Lecture 6
State Machine Basics

David Garlan

Carnegie Mellon University

Models of Software Systems
Fall 2016

State Machines

- **The next few lectures will show you how to model systems using *State Machines***
  - > **State Machine Basics**
    - » **motivation, basic concepts, notation, simple examples**
  - > **State Machine Variations**
    - » **modeling complex systems, variables, actions, non-determinism**
  - > **Reasoning about State Machines**
    - » **invariants and constraints**
  - > **Later … Statecharts and other applications**
    - » **a specific, popular graphical approach**

Why State Machines?

- Goal: provide simple abstractions of complex systems
- All computer systems are state machines
  - > registers and memory are state
  - > changes are transitions between states
  - > a program defines the way in which initial states are transformed into final states
  - > a programming language determines a set of programs (and hence, a set of machines)
- Primary challenge will be to represent these very complex machines with simpler (more abstract) machines that we can reason about

State Machines Are Often Used

- **When it is possible to abstract away irrelevant details, leaving only a small number of states**
- **When we want to examine every possibility using exhaustive checking**
- **For communication protocols and complex distributed algorithms (e.g., cache coherency)**

Some Notes

- State machines are the foundation for all of the other forms of modeling that we will be looking at in this course (including Z, CSP, Temporal Logic, etc.)
- There is no standard notation, although the basic concepts are widely agreed upon
- The level of rigor that we use will vary depending on needs and mood

Informally ...

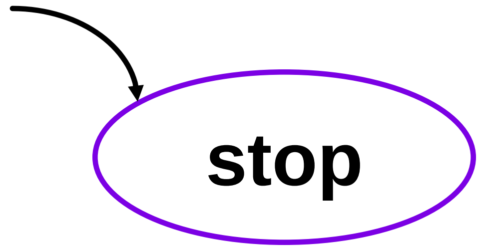
- **A state machine captures the idea that a system progresses through a set of states by performing (or responding to) a set of actions.**
- **Thus there are two key concepts**
  - > ***States***
  - > ***Actions***
- **A state machine definition must say**
  - > **what are the possible *states***
  - > **what *initial states* may the machine start in**
  - > **what are the possible *actions***
  - > **how the state *changes* when actions occur**

What is a State?

- **A snapshot of the system**
  - > **values of memory, registers**
- **Set of values for variables**
  - > **snapshot of a system’s data**
- **Control location(s)**
  - > **snapshot of where a program is in its execution sequence(s)**
- **Contents of communication channels**
  - > **snapshot of a communication state**

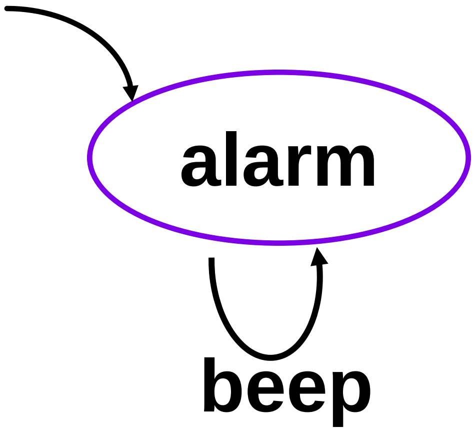
A (Very) Simple State Machine

- World's simplest (and most boring) state machine



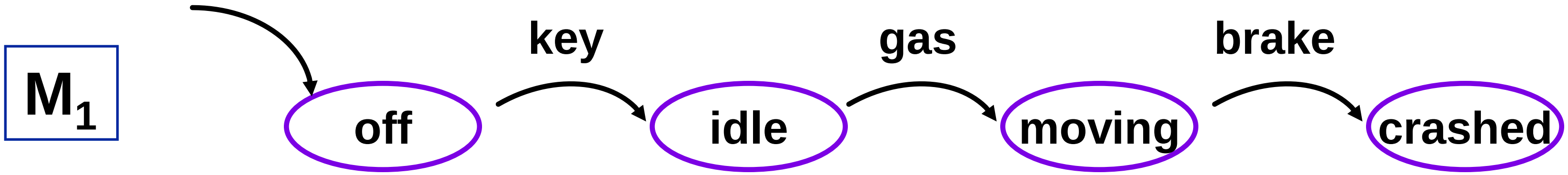
- States: {stop}
- Actions: {}
- Initial state: {stop}
- State changes: none

Another Simple Example



- States: {alarm}
- Actions: {beep}
- Initial state: {alarm}
- State changes: alarm to alarm on beep

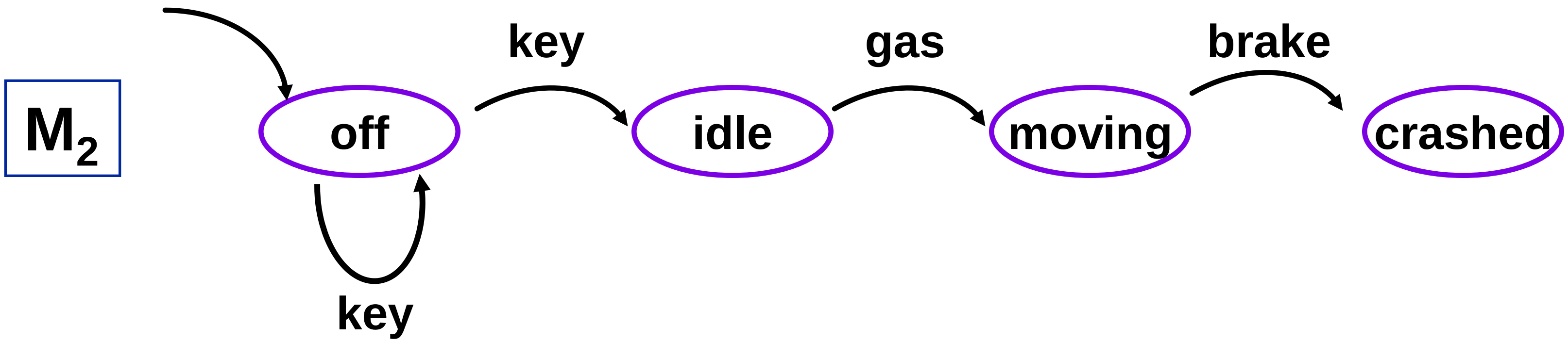
A More Interesting Example



- **States: {off, idle, moving, crashed}**
- **Actions: {key, gas, brake}**
- **Initial state: {off}**
- **State changes:**

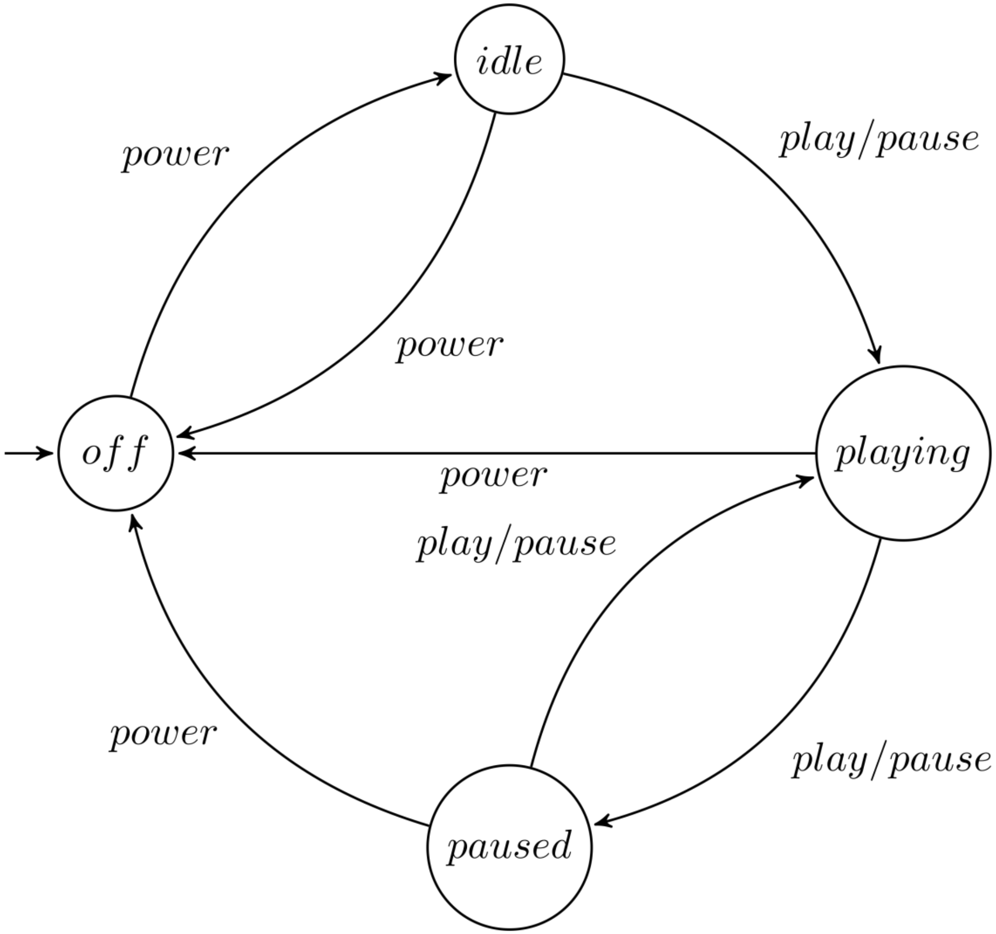
  ***off* to *idle* on *key*, *idle* to *moving* on *gas*, …**

A Small Variation



- **One action may be involved in alternative transitions from the same state**
- **This is referred to as *non-determinism***
- **Note: this state machine now has a possibly infinite number of steps that it can take**

A More Interesting Example

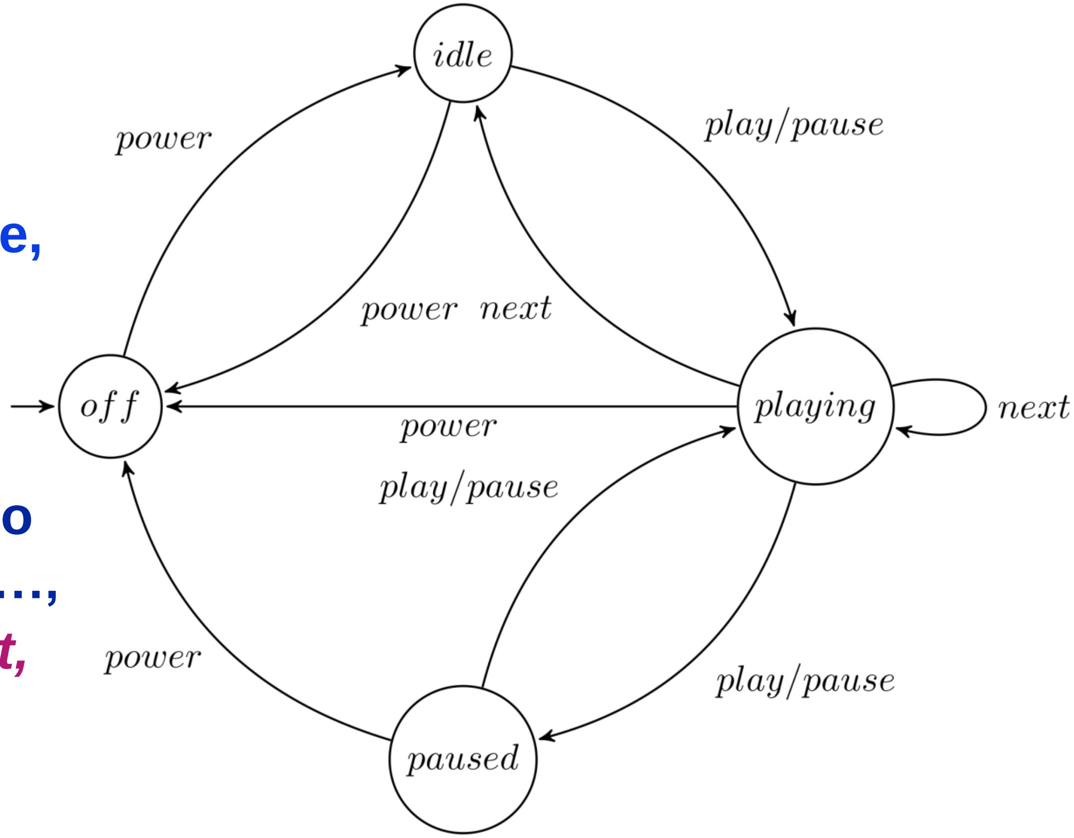


Portable Music Player

- **States: {off, idle, playing, paused}**
- **Actions: {power, play/pause}**
- **Initial state: {off}**
- **State changes:**
  off **to** ***idle*** **on** ***power*****,** ***idle*** **to** ***playing*** **on** ***play/pause*****, …**

A Small Variation

- **States: {off, idle, playing, paused}**
- **Actions: {power, play/pause, next}**
- **Initial state: {off}**
- **State changes:**
  off **to** ***idle*** **on** ***power***, ***idle*** **to** ***playing*** **on** ***play/pause***, **…,**
  playing to playing on next,
  playing to idle on next,...



- **One action may be involved in alternative transitions from the same state**
- **This is referred to as *non-determinism***

Formal Definition

- A ***state machine*** **M is a 4-tuple (S, I, A, T) with**
  - **S : set of possible states**
  - **I : set of initial states (subset of S)**
  - **A : set of actions**
  - **T : transition “relation” over S x A x S**
- **When S is finite, we say that M is a *finite state machine***

Notes on the Definition

- **T is often represented by the symbol δ (Greek delta).**
- **Alternative forms for T**

  (S x A) $\leftrightarrow$ S, representing a binary relation

  (S x A) $\rightarrow \mathbb{P}$ S, representing a function
- **This form of a finite state machine is sometimes called a Labeled Transition System**

Transitions

- **(s1, a, s2) is in T when there is an arrow from *s1* to *s2* labeled *a***
- **When viewed as a *binary* relation (S x A ↔ S), T can map the same state-label pair, to different targets – representing non-determinism**
- **When T is a *function* we say M is a *deterministic state machine***
- **Actions are sometimes called *labels*, *actions*, or state *transitions***
- **A is called the *alphabet* of M**

Car Example (M_1)

- **S = { off, idle, moving, crashed }**
- **I = { off }**
- **A = { key, gas, brake }**
- **T = { (off, key, idle), (idle, gas, moving), (moving, brake, crashed) }**

```
Car == (
      { off, idle, moving, crashed },
      { off },
      { key, gas, brake },
      { (off, key, idle), (idle, gas, moving), (moving,
              brake, crashed) }
 )
```

Executions

- Each triple, (s,a,s') in T is called a ***step*** of M
- An ***execution*** is a finite or infinite sequence
    of the form < s_0 >, or <s_0, a_1, s_1, a_2, s_2, ... , >, where s_0 is an initial state of M, and for all i, (s_i, a_{i+1}, s_{i+1}) is in T
- Examples

  <off>

  <off, key, idle>

  <off, key, idle, gas, moving, brake, crashed>

  <off, key, off, key, off, key, ...>
- For a finite execution, the last state is called a ***final state*** of M (note: often defined differently elsewhere)
- A state is ***reachable*** if it is the final state of some execution

Traces

- **A (event-based) *trace* is the sequence of actions of an execution**

  < key, gas, brake >

  <key>

  <key, key, key, ... >

- **A (state-based) *trace* is the sequence of states of an execution, or the sequence <s> for any initial state, s, in I, the set of initial states**

  < off, idle, moving, crashed>

  <off>

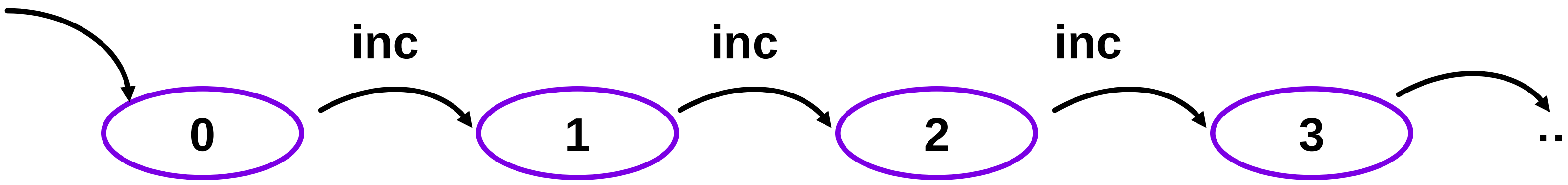
  <off, off, off, ... >

Behavior

- **The *behavior* of a machine M (Beh(M)) is the set of all traces of M.**
  - **Event based:**
    - **{<>, <key>, <key, gas>, <key, gas, brake>}**
    - **{<>, <key>, <key, key>, <key, gas>, <key, key, gas>, …}**
- **Behaviors are *prefix-closed*** (why?)
- **If not indicated otherwise, “behavior” is taken to be event-based**
- **Note: a finite state machine can have infinite behavior** (why?)
- **Question: can an infinite state machine have finite behavior?**

Infinite State Machines

- In general, state machines may not have a finite numbers of states
- Example: integer counter



- How can we write this state machine down?

Informally

We want something like this:

SimpleCounter == (
{0, 1, 2, ...},
{0},
{inc},
{(0,inc,1), (1,inc,2), (2,inc,3), ...}
)

But "..." is not very precise

Solution: you already know how (I hope)

Using Sets and Logic to Describe State Machines

- **The set of states is easy: $\mathbb{N}$**
- **The transition for inc can be described by:**

  T_{inc} == { (s,a,s'): $\mathbb{N}$ x {inc} x $\mathbb{N}$ | s' = s +1 }

  where T_{inc} is the part of T that deals with inc
- **In general, we have**

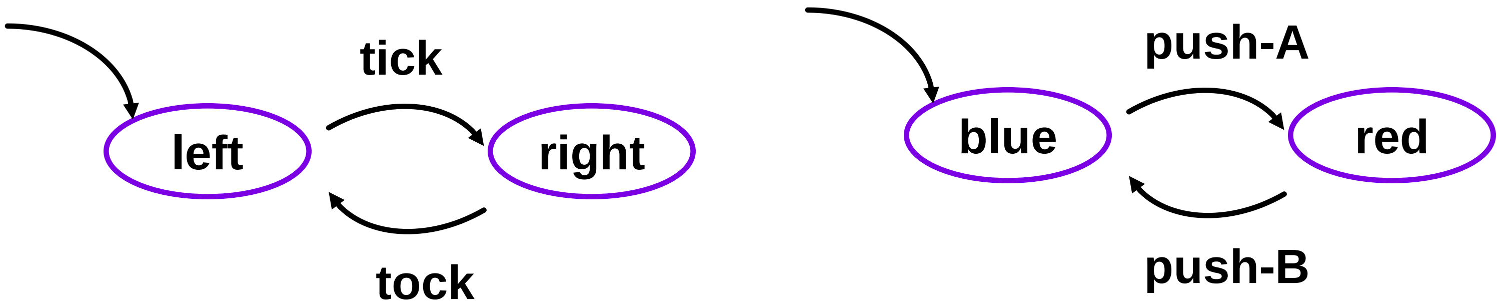
  $$T = \bigcup_{a \in A} T_a$$
- **So,**

  SimpleCounter ==

  ($\mathbb{N}$, {0}, {inc}, { (s,a,s'): $\mathbb{N}$ x {inc} x $\mathbb{N}$ | s' = s +1 })

Interfaces and Environments

- What is the difference between these machines?



- In general, a machine operates in an environment that can either observe an event or cause an event to occur.
- Sometimes it helps to distinguish between observed and initiated events, or input and output actions

Abstraction

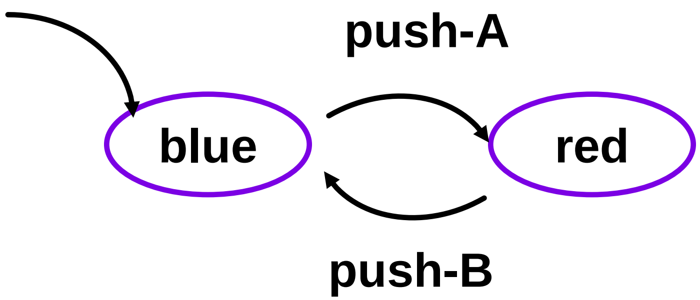
- In all of the examples we are obviously representing only some of the behavior of a system. This is called ***abstraction***.
- Deciding what details to ignore is determined by
  - what will fit on a page
  - what you want to communicate to others
  - what you want to reason about
  - what can be checked by tools
  - experience and practice
  - taste

Scalability

- State can get huge even for very simple systems
- Often we need more compact representations
  > Describe the set of states from which a given transition exists using a *predicate*
  > Describe the `target' in terms of changes to the source
  > Use *text* and not pictures
  > Focus on *specialized aspects* (such as event traces)
- The rest of this course can be seen as an elaboration of this point

Unexpected Actions

- Consider the following machine



- What happens if I push button B when the machine is in the blue state?
  > Nothing
  > It is undefined -- anything can happen
  > It is an error
  > It can't happen

We'll adopt this interpretation

Another Example

- Infusion pump
  - An infusion pump is a device used in hospitals to feed fluids intravenously to patients through one of several “infusion lines.” Each line is a physical tube connected to a patient.
  - What would the state machine look like for pump that operates a single infusion line?
    - what aspects of the situation should be modeled?
    - what are the states?
    - what are the actions?

For Next Time

- **Chapter 9 on State Machine Variations**