Operational specifications
Finite state modeling / state diagrams

Recall
Operational specifications describe behavior of a system via an abstract machine.
With data-flow diagrams, the abstract machine is an assembly of functions connected by data flows initiated by some actor(s).

   - useful for incremental analysis of requirements of information systems
   - however, incapable of expressing control aspects

Finite-state models
Describe temporal/behavioral view of a system

Specify control:
   - Sequence operations in response to stimuli
   - Distinguish states, events, and transitions
   - Especially useful during design

Lots of variants:
   - E.g., StateCharts, UML state diagrams
   - E.g., FSP (textual notation)

Key terms
Event: occurrence at a point in time
   - instantaneous
   - often corresponds to verb in past tense
     - e.g., alarm set, power turned on
   - or onset of a condition
     - e.g., paper tray becomes empty, temperature drops below freezing

State: behavioral condition that persists in time
   - often corresponds to verbs with suffix of "-ing"
     - E.g., Boiling, Waiting, Dialing
   - abstraction of values of attributes and configuration of objects

Transition: instantaneous change in state
   - triggered by an event

State diagrams
Graphical state-modeling notation:
   - States: labeled roundtangles
   - Transitions: directed arcs
   - Events: parameterized actions or conditions

Example:

\[ \text{S} \quad \text{T} \]

\[ \text{Transition} \]

\[ \text{S} \quad \text{T} \]
State diagrams

Graphical state-modeling notation:
- States: labeled roundtangles
- Transitions: directed arcs
- Events: parameterized actions or conditions

Example:

<table>
<thead>
<tr>
<th>States: labeled roundtangles</th>
<th>Transitions: directed arcs</th>
<th>Events: parameterized actions or conditions</th>
</tr>
</thead>
</table>

Enabling and firing of transitions

Transition is:
- enabled when source state is active and conditions are satisfied
- fires when enabled and the action is observed

Example below:
- enabled when current state is Editing and the form is complete
- fires when the user presses the "OK" button

Example

Chess game

White’s turn
- Black wins
- [checkmate]
- Draw
- [stalemate]

Black’s turn
- White wins
- [checkmate]
- Draw
- [stalemate]

Problems with FSMs

Difficult to read with lots of states and transitions
Two sources:
- Multiple transitions with same event but different source and/or target states
- State explosion due to concurrency

Ameliorated somewhat by modularity features:
- State generalization
- Parallel composition
Example: Automatic transmission

Problem: Multiple similar transitions

Solution: State generalization

State generalization

Introduces an abstract “super state”:
- decomposes into multiple substates
- when super state is active, exactly one of its substates is active

Outbound transition incident on superstate abbreviates set of transitions, one from each substate

Inbound transition incident on superstate enters substate that is distinguished as the start state

Problem: Composite behaviors

Consider an automobile with multiple options:
- Automatic transmission
- Temperature control (heating/air)
- Rear-window defroster
- Stereo system

Suppose we wish to construct a state diagram for the automobile:
- Assume car starts with transmission in neutral and temp control, rear defroster, and stereo are all off
- What are the possible next states?

Example: Automobile states
State explosion problem

Number of states in a composite diagram is product of the number of states in component diagrams

Major impediment to understanding:
- Impossible to visualize in any meaningful way
- Requires the use of analysis tools to verify properties

Managing state explosion:
- Concurrent state diagrams
- Highly effective when diagram can be separated into truly orthogonal components

Example

Semantics of parallel composition

Multiple interpretations:
- Concurrent regions execute independently
  - What happens if transitions in different regions are triggered by same event?
  - Do both execute simultaneously? Does one “consume” the event to the exclusion of the other?
- Concurrent regions communicate with one another, synchronizing on common events
  - Regions can only proceed when all are ready to proceed
  - Regions transfer data values during a concurrent transition
- Do we distinguish internal and external events?

Communicating sequential regions

We will now do an example that assumes regions communicate and synchronize with one another on common actions:
- E.g., if two (or more) regions are at any time capable of observing action A then all must observe A and transition accordingly at the same time

Note: This interpretation followed by FSP and some variants of StateCharts, not by UML 2.0

Producer–consumer system

Producer write $p_1$ produce $p_2$  Consumer consume $c_1$ read $c_2$

Two-slot buffer

System

Producer write $p_1$ produce $p_2$  Consumer consume $c_1$ read $c_2$

Two-slot buffer
Analyzing behavior

Diagrams with communicating regions have tricky semantics
- Diagram may “deadlock” if two or more regions are mutually waiting on each other to offer an action or set a condition
- Regions may enter “unsafe” configurations
  * E.g., “writer” region modifying shared data concurrently with “reader” regions

To detect problems we derive a product state diagram and search for anomalies

State explosion problem

Number of states in a product diagram quickly explodes
- why we introduced concurrent regions in first place!
Impossible to find problems by inspection
Must use analysis tools to compute and exhaustively analyze these state spaces