CSE P503: Principles of Software Engineering

David Notkin
Autumn 2007

Requirements and Specifications … or

Man is the only animal whose desires increase as they are fed; the only animal that is never satisfied.
--Henry George

The designer of a new kind of system must participate fully in the implementation.
--Donald Knuth

I have yet to see any problem, however complicated, which, when you looked at it in right way, did not become still more complicated.
--Poul Anderson

Some announcements and reminders

- If you are not receiving email from the class list, then make sure to let Jonathan or me know so we can add you – and messages are now being archived
- I will start to place more material on the wiki – again, if you cannot access it, let Jonathan or me know
  - In particular, I'll try to place material that is likely to be "discussable" there, in an attempt to encourage an exchange of viewpoints
- Format and citations for essay and papers: I don't really care, as long as it is reasonable – as a very rough guide, you might consider articles in *IEEE Software* and in *CACM* for this

Our plan of attack: this week

- Analysis of state machine based specifications (model checking)
- Michael Jackson on video: “The World and the Machine”

Interlude: but what should we *do*?

- A student writes (roughly):
  - "*Software engineering seems very good at telling me what not to do.*
  - *"How *do* you do things, instead of what *doesn't* work properly."*
- Back at you: take two minutes with another student or two and produce one or two imaginable "actionable" principles that would be of the form you'd like to have
  - Example: "The application of test-driven development has been shown in some studies to reduce bug counts by an _order of magnitude_ over standard techniques where tests are written after the fact." [from a student in class]
  - Your examples don't have to be "true" – just in a form that captures what you'd like to see

What is dependability

- Based on experience as part of a large NASA-funded project on dependability, it became clear to me that
  - Dependability is different things to different people
  - There are two camps
    - Use technology to improve dependability
    - Build a “culture of dependability”
- Without a “designation” of dependability, surely efforts to increase dependability will be complicated and perhaps compromised
- Surely it’s a combination of culture and of technology; we’ll focus on one technology tonight

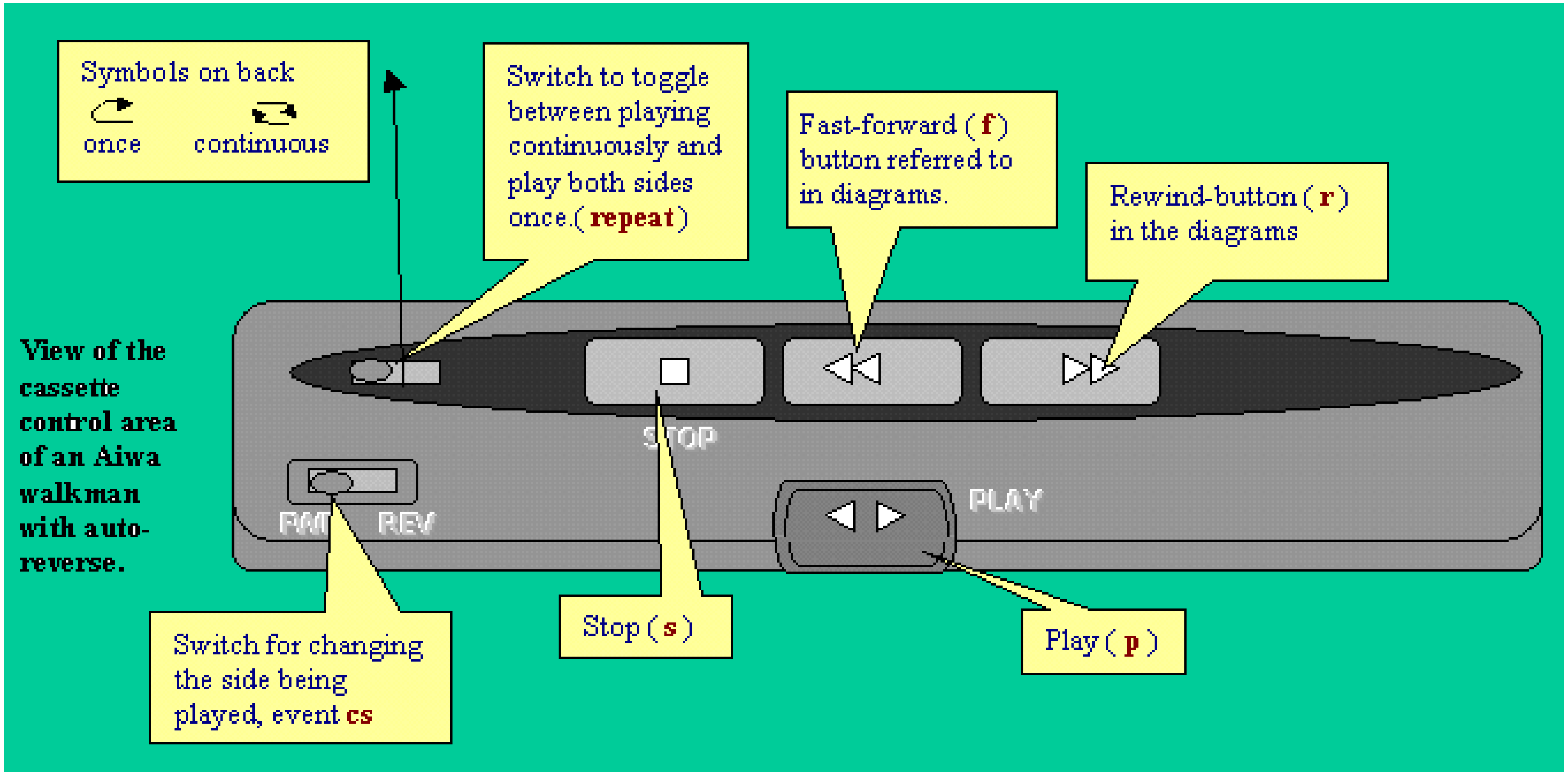
Finite state machines

- There is a large class of specification languages based on finite state machines
  - A finite set of states
  - A finite alphabet of symbols
  - A start state and zero or more final states
  - A transition relation
- Often used for describing the control aspects of reactive systems (and much, much more!)
- The theoretical basis is very firm
- Many models including Petri nets, communicating finite state machines, statecharts, RSML, …

State machines: event-driven

- External events (actions in the external environment, such as “button pushed”, “door opened”, “nuclear core above safe temperature”, etc.)
- Internal events (actions defined in the internal system to cause needed actions)
- Can generate external events that may drive actuators in the environment (valves may be opened, alarms may be rung, etc.)
- Transitions can have guards and conditions that control whether or not they are taken

Walkman example
(due to Alistair Kilgour, Heriot-Watt University)

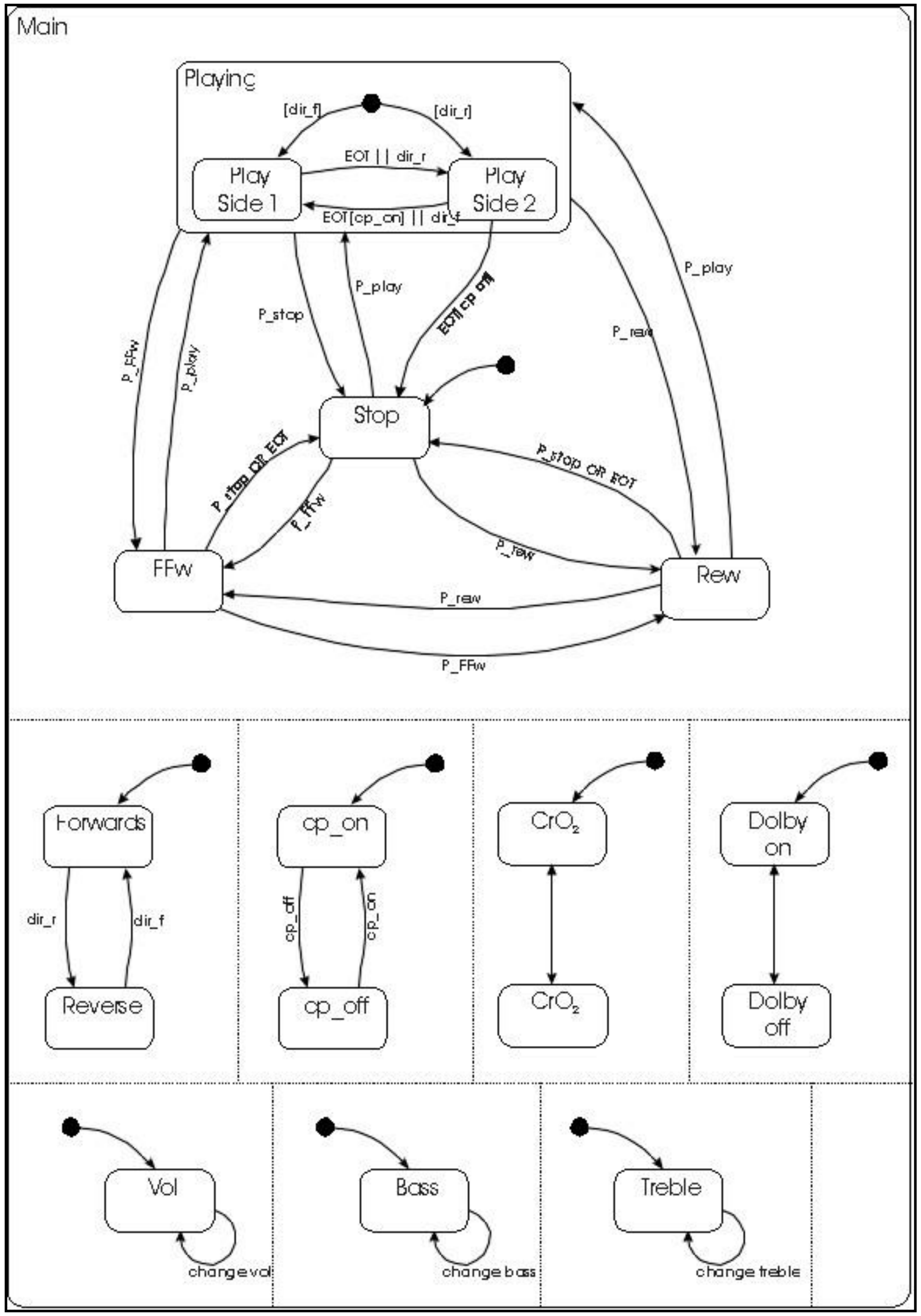

Symbols on back
once
continuous
Switch to toggle between playing continuously and play both sides once.(repeat)
Fast-forward (f) button referred to in diagrams.
Rewind-button (r) in the diagrams
View of the cassette control area of an Aiwa walkman with auto-reverse.
STOP
PLAY
FWD
REV
Switch for changing the side being played, event cs
Stop (s)
Play (p)

A common problem

- It is often the case that conventional finite state machines blow-up in size for big problems, in two senses
  - The actual description of the machine can get very large
  - The state space represented by the machine can get to be enormous
- This is especially true for
  - deterministic machines (which are usually desirable) and
  - machines capturing concurrency (because of the potential interleavings that must be captured)

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
- Now part of UML



Main
Playing
[dir_f]
[dir_r]
Play Side 1
Play Side 2
EOT || dir_r
EOT[cp_on] || dir_f
P_stop
P_play
EOT[cp off]
P_FFw
P_play
P_rew
P_play
Stop
P_stop OR EOT
P_FFw
P_stop OR EOT
P_rew
FFw
Rew
P_rew
P_FFw
Forwards
Reverse
dir_r
dir_f
cp_on
cp_off
cp_off
cp_on
CrO_2
CrO_2
Dolby on
Dolby off
Vol
change vol
Bass
change bass
Treble
change treble

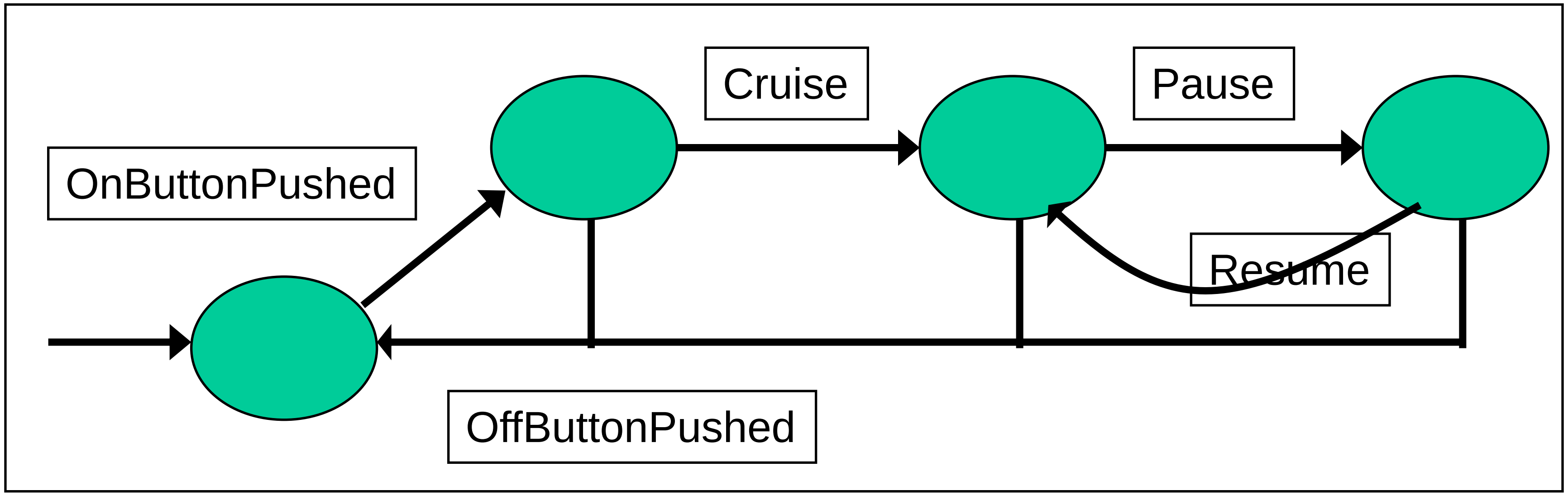
Tools

- Statecharts have a set of supporting tools from i-Logix (STATEMATE, Rhapsody)
  – Editors
  – Simulators
  – Code generators
    - C, Ada, Verilog, VHDL
  – Some analysis support
- UML tools and environments…

Classic examples

- Specifying a cruise control
- Specifying the traffic lights at an intersection
- Specifying trains on shared tracks
  - Could be managing the bus tunnel in Seattle
- Etc.

A snippet of cruise control



- Completely incomplete
  - There should be guards and conditions on transitions
  - Lots of other information left out

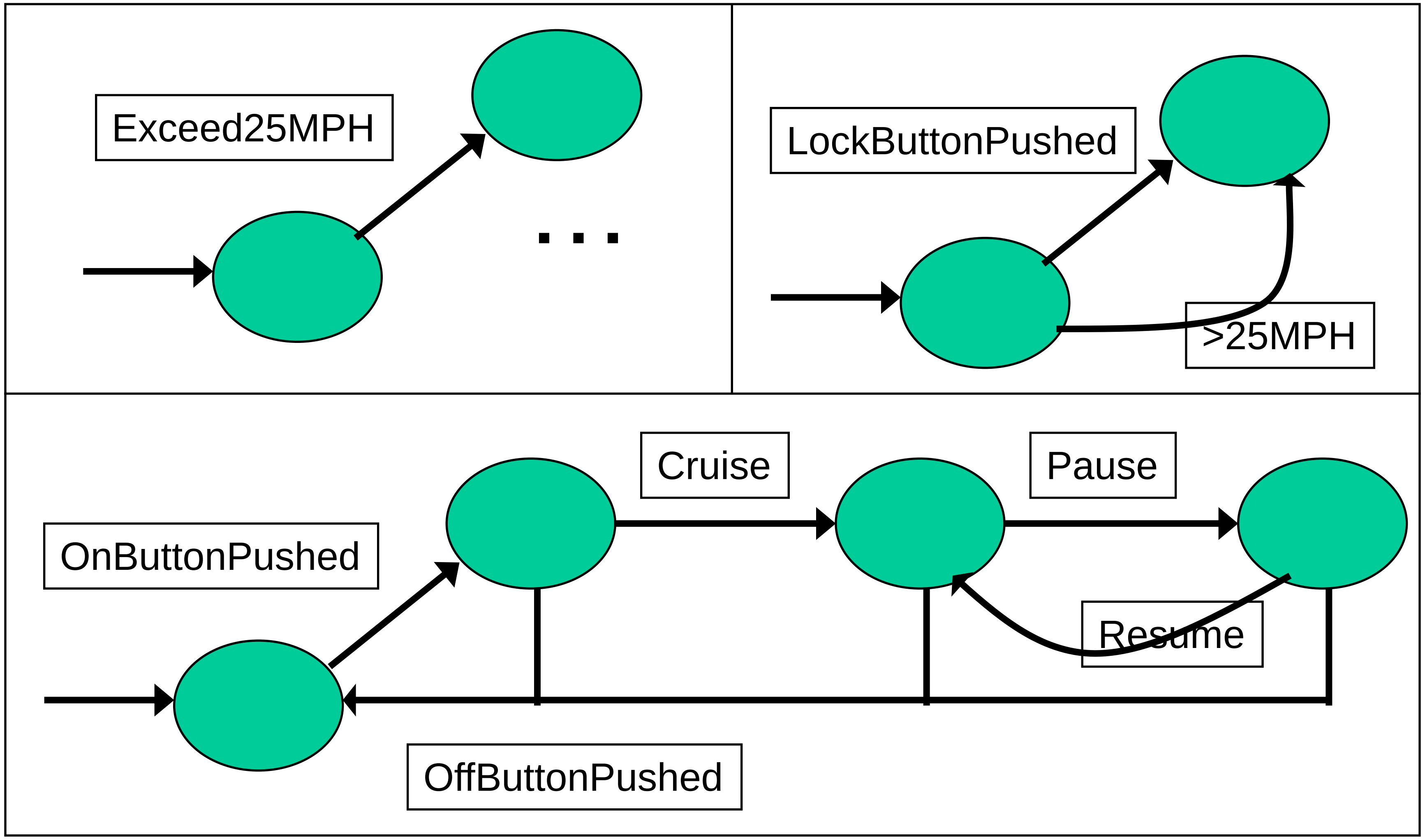
More cruise control

- What if your state machine also tracked speed?
  - Maybe the cruise control doesn’t work at low speeds
  - Anyway, it needs to remember a speed so it can resume properly
- What if it also interacted with the door locking system?
- You might have to modify almost every state to track not only the state on the previous slide, but the speed, too
  - Essentially, you need to build a cross product of all combinations of states
- This is the kind of issue that can cause the machine to blowup in size
  - It’s not the best example, but it’s adequate

Statecharts

- The idea of statecharts [Harel] is to provide a rich, visual representation for defining finite state machines that capture the essence of complex, reactive systems
- There isn’t a simple, easy-to-get, reference
  - “The” statecharts paper, but long and a bit hard to find
    D. Harel, "Statecharts: A Visual Formalism for Complex Systems, " Science of Computer Programming (1987)
  - A general paper on statechart-like formalisms
    D. Harel. "On Visual Formalisms," Comm. of the ACM (1988)

Key idea: hierarchy



Parallel AND-machines

- The state of the overall machine is represented by one state from each of the parallel AND machines
  – In a cruise control state AND in a speed state AND in a door lock state
- Transitions can take place in all substates in parallel
  – Events in one substate can cause transitions in another substate

A few statechart features

- Default entry states for each substate
  - Indicated by an arrow with no initial state
- When any of the parallel machines is exited, the entire machine is exited
- You can have “history” states, which remember where you were the last time you were in a machine
- The “STATEMATE semantics” are the standard semantics
  - This is largely a question of which enabled transitions are taken, and when
  - At this level, you surely don’t care

Leap of faith

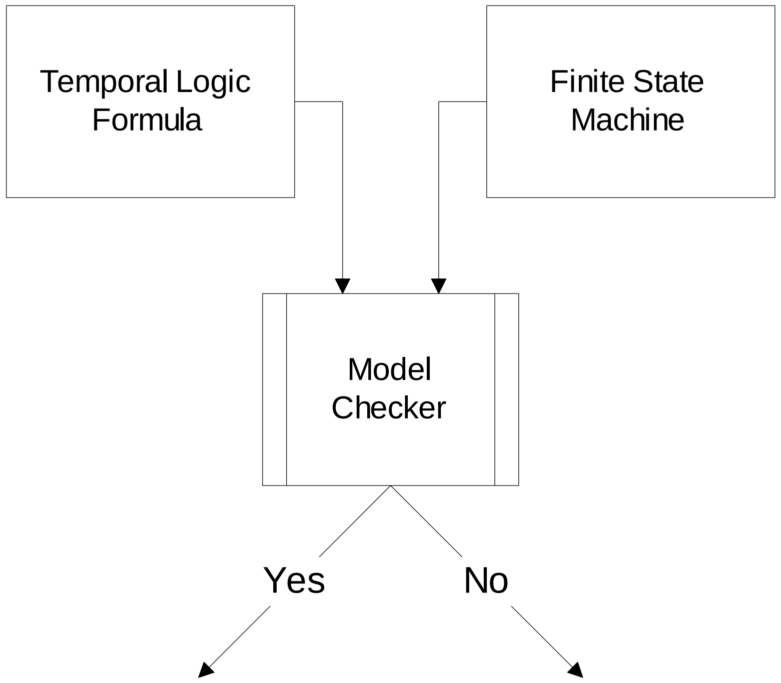
- Statecharts (and variants) can be used to specify important, complex systems
- (Not all software-based systems, nor all aspects of many software-based systems)

Question

- So we have a big statecharts-like specification
- How do we know it has properties we want it to have?
  - Ex: is it deterministic?
  - Ex: can you ever have the doors unlock by themselves while the car is moving?
  - Ex: can you ever cause an emergency descent when you are under 500 feet above ground level?
- Three main approaches
  - Human inspection
  - Simulation
  - Analysis: the most promising of these is model checking

Model checking

- Evaluate temporal properties of finite state systems
  - Guarantee a property is true or return a counterexample
  - Ex: Is it true that we can never enter an error state?
  - Ex: Are we able to handle a reset from any state?
- Extremely successfully for hardware verification
  - Intel got into the game after the FDIV error
- Open question: applicable to software specifications?



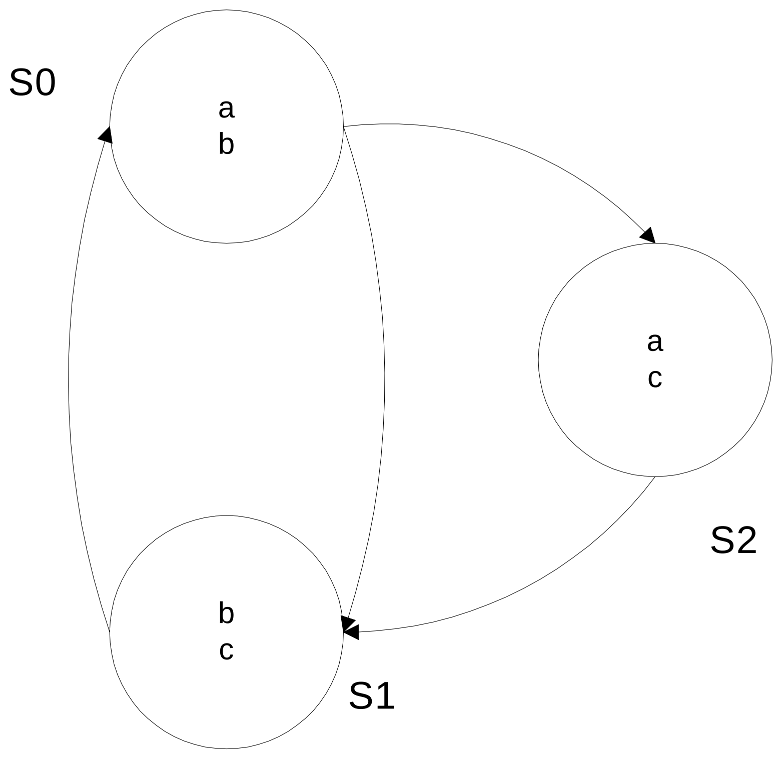
State Transition Graph

- One way to represent a finite state machine is as a state transition graph
  - S is a finite set of states
  - R is a binary relation that defines the possible transitions between states in S
  - P is a function that assigns atomic propositions to each state in S (e.g., that a specific process holds a lock)
- Other representations include regular expressions, etc.

Example

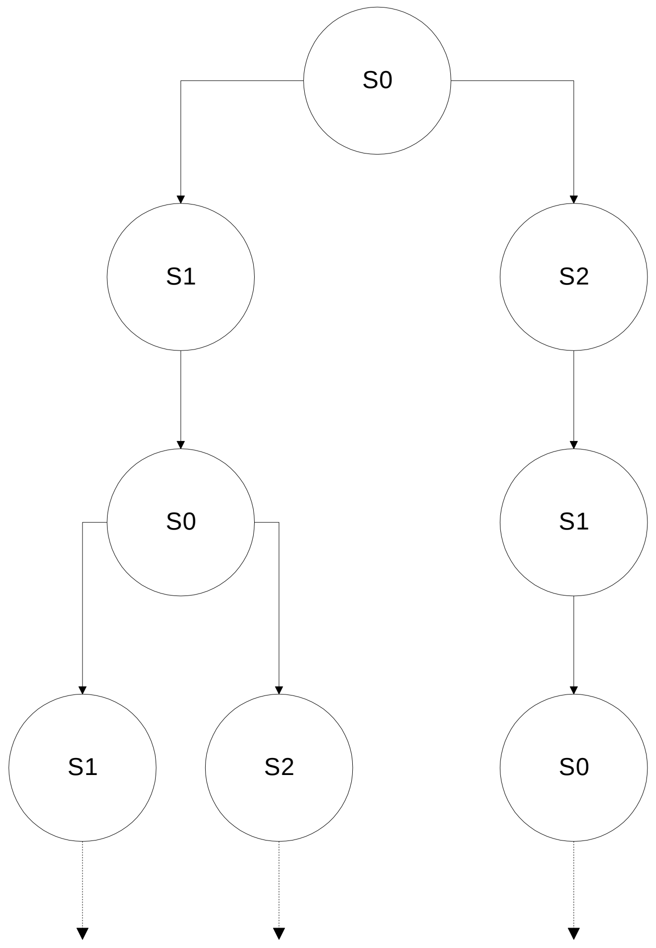
- Three states
- Transitions as shown
- Atomic properties a, b and c

- Given a start state, you can consider legal paths through the state machine



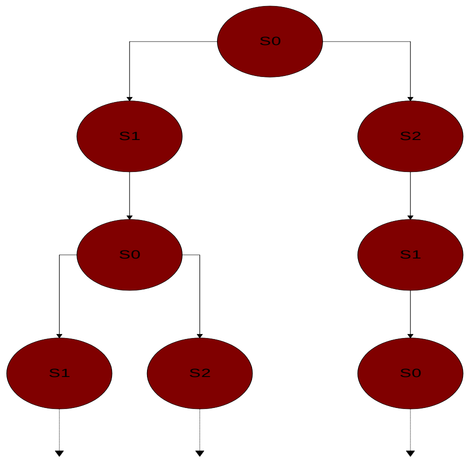
A computation tree

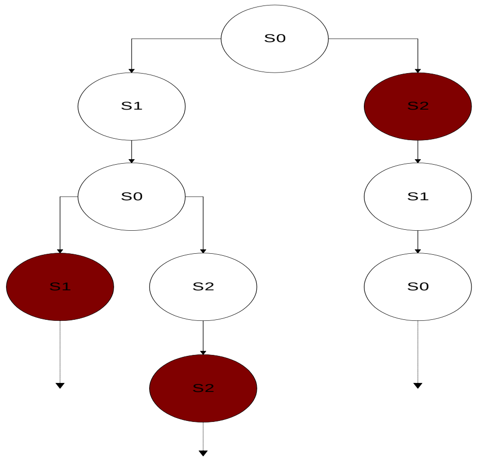
- From a given start state, you can represent all possible paths with an infinite computation tree
- Model checking allows us to answer questions about this tree structure



Temporal formulae

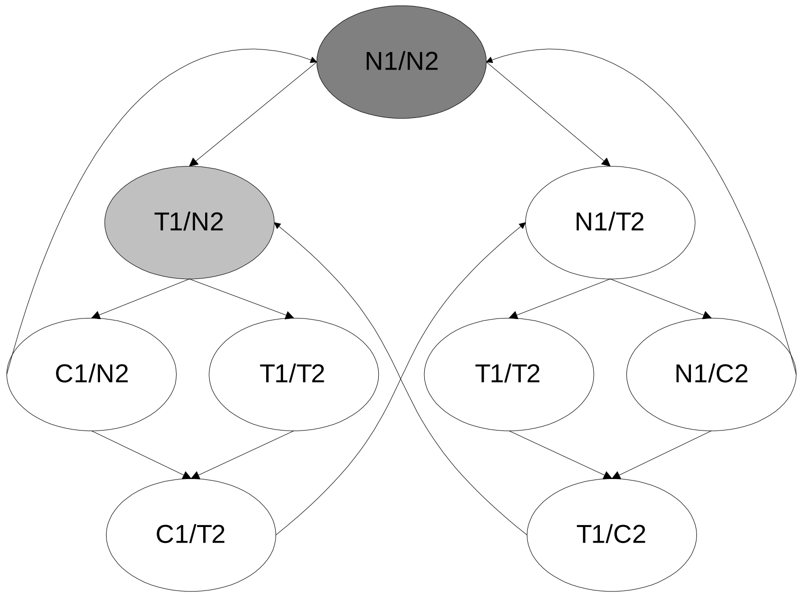
- Temporal logics allow us to say things like
  - Does some property hold true globally?
    - Top figure
  - Does some property inevitably hold true?
    - Bottom figure
  - Does some property potentially hold true?





Mutual exclusion example

- N1 & N2, non-critical regions of Process 1 and 2
- T1 & T2, trying regions
- C1 and C2, critical regions
- AF(C1) in lightly shaded state?
  - C1 always inevitably true?
- EF(C1 $\wedge$ C2) in dark shaded state?
  - C1 and C2 eventually true?



How does model checking work? (in brief!)

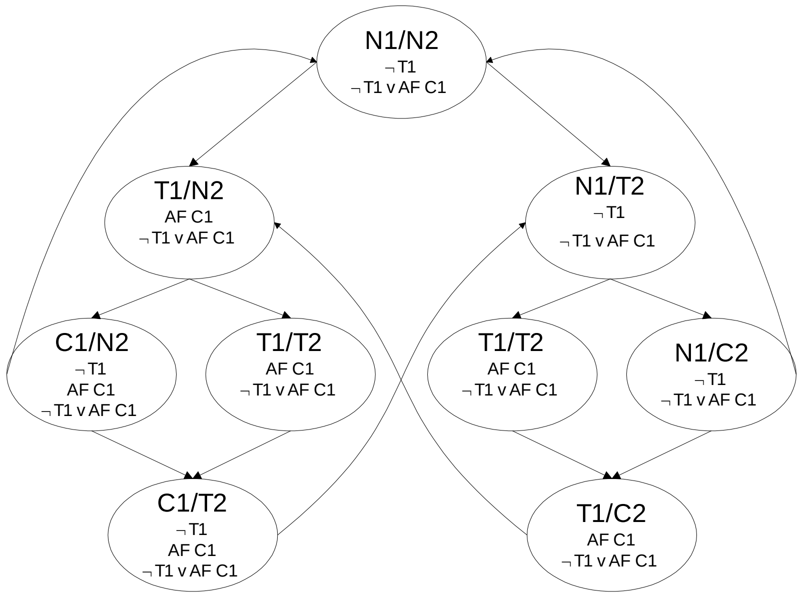
- An iterative algorithm that labels states in the transition graph with formulae known to be true
- For a query Q
  - the first iteration marks all subformulae of Q of length 1
  - the second iteration marks them of length 2
  - this terminates since the formula is finite
- The details of the logic indeed matter
  - But not at this level of description

Example

- $Q = T1 \Rightarrow AF\ C1$
  - If Process 1 is trying to acquire the mutex, then it is inevitably true it will get it sometime
- $Q = \neg T1 \vee AF\ C1$
  - Rewriting with DeMorgan's Laws
- First, label all the states where T1, $\neg$T1, and C1 are true
  - These are atomic properties

Example

- Next mark all the states in which AF C1 is true, etc.
  - The algorithm tracks states visited using depth-first search
  - Slight variations for AF, AG, EF, EG, etc.
- At termination, $\neg T1 \vee AF\ C1$ is true everywhere
  - Hence the temporal property is true for the state machine



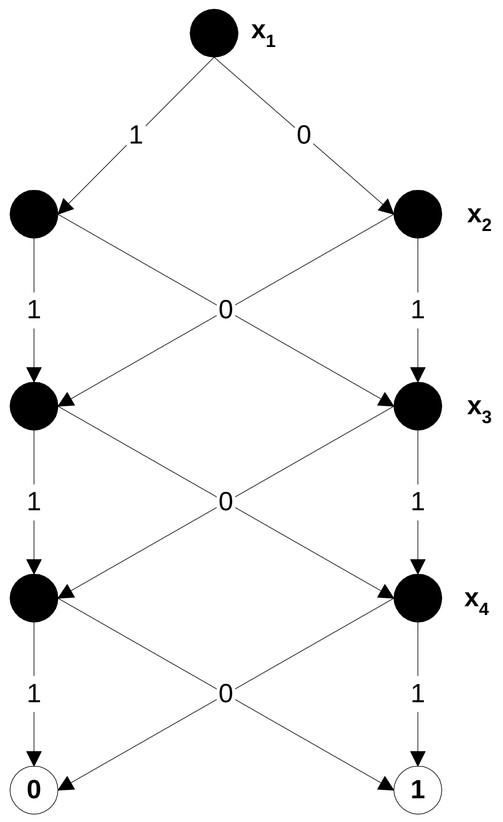
Symbolic model checking

- State space can be huge ($>2^{1000}$) for many systems
- Key idea: use implicit representation of state space
  - Data structure to represent transition relation as a boolean formula
- Algorithmically manipulate the data structure to explore the state space
- Key: efficiency of the data structure

Binary decision diagrams (BDDs)

- "Folded decision tree"
- Fixed variable order
- Many functions have small BDDs
  - Multiplication is a notable exception
- Can represent
  - State machines (transition functions) and
  - Temporal queries

Due to Randy Bryant



Odd Parity

BDD-based model checking

- Iterative, fixed-point algorithms that are quite similar to those in explicit model checking
- Applying boolean functions to BDDs is efficient, which makes the underlying algorithms efficient
  - $\wedge$ becomes set intersection ($\cap$), $\vee$ becomes set union ($\cup$), etc.
- When the BDDs remain small, that is
  - Variable ordering is a key issue

BDD-based successes in HW

- IEEE Futurebus+ cache coherence protocol
- Control protocol for Philips stereo components
- ISDN User Part Protocol
- But what about software?
  - Software is often specified with infinite state descriptions
  - Software specifications may be structured differently from hardware specifications
    - Hierarchy
    - Representations and algorithms for model checking may not scale

Our approach at UW—try it!

- Applied model checking to the specification of TCAS II
  - Traffic Alert and Collision Avoidance System
    - In use on U.S. commercial aircraft
  - Issue resolution advisories only
    - Vertical resolution only
    - Relies on transponder data
    - http://www.faa.gov/and/and600/and620/newtcas.htm
  - FAA adopted specification
  - Initial design and development by Leveson et al.
- Later applied it to a statecharts description of an electrical power distribution system model of the B777

Interlude: people make mistakes
Gerard Holzmann

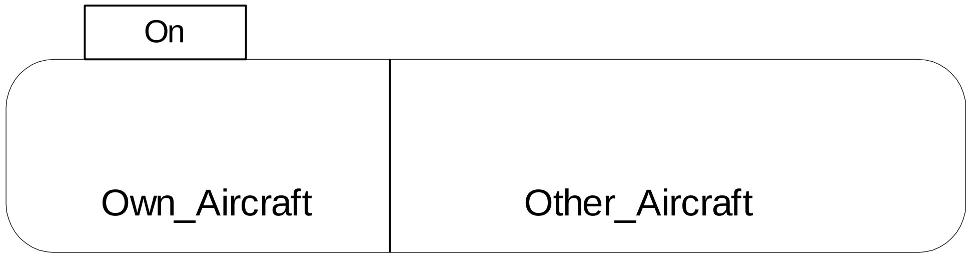
- Average software tends to have about 1-10 residual defects for every 1,000 lines of non-comment code (defects found *after* testing)
- An average issue of ***New York Times*** has about 10,000 sentences **+** about 10 corrections to the preceding issue → about 1 *detected* mistake per 1,000 (proofread) sentences

- What are the consequences of this on software engineering?

TCAS specification

- Irvine Safety Group (Leveson et al.)
  - Specified in RSML as a research project
    - RSML is in the Statecharts family of hierarchical state machine description languages
  - FAA adopted RSML version as official
- Specification is about 400 pages long
- This study uses: Version 6.00, March 1993
  - Not the current FAA version

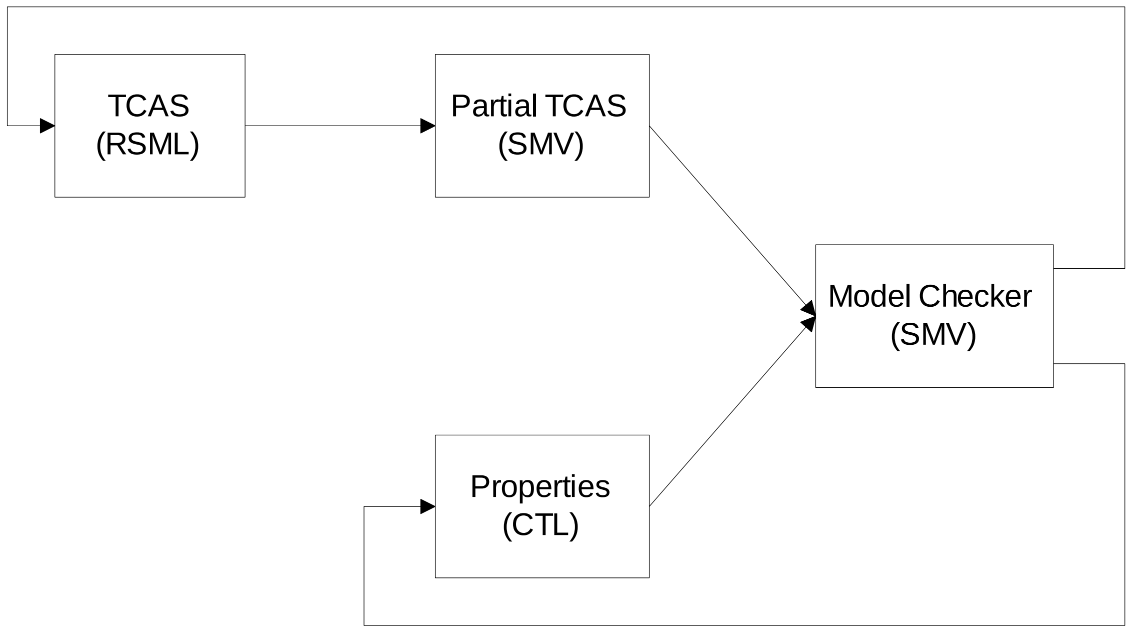
TCAS—high-level structure



- Own_Aircraft
  - Sensitivity levels, Alt_Layer, Advisory_Status
- Other_Aircraft
  - Tracked, Intruder_State, Range_Test, Crossing, Sense Descend/Climb

Using SMV

- SMV is a BDD-based model checker
- It checks CTL formulas
  - A specific temporal logic



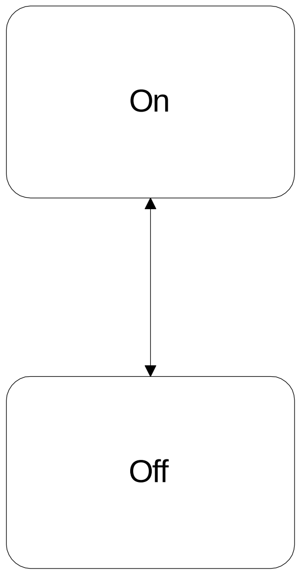
Iterative process

- Iterate SMV version of specification
- Clarify and refine temporal formula
- Model environment more precisely
- Refine specification

Use of non-determinism

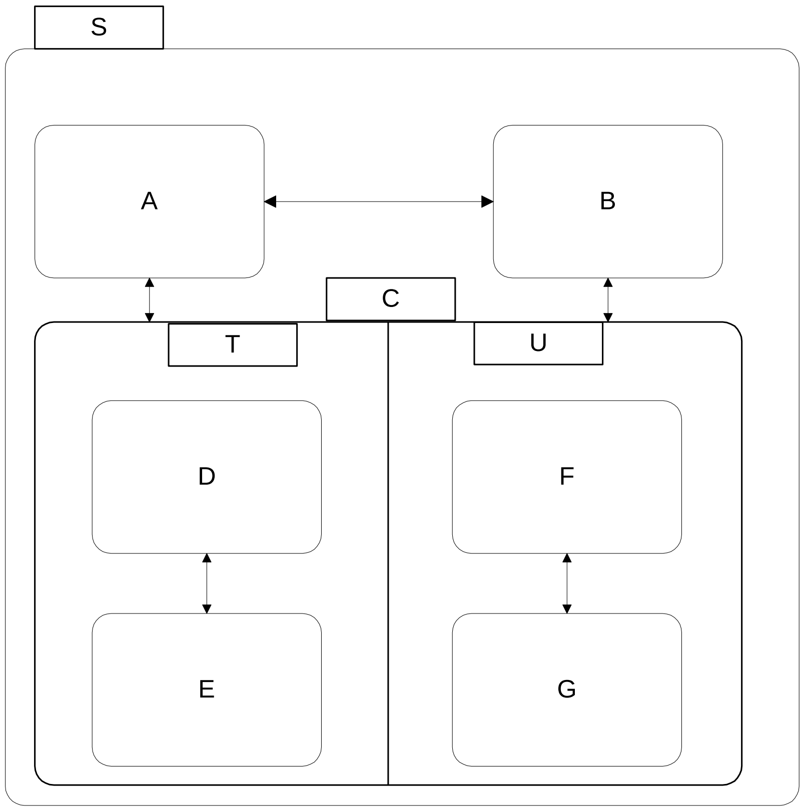
- Inputs from environment
  - **`Altitude := {1000…8000}`**
- Simplification of functions
  - **`Alt_Rate := 0.25*(Alt_Baro-ZP)/Delta_t`**
  - **`Alt_Rate := {-2000…2000}`**
- Unmodelled parts of specification
  - States of Other_Aircraft treated as non-deterministic input variables

Translating RSML to SMV



```
MODULE main
   VAR
   state:{ON,OFF};
   on_event: boolean;
   off_event: boolean;
ASSIGN
   init(state)  := OFF;
   next(state)  := case
                        state = ON &
                     off_event: OFF;
                 state = OFF &
                     on_event: ON;
                        1 : state;
   esac;
```

State encoding



- Flatten nested AND and nested OR states
- One variable for each OR state
  - An enumerated type of the alternatives

- **VAR**
```
    S:  {A,B,C};
    T:  {D,E};
    U:  {F,G};
```

Events

- External—interactions with environment
- Internal—micro steps
- Synchrony hypothesis
  - External event arrives
  - Triggers cascade of internal events (micro steps)
  - Stability reached before next external event
- Technical issues with micro steps

Non-deterministic transitions

- A machine is deterministic if at most one of **`T_A_B`**, **`T_A_C`**, etc. can be true
  - **`T_A_B`** represents the conditions under which a transition is taken from state A to state B
  - Else non-deterministic

Checking properties

- Initial attempts to check any property generated BDDs of over 200MB
- First successful check took 13 hours
  - Was reduced to a few minutes
- Partitioned BDDs
- Reordered variables
- Implemented better search for counterexamples

Property checking

- Domain independent properties
  - Deterministic state transitions
  - Function consistency
- Domain dependent
  - Output agreement
  - Safety properties
- We used SMV to investigate some of these properties on TCAS' **`Own_Aircraft`** module

Deterministic transitions

- Do the same conditions allow for non-deterministic transitions?
- Inconsistencies were found earlier by other methods [Heimdahl and Leveson]
  - Identical conditions allowed transitions from Sensitivity Level 4 to SL 2 or to SL 5
- Our formulae checked for all possible non-determinism; we found this case, too

```
V_254a := MS = TA_RA | MS = TA_only | MS =3 | MS = 4 |
          MS = 5 | MS = 6 | MS = 7;
V_254b := ASL = 2 | ASL = 3 | ASL = 4 | ASL = 5 |
          ASL = 6 | ASL = 7;
T_254  := (ASL = 2 & V_254a) | (ASL = 2 & MS = TA_only) |
          (V_254b & LG = 2 & V524a);
V_257a := LG = 5 | LG = 6 | LG = 7 | LG = none;
V_257b := MS = TA_RA | MS = 5 || MS = 6 | MS = 7;
V_257c := MS = TA_RA | MS = TA_only | MS = 3 | MS = 4 |
          MS = 5 | MS = 6 | MS = 7;
V_257d := ASL = 5 | ASL = 6 | ASL = 7;
T_257  := (ASL = 5 | V_257a | V_257b) |
          (ASL = 5 & MS = TA_only) |
          (ASL = 5& LG = 2 & V_257c) |
          (V_257d & LG = 5 & V_257b) |
          (V_257d & V_257a & MS = 5);
```

Function consistency

- Many functions are defined in terms of cases
- A function is inconsistent if two different conditions C_i and C_j and be true simultaneously

$$F = \begin{cases} V_1 \text{ if } C_1 \\ V_2 \text{ if } C_2 \\ V_3 \text{ if } C_3 \end{cases}$$

$$\begin{aligned} \text{AG } !((C_1 \ \&\ C_2) \ | \\ (C_1 \ \&\ C_2) \ | \\ (C_2 \ \&\ C_3)) \end{aligned}$$

$$
\text{Displayed\_Model\_Goal} =
\begin{cases}
0 & \textbf{if } \text{Composite\_RA } \textbf{not in state } \text{Positive} \\[2ex]
\begin{array}{l}\textbf{Max}(\text{Own\_Track\_Alt\_Rate},\\ \text{PREV(Displayed\_Model\_Goal)},\\ 1500 \text{ ft/min})\end{array} & \begin{array}{l}\textbf{if } (\text{New\_Climb } \textbf{or } \text{New\_Threat}) \textbf{ and}\\ \ \textbf{not } \text{New\_Increase\_Climb } \textbf{and}\\ \ \textbf{not } (\text{Increase\_Climb\_Cancelled } \textbf{or}\\ \text{Increase\_Descend\_Cancelled}) \textbf{ and}\\ \text{Composite\_RA } \textbf{in state } \text{Climb}\end{array} \\[2ex]
\begin{array}{l}\textbf{Min}(\text{Own\_Track\_Alt\_Rate},\\ \text{PREV(Displayed\_Model\_Goal)},\\ -1500 \text{ ft/min})\end{array} & \begin{array}{l}\textbf{if } (\text{New\_Descend } \textbf{or } \text{New\_Threat}) \textbf{ and}\\ \ \textbf{not } \text{New\_Increase\_Descend } \textbf{and}\\ \ \textbf{not } (\text{Increase\_Climb\_Cancelled } \textbf{or}\\ \text{Increase\_Descend\_Cancelled}) \textbf{ and}\\ \text{Composite\_RA } \textbf{in state } \text{Descend}\end{array} \\[2ex]
2500 \text{ ft/min} & \textbf{if } \text{New\_Increase\_Climb} \\[2ex]
-2500 \text{ ft/min} & \textbf{if } \text{New\_Increase\_Descend} \\[2ex]
\begin{array}{l}\textbf{Max}(\text{Own\_Track\_Alt\_Rate},\\ 1500 \text{ ft/min})\end{array} & \begin{array}{l}\textbf{if } \text{Increase\_Climb\_Cancelled } \textbf{and}\\ \ \textbf{not } \text{New\_Increase\_Climb } \textbf{and}\\ \text{Composite\_RA } \textbf{in state } \text{Positive}\end{array} \\[2ex]
\begin{array}{l}\textbf{Min}(\text{Own\_Track\_Alt\_Rate},\\ -1500 \text{ ft/min})\end{array} & \begin{array}{l}\textbf{if } \text{Increase\_Descend\_Cancelled } \textbf{and}\\ \ \textbf{not } \text{New\_Increase\_Descend } \textbf{and}\\ \text{Composite\_RA } \textbf{in state } \text{Positive}\end{array} \\[2ex]
\text{P{\scriptsize REV}(Displayed\_Model\_Goal)} & \text{Otherwise}
\end{cases}
$$

Display_Model_Goal

- Tells pilot desired rate of altitude change
- Checking for consistency gave a counterexample
  - **`Other_Aircraft`** reverse from an **`Increase-Climb`** to an **`Increase-Descend`** advisory
  - After study, this is only permitted in our non-deterministic modeling of **`Other_Aircraft`**
  - Modeling a piece of **`Other_Aircraft`**'s logic precludes this counterexample

Output agreement

- Related outputs should be consistent
  - Resolution advisory
    - **`Increase-Climb, Climb, Descend, Increase-Descend`**
  - **`Display_Model_Goal`**
    - Desired rate of altitude change
    - Between -3000 ft/min and 3000 ft/min
  - Presumably, on a climb advisory, **`Display_Model_Goal`** should be positive

Output agreement check

- `AG (RA = Climb ⇒ DMG > 0)`
  - If **`Resolution Advisory`** is **`Climb`**, then **`Display_Model_Goal`** is positive
- Counterexample was found
  - `t0 : RA = Descend, DMG = -1500`
  - `t1 : RA = Increase-Descend,`
    `DMG = -2500`
  - `t2 : RA = Climb, DMG = -1500`

Limitations

- Can’t model all of TCAS
  - Pushing limits of SMV (more than 200 bit variables is problematic)
  - Need some non-linear arithmetic to model parts of **`Other_Aircraft`**
    - New result that represents constraints as BDD variables and uses a constraint solver
- How to pick appropriate formulae to check?

Whence formulae?

- “There have been two pilot reports received which indicated that TCAS had issued Descend RA's at approximately 500 feet AGL even though TCAS is designed to inhibit Descent RAs at 1,000 feet AGL. All available data from these encounters are being reviewed to determine the reason for these RAs.”
  –TCAS web

Whence formulae?

- Jaffe, Leveson et al. developed criteria that specifications of embedded real-time systems should satisfy, including:
  - All information from sensors should be used
  - Behavior before startup, after shutdown and during off-line processing should be specified
  - Every state must have a transition defined for every possible input (including timeouts)
    - Predicates on the transitions must yield deterministic behavior
- Essentially a check-list, but a very useful one

Model checking wrap up

- The goal of model checking is to allow finite state descriptions to be analyzed and shown to have particular desirable properties
  - Won’t help when you don’t want or need finite state descriptions
  - Definitely added value when you do, but it’s not turnkey yet
    - There’s still a real art in managing model checking
  - Definitely feasible on modest sized systems
  - More later on applications to code

I know this was quick

- My goal isn’t to make you into model checking experts
  - But it might titillate one or two of you to learn more
- But rather to understand the sketches of what model checking is and why it is so promising for checking some classes of specifications

Michael Jackson: The World and the Machine

- Wikipedia lists the following well-known people named Michael Jackson
- Janet Jackson’s brother
- Record producer, known for some Kiss albums
- National Football League linebacker, 1979-1986
- Sacramento Kings player, 1987–1990
- English football (soccer) player
- National Football League wide receiver, 1991–1998
- Major League Baseball relief pitcher
- Beer Hunter show host, beer and whisky expert
- Radio talk show host, interviewer, KNX, Los Angeles
- British television executive
- Child actor who appeared in 1976 musical film Bugsy Malone
- U.S. Deputy Secretary, 2005
- Republican candidate in the 2003 California recall Gubernatorial election
- Bishop of Clogher, 2002–
- Professor of social anthropology and writer
- ***Developer of software development methods***

- “Action”!

Good night

- Let’s leave discussion to the wiki