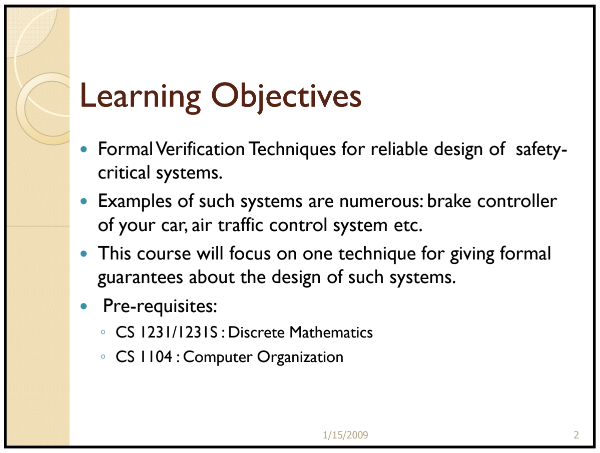
CS4271 Critical Sys. and their Verification

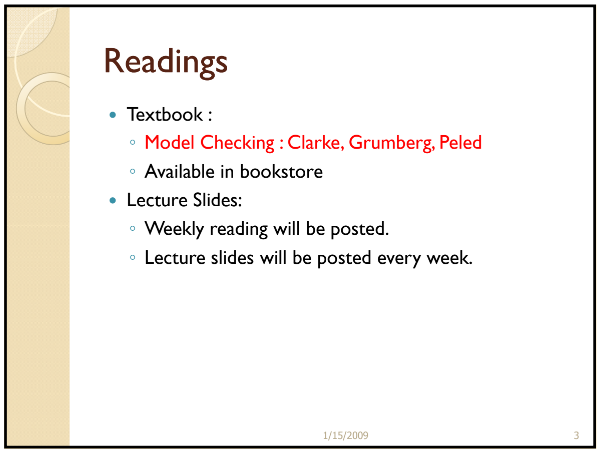
Abhik Roychoudhury
abhik@comp.nus.edu.sg

Introductory Lecture



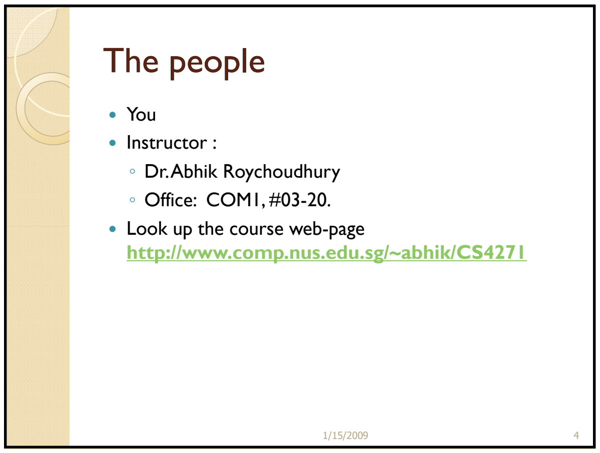
Learning Objectives

- Formal Verification Techniques for reliable design of safety-critical systems.
- Examples of such systems are numerous: brake controller of your car, air traffic control system etc.
- This course will focus on one technique for giving formal guarantees about the design of such systems.
- Pre-requisites:
  - CS 1231/1231S : Discrete Mathematics
  - CS 1104 : Computer Organization



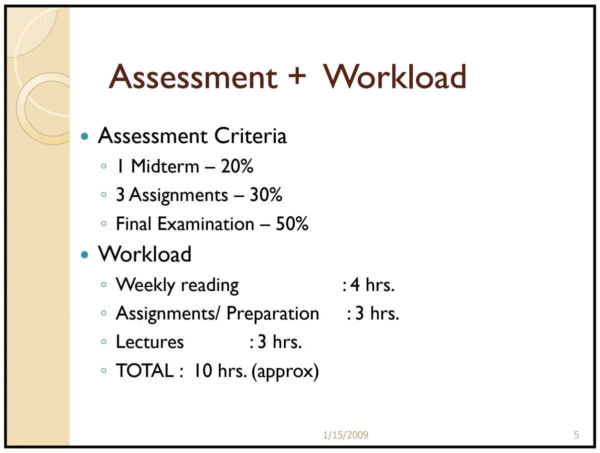
Readings

- Textbook :
  - Model Checking : Clarke, Grumberg, Peled
  - Available in bookstore
- Lecture Slides:
  - Weekly reading will be posted.
  - Lecture slides will be posted every week.



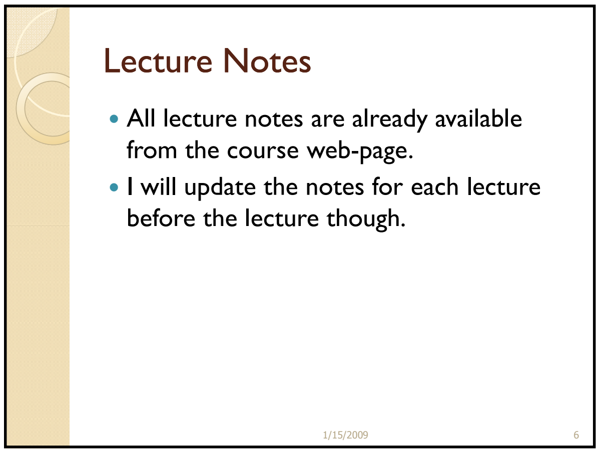
The people

- You
- Instructor :
  - Dr. Abhik Roychoudhury
  - Office: COM1, #03-20.
- Look up the course web-page http://www.comp.nus.edu.sg/~abhik/CS4271



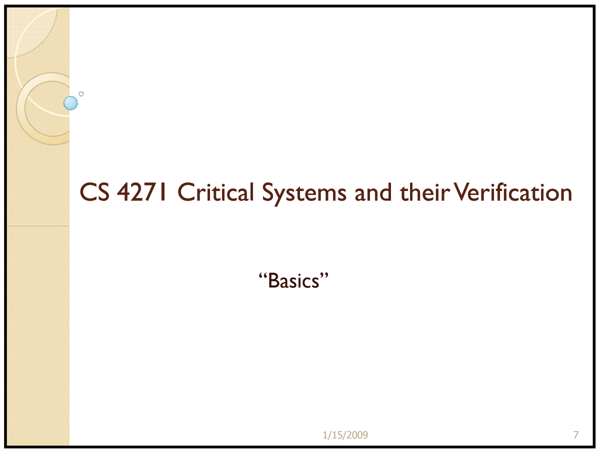
Assessment + Workload

- Assessment Criteria
  - 1 Midterm – 20%
  - 3 Assignments – 30%
  - Final Examination – 50%
- Workload
  - Weekly reading : 4 hrs.
  - Assignments/ Preparation : 3 hrs.
  - Lectures : 3 hrs.
  - TOTAL : 10 hrs. (approx)



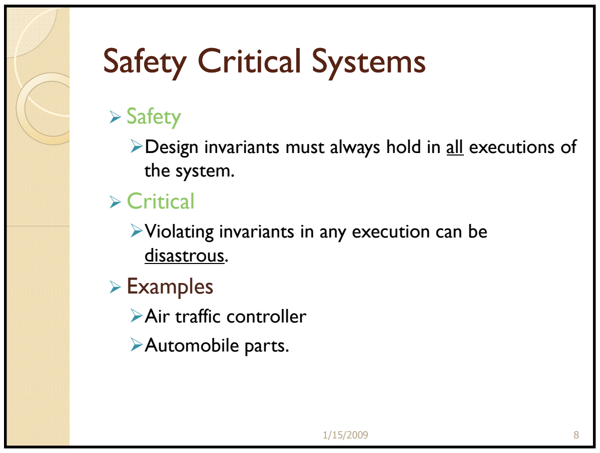
Lecture Notes

- All lecture notes are already available from the course web-page.
- I will update the notes for each lecture before the lecture though.



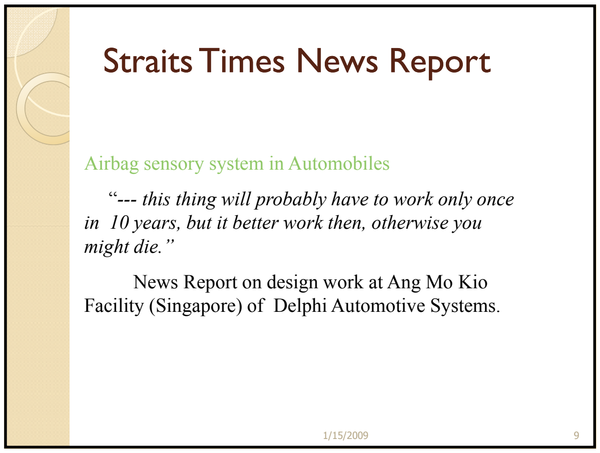
CS 4271 Critical Systems and their Verification

"Basics"



Safety Critical Systems

- Safety
  - Design invariants must always hold in <u>all</u> executions of the system.
- Critical
  - Violating invariants in any execution can be <u>disastrous</u>.
- Examples
  - Air traffic controller
  - Automobile parts.

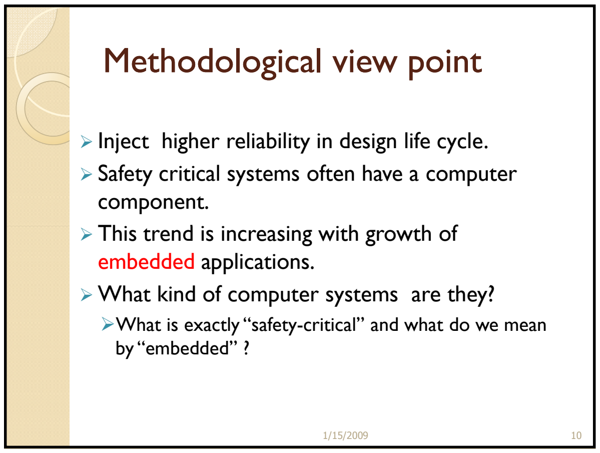


Straits Times News Report

Airbag sensory system in Automobiles

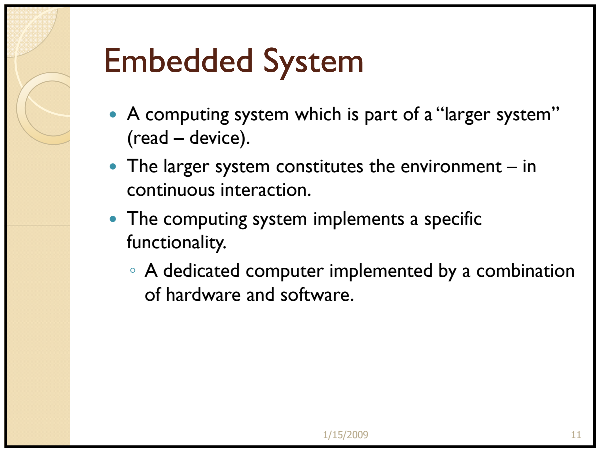
"--- this thing will probably have to work only once in 10 years, but it better work then, otherwise you might die."

News Report on design work at Ang Mo Kio Facility (Singapore) of Delphi Automotive Systems.



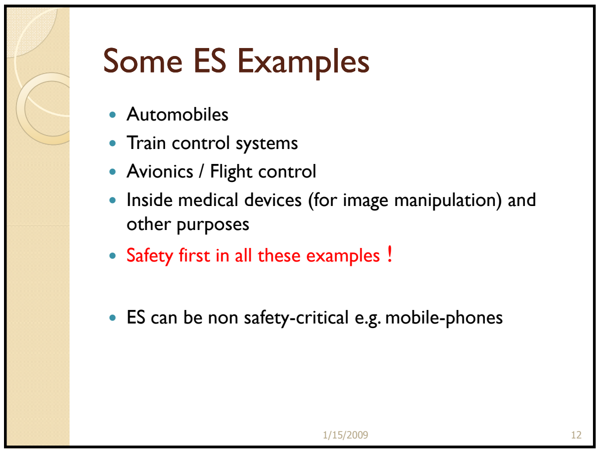
Methodological view point

- Inject higher reliability in design life cycle.
- Safety critical systems often have a computer component.
- This trend is increasing with growth of embedded applications.
- What kind of computer systems are they?
  - What is exactly "safety-critical" and what do we mean by "embedded" ?



Embedded System

- A computing system which is part of a "larger system" (read – device).
- The larger system constitutes the environment – in continuous interaction.
- The computing system implements a specific functionality.
  - A dedicated computer implemented by a combination of hardware and software.



Some ES Examples

- Automobiles
- Train control systems
- Avionics / Flight control
- Inside medical devices (for image manipulation) and other purposes
- Safety first in all these examples !

- ES can be non safety-critical e.g. mobile-phones

ES Characteristics

- Real-time and/or Reactive
  - Often combines hard and soft real-time
  - Timing constraints on the response
- Low power budget
  - Novel architectures etc.
- Low code size
  - Aggressive Code compression possible.

But what is "reactive" ?

- Continuously interacts with its environment.
- Interaction with env. is asynchronous.
- Often, its response to environment needs to obey time constraints.
- Often consists of a concurrent composition of processes.

OK, but why study ES now ?

- Embedded systems
  - Using a computer component as part of a bigger system becoming pervasive.
- Many are *safety-critical* e.g. automobile parts
- Current validation techniques do not suffice.
  - Perceived as *intrusive* to design process.

Validation Techniques

- In circuit Emulator (ICE)
- Logic Analyzer
- Model based simulation
- Formal verification techniques
  - Model Checking
  - *Deduction*
  - *Combinations of the two*

In circuit Emulator (ICE)

- Used widely in industry for designs where a microprocessor interacts with potpourri of peripherals.
- ICE is a dedicated hardware for a particular processor which allows its internals to be read.
- Response of processor (to environment) observed by *physically* inserting the ICE.

Logic Analyzer

- Used for sampling many signals simultaneously in a complex design.
- Can snoop on a bus to observe interactions of a microprocessor with its environment.
- ICE and Logic Analyzer do not work when:
  - Processor, peripherals, bus all integrated in a chip.
  - *System-on-Chip (SoC)*

Model based simulation

- Simulate and observe the behaviors of a system model, rather than the system itself.
- Takes validation/debugging higher in the design life-cycle.
- Since a model is validated, can take place prior to system integration
  - Hardware software co-simulation (POLIS)

Model Checking – What ?

- Same as model based simulation except that you check all possible behaviors.
- Needed for checking critical properties.
- Can be used if model has finite states.

- Many realistic systems are infinite-state
  - e.g. real-time systems.
- For these systems, need extensions of model checking (not discussed in this course).

Model Checking – What ?

- M $|= \varphi$?
  - M is the system to be verified,
  - φ is the property to be verified,
  - $|=$ is the satisfaction relation.

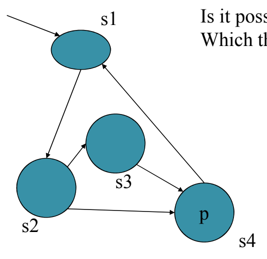
Specification, Implementation

- Need a specification language
  - property to be verified.
- Need an implementation language
  - system to be verified.
- Such a language describes reactive system *behaviors*.
  - Temporal Logic
    - Linear - LTL, Branching - CTL
  - Process Algebra/Calculus
    - CCS (Milner 1989), CSP (Hoare)
  - Finite state automata over infinite inputs.

Model Checking – How ?

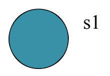
- Inputs:
  - **finite state** concurrent system (implementation as FSM)
  - Temporal logic formula (specification language)
- Output:
  - True if the specification holds
  - A **counterexample behavior** if it does not
- Technique:
  - Implementation FSM is a finite graph.
  - Unfold and search this finite graph to check all behaviors.

Example



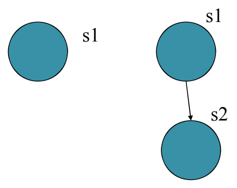
Is it possible to reach a state in Which the property p holds ?

State space search



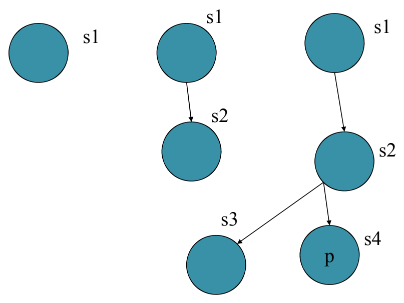
1/15/2009 25

State space search

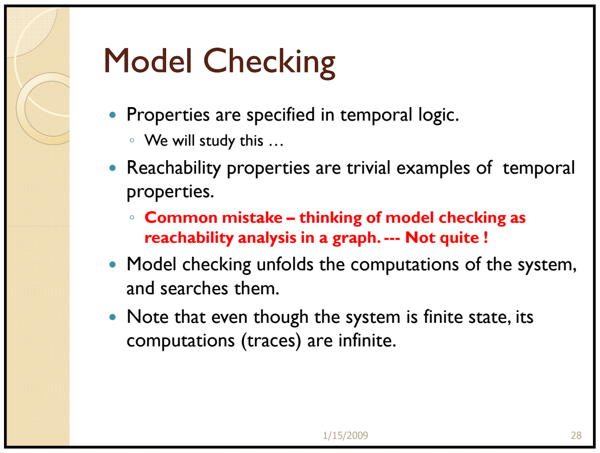


1/15/2009 26

State space search



1/15/2009 27

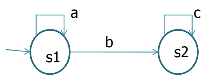


Model Checking

- Properties are specified in temporal logic.
  - We will study this …
- Reachability properties are trivial examples of temporal properties.
  - **Common mistake – thinking of model checking as reachability analysis in a graph. --- Not quite !**
- Model checking unfolds the computations of the system, and searches them.
- Note that even though the system is finite state, its computations (traces) are infinite.

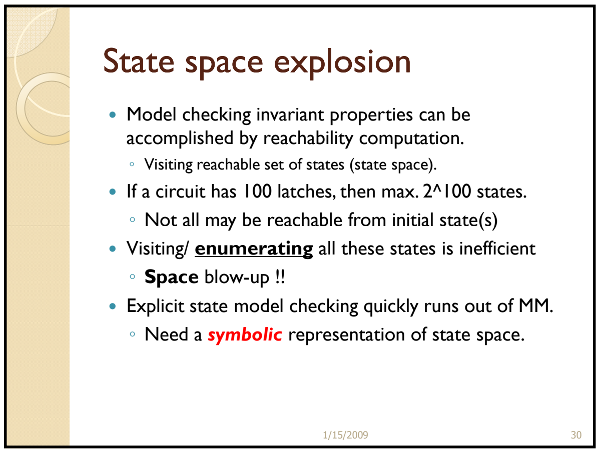
1/15/2009 28

A Simple Example



- What are the states? {s1, s2}
- How many traces? Infinitely many
  - a^ω
  - a^*bc^ω
- Length of each trace? Infinite

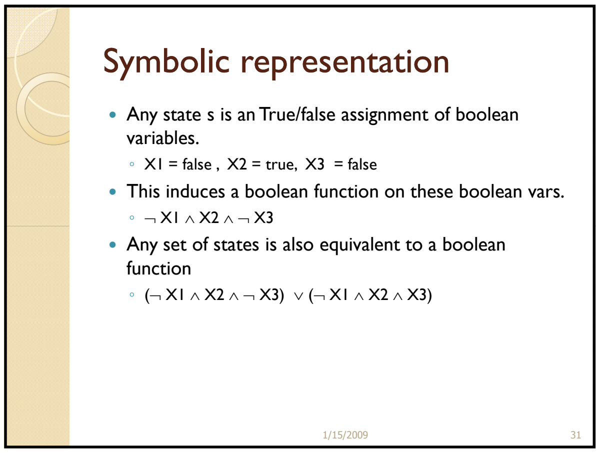
1/15/2009 29



State space explosion

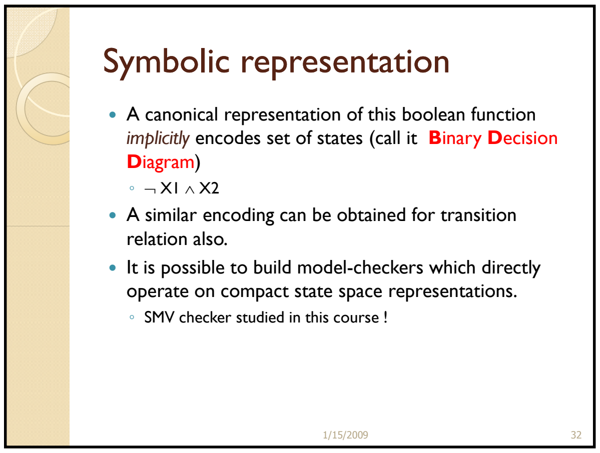
- Model checking invariant properties can be accomplished by reachability computation.
  - Visiting reachable set of states (state space).
- If a circuit has 100 latches, then max. 2^100 states.
  - Not all may be reachable from initial state(s)
- Visiting/ **enumerating** all these states is inefficient
  - **Space** blow-up !!
- Explicit state model checking quickly runs out of MM.
  - Need a ***symbolic*** representation of state space.

1/15/2009 30



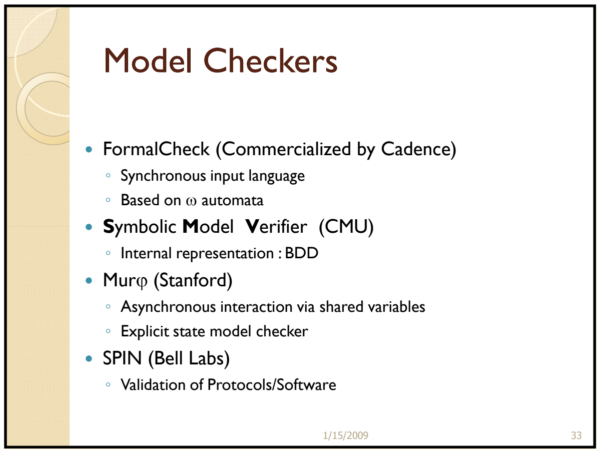
Symbolic representation

- Any state s is an True/false assignment of boolean variables.
  - $X1$ = false , $X2$ = true, $X3$ = false
- This induces a boolean function on these boolean vars.
  - $\neg X1 \wedge X2 \wedge \neg X3$
- Any set of states is also equivalent to a boolean function
  - $(\neg X1 \wedge X2 \wedge \neg X3) \vee (\neg X1 \wedge X2 \wedge X3)$



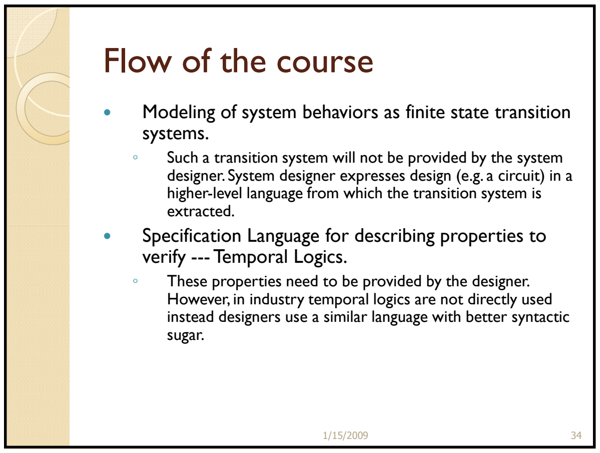
Symbolic representation

- A canonical representation of this boolean function *implicitly* encodes set of states (call it **B**inary **D**ecision **D**iagram)
  - $\neg X1 \wedge X2$
- A similar encoding can be obtained for transition relation also.
- It is possible to build model-checkers which directly operate on compact state space representations.
  - SMV checker studied in this course !



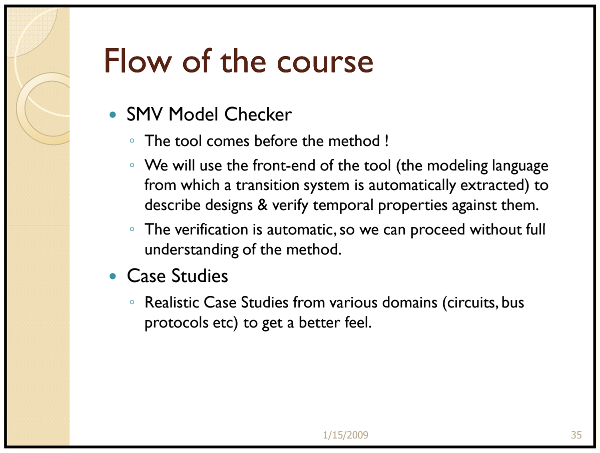
Model Checkers

- FormalCheck (Commercialized by Cadence)
  - Synchronous input language
  - Based on ω automata
- **S**ymbolic **M**odel **V**erifier (CMU)
  - Internal representation : BDD
- Murφ (Stanford)
  - Asynchronous interaction via shared variables
  - Explicit state model checker
- SPIN (Bell Labs)
  - Validation of Protocols/Software



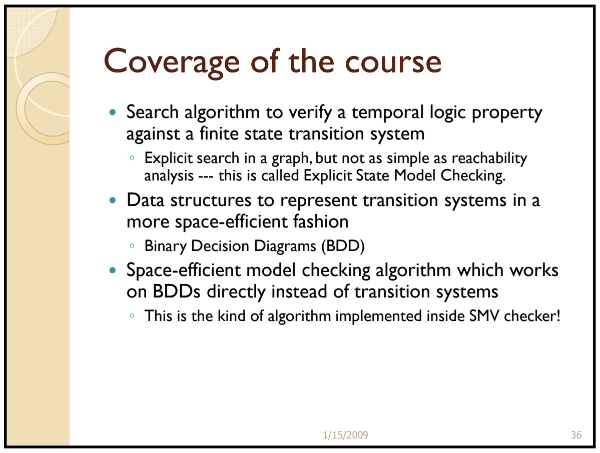
Flow of the course

- Modeling of system behaviors as finite state transition systems.
  - Such a transition system will not be provided by the system designer. System designer expresses design (e.g. a circuit) in a higher-level language from which the transition system is extracted.
- Specification Language for describing properties to verify --- Temporal Logics.
  - These properties need to be provided by the designer. However, in industry temporal logics are not directly used instead designers use a similar language with better syntactic sugar.



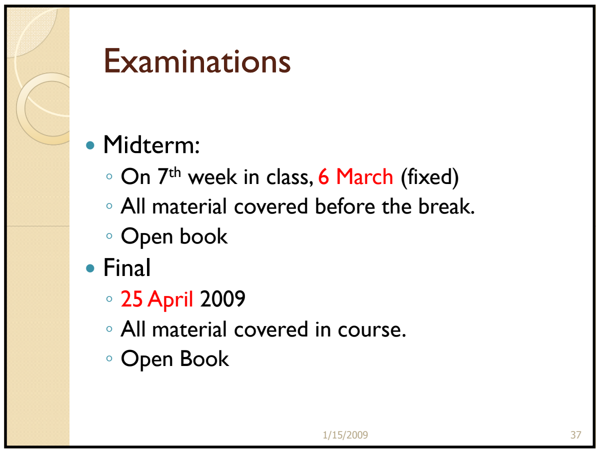
Flow of the course

- SMV Model Checker
  - The tool comes before the method !
  - We will use the front-end of the tool (the modeling language from which a transition system is automatically extracted) to describe designs & verify temporal properties against them.
  - The verification is automatic, so we can proceed without full understanding of the method.
- Case Studies
  - Realistic Case Studies from various domains (circuits, bus protocols etc) to get a better feel.



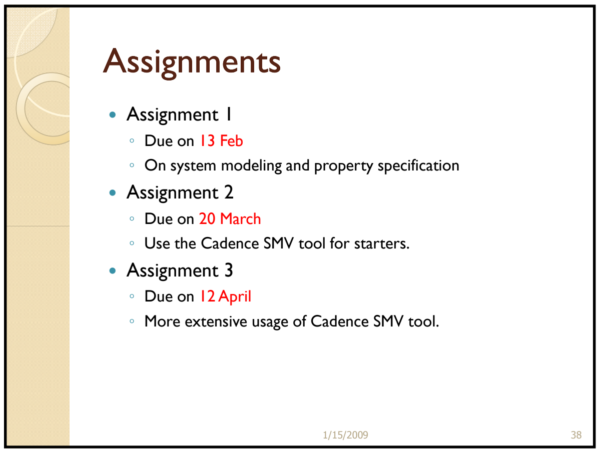
Coverage of the course

- Search algorithm to verify a temporal logic property against a finite state transition system
  - Explicit search in a graph, but not as simple as reachability analysis --- this is called Explicit State Model Checking.
- Data structures to represent transition systems in a more space-efficient fashion
  - Binary Decision Diagrams (BDD)
- Space-efficient model checking algorithm which works on BDDs directly instead of transition systems
  - This is the kind of algorithm implemented inside SMV checker!



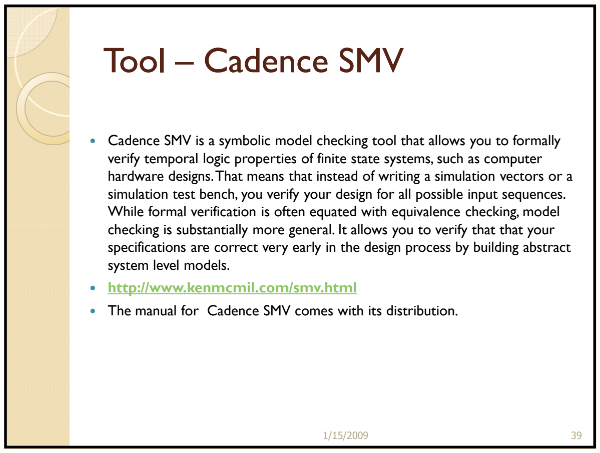
Examinations

- Midterm:
  - On 7th week in class, 6 March (fixed)
  - All material covered before the break.
  - Open book
- Final
  - 25 April 2009
  - All material covered in course.
  - Open Book



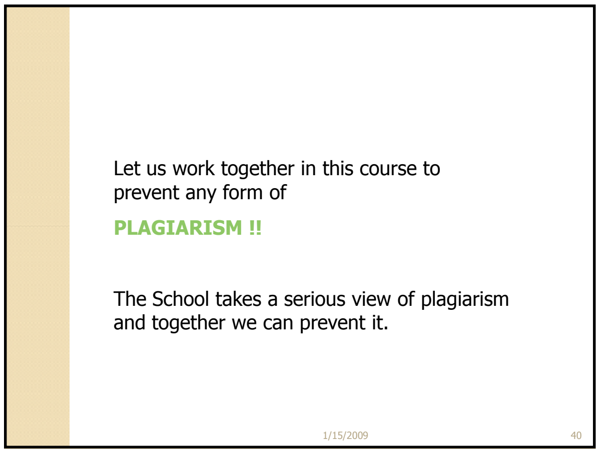
Assignments

- Assignment 1
  - Due on 13 Feb
  - On system modeling and property specification
- Assignment 2
  - Due on 20 March
  - Use the Cadence SMV tool for starters.
- Assignment 3
  - Due on 12 April
  - More extensive usage of Cadence SMV tool.



Tool – Cadence SMV

- Cadence SMV is a symbolic model checking tool that allows you to formally verify temporal logic properties of finite state systems, such as computer hardware designs. That means that instead of writing a simulation vectors or a simulation test bench, you verify your design for all possible input sequences. While formal verification is often equated with equivalence checking, model checking is substantially more general. It allows you to verify that that your specifications are correct very early in the design process by building abstract system level models.
- http://www.kenmcmil.com/smv.html
- The manual for Cadence SMV comes with its distribution.



Let us work together in this course to prevent any form of

PLAGIARISM !!

The School takes a serious view of plagiarism and together we can prevent it.