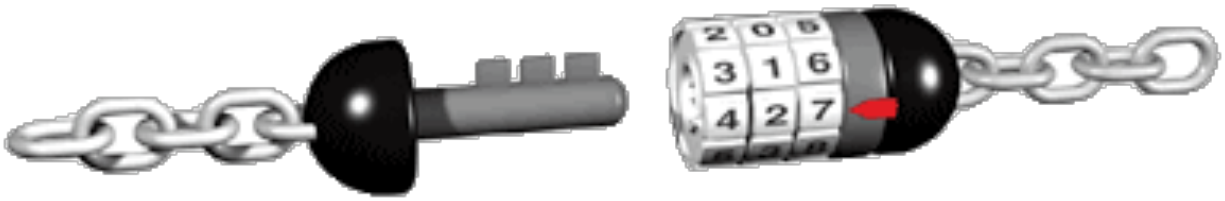
Definição Informal

- Um requisito chave é que o dispositivo deve ter alguma memória interna que possibilita lembrar de sequências de entradas de tal forma que as saídas não sejam somente dependentes da combinação de valores de entrada, mas também da ordem no qual eles foram aplicados

Vantagens de uso das MEF

- É aplicável a uma **grande variedade** de tipos diferentes de projetos de software
- Pode ser usado em **muitos estágios do desenvolvimento,** desde projeto de alto nível até codificação modular
- Provê uma **estratégia rigorosa,** mas que resulta em uma estrutura de **código simples e consistente**
- Cria uma **documentação** (útil principalmente na manutenção) que reflete a organização do código
- Incorpora uma **ferramenta de depuração** efetiva com mínimo esforço
- Facilita o **projeto cooperativo** e **revisão em pares**

Dispositivo combinacional



Abrirá quando os três dígitos forem colocados de acordo com a senha numérica. Não importa a ordem de colocação dos dígitos. Este dispositivo não precisa de memória. Consequentemente, **não é uma MEF**, mas somente um dispositivo combinacional.

Cadeado é uma MEF

O cadeado tem a habilidade de lembrar-se da ordem de entrada do números. É um bom exemplo de uma MEF, que são também chamadas de máquinas sequenciais.

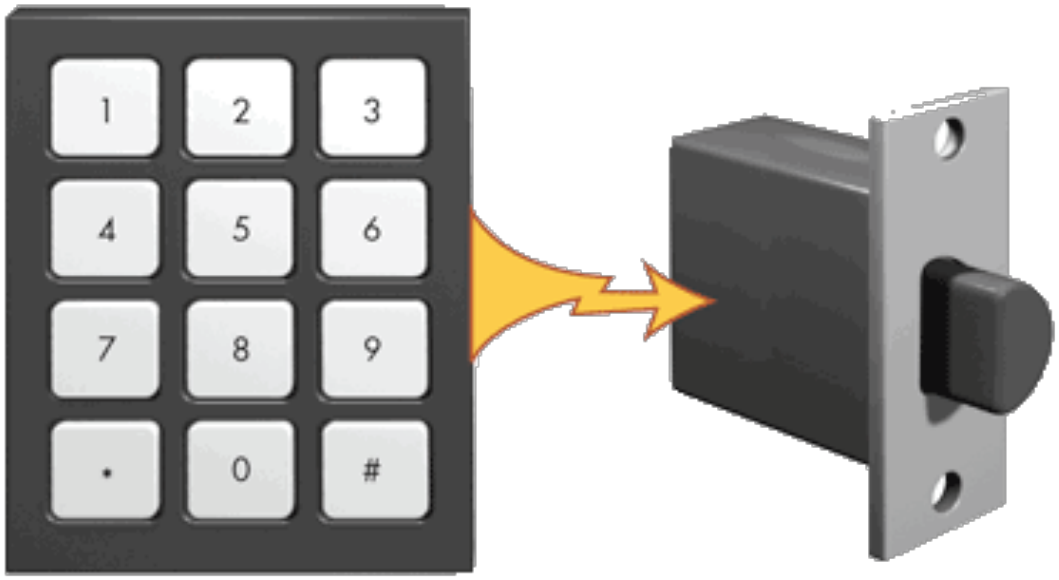
Características do cadeado

- Não é necessário saber a respeito do trabalho mecânico do cadeado
- Tem-se configurações internas (ou **estados**) geradas de certas entradas e que são também dependentes da história dos valores entrados
- Um estado pode ser pensado como o encapsulamento de uma sequência específica de entradas passadas.
- Duas ou mais sequências de entrada podem mapear para o mesmo estado

Telefone também é uma MEF

- Um telefone pode ser considerado uma MEF
- Quando o recebedor é escolhido para iniciar uma conversa, um tom de discagem é ouvido (o estado pode ser chamado "**tom de discagem**")
- Após uma sequência apropriada de dígitos serem entrados e o receptor atende a ligação, o estado é trocado para "falando"
- Neste ponto, entrando mais dígitos modificará a sequência de entrada mas o estado "falando" permanecerá o mesmo

Cadeado com teclado numérico



Substituindo o exemplo do cadeado por uma versão eletrônica, que controla uma trava através de um teclado numérico, onde o teclado é o dispositivo de entrada e a trava é o dispositivo de saida

Cadeado com teclado numérico

- A combinação para desbloquear é 4-9-1 e que inicialmente o dispositivo esteja bloqueado. O dispositivo está no estado "**inicial**"
- O estado inicial implica que o dispositivo espera o primeiro dígito correto. Após pressionar o "4", o dispositivo reconhece o inicio de uma sequência correta e reage criando um novo estado "**aceitou primeiro dígito**" (trava continua bloqueada)
- Se o "9" for pressionado, o novo estado pode ser chamado "**aceitou primeiro e segundo dígitos**" (trava continua bloqueada)
- Quando pressionar o "1", a **fechadura** emitirá um sinal para a trava desbloquear e o sistema passa a estar no estado inicial novamente

Cadeado com teclado numérico

- Um ponto crucial neste exemplo é que os **estados** são **noções abstratas** que permite um projeto conceitual
- Dessa forma, o projeto **pode ser totalmente desvinculado** da implementação real da fechadura

Tabela de Transição de Estados

	STATE		
INPUT	A: Initial	B: Got 1st no.	C: Got 1st & 2nd no.
4	B, Lock	A	A
9	A, Lock	C, Lock	A
1	A, Lock	A	A, Unlock
Default	A, Lock	A	A

Tabela de Transição de Estados

	STATE		
INPUT	A: Initial	B: Got 1st no.	C: Got 1st & 2nd no.
4	B, Lock	A	A
9	A, Lock	C, Lock	A
1	A, Lock	A	A, Unlock
Default	A, Lock	A	A

* após destravar sempre volta para o estado inicial

Para uma dada combinação de entrada e estado atual, cada célula mostra o próximo estado e uma lista de ações ou saídas
Em projetos mais complexos, cada célula pode listar mais de uma ação

Tabela de Transição de Estados

- **Por que uma tabela** e não um diagrama de estados?
- Qualquer gráfico contendo muitos estados e transições podem rapidamente tornar-se **complexo** e **confuso**
- Não existe um padrão no desenho de um diagrama

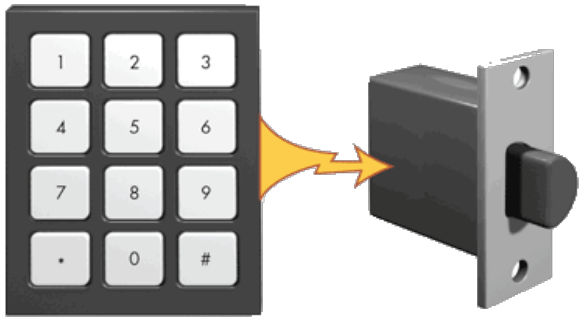
Tabela de Transição de Estados

- Gráficos **são mais difíceis de editar** e manter do que tabelas
- Diagramas de estado podem **não ser tão fácil de atualizar** com as alterações dos requisitos
- Pode até **cair em desuso**, tanto na documentação quanto como ferramenta de manutenção
- A conexão entre o **diagrama de estado e o código correspondente** não é sempre óbvia

Codificação automática

A partir de uma Máquina de Estados Finitos

```
Set_bolt (LOCK)                          // initialize device to known output . . .
  State = A                              // . . . and state
  LOOP (Forever) {
    Read (Input)                         // wait until some key is pressed
    IF(Debug) write Input, State         // used to create a debug trace
    SWITCH (State) {
      Case A:
        SWITCH (Input){
            Case 4:
                   State = B
                   Set_bolt(LOCK)        // needed if previously unlocked
            Default:
                   State = A
                   Set_bolt(LOCK)        // needed if previously unlocked
        }   // end SWITCH (Input)
      Case B:
        SWITCH (Input){
            Case 9:
                   State = C
            Default:
                   State = A
        }   // end SWITCH (Input)
      Case C:
        SWITCH (Input){
            Case 1:
                   State = A
                   Set_bolt(UNLOCK)      // SUCCESS!
            Default:
                   State = A
        }   // end SWITCH (Input)
    }   // end outer SWITCH (State)
  }   // end LOOP (Forever)
```



```
Set_bolt (LOCK)                    // initialize deadbolt to known output . . .
State = A                          // . . . and state
LOOP (Forever) {
  Read (Input)                     // wait until some key is pressed
  IF(Debug) write Input, State     // used to create a debug trace
  SWITCH (State .AND. Input) {
    Case A .AND. 4:
      State = B
      Set_bolt(LOCK)               // needed if previously unlocked
    Case B .AND. 9:
      State = C
    Case C .AND. 1:
      State = A
      Set_bolt(UNLOCK)             // SUCCESS!!
    Default:                       // new State & Input cases go above here
      State = A
      Set_bolt(LOCK)               // needed if previously unlocked
  }                                // end SWITCH (State .AND. Input)
}                                  // end LOOP (Forever)
```

Isto não é normalmente possível em C (ou outra linguagem de programação). Mas pode ser facilmente resolvida pela troca da estrutura **SWITCH** com uma série de instruções **IF** e **ELSE IF** tal que: **IF((State==A) && I(NPUT==4))** e assim por diante; ou uma função pode ser criada que retorne algum valor combinado de **State** e **Input**.

Debug

- A 5a. Linha do código fonte inclui a instrução **IF(Debug) write Input, State** que cria um *trace* de estados e entradas, tanto na tela quanto em arquivo, quando o flag "**Debug**" estiver setado
- Esta sequencia de estados e entrada revela o que acontece no código e deve confirmar a afirmação que o paradigma MEF pode incorporar ferramentas de depuração somente pela adição de uma simples linha de código

Debug

- No caso de aplicações de tempo-real, onde escrita em impressora ou arquivo pode gerar imprecisão temporal, pode-se armazenar o estado e variáveis de entrada em **um buffer de memória (RAM)** para visão posterior
- Eventualmente, vai requerer um pouco de código adicional

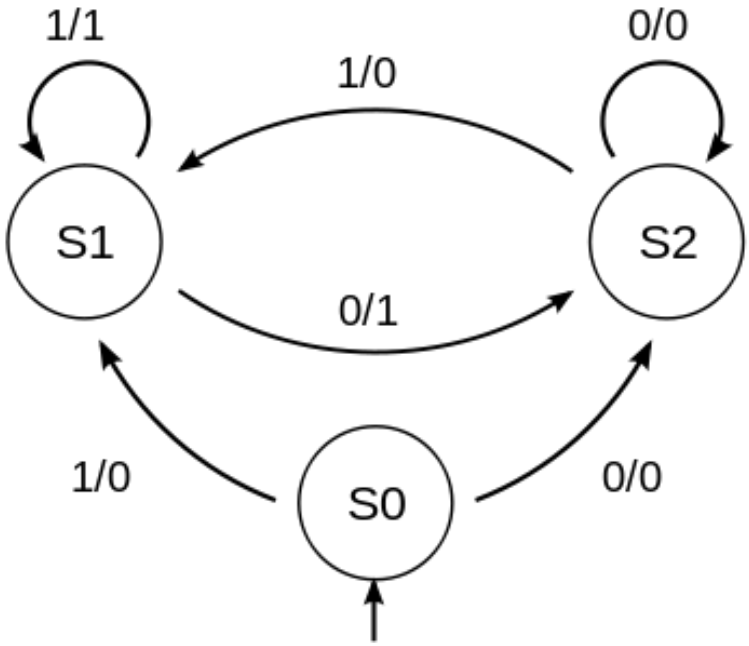
Tipos de Máquinas de Estados Finitos

Máquina de Mealy

Uma máquina de Mealy é um 6-tuplo $(S, S_0, \Sigma, \Lambda, T, G)$, que consistem em

- um conjunto finito de estados (S)
- um estado inicial S_0 que é um elemento de S
- um conjunto finito chamado o alfabeto de entrada (Σ),
- um conjunto finito chamado o alfabeto de saída (Λ)
- uma função de transição ($T : S \times \Sigma \rightarrow S$)
- uma função de saída de dados ($G : S \times \Sigma \rightarrow \Lambda$)

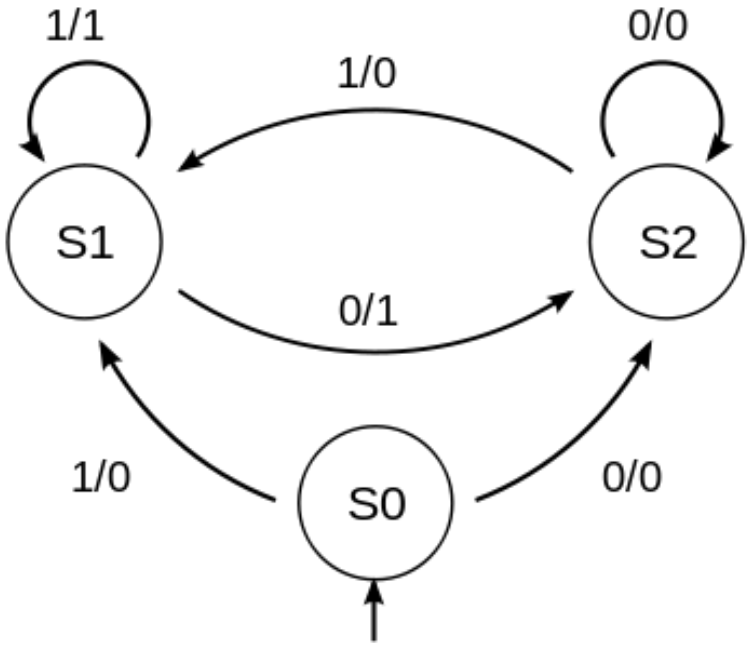
Máquina de Mealy



Uma máquina de **Mealy** é uma máquina de estado finito que produz um resultado (saída de dados) baseando-se no estado em que se encontra e na entrada de dados

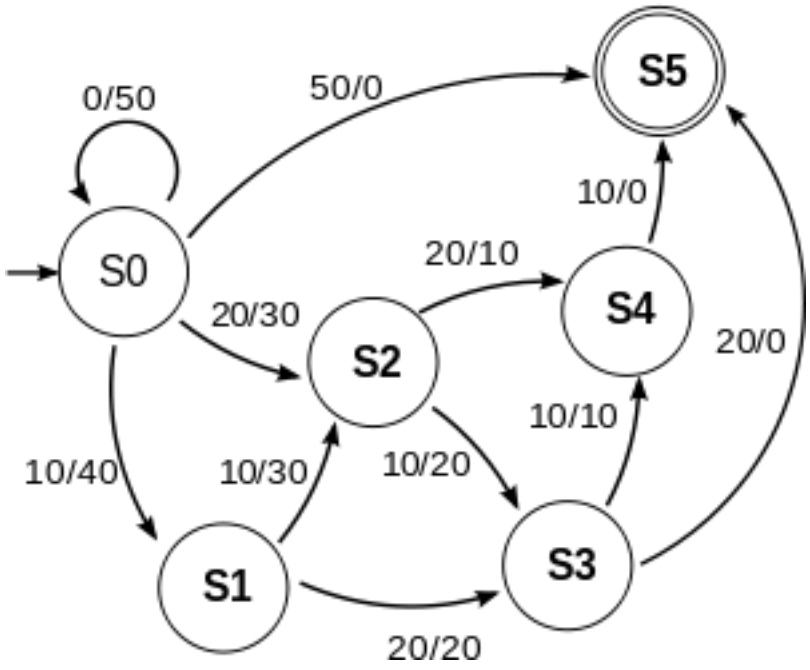
Isto significa que o diagrama de estados irá incluir tanto o **sinal de entrada** como **o de saída** para **cada vértice** de transição

Máquina de Mealy



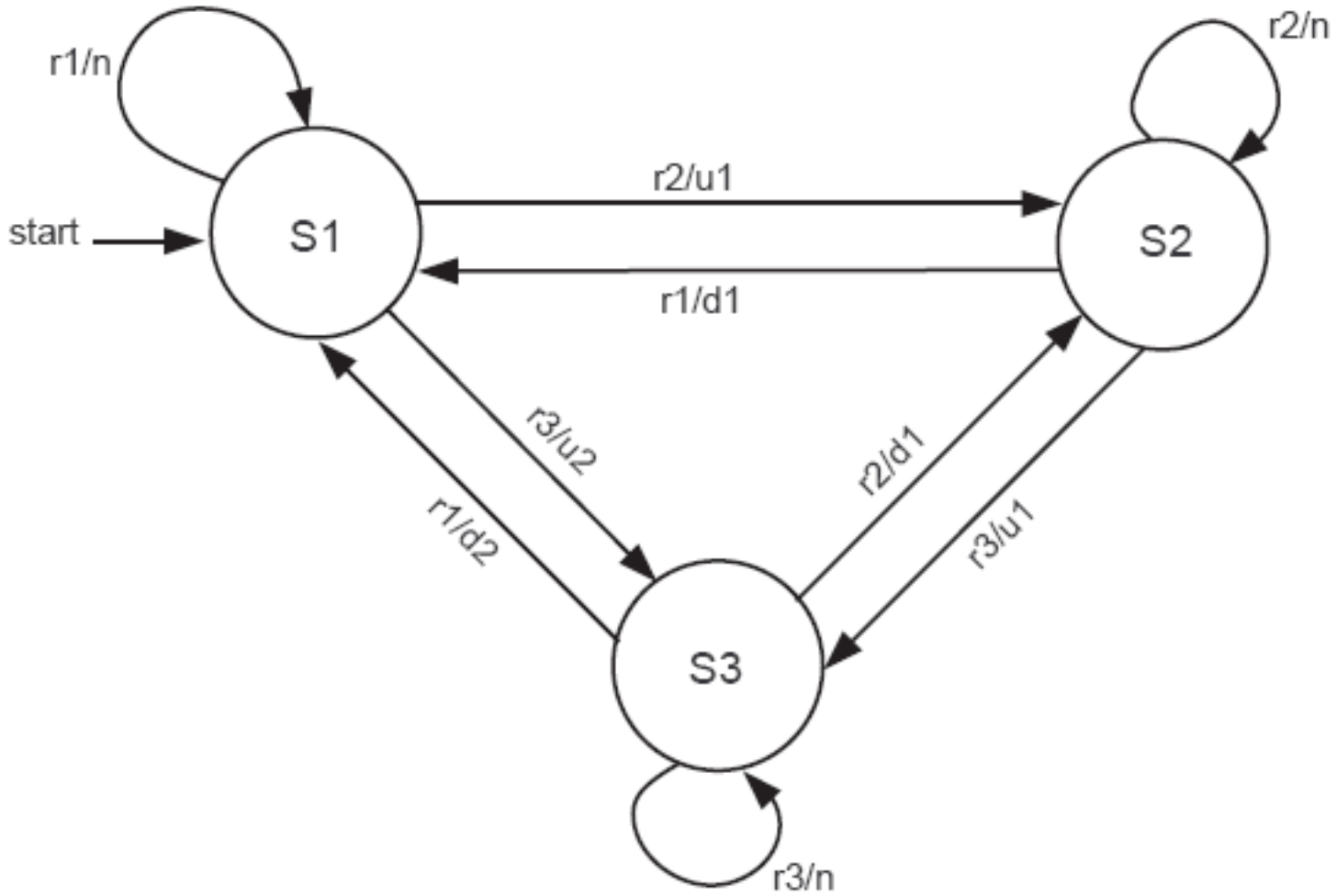
Para a entrada 00101010 produz-se com saída 00010101, que modela um deslocamento de um bit para a direita (ou divisão por dois)

Máquina de Mealy

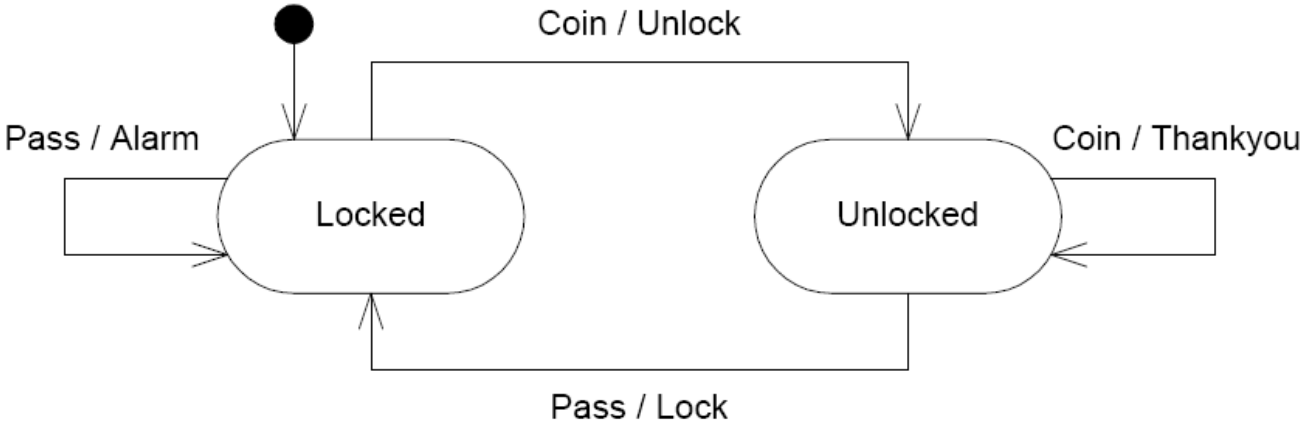


Qual seria a saída para as seguintes entradas: (i) 50; (ii) 20, 10, 20; e (iii) 10,10,10,20? O que faz esta máquina de estados?

Máquina de Mealy



Máquina de Mealy



Entrada	Locked	Unlocked
Pass	Alarm(), Locked	Lock(), Locked
Coin	Unlock(), Unlocked	Thankyou(), Unlocked

Entrada	Locked	Unlocked
Pass	Alarm(), Locked	Lock(), Locked
Coin	Unlock(), Unlocked	Thankyou(), Unlocked


```
enum State {Locked, Unlocked};
enum Event {Pass, Coin};
void Unlock();
void Lock();
void Thankyou();
void Alarm();

void Transition(Event e)
{
  static State s = Locked;
  switch(s)
  {
    case Locked:
      switch(e)
      {
        case Coin:
          s = Unlocked;
          Unlock();
          break;
        case Pass:
          Alarm();
          break;
      }
      break;
    case Unlocked:
      switch(e)
      {
        case Coin:
          Thankyou();
          break;
        case Pass:
          s = Locked;
          Lock();
          break;
      }
      break;
  }
}
```

Máquina de Moore