CSE503: Software Engineering

David Notkin
University of Washington
Department of Computer Science & Engineering
Winter 2002

1

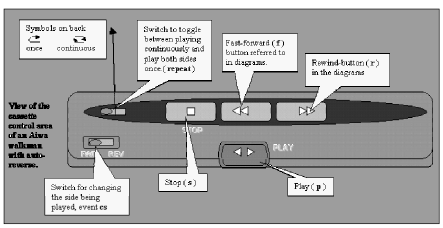
Finite-State Specifications

- There is a large class of specification languages based on finite state machines
- Often primarily for describing the control aspects of reactive systems
- The theoretical basis is very firm
  - Lots of theory on finite-state machines, plus analysis support from theorem proving and model checking
  - As we'll see during the lectures on tools and analysis, modeling checking is increasing feasible for analyzing this kind of specification

2

Walkman example

(due to Alistair Kilgour, Heriot-Watt University)



- What happens when…?
- The implementers need to know, the users need to know, …

3

Reactive systems

- Essentially event-driven systems that responds to both external (from the environment) and internally-generated stimuli, and also provides stimuli to the external environment
- These are generally embedded systems in which we care about the behavior of the overall system, not the software per se
- As fewer and fewer complex systems are built without software — one can legitimately view this as inappropriate and, in some cases, perhaps even unethical — the pressures on properly specifying (and analyzing) reactive systems increases

4

Many, many models

- Standard finite state machines
  - Set of states
  - One initial state
  - Zero or more termination states
  - Finite alphabet
  - Transition relation
- Petri nets
- Communicating finite state machines
- Statecharts
- RSML
- …

5

A common problem

- It is often the case that conventional finite state machines blow-up in size for big problems
- This is especially true for deterministic machines
  - And these are usually preferable to non-deterministic ones, because they don't allow implementers to make decisions about the behavior of the specified system
- And for machines capturing concurrency (because of the potential interleavings that must be captured)

6

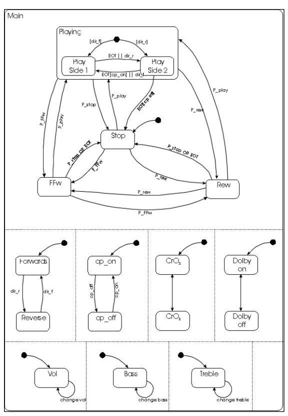
State explosion

- The state explosion problem is very similar to the potential blow-up that arises when transforming a non-deterministic finite-state machine to a deterministic one
- There is a potential exponential blowup: an N-state machine can become an 2^N-state machine
- As a high-level example think
  - of a state machine that tracks the amount of money put into a vending machine and
  - of a state machine that tracks the buttons pushed on the vending machine to indicate which product to purchase
  - if money can be entered and buttons can be pushed in an interleaved fashion, consider the fully expanded single state machine that composes these two sub-machines

Statecharts (Harel)

- A visual formalism for defining finite state machines
- A hierarchical mechanism allows for complex machines to be defined by smaller descriptions
  - Parallel states (AND decomposition)
  - Conventional OR decomposition
- The reduced size of the description is a central piece of the leverage of statecharts

Walkman example: statechart



Communicating state machines

- In conventional state machines, precisely one state must be occupied at a given time
- In communicating state machines (including statecharts), every machine in a composition must occupy one state at a given time
  - This allows (in part) the blow-up of representation to be mitigated, because now a pair of communicating state machines can represent NxM states in the overall machine using N+M states

Hierarchical state machines

- Harel's additional insight was to allow the hierarchical definition of state machines
  - It's basically an and-or tree of state machines
  - Machines separated by dotted lines are "and" machines, where each of the machines occupies exactly one state at a time; it's easy to imagine taking the cross product to create a flattened machine
  - Everything else is an "or" machine, essentially like a standard state machine (although they can in turn be nested "and" machines)

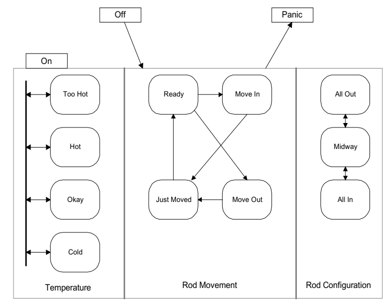
Tons of details

- As you noted in the paper, there are many details
- What are the start states upon entering an "and" machine?
  - These notations usually have an arrow with nothing at the tail pointing at the start states.
- What happens upon exits from a nested state?
  - Nested states are allowed to cause exits from the enclosing "and" machines (usually by showing a transition to the edge of the enclosed box)
- And more, more, more!

An RSML example

- The following slide shows a very rough "statechart" from RSML
  - RSML is a variant of statecharts developed specifically for the specification of TCAS (Traffic Collision Avoidance System)
  - I will call all descriptions in these similar languages "statecharts"
- Three high-level states: on, off, and panic
- The on state is expanded and has three parallel states: temperature, rod movement, and rod configuration
- The only non-traditional statecharts feature in this description is the temperature state, which uses a bus that connects all substates (too hot, hot, okay, cold) to one another
- There are six events listed at the bottom (this is an incomplete list)
  - Each event has a name, a description of how it is generated (externally or by a specific sub-machine in the description), and a list of the sub-machines that react to that event

13



Events

Temp_Reading	External	Temperature
Initiate_Move	Rod_Move	Rod_Config
Move_Finished	External	Rod_Config
Rod_Updated	Rod_Config	Rod_Move
Clock_Event	External	Rod_Move
Temp_Update	Temperature	Rod_Move

Sample transitions

On → Panic

```
Trigger_Event: Temp_Update
Condition: Temperature in Too Hot
Output Action: Panic_Event
```

Ready → Move In

```
Trigger_Event: Temp_Update
Condition: Rod_Movement in Ready and Temperature in Hot
Output Action: Initiate_Move
```

Just Moved → Ready

```
Trigger_Event: Clock_Event
Condition: Rod_Movement in Just_Moved and
           t > t(entered(Just_Moved))+ Move_Delay
```

- This slide shows three sample transitions
- Conditions on the transitions are common
- Output actions are also listed here

15

Events

- External—interactions with environment
- Synchrony hypothesis (from Esterel)
  - External event arrives
  - Triggers cascade of internal events (micro steps)
  - Stability reached before next external event
- RSML requires the synchrony hypothesis
- Statecharts gives a choice

16

Synchrony hypothesis

- Accept a single external event and then propagate all internal events until the machine stabilizes, and then accept another external event, etc.
- One model of this is to think of the machine as executing infinitely fast
- The alternative is to allow external and internal events to interleave
- The latter alternative appears to be used in hardware specifications more frequently, and the former in software specifications (so we will consider the synchrony hypothesis as a rule)

17

Semantics

- What to do when there are multiple events available: which of the enabled transitions should be taken?
- There are literally dozens of (published) choices, with subtle distinctions
- Some of the more theoretically pleasing semantics seem, unfortunately, to be less intuitive to people
- It is, however, critical to have a well-defined semantics; after all, these are specification languages
  - The most common semantics are the "Statemate semantics", Harel and Naamad, which define the formal semantics of statecharts in terms of the operational semantics defined by the Statemate tool
- At the same time, for most "normal" examples, the differences among the semantics are not significant

18

Reasoning

- The definition of precise semantics allows reasoning of the meaning of statecharts
- Given an initial state
  - And a set of possible external events
  - What states can be reached?
- Again, not that different from program correctness, model-based specifications, or algebraic specifications: reason inductively

19

Differences

- But state-based specifications are fundamentally different from model-based and algebraic-specifications
- More importantly, a central focus on specifying control (as opposed to state, or pseudo-state as in algebraic specifications)
- The computations represented at specific nodes (states) in statecharts are generally not part of the basic specification and reasoning
  - But they are, of course, important
  - And they are addressed by some notations and tools

20