

# CSci 780 Advanced Software Engineering

## State-Based Model Checking of Event-Driven System Requirements

Joanne M. Atlee and John Gannon

11/12/2003

1

## Outline

- Background: Model Checking
- Event-Driven Systems
- SCR
- CTL
- Model Checking SCR Specifications
- Case Studies
- Evaluation and Conclusion
- Discussion

11/12/2003

2

## Background: Model Checking

11/12/2003

3

## Problem

- Errors often due to unknown, undocumented, or wrong requirements
- Specification faults lead to safety errors
- Errors cheaper to fix if found early
- Testing fails if it tests a buggy specification

11/12/2003

4

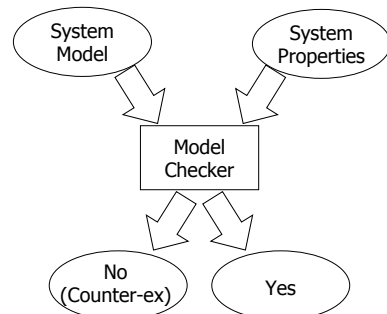
## Model Checking

- Construct a finite state machine from the specification, and check that desired properties hold
  - Reachability analysis (application-independent properties)
  - Behavioral model checking (application-dependent control properties)
  - Relational model checking (application-dependent data properties)

11/12/2003

5

## How It Works



11/12/2003

6

## Pros and Cons

---

- Pros
  - Identifies subtle bugs
  - Faster than theorem proving
  - Provides a counter-example
  - Doesn't require a lot of math
  - Successful in hardware verification
- Cons
  - Suffers from state explosion problem
  - Proof negative, but no proof positive

11/12/2003

7

## Event-Driven Systems

11/12/2003

8

## Event-Driven Systems

---

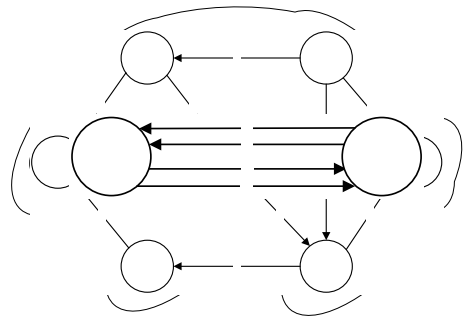
- States: System can occupy different states
- Transitions: State changes occur when events happen
- Modes: Abstract state details by constructing a set of related states
- Modeclass: A partition of modes
- Mode transition: Change between modes

11/12/2003

9

## Example

---



11/12/2003

10

## More On Finite State Machines

---

- Nondeterminism is allowed
- Transitions occur in zero time
- Modes are just one approach to dealing with state explosion
  - e.g. Can use better data structures (BDDs, MBDDs)

11/12/2003

11

## SCR

11/12/2003

12

## Software Cost Reduction (SCR) Specs

- A requirements-oriented specification language
- Table-based specification of finite state machine
  - Table specifies transitions based on conditions
  - Invariants express condition value relationships

11/12/2003

13

## Events and Conditions

- Condition: Boolean state value
- Event: Change in condition's value at a time
  - @T(A): Condition A has become true
  - @F(A): Condition A has become false
- Guarded events:
  - @T(A) WHEN B: Condition A has become true while condition B is also true
  - @T(A ∧ B) ≠ @T(A) when B: B true in  $[t_{@T(A)} - \epsilon, t_{@T(A)}]$

11/12/2003

14

## Example: Temperature Control

Current Mode	Running	BelowDesiredTemp	TempOK	AboveDesiredTemp	New Mode
Off	@T	-	t	-	Inactive
	@T	t	-	-	Heat
	@T	-	-	t	AC
Inactive	@F	-	-	-	Off
	-	@T	-	-	Heat
	-	-	-	@T	AC
Heat	@F	-	-	-	Off
	-	-	@T	-	Inactive
	@F	-	-	-	Off
AC	@F	-	-	-	Off
	-	-	@T	-	Inactive

11/12/2003

15

## Formalization

- User often omits detail in table
  - Leads to nondeterminism, underspecification
- User knows global constraints
  - e.g. Temperature must be below, ok, or above
- "Formalization" process augments the user-given table
  - Invariants add additional condition constraints
  - Instantaneous transitions simplified and added
  - Nondeterminism detected and warned

11/12/2003

16

## Example: Invariants, Transitions, Determinism

Current Mode	Running	BelowDesiredTemp	TempOK	AboveDesiredTemp	New Mode
Off	@T	f	t	f	Inactive
	@T	t	f	f	Heat
	@T	f	f	t	AC
Inactive	@F	-	-	-	Off
	@F	@T	@F	f	Heat
	@F	f	@F	@T	AC
	□	@T	@F	f	Heat
	□	f	@F	@T	AC
Heat	@F	-	-	-	Off
	@F	@F	@T	f	Inactive
	□	@F	@T	f	Inactive
AC	@F	-	-	-	Off
	@F	@F	@T	f	Inactive
	□	f	@T	@F	Inactive

11/12/2003

17

CTL

11/12/2003

18

## Recall: Linear Temporal Logic (LTL)

- Future
    - $\bigcirc$ : Next
    - $\diamond$ : Future
    - $\square$ : Global
  - Past
    - $\ominus$ : Previous
    - $\diamond$ : Once
    - $\boxminus$ : Always been
- State Formulae +  
Boolean Operators =  
Linear Temporal Logic  
(LTL)

11/12/2003

19

## Computation Tree Logic (CTL)

- Future
    - X: Next
    - F: Future
    - G: Global
    - U: Until
  - Past
    - P: Previous
    - O: Once
    - B: Always been
    - S: Since
  - States/Paths
    - A: All states
    - E: Some state
- (+ Boolean Operators)

11/12/2003

20

## Computation Tree Logic (CTL)

- CTL can express properties LTL can't (and vice-versa)
- Examples
  - $AX(f)$ : property  $f$  holds in all next states
  - $E(f U g)$ : there is some path in which  $f$  is true up to a state in which  $g$  is true
  - $AG(f)$ :  $f$  is true in every state in every path

11/12/2003

21

## CTL Example

- Global constraints from the user
  - $OFF \Rightarrow \neg \text{Running}$
  - $INACTIVE \Rightarrow (\text{Running} \wedge \text{TempOK})$
  - $HEAT \Rightarrow (\text{Running} \wedge \text{BelowDesiredTemp})$
  - $AC \Rightarrow (\text{Running} \wedge \text{AboveDesiredTemp})$
  - $(\text{Running} \wedge \text{BelowDesiredTemp} \Rightarrow (\text{Heat} \vee \bigcirc(\text{Heat})))$
  - $(\text{Running} \wedge \text{AboveDesiredTemp} \Rightarrow (AC \vee \bigcirc(AC)))$
- Global constraints must hold in all states (CTL "A") and all paths (CTL "G")

11/12/2003

22

## CTL Example (cont)

- CTL versions
  - $AG( OFF \rightarrow \sim \text{Running} )$
  - $AG( INACTIVE \rightarrow (\text{Running} \ \& \ \text{TempOK}) )$
  - $AG( HEAT \rightarrow (\text{Running} \ \& \ \text{BelowDesiredTemp}) )$
  - $AG( AC \rightarrow (\text{Running} \ \& \ \text{AboveDesiredTemp}) )$
  - $AG( (\text{Running} \ \& \ \text{BelowDesiredTemp} \rightarrow (\text{Heat} \ | \ AX(\text{Heat}))) )$
  - $AG( (\text{Running} \ \& \ \text{AboveDesiredTemp} \rightarrow (AC \ | \ AX(AC))) )$

11/12/2003

23

## Model Checking SCR Specifications

11/12/2003

24

## Model Checking of SCR Specifications

- User writes SCR table and CTL properties
- Model checker
  - Converts SCR into CTL machine
  - Feeds CTL machine and CTL properties into MCB model checker

11/12/2003

25

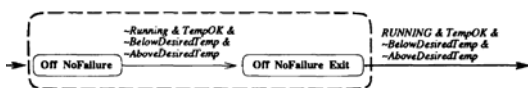
## Mapping SCR to CTL Machines

- Modes → states
- Mode transitions → state transitions
- Conditions → input variables
- Events → ?

11/12/2003

26

## Events

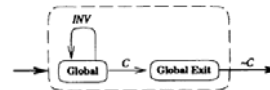


- Need to detect changes in condition values and ensure that the state transitions are activated
- Use exit states to ensure events are false, then become true (when conditions always true)

11/12/2003

27

## Mode Invariants

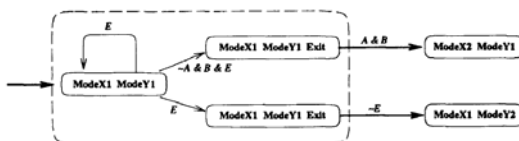


- Invariants must be true when entering a mode, and remain true while in the mode.
- CTL transition condition may be false on entering the state, violating the invariant
- So we add an invariant self-loop to make sure that the machine allows the invariant to be true

11/12/2003

28

## Mode Conjunction



- For multiple modeclasses, the combined machine may be missing invariants
- Add invariants of each modeclass to other transitions

11/12/2003

29

## Conversion Algorithm

1. Each environmental condition becomes an input condition.
2. Each system mode becomes an output proposition (Exit is also an output proposition)
3. For each mode
  1. Create a mode state and annotate it with the mode name.
  2. Create a transition from each mode state to itself, annotated with the mode's invariant properties
  3. Create an exit state for each transition annotated with the mode name and exit
  4. Create a transition from each mode state to the exit states annotated with the when conditions, the negated trigger conditions and the mode invariants.
  5. Create a transition from each exit state to the destination mode state, annotated with the triggered conditions and the when conditions.
4. Create an initial exit state with a transition to the initial mode annotated with the initial conditions.

11/12/2003

30

## Example: Temperature Control System



11/12/2003

31

## Case Studies

11/12/2003

32

## Case Study #1: Cruise Control

Current Mode	Ignited	Running	TooFast	Brake	Activate	Deactivate	Resume	New Mode
Off	@T	-	-	-	-	-	-	Inactive
Inactive	@F	-	-	-	-	-	-	Off
	t	t	-	f	@T	-	-	Cruise
Cruise	@F	-	-	-	-	-	-	Off
	-	@F	-	-	-	-	-	Inactive
	-	-	@T	-	-	-	-	Override
	-	-	-	@T	-	-	-	Override
Override	@F	-	-	-	-	-	@T	Off
	-	@F	-	-	-	-	-	Inactive
	t	t	-	f	@T	-	-	Cruise
	t	t	-	f	-	-	@T	Cruise

Initial Mode: Off

11/12/2003

33

## Case Study #1: Formalization Results

- Formalization revealed unintended nondeterminism
- In Cruise, "@F(Running) & @T(Brake)" leads to either Inactive or Override
- But you should only be in override mode when the car is running (Override ⇒ Running)
- Need to add "Running=true" guard on @T(Brake)

11/12/2003

34

## Case Study #1: Model Checking Results

- Ignited not always false in initial OFF mode
  - Add ~ignited as a starting condition
- System wouldn't leave Cruise mode when Brake becomes true if TooFast is true (!)
  - Add ~TooFast a condition of all transitions to Cruise (ensures that ~TooFast is a Cruise invariant)
- Cruise mode not entered if Ignition is true, Engine is true, and Brake is true when activated
  - Invariant changed to make this okay

11/12/2003

35

## Case Study #2: Water Level Monitor

Current Mode	InadeqByRange	WithinLimits	SFTatPressure	SFTatInterval	T atInterval	ResetInterval	ShutdownLockTime	New Mode
Standby	t	t	-	-	-	@T	-	Operating
	-	-	t	@T	-	-	-	Test
Operating	f	@F	f	f	-	-	-	Shutdown
	-	-	t	@T	-	-	-	Test
Shutdown	@T	t	f	f	-	-	f	Operating
	-	-	f	f	-	-	@T	Standby
	-	-	t	@T	-	-	-	Test
Test	-	-	-	@T	-	-	-	Standby

Initial Mode: Standby (~SFTatInterval & ~TestInterval & ~ResetInterval & ~ShutdownLockTime)

11/12/2003

36

## Case Study #2: Model Checking Results

---

- When not in test mode, press test and reset at the same time; system goes operational
  - Add  $\sim$ SIfTestInterval condition to other transitions
- When in test mode, hold test button; system goes into Standby with SIfTestInterval true
  - Change spec so Test isn't left until button released
- System won't detect errors and enter Shutdown if test button being held down, then released before Test mode entered
  - Enter Shutdown even if test button pressed

11/12/2003

37

## Evaluation and Conclusion

11/12/2003

38

## Evaluation

---

- Pros
  - Specification language is intuitive
  - Analysis is automated
  - Reveals subtle specification bugs
- Cons
  - Only a single modeclass shown. Multiple modeclasses lead to exponential state growth
  - MCB model checker restricted to FSMs

11/12/2003

39

## Conclusion

---

- Demonstrates the feasibility of model checking safety properties of event-driven systems
- Combines
  - Event-driven requirements expressed using SCR
  - Property checking using CTL
- Fully automated
- Found subtle errors in significant systems

11/12/2003

40