CSE584: Software Engineering

Lecture 7 (November 17, 1998)

David Notkin
Dept. of Computer Science & Engineering
University of Washington
www.cs.washington.edu/education/courses/584/CurrentQtr/

This week

- A few formal methods in action
- Repeat: little attention has been paid to "non-functional" requirements
  - These have, however, been very high on your radar during class discussions
  - So far, formal methods have relatively little to contribute in this (very difficult) area
- Oh yeah…I'll work the whiteboard a bit today, too

Formal methods (reprise)

- The original use of formalism in software engineering was for proving the equivalence between a specification and an implementation
  - This had a number of problems
- But there has been a resurgence of interest in formal methods
  - Mostly due to potential usefulness in specification
  - And a few success stories

Potential benefits

- Increased clarity
- Ability to check for internal consistency
- Ability to prove properties about the specification (related to Jackson's refutable descriptions)
- Provides basis for falsification (perhaps more useful than verification)
- But not always worth the effort

C.A.R. Hoare, 1988

- *Of course, there is no fool-proof methodology or magic formula that will ensure a good, efficient, or even feasible design. For that, the designer needs experience, insight, flair, judgement, invention. Formal methods can only stimulate, guide, and discipline our human inspiration, clarify design alternatives, assist in exploring their consequences, formalize and communicate design decisions, and help to ensure that they are correctly carried out.*

Z ("zed")

- Perhaps the most widely known and used model-based specification language
- Good for describing state-based abstract descriptions roughly in the abstract data type style
- Based on typed set theory and predicate logic
- A few commercial successes
  - I'll come back to one reengineering story afterwards

The basic idea

- Static schemas
  - States a system can occupy
  - Invariants that must be maintained in every system state
- Dynamic schemas
  - Operations that are permitted
  - Relationship between inputs and outputs of those operations
  - Changes of states

The classic example

- A “birthday book” that tracks people’s birthdays and can issue reminders of those birthdays
  - There are tons of web-based versions of these now
- There are two basic types of atomic elements in this example
  - *[NAME, DATE]*
  - Nothing about formats, possible values, etc.

To the whiteboard

Z/CICS

- Z was used to help develop the next release of CICS/ESA_V3.1, a transaction processing system
  - Integrated into IBM's existing and well-established development process
  - Many measurements of the process indicated that they were able to reduce their costs for the development by almost five and a half million dollars
  - Early results from customers also indicated significantly fewer problems, and those that have been detected are less severe than would be expected otherwise

1992 Queen’s Award for Technological Achievement

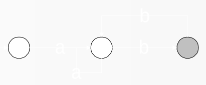
- “Her Majesty the Queen has been graciously pleased to approve the Prime Minister's recommendation that The Queen's Award for Technological Achievement should be conferred this year upon Oxford University Computing Laboratory.
- “Oxford University Computing Laboratory gains the Award jointly with IBM United Kingdom Laboratories Limited for the development of a programming method based on elementary set theory and logic known as the Z notation, and its application in the IBM Customer Information Control System (CICS) product. ...

…

- “The use of Z reduced development costs significantly and improved reliability and quality. Precision is achieved by basing the notation on mathematics, abstraction through data refinement, re-use through modularity and accuracy through the techniques of proof and derivation.
- “CICS is used worldwide by banks, insurance companies, finance houses and airlines etc. who rely on the integrity of the system for their day-to-day business.”

Finite state machines

- There is a large class of specification languages based on finite state machines
- Often primarily for describing the control aspects of embedded systems
- The theoretical basis is very firm



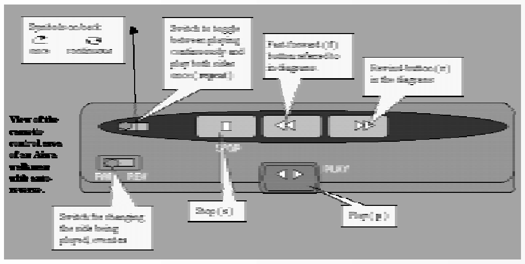
- A finite set of states
- A finite alphabet of symbols
- A start state and one or more final states
- A transition relation

Many, many models

- Petri nets
- Communicating finite state machines
- Statecharts
- RSML
- …
- …

Walkman example

(due to Alistair Kilgour, Heriot-Watt University)

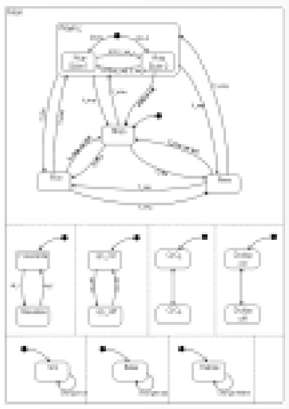

Symbols on back
once
continuous
Switch to toggle between playing continuously and play both sides once (repeat)
Fast-forward (ff) button referred to in diagram
Rewind button (rr) in the diagram
View of the cassette control area of an Aiwa walkman with auto-reverse.
STOP
PLAY
FWD
REV
Switch for changing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

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 (which are usually desirable)
  - And for machines capturing concurrency (because of the potential interleavings that must be captured)

Statecharts (Harel)

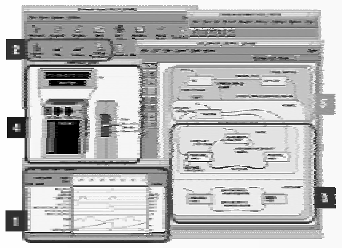
- A visual formalism for defining finite state machines
- A hierarchical mechanism allows for complex machines to be defined by smaller descriptions
  - Parallel states (AND decomposition)
  - Conventional OR decomposition



Tools

- Statecharts have a set of supporting tools from i-Logix (STATEMATE, Rhapsody)
  - Editors
  - Simulators
  - Code generators
    - C, Ada, Verilog, VHDL
  - Some analysis support
- Statecharts (roughly) are a central part of UML

i-Logix screenshot

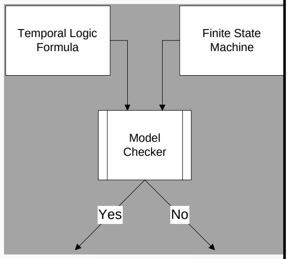


Analysis

- Given a Statecharts description, how can one tell if it has some desirable properties?
  - For instance, is it deterministic?
  - Are there deadlocks?
  - And domain-specific properties, too
- Perhaps the most promising technology for helping with this is model checking

Model checking

- Evaluate temporal properties of finite state systems
  - Guarantee a property is true or return a counterexample
- Extremely successfully for hardware verification
- Open question: applicable to software specifications?

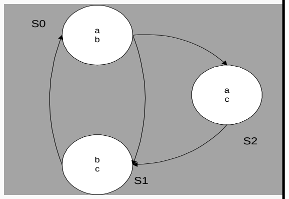


State Transition Graph

- 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.

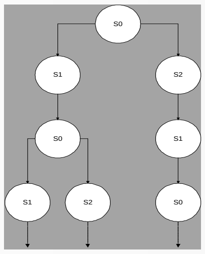
Example

- Three states
- Transitions as shown
- Atomic properties a, b and c
- Given a start state, you can consider legal paths through the state machine



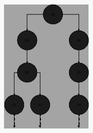
A computation tree

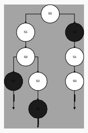
- From a given start state, you can represent all possible paths with an infinite computation tree
- Model checking allows us to answer questions about this tree structure



Temporal formulae

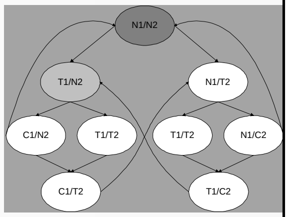
- Temporal logics allow us to say things like
  - Does some property hold true globally?
    - Top figure
  - Does some property inevitably hold true?
    - Bottom figure
  - Does some property potentially hold true?





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?



Model checking

- A model checker takes as input the state machine description and a temporal logic formula and
  - either returns "true" or
  - returns "false" and gives a counterexample
    - a description of state transitions that leads to a counterexample of the temporal formula

How does it work? (in brief)

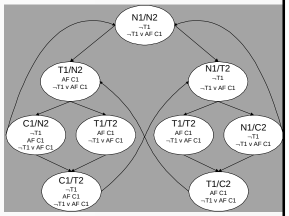
- An iterative algorithm that labels states in the transition graph with formulae known to be true
- For a query Q
  - the first iteration marks all subformulae of Q of length 1
  - the second iteration marks them of length 2
  - this terminates since the formula is finite
- The details of the logic 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

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

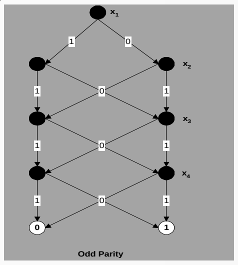


Symbolic model checking

- State space can be huge (>2^{1000}) for many systems
- Use implicit representation
  - 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)
  - Temporal queries



Due to Randy Bryant

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
- 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
- …

Software model checking

- Finite state software specifications
  - Reactive systems (avionics, automotive, etc.)
  - Hierarchical state machine specifications
    - Statecharts (Harel), RSML (Leveson)
- Not intended to help with proving consistency of specification and implementation

Why might it fail?

- Software is often specified with infinite state descriptions
  - We'll come back to this later (counterexample checking)
- Software specifications may be structured differently from hardware specifications
  - Hierarchy
  - Representations and algorithms for model checking may not scale

An 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.

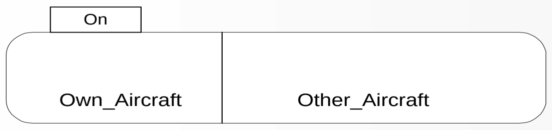
TCAS

- 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

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

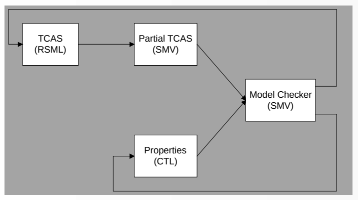
TCAS—high-level structure



- 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



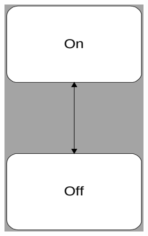
Iterative process

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

Use of non-determinism

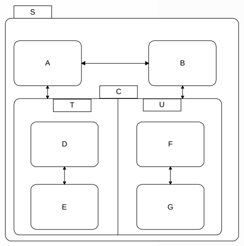
- Inputs from environment
  - `Altitude := {1000…8000}`
- Simplification of functions
  - `Alt_Rate := 0.25*(Alt_Baro-ZP)/Delta_t`
  - `Alt_Rate := {-2000…2000}`
- Unmodelled parts of specification
  - States of `Other_Aircraft` treated as non-determinstic input variables

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};
```

Synchrony hypothesis

- Handling an external event

```
DEFINE
  Stable := !Initiate_Move &
               !Move_Finished &
               !Rod_Updated & !Clock_Event
  ASSIGN
  next(Move_Finished) := case
        Stable : {0,1};
        1      : 0;
esac;
…for other external events…
```

Transitions

```
VAR RC: {Out, Mid, In};

  ASSIGN
  T_Out_Mid : Mid;     T_Mid_In : In;
  T_Mid_Out : Out;     T_In_Mid : Mid;
  1 : RC;
esac;
```

Non-deterministic transitions

- A machine is deterministic if at most one of `T_A_B`, `T_A_C`, etc. can be true
  - Else non-deterministic
- Can encode non-deterministic transitions

  -
    ```
    next(S) := case
        T_A_B & T_A_C: {B,C};
        T_A_B : B; T_A_C : C;
        1 : S;
    esac;
    ```

Checking properties

- Initial attempts to check any property generated BDDs of over 200MB
- First successful check took 13 hours
  - Has been 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

Disclaimer

The intent of this work is 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 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

Note: Earlier version of TCAS spec

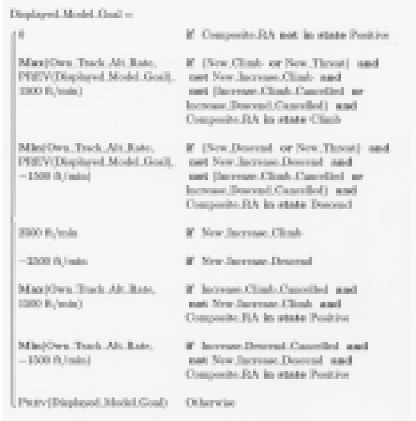
```
V_254a := MS = TA_RA | MS = TA_only | MS =3 | MS = 4 |
          MS = 5 | MS = 6 | MS = 7;
V_254b := ASL = 2 | ASL = 3 | ASL = 4 | ASL = 5 |
          ASL = 6 | ASL = 7;
T_254  := (ASL = 2 & V_254a) | (ASL = 2 & MS = TA_only) |
          (V_254b & LG = 2 & V524a);
V_257a := LG = 5 | LG = 6 | LG = 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

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

```
AG !((C1 & C2) |
     (C1 & C2) |
     (C2 & 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 -> 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 site

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

More criteria

- Timing criteria
- Data validity criteria
- Degradation criteria
- Feedback criteria
- Reachability criteria

What about infinite state?

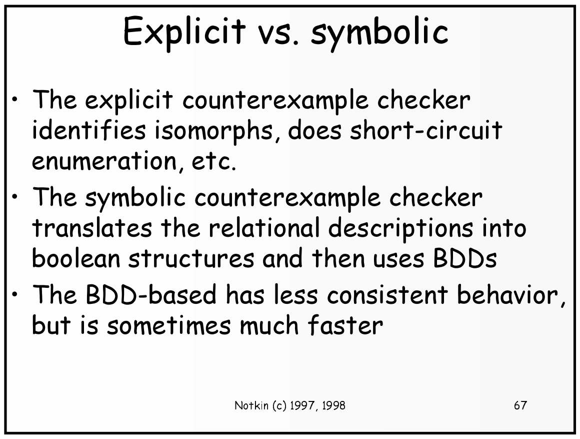
- Model checking does not apply to infinite state specifications
  - The iterative algorithm will not reach a fixpoint
- Theorem proving applies well to infinite state specifications, but has generally proved to be unsatisfactory in practice
- One approach is to abstract infinite state specifications into finite state ones
  - Doing this and preserving properties is hard

A middle ground

- Jackson and Damon have found an interesting middle ground
- Write infinite state specs (in the style of Z)
- Use "model checking" on all instances of the specifications up to a certain size
  - Report counterexamples, if found
  - Success doesn't guarantee that the properties hold in the specification (beyond the checked sizes)

Nitpick

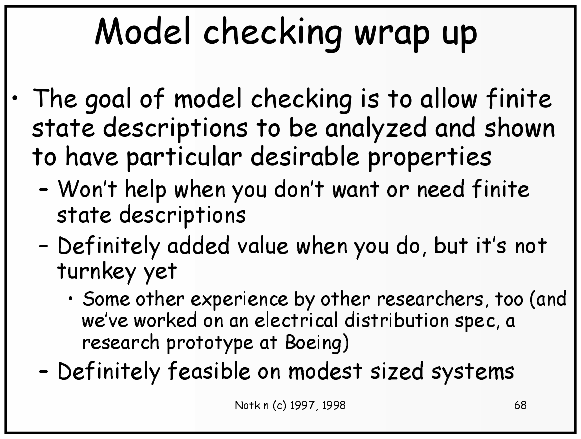
- The tool that checks for counterexamples given a (subset of) Z specification
- Examples include
  - Paragraph style mechanisms
  - Telephone switch structures
- Two variants—explicit state space enumeration and BDD-based checking
  - More recent version using a boolean satisfiability engine (Walksat)



Explicit vs. symbolic

- The explicit counterexample checker identifies isomorphs, does short-circuit enumeration, etc.
- The symbolic counterexample checker translates the relational descriptions into boolean structures and then uses BDDs
- The BDD-based has less consistent behavior, but is sometimes much faster

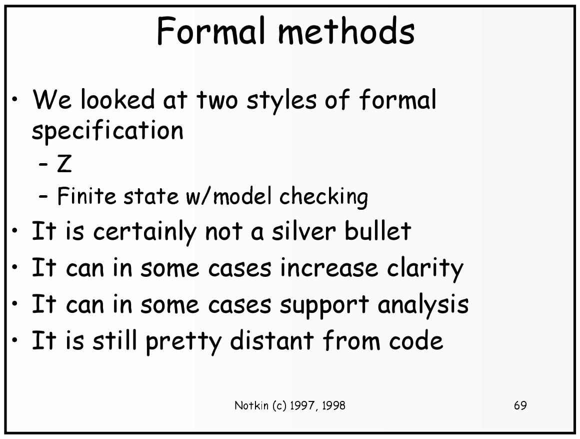
Notkin (c) 1997, 1998 67



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
    - Some other experience by other researchers, too (and we've worked on an electrical distribution spec, a research prototype at Boeing)
  - Definitely feasible on modest sized systems

Notkin (c) 1997, 1998 68



Formal methods

- We looked at two styles of formal specification
  - Z
  - Finite state w/model checking
- It is certainly not a silver bullet
- It can in some cases increase clarity
- It can in some cases support analysis
- It is still pretty distant from code

Notkin (c) 1997, 1998 69