Review

Part II:
The Power Of
Mathematical Notations

The Power Of Mathematical Notations

- Text book for this part:
  Harold Thimbleby (UCL Interaction Centre, London):
  "Press On"
  - To be published, pre-print PDF version at
    http://www.uclic.ucl.ac.uk/harold/book/index.html

Culture

- Computers are like the Winchester Mistery House
  - Staircases leading nowhere
  - Cupboards with nothing behind their door

Culture

- The Magic Machine
- Externalizing costs
- Software warranties
- Bad interaction design –> formalize!

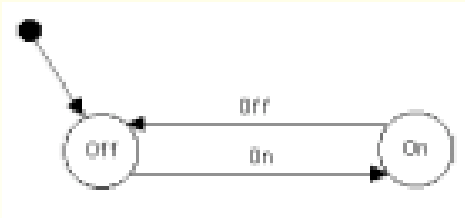
- *Book key point: Describe UI behavior mathematically to improve usability in a predictable way*

State machines (FSMs)

- Describe UIs (discrete systems) by *states* and *actions*
  - user generates actions *(*pressing buttons…) which cause effects
- Mode
  - in a given mode, an action has a unique effect
  - a mode tells what a button will do (e.g., on/off button)
- State
  - in the same state, the same actions have exactly the same effects
  - a state tells what the system will do
  - e.g., television state:
    <on/off, channel, sound level, color, brightness…>
- Timeouts and synchronization problems in many systems!
  - system resets after certain time, user cannot find a certain state

Drawing state machines

- *Circles* represent states
- *Arrows* represent actions
- Indicate *default state* with special arrow



- Example: torch
  - 2 states: on, off
  - 2 actions: switch on, switch off
  - more detailed analysis reveals additional states / actions (e.g., dead bulb, no batteries, broken, replace bulb…)

- Number of states and actions depend on what we try to achieve as UI analysts!
- Some states are unimportant to our needs
- Computer has too many states—clump them together
- Example: alarm clock has 4 million states
  - How could users check?

Rules for drawing simple state diagrams

- Every arrow starts and finishes at a state circle
- A state has as many arrows pointing from it as possible actions are available
- Only one initial state exists
- Arrows can start and finish at the same state
- Terminal states have no outgoing arrows (error!)
- States without incoming arrows are never reached (error!)
- Strong connectivity: all states must be reached from all other states following arrows

Statecharts

- Goal: Simplify drawings for complex state machines
  - Example: Saving arrow to Off state from every state
- Here: Basic statecharts only
  - More general statecharts in **UML**

- States can be collected into **state clusters**

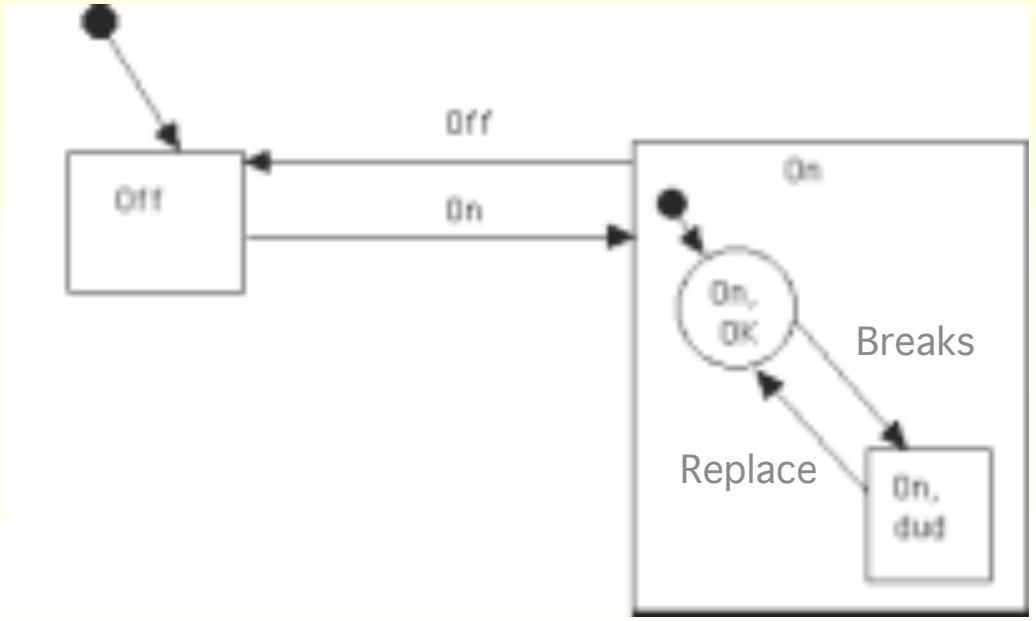
cluster

- A state cluster represents a **mode** for an action iff we can draw an arrow for that action from the cluster
  - See also Raskin

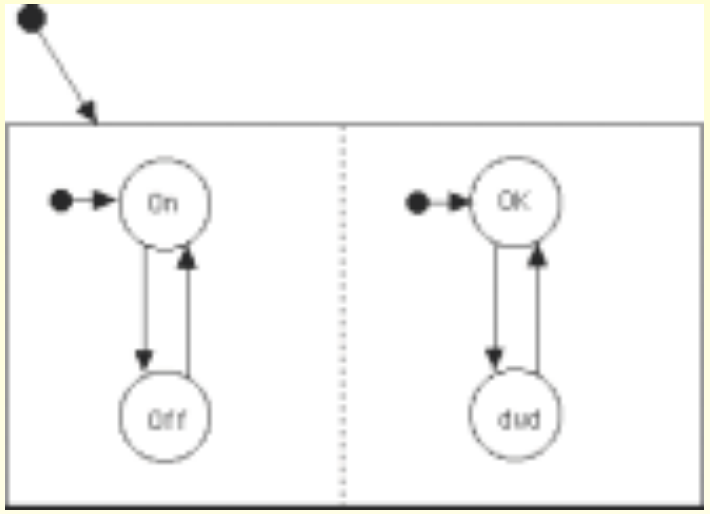
On and Off state clusters of our torch

Multilevel Statecharts

- What is the default state inside the On cluster?
  - Make On arrow point to a state inside On cluster
  - Or mark default state as usual
- State clusters can contain state clusters
  - Example: More detailed On state for our torch



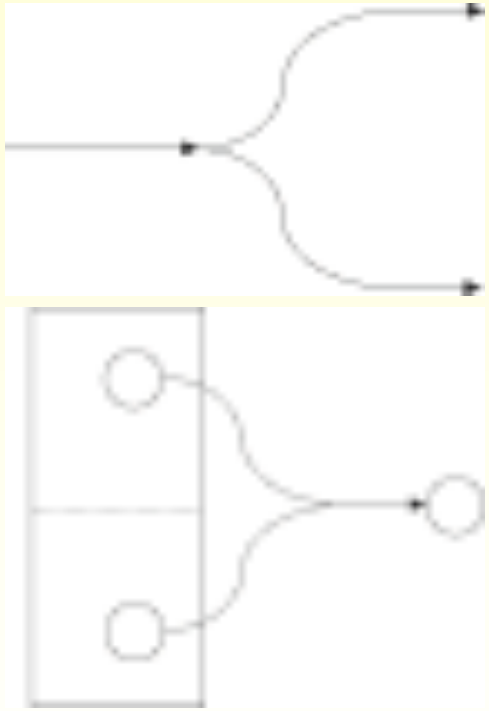
AND States



- Torch: Off state should be symmetrical to On state
  - Bulb can be OK or broken
  - Breaking and replacing it also does the same
  - On and Off switches work independently of bulb health
- *AND* states can represent this "repetition"
  - So far, state machines were always in exactly **one** state at a time (coin analogy)
  - State cluster divided by **dotted line:** actions on both sides of the line can happen independently (two coins); saves arrows

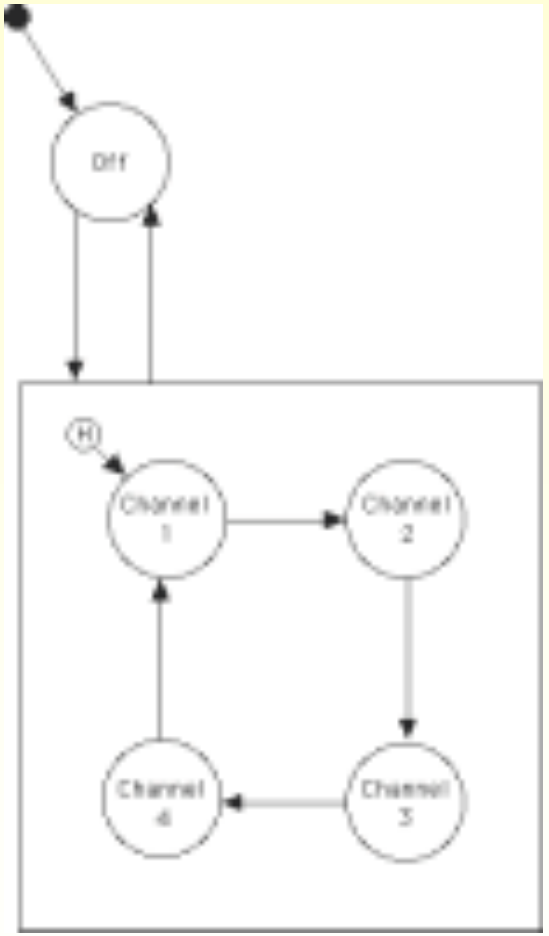
Joint Connectors

- Entering several AND states upon an action:
- Only allowing an action if several AND states are active (e.g., only allow bulb change if broken and off):



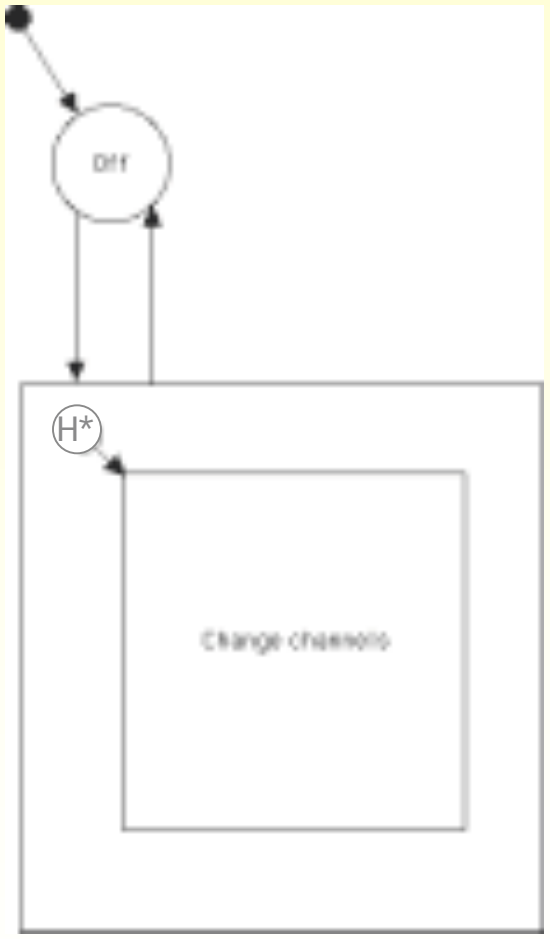
History Entrances

- Example: TV set that remembers channel while off
- When cluster is entered, go to the state that cluster was in last (remember cluster state)
- Variant of Default arrow, marked with an "H"
- Imagine leaving coins in clusters (maybe flipped over for "inactive")
- More general: Petri Nets



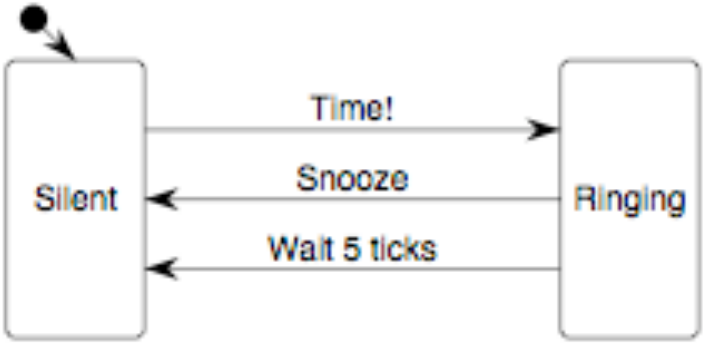
Deep History

- For nested clusters, need to specify nested history
- Shortcut: H* marks **Deep History** arrow that reaches all the way into a nested cluster.



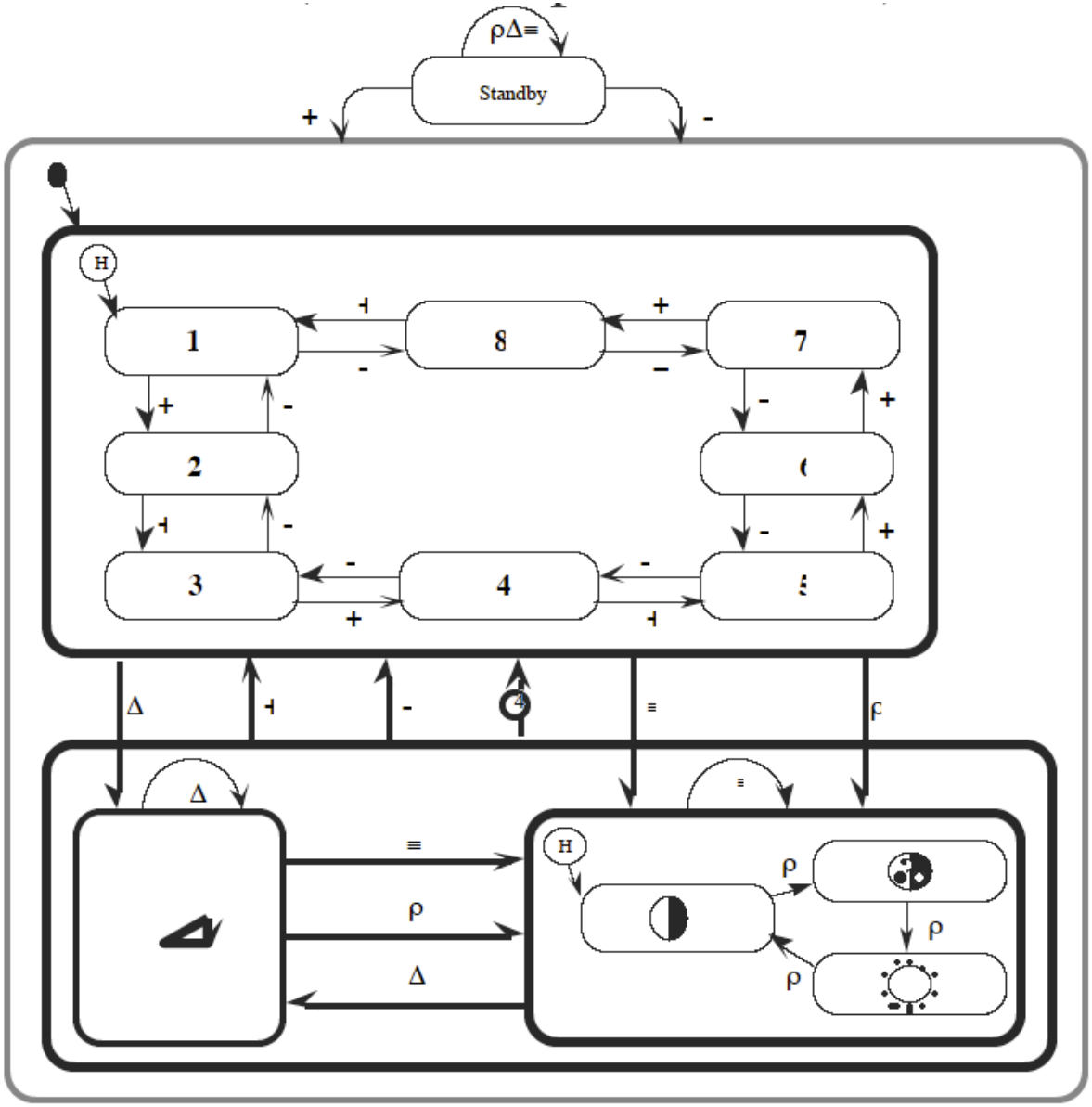
Delays and Conditions

- Delays *(almost always evil!):*
  - Trigger if nothing happened for a while
  - Or delay action for a while after trigger
- Conditions:
  - Action can only occur when certain conditions hold true
  - Can always be replaced with explicit states, but sometimes saves drawing lots of states
  - Alarm clock example



Example: Sony TV

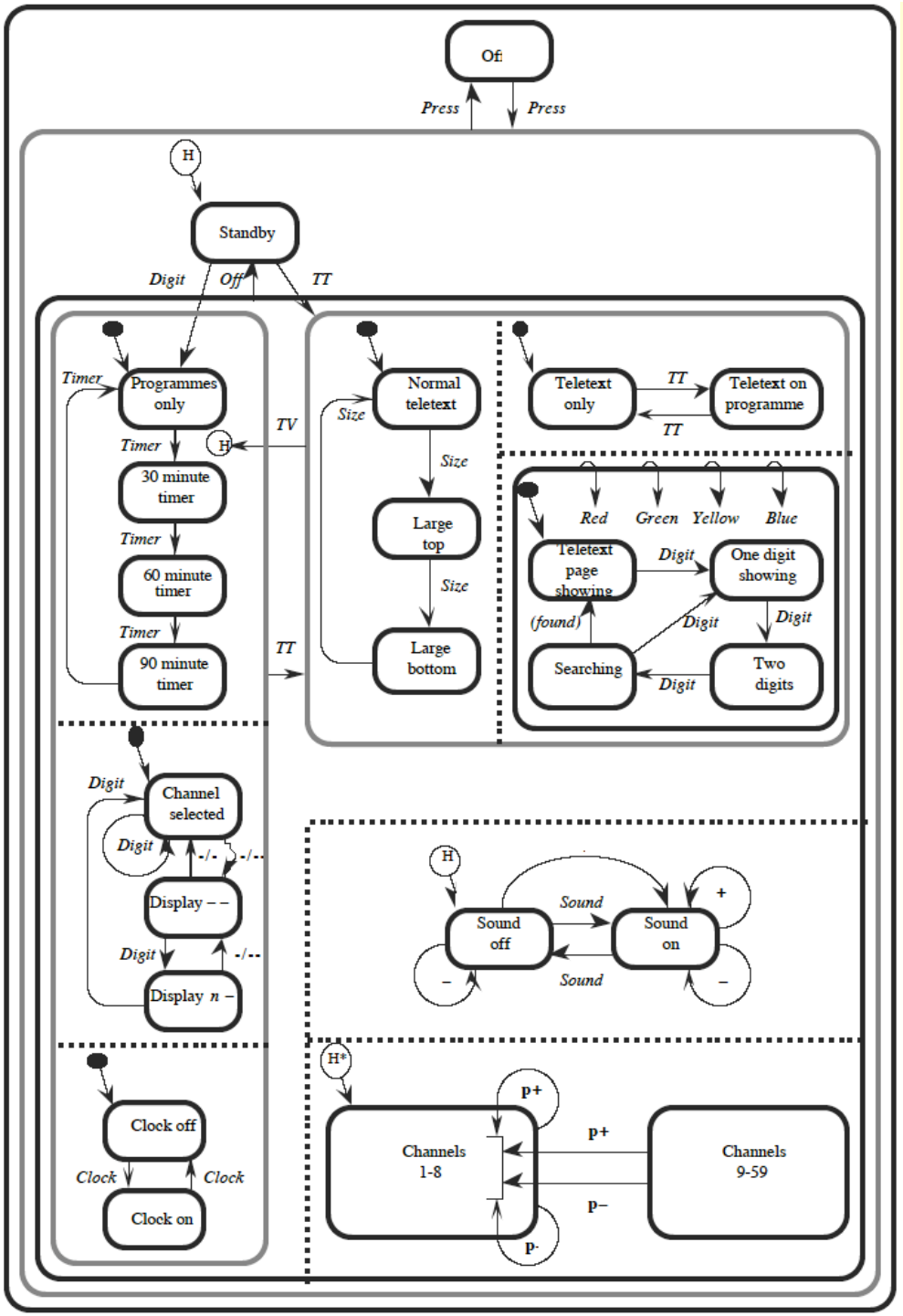
- 8 channels
- Buttons:
  - Standby
  - Channel: +/–
  - Volume: Δ+, Δ–
  - Contrast/Color/ Brightness: ρ
  - Pict. Adjust: ≡+, ≡–
- Incomplete
  - Missing details for volume / brightness / color / contrast selection



Note. ≡ means either of the ≡+ or ≡– buttons;
Δ means either of the Δ+ or Δ– buttons

Sony Remote

- What do you notice?
- Different from its TV
  - More complex
  - Why?
- Strange channel split
- Missing details
  - Channels
  - Clock



Undo

- What does Undo look like in a state chart?
- Back arrows with inverse action â
- Toggle switches are easier than single toggle buttons
- What do several switches on a device look like?
- Divided by dotted AND line
- Number of states drawn: m+n
- Number of states posible: m*n
- What does an UNDO button look like?
- Lots of new states and arrows
- Therefore, mark statechart as "undoable", then every arrow implicitly has an undoable action (cross through exceptions)

Books on Statecharts

- Harel, Politi: *Modeling Reactive Systems with Statecharts* (the definitive book)
- Horrocks: *Constructing the User Interface with Statecharts* (more practical, programming-oriented)
- Fowler, Scott: *UML Distilled* (UML introduction)

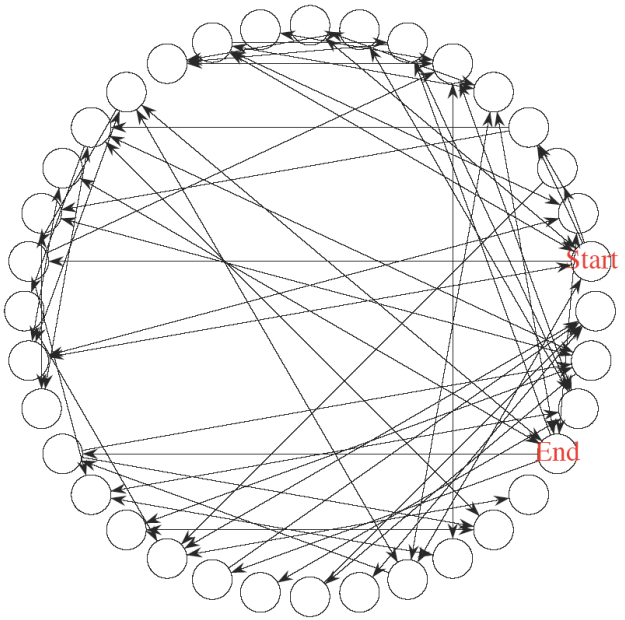
Programming With States

- State may be represented by a variable
- Actions may be represented by functions
  - function off() { state = 0 };
- The FSM can be represented as a matrix
- Example: Light bulb (off, dim, on) with 3 buttons:

  0 1 2
  0 1 2
  0 1 2

  When in state x and button y is pressed, go to state (x,y)

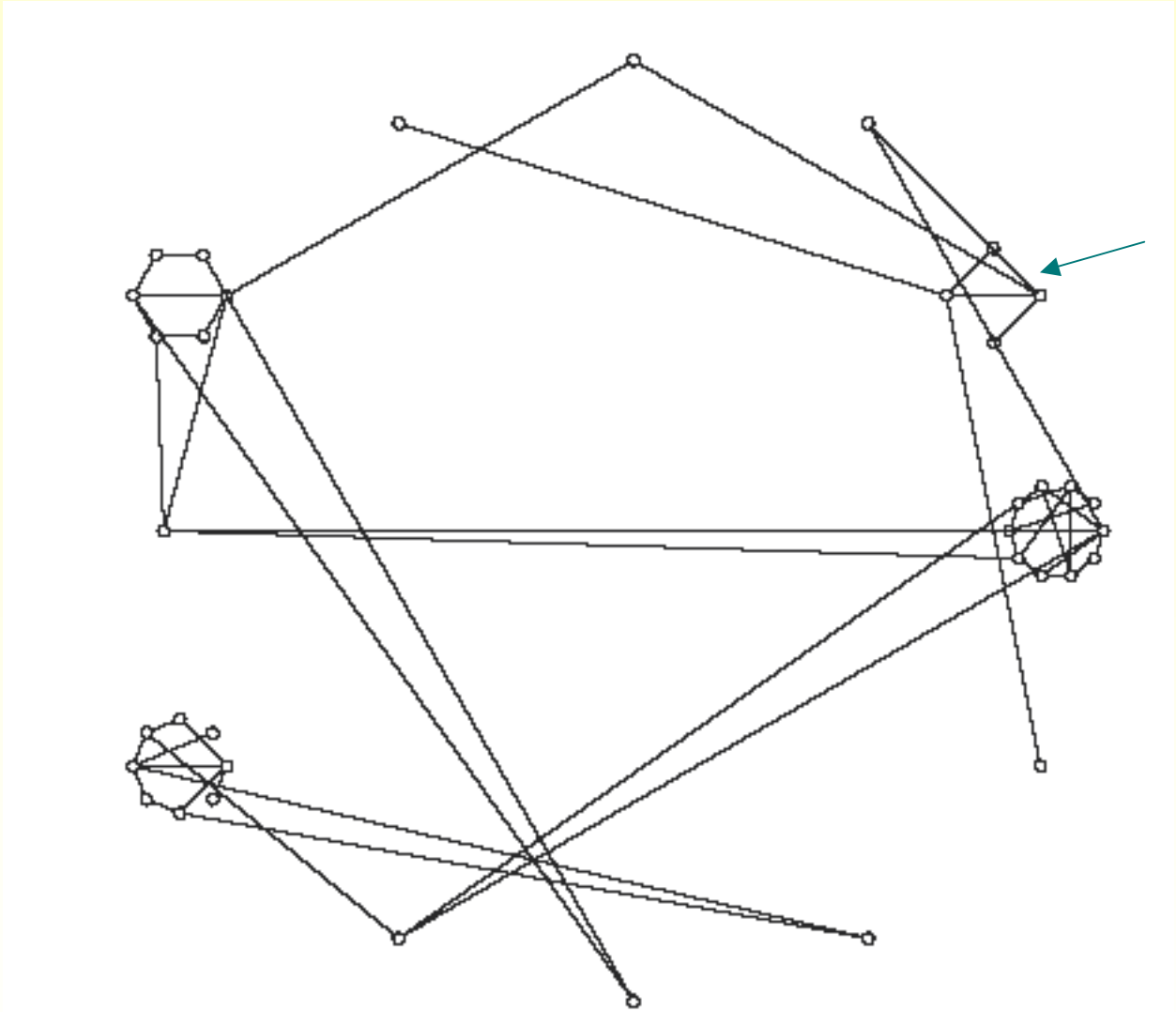
Strong Connectivity



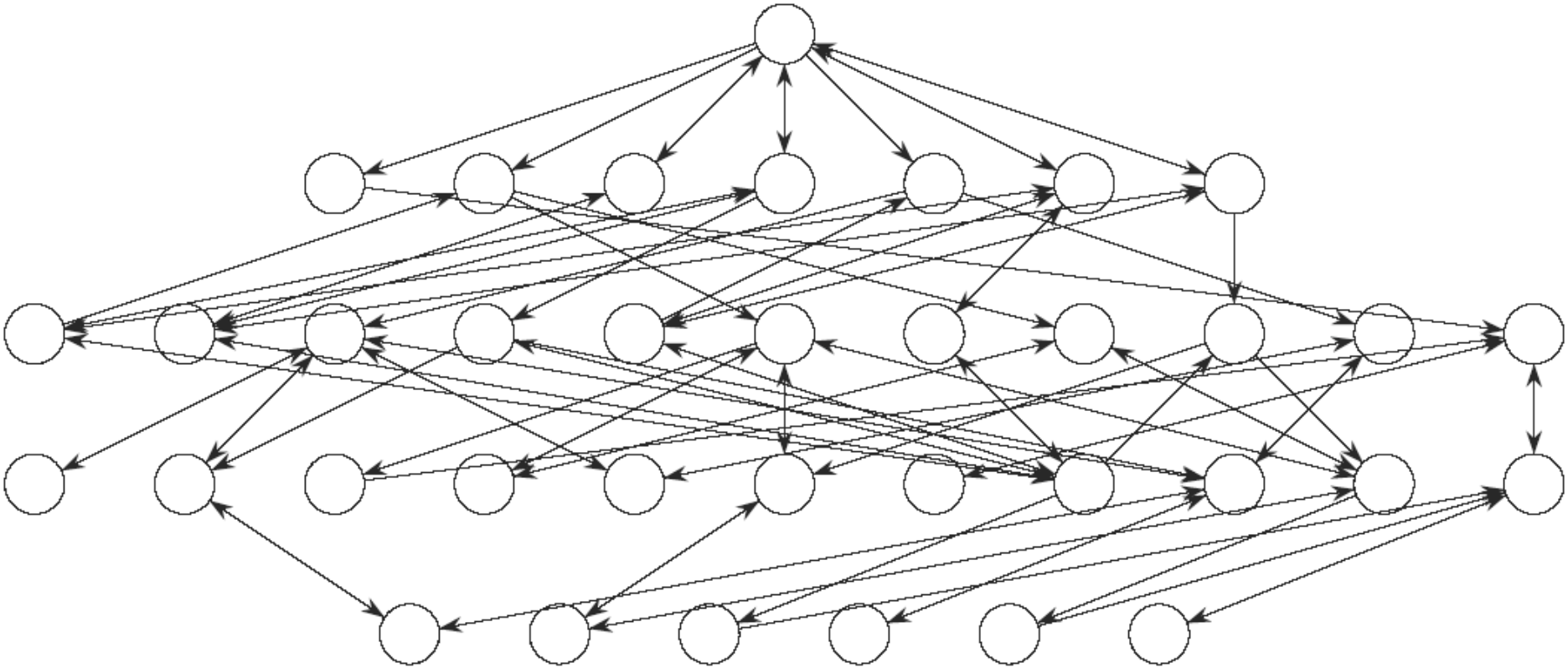
- What does the Farmer's Problem (farmer, wolf, cabbage) look like as an FSM?
- Hard because need to find *route* through FSM
- A *strongly connected* system is a system where the user can get from any state to any other state
- The Farmer's Problem is not strongly connected
  - Cannot go back when you made a mistake
- A *strongly connected component* is a subset of states in a statechart that is strongly connected

Strong Connectivity

- Algorithms for finding them well known
- Important for usability
- Hard to find by users or empirical testing
  - Designers' responsibility!
- Farmer's Problem has 12 strongly connected components of various sizes (can you find them?)



The twelve strongly connected components arranged around a clock



- Same diagram, arranged in rows by distance from start state at the top
- Useful: An optimal solution to the problem is one that only goes down, never up or sideways in this graph
  - Otherwise a shorter route would have been possible
- Graph shape also gives a feel for complexity of using device

Connectivity

- Problem hard because some states are one-way (not the end state btw.)
- So: Remove these states to get an easier to solve problem diagram
- Can be done automatically!
- For actual devices, this would remove states in which the user could get stuck (good idea)