Presentation for 4th International Workshop on Formal Methods for Interactive Systems: FMIS 2011, Limerick, Ireland, 21 June 2011

Formal Modeling and Analysis For Interactive Hybrid Systems

Ellen J. Bass

Systems and Information Engineering, University of Virginia

Karen M. Feigh

School of Aerospace Engineering, Georgia Institute of Technology

Elsa Gunter

Department of Computer Science, University of Illinois, Urbana-Champaign

John Rushby

Computer Science Laboratory, SRI International, Menlo Park, California

Supported by NSF grant CNS-0720908 and NASA contract NNA10DE79C

Premise

- Human interactions with automated systems are guided by mental models (Craik 1943)
- Exact nature of the models is a topic of debate and research
  - Behavioral representation that allows mental simulation
    - ⋆ e.g., state machine
  - Stimulus/response rules
  - Both

  We'll assume the first of these
- An automation surprise can occur when the behavior of the real system and the mental model diverge
- Can discover potential surprises by model checking
  - Build state machines for the system and its model, explore all possible behaviors looking for significant divergences
- This works! (Rushby 1997)

Mental Models

- Aviation psychologists elicit pilot's actual mental models
- However, a well-designed system should induce an effective model, and the purpose of training is to develop this
- So can construct plausible mental models by extracting state machines from training material, then applying known psychological simplification processes (Javaux 1998)
  - Frequential simplification
  - Inferential simplification
- But there are some basic properties that should surely be true of any plausible mental model
  - e.g., pilots can predict whether their actions will cause the plane to climb or descend
- Yet many avionics systems are so poor that they provoke an automation surprise even against such core models
- We will use models of this kind

System Models

- The real system will have many parts, and possibly complex internal behavior
- But there is usually some externally visible physical plant
  - e.g., a car, airplane, vacuum cleaner, iPod
- And what humans care about, and represent in their mental models, is the behavior of the plant
- And divergence between a mental model and the real system should be in terms of this plant behavior
  - e.g., does the car or plane go in the right direction, does the vacuum cleaner use the brush or the hose, does the iPod play the right song?
- So our analysis should model the plant behavior
- Did not do this previously, just the plant controller

Hybrid Systems

- Many plants are modelled by differential equations
  - e.g., 6 DOF models for airplanes
- Compounded by different sets of equations in different discrete modes
  - e.g., flap extension
- These models are called hybrid systems
  - Combine discrete (state machine) and continuous (differential equation) behavior
- The full system model will be the composition of the hybrid plant model with its controller and its interface and. . .
- Can do accurate simulations (e.g., Matlab)
- But that's just one run at a time, we need all runs
- And formal analysis of hybrid systems is notoriously hard

Relational Abstractions

- We need to find suitable abstractions (i.e., approximations) for hybrid systems that are sufficiently accurate for our purposes, and are easy to analyze
- Several abstractions available for hybrid systems, we use a very recent kind called relational abstractions (Tiwari 2011)
- For each discrete mode, instead of differential equations to specify evolution of continuous variables, give a relation between them that holds in all future states (in that mode)
- Accurate relational abstractions for hybrid systems require specialized invariant generation and eigenvalue analysis
- But for our purposes, something much cruder suffices
  - e.g., if pitch angle is positive, then altitude in the future will be greater than it is now
- Rather than derive these rel'ns, we assert them as our spec'n

Model Checking Infinite State Systems

- Our relational abstractions get us from hybrid systems back to state machines
- But these state machines are still defined over continuous quantities (i.e., mathematical real numbers)
  - Altitude, roll rate, etc.
- How do we model check these?
  - i.e., do fully automatic analysis of all reachable states
  - When there's potentially an infinite number of these
- We can do it by Bounded Model Checking (BMC) over the theories decided by a solver for Satisfiability Modulo Theories (SMT)
  - This is infinite BMC

SMT Solvers: Disruptive Innovation in Theorem Proving

- SMT solvers extend decision procedures with the ability to handle arbitrary propositional structure
  - Previously, case analysis was handled heuristically or interactively in a front end theorem prover
    - ⋆ Where must be careful to avoid case explosion
  - SMT solvers use the brute force of modern SAT solving
- Or, dually, they generalize SAT solving by adding the ability to handle arithmetic and other decidable theories
- Typical theories: uninterpreted functions with equality, linear arithmetic over integers and reals, arrays of these, etc.
- There is an annual competition for SMT solvers
- Very rapid growth in performance
- Biggest advance in formal methods in last 25 years

Bounded Model Checking (BMC)

- Given system specified by initiality predicate I and transition relation T on states S
- Is there a counterexample to property P in k steps or less?
- i.e., can we find an assignment to states $s_0, \ldots, s_k$ satisfying

$$I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg(P(s_1) \wedge \cdots \wedge P(s_k))$$

- Try for $k = 1, 2, \ldots$
- Given a Boolean encoding of I, T, and P (i.e., circuits), this is a propositional satisfiability (SAT) problem
- If I, T, and P are over the theories decided by an SMT solver, then this is an SMT problem
  - Then called Infinite Bounded Model Checking (inf-BMC)
- Works for LTL (via Büchi automata), not just invariants
- Extends to verification via k-induction

Specifying Relations

- Most model checking notations specify state variables of new state in terms of those in the old; may be nondeterministic
- For example, guarded command in SAL
  - `pitch > 0 --> alt' IN {x: REAL | x > alt}`

  If `pitch` is positive, new value of `alt` is bigger than old one
- But how do we say that `x` and `y` get updated such that
  - `x*x + y*y < 1` ?
- Various possibilities, depending on the model checker, but one way that always works is to use a synchronous observer
- Main module makes nondeterministic assignments to `x` and `y`
- Observer module sets ok false if relation is violated
  - `NOT(x*x + y*y < 1) --> ok' = FALSE`
- Model check for the property we care about only when ok is true: `G(ok IMPLIES property)`

Example: Airbus Speed Protection

- Systems similar to that described below were used in A310, A320, A330, and A340 airplanes; this is the A320 version
- Autothrottle modes
  - SPD: try to maintain speed set in the FCU
- Autopilot vertical modes and submodes
  - VS/FPA: fly at the fight path angle specified in the FCU
  - OP CLB: climb toward target altitude set in the FCU, using max thrust at the FPA that maintains set airspeed
  - OP DES: ...if target altitude is lower than current
- Speed protection
  - On descent in SPD VS/FPA modes, allow overspeed
  - But if it exceeds the MAX, change to OP mode
  - Will be OP CLB if target altitude is above current
  - MAX speed is lower when flaps are extended

Modeling Airbus Speed Protection

- Composition of three main components
  - Pilots: nondeterministically set vertical mode, dial values into FCU, deploy flaps
    - ⋆ Organized by mental mode (descend, climb, level)
  - Automation: determines actual mode and applies control laws to determine thrust and pitch
  - Airplane: uses thrust and pitch values, and flap setting, to calculate airplane trajectory (altitude and airspeed)
- Plus constraints, which is an observer that sets `ok` to enforce plausible relations among pitch, altitude, etc.
- And observer, which sets `alarm` if airplane climbs while mental mode is `descend`
- Model check for `G(ok IMPLIES NOT alarm)`

Fragment of Pilots Module

```
INPUT
  airspeed:    speedvals,    altitude:    altvals
INITIALIZATION
  mental_mode = level;  fcu_mode = other;   flaps = retracted;
TRANSITION
[  extend_flaps: mental_mode = descend and flaps = retracted -->
    flaps' = extended
[] retract_flaps: mental_mode = climb and flaps = extended -->
    flaps' = retracted
[] dial_fcu_alt: fcu_mode = other --> fcu_alt' IN {x: altvals | TRUE}
[] dial_descend: mental_mode /= descend -->
    mental_mode' = descend;     fcu_mode' = vs_fpa;
    fcu_fpa' IN {x: pitchvals | x < 0};
[] dial_climb: mental_mode /= climb -->
    mental_mode' = climb;    fcu_mode' = vs_fpa;
    fcu_fpa' IN {x: pitchvals | x > 0};
[] pilots_idle: TRUE -->
] END;
```

Fragment of Automation Module

```
DEFINITION
  max_speed = IF flaps = retracted THEN VMAX ELSE Vfe ENDIF;
TRANSITION
[  track-fcu-mode: fcu_mode' /= fcu_mode --> actual_mode' = fcu_mode'
[] mode_reversion: actual_mode = vs_fpa AND airspeed > max_speed -->
    actual_mode' = IF fcu_alt > altitude THEN op_clb ELSE op_des ENDIF;
[] vs_fpa_mode: actual_mode = vs_fpa AND airspeed <= max_speed -->
    pitch' IN vs_fpa_pitch_law(...)
[] op_clb_mode: actual_mode = op_clb --> pitch' IN op_clb_pitch_law(...)
[] op_des_mode: actual_mode = op_des --> pitch' IN op_des_pitch_law(...)
[] automation_idles: ELSE -->
] END;
```

Fragment of Airplane Module

```
INITIALIZATION
  airspeed = 200;       altitude = 3000;
TRANSITION
[  flying_clean: flaps = retracted -->
    airspeed’ IN
      speed_dynamics_clean(airspeed, altitude, thrust, pitch);
    altitude’ IN alt_dynamics_clean(...);
[] flying_flaps: flaps = extended -->
    airspeed’ IN speed_dynamics_flaps(...);
    altitude’ IN alt_dynamics_flaps(...);
] END;
```

Fragment of Constraints Module

```
INITIALIZATION
   ok = TRUE;
TRANSITION
[  actual_mode = op_des AND pitch > 0 --> ok' = FALSE;
[] actual_mode = op_clb AND pitch < 0 --> ok' = FALSE;
[] actual_mode = vs_fpa AND fcu_fpa <= 0 AND pitch > 0 --> ok' = FALSE;
[] actual_mode = vs_fpa AND fcu_fpa >= 0 AND pitch < 0 --> ok' = FALSE;
[] pitch > 0 AND altitude' < altitude --> ok' = FALSE;
[] pitch < 0 AND altitude' > altitude --> ok' = FALSE;
[] pitch=0 AND altitude' /= altitude --> ok' = FALSE;
[] ELSE -->
] END;
```

Observer Module

```
observer: MODULE =
BEGIN
 OUTPUT
  alarm: BOOLEAN
 INPUT
  mental_mode: mental_modes,   altitude:    altvals
 INITIALIZATION
   alarm = FALSE
 TRANSITION
   alarm' = alarm OR (mental_mode = descend AND altitude' - altitude > 90)
END;
```

The System, the Property, the Analysis

```
system: MODULE = airplane || automation || pilots || constraints || observer;

surprise: THEOREM system |- G(ok IMPLIES NOT alarm);

sal-inf-bmc a320sp.sal surprise -v 3 -it -d 20
```

First Counterexample

step	act_mde	airspd	alt	fcu_alt	fcu_fpa	fcu_md	flaps	mx_spd	mntl_md	pitch
1	other	200	3000	3001	-1	other	rtrctd	400	level	0
	Commands: flying_clean, track_fcu_md, dial_descend									
2	vs_fpa	401	3000	3001	-2	vs_fpa	rtrctd	400	descend	0
	Commands: flying_clean, mode_reversion, extend_flaps									
3	op_clb	180	3000	3001	-2	vs_fpa	extnd	180	descend	0
	Commands: flying_flaps, op_clb_mode, pilots_idle									
4	op_clb	0	3000	3001	-2	vs_fpa	extnd	180	descend	1
	Commands: flying_flaps, op_clb_mode, pilots_idle									
5	op_clb	0	3091	3001	-2	vs_fpa	extnd	180	descend	0

- Mode reversion has occurred
- Causing a climb while the `mental_mode` is `descend`
- But it is due to `airspeed` abruptly increasing from 200 to 401
- Also, in steps 4 and 5 the airspeed decays to 0
- Our abstraction is too crude: need more constraints

Additional Constraints

```
[] airspeed' > airspeed+10 OR airspeed' < airspeed-10 --> ok' = FALSE;
[] pitch > 0 AND altitude' < altitude+10*pitch --> ok' = FALSE;
[] pitch < 0 AND altitude' > altitude+10*pitch --> ok' = FALSE;
[] pitch=0 AND
 (altitude' > altitude+10 OR altitude' < altitude-10) --> ok' = FALSE;
```

- Want airspeed changes to be gradual
- And altitude coupled more closely to pitch

Second Counterexample

step	act_mde	airspd	alt	fcu_alt	fcu_fpa	fcu_md	flaps	mx_spd	mntl_md	pitch
1	other	200	3000	3291	-1/50	other	rtrctd	400	level	-1/100
	Commands: flying_clean, track_fcu_md, dial_descend									
2	vs_fpa	201	2989	3291	-1/100	vs_fpa	rtrctd	400	descend	-1/100
	Commands: flying_clean, vs_fpa_mode, extend_flaps									
3	vs_fpa	200	2988	3291	-1/100	vs_fpa	extnd	180	descend	0
	Commands: flying_flaps, mode_reversion, pilots_idle									
4	op_clb	201	2989	3291	-1/100	vs_fpa	extnd	180	descend	0
	Commands: flying_flaps, op_clb_mode, pilots_idle									
5	op_clb	200	2990	3291	-1/100	vs_fpa	extnd	180	descend	1/50
	Commands: flying_flaps, op_clb_mode, pilots_idle									
6	op_clb	190	3291	3291	-1/100	vs_fpa	extnd	180	descend	3/100

- The `fcu_alt` is set to 3291 while the aircraft is flying at 3000
- The `pilots` decide to `descend` and enter a negative `fcu_fpa`
- Then extend the `flaps`
- Causes overspeed and a mode reversion to `op_clb` mode
- Which in turn causes a strong climb.

That Scenario Is Real

- It happened on 24 September 1994 to an Airbus A310, registration YR-LCC, operating as Tarom Flight 381 from Bucharest to Paris Orly
- Take a look at the following video of the incident http://www.youtube.com/watch?v=VqmrRFeYzBI
  - First part is a reconstruction based on information from the flight data recorder
  - The second part is actual video taken from the ground
  - sound track from the voice data recorder is synchronized to both parts
- Official incident report is available here http://www.bea.aero/docspa/1994/yr-a940924a/htm/yr-a940924a.html
- Due to this and other similar incidents, Airbus modified its speed protection package

Workflow

- Although it is very approximate, our modeling is sound
  - We include all real behaviors
- Idea is to refine the constraints until we get a realistic scenario that we can take to a high-fidelity simulation
  - Or discover that the counterexample was due to excessive approximation
- Formally equivalent, but a conceptual distinction between constraints that truly refine the model and those that serve merely to nudge the counterexample in a preferred direction
  - If desired, the latter can be placed in a separate constraints module
  - e.g., the values for `pitch` and `fcu_fpa` in our example are implausible

Conclusion

- Model checking systems against mental models is an effective way to discover automation surprises
- Extending the models to include hybrid systems increases range of systems for which approach is feasible and realistic
- Approximate modeling is ok: we are not analyzing performance of a control system
- Approach using relational abstractions is simple and effective
  - Synchronous observer allows easy spec'n of constraints
  - Although we used infinite BMC, could approximate with finite integers (bitvectors) and reproduce in any model checker—though performance problems are likely

Future Work

- We participate in a project called NextGen Authority and Autonomy (NextGenAA)
    - NextGen is decentralized air traffic control
- Intend to explore new procedures such as Continuous Descent Approach (CDA) and Oceanic Airspace In-Trail Procedure (ATSA ITP)
- Have developed a notation for task analytic models (EOFM)
- Will couple it to this method of analysis
- No conceptual problem with multiple human actors (e.g., pilots plus air traffic controllers) or systems (e.g., multiple airplanes), but we need to do examples of this kind