CSE503: Software Engineering

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

1

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?

2

Standard answers include

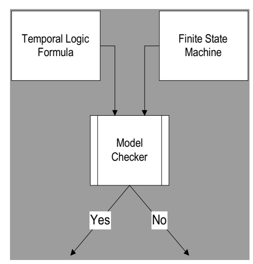
- Human inspection
- Simulation
- Analysis

- Aside: especially for safety-critical systems, I cannot imagine using only a single approach

3

An alternative: 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?



4

State Transition Graph

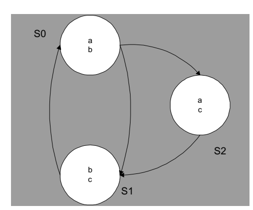
- One way to represent a finite state machine is as a state transition graph
  - S is a finite set of states
  - R is a binary relation that defines the possible transitions between states in S
  - P is a function that assigns atomic propositions to each state in S
    - e.g., that a specific process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

5

Example

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

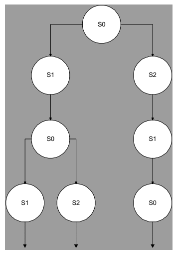
- Given a start state (say, S0), you can consider legal paths through the state machine



6

A computation tree

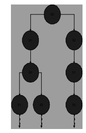
- From a given start state, you can represent all possible paths with an infinite computation tree
- Model checking allows us to answer questions about this tree structure
- Because the underlying machine is finite-state, the structure of the computation tree is constrained

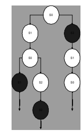


7

Temporal formulae: we can say things like

- Does some property hold true globally (e.g., in every state)?
  - Top figure
- Does some property inevitably hold true (e.g., along every path)?
  - Bottom figure
- Does some property potentially hold true?

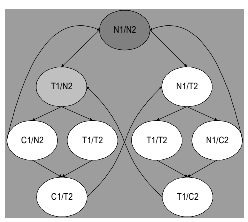




8

Mutual exclusion example

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



9

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

10

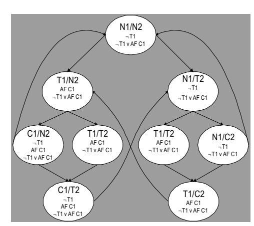
Example

- **Q == T1 implies AF C1**
  - If Process 1 is trying to acquire the mutex, then it is inevitably true it will get it sometime
- **Q ==** (**not T) OR AF C1**
  - Rewriting with DeMorgan's Laws
- First, label all the states where **T1**, **not T1**, and **C1** are true
  - These are atomic properties

11

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, **(not T1) OR AF C1** is true everywhere
  - Hence the temporal property is true for the state machine



12

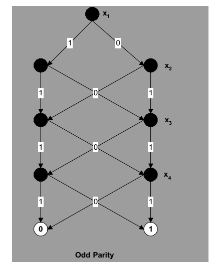
Symbolic model checking

- 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

13

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

14

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
  - **AND** becomes set intersection, **OR** becomes set union, etc.
- When the BDDs remain small, that is
  - The ordering of the variables is a key issue

15

BDD-based successes in HW

- IEEE Futurebus+ cache coherence protocol
- Control protocol for Philips stereo components
- ISDN User Part Protocol
- ...

16

Software model checking

- Finite state software specifications
  - Reactive systems (avionics, automotive, etc.)
  - Hierarchical state machine specifications
- Not intended to help with proving consistency of specification and implementation
  - Rather, checking properties of the specification itself

17

Why might it fail?

- 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

18

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
    - 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
- The vast bulk of this work was due to William Chan
  - Along with Mike Ernst won honorable mention in the 2000 ACM Dissertation Award competition
  - Died in a tragic automobile accident a week after defending his dissertation

19

TCAS

- Warn pilots of traffic
  - Plane to plane, not through ground controller
  - On essentially all commercial aircraft
- Issue resolution advisories only
  - Vertical resolution only
  - Relies on transponder data

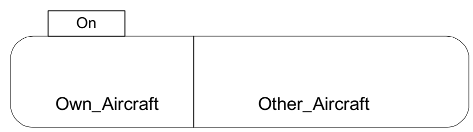
20

TCAS specification

- Irvine Safety Group (Leveson et al.)
  - Specified in RSML as a research project
  - 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

21

TCAS—high-level structure



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

22

Using SMV

- SMV is a BDD-based model checker
- It checks CTL formulas
  - A specific temporal logic
- We developed reasonably efficient techniques for mapping RSML to SMV, including the state hierarchies

23

Iterative process

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

24

Use of non-determinism: needed to reduce size of the BDDs

- 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

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
- Techniques included
  - 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

Disclaimer

•The intent of this work was to evaluate symbolic model checking of state-based specifications, not to evaluate the TCAS II specification. Our study used a preliminary version of the specification, version 6.00, dated March, 1993. We did not have access to later versions, so we do not know if the issues identified here are present in later versions.

Deterministic transitions

- Do the same conditions allow for non-deterministic transitions?
- Inconsistencies were found earlier (in an earlier version of TCAS) 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
  - If C1 is true then F returns V1
  - If C2 is true then F returns V2
  - If C3 is true then F returns V3
- A function is inconsistent if two different conditions Ci and Cj can be true simultaneously
- So, check the formula (for three cases)
  - **AG NOT ((C1 AND C2) OR (C1 AND C3) OR (C2 AND C3))**

Displayed_Model_Goal =

0	**if** Composite_RA **not in state** Positive
Max(Own_Track_Alt_Rate, PREV(Displayed_Model_Goal), 1500 ft/min)	**if** (New_Climb **or** New_Threat) **and** **not** New_Increase_Climb **and** **not** (Increase_Climb_Cancelled **or** Increase_Descend_Cancelled) **and** Composite_RA **in state** Climb
Min(Own_Track_Alt_Rate, PREV(Displayed_Model_Goal), −1500 ft/min)	**if** (New_Descend **or** New_Threat) **and** **not** New_Increase_Descend **and** **not** (Increase_Climb_Cancelled **or** Increase_Descend_Cancelled) **and** Composite_RA **in state** Descend
2500 ft/min	**if** New_Increase_Climb
−2500 ft/min	**if** New_Increase_Descend
Max(Own_Track_Alt_Rate, 1500 ft/min)	**if** Increase_Climb_Cancelled **and** **not** New_Increase_Climb **and** Composite_RA **in state** Positive
Min(Own_Track_Alt_Rate, −1500 ft/min)	**if** Increase_Descend_Cancelled **and** **not** New_Increase_Descend **and** Composite_RA **in state** Positive
PREV(Displayed_Model_Goal)	Otherwise

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) implies (DMG > 0))**
  - If Resolution Advisory is `Climb`, then `Display_Model_Goal` is positive
- Counterexample was found
  - t_0 : `RA = Descend, DMG = -1500`
  - t_1 : `RA = Increase-Descend, DMG = -2500`
  - t_2 : `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

37

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

38

What about infinite state?

- Model checking does not apply to infinite state specifications
  - The iterative algorithm will not reach a fixpoint
- Theorem proving applies well to infinite state specifications, but has generally proved to be unsatisfactory in practice
- One approach is to abstract infinite state specifications into finite state ones
  - Doing this while preserving properties is hard
- D. Jackson et al.'s Nitpick approach
  - Find counterexamples (errors), but don't "prove" anything

39

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
- This was fast: my goal wasn'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

40