Overview of Model Checking

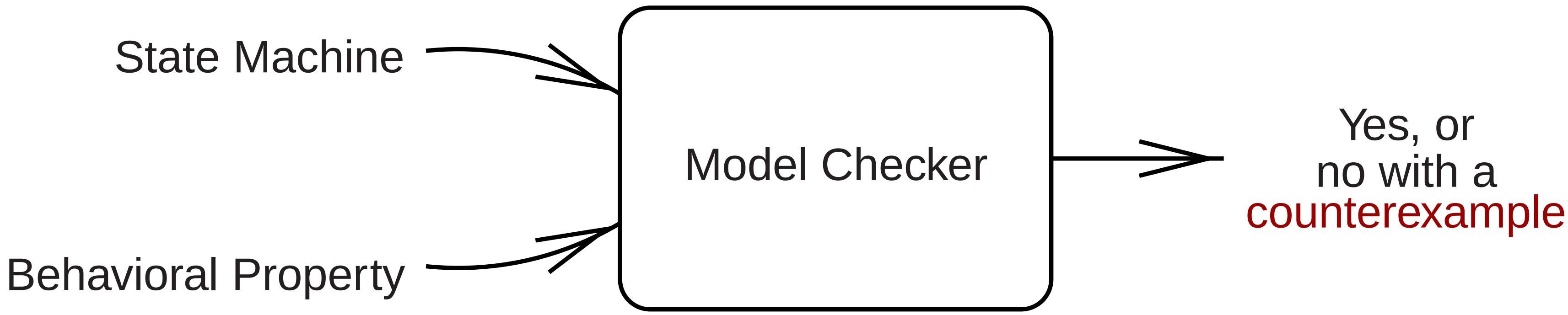
William Chan
wchan@cs.washington.edu

Department of Computer Science & Engineering
University of Washington

Outline

- Basics of model checking and temporal logic
- The *symbolic* variant
- Applications to specifications of reactive software
- Some lessons

Temporal-Logic Model Checking [Clarke & Emerson 81]



Some properties expressible in temporal logics:

- Error states not reached (invariant).
- Eventually ack for each request (liveness).
- Can always restart the machine.

Computation Tree Logic (CTL)

- The usual Boolean operators: $\wedge$, $\vee$, $\neg$, etc., plus:

  A: for all paths, E: for some path,
  G: globally on the path, F: in a future state on the path, and some more.

- Examples:

Error states not reached	$\mathrm{AG}\neg Err$
Eventually ack for each request	$\mathrm{AG}(req \rightarrow \mathrm{AF}ack)$
Can always restart the machine	$\mathrm{AGEF}restart$

Many other temporal logics exist.

- Decade-long debate: expressiveness and complexity.

Why Temporal Logics?

What's wrong with partial correctness and termination?

- Not suitable for reactive systems.
- e.g., cannot express liveness and fairness.

The introduction of temporal logic is an award-winning idea. (Pnueli)

Why model checking?

- "Easy" for finite-state machines.
- Fancy graph traversals, linear in # states & transitions.
- You already know how to evaluate AG$\neg Err$.

State Explosion

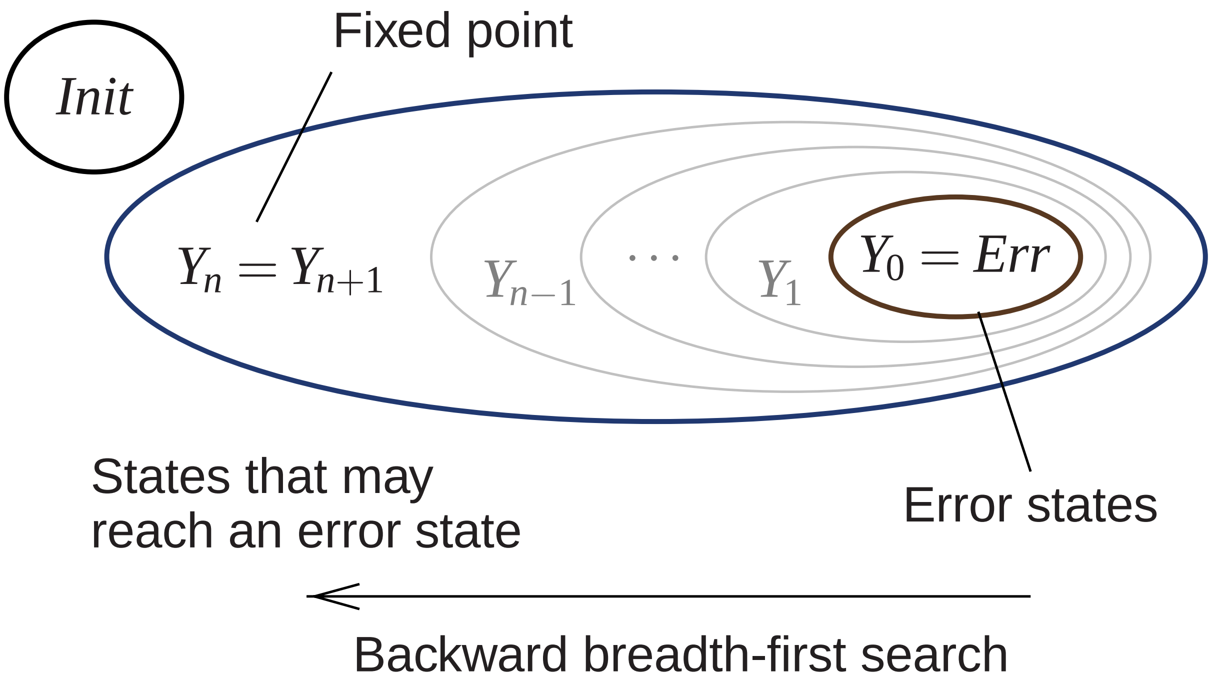
states grow exponentially with # components.

Attacks:

- Abstraction and composition (Cospan)
- Symmetry reduction (Murφ)
- Partial-order reduction (Spin)
- **Symbolic search** (SMV, VIS):
  - Represent a set of states symbolically without enumerating the states individually.

Invariant Checking as Set Manipulations

Compute $Y_{i+1} = Pre(Y_i) \cup Y_i$.   Check if $Y_n \cap Init = \emptyset$.



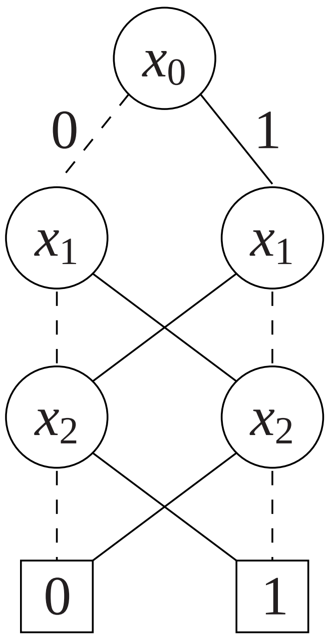
Symbolic Search [Burch et al. 90, Coudert et al. 89]

- Define Boolean state variables X.
  - e.g., define $x_{n-1}, x_{n-2}, \ldots, x_0$ for an n-bit integer.
- A set of states: a Boolean function $S(X)$.
  - e.g., $\neg x_0$ for the set of n-bit even integers.
- Set operations ($\cup$, $\cap$) becomes Boolean operations ($\vee$, $\wedge$).
- Transition relation: $R(X, X')$.
- Compute predecessors also using Boolean operations:

$$Pre(S) = \lambda X.\ \exists X'.\ S(X') \wedge R(X, X').$$

Binary Decision Diagrams (BDDs) [Bryant 86]

BDD for odd parity



- Generalization of binary decision trees to DAGs.
- Restrictions:
  - Reduced: isomorphic subgraphs merged.
  - Ordered: every path conforms to a common variable order.
- Properties:
  - Canonical.
  - Operations poly-time in BDD size.

BDDs are Wild

BDD size not direclty related to numbers of states or variables.

✓ Usually small. Some large state spaces (10^{20}) can be handled.

✓ Reduce the amount of manual abstraction needed.

✗ Sensitive to implementation details like variable order.

✗ Some well-known limitations (e.g., exponential size for $x > yz$).

✗ Few theoretical results known for general control systems.
Performance can be unpredictable.

Why Might BDDs Not Work Well for Software?

Common view:

	Hardware	Software
Data	Simple	Complex
States	Finite	Infinite
Concurrency	Synchronous (aka Simultaneous)	Asynchronous (aka Interleaving)
Strategy	Use BDDs	Abstract and search explicitly

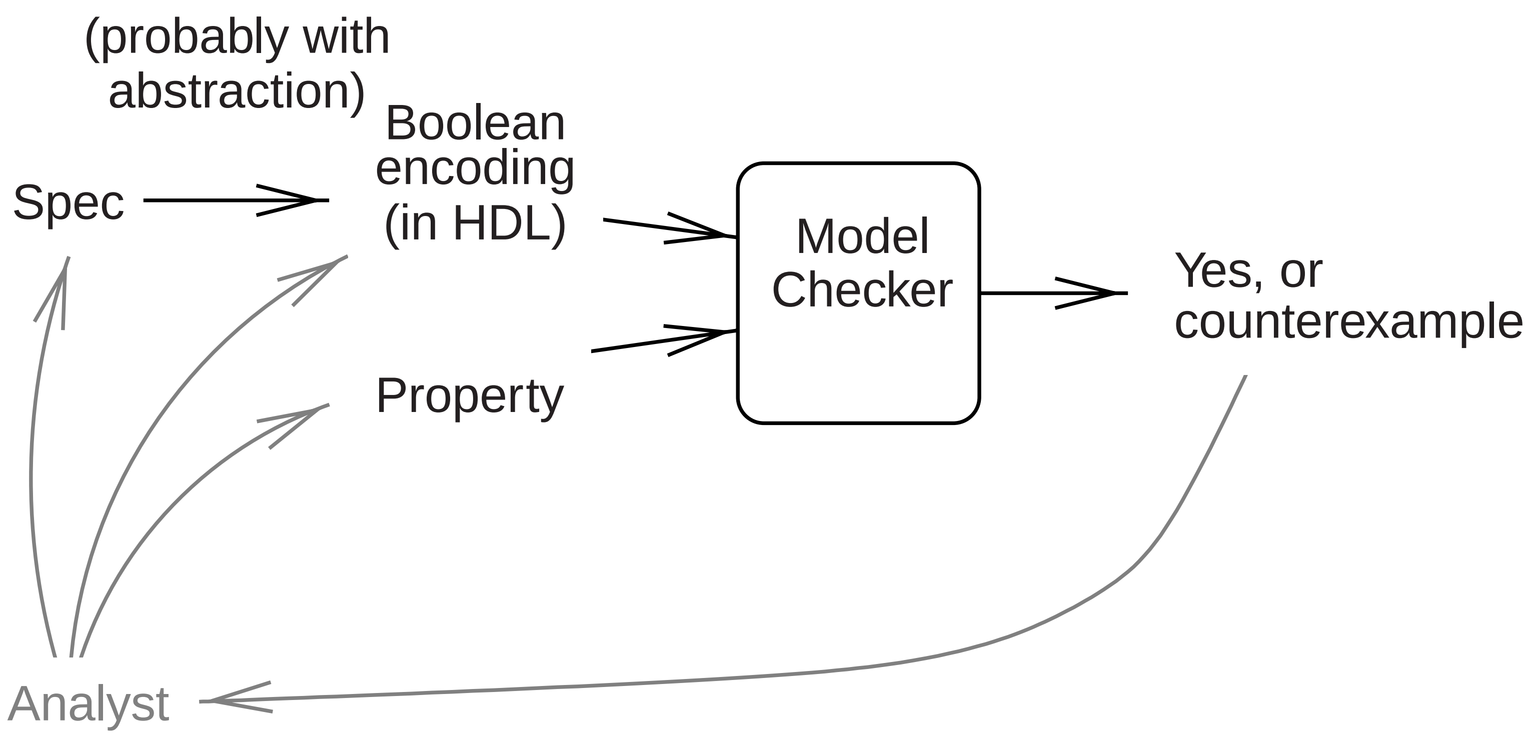
This may be true for software like multi-threaded programs, but . . . .

Consider Many Safety-Critical Software Specs

	Hardware	Spec	Multi-threaded Code
States	Finite	Finite (except numbers)	Possibly infinite
Data	Simple	Simple (except numbers)	Often complex
Concurrency	Synchronous	Synchronous	Asynchronous

Perhaps BDDs would work for such specs?

The Iterative Process

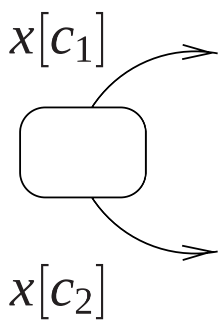


TCAS II

- Traffic Alert and Collision Avoidance System
  - Warns pilots of traffic. (Does not control airciraft.)
  - Issues vertical resolution advisories (RAs)
    e.g., Climb, Descend, Increase-Climb, Do Not Descend $>$ 500 ft/min.
  - Required on most commercial aircraft in USA.
  - One of the most complex systems on commercial aircraft.
- 400-page specification reverse-engineered from pseudo-code.
- Written in RSML [Leveson et al. 94], based on statecharts.
- Complexity in guarding conditions, not hierarchy or synchronization.

Analysis of TCAS II [FSE 96, TSE 98]

- Around 200 Boolean variables, 10^{60} states.
- Used model checker SMV. [McMillan 93]
- Domain-independent properties:
  - Transition consistency:
    $\mathrm{AG}\neg(x \wedge c_1 \wedge c_2)$



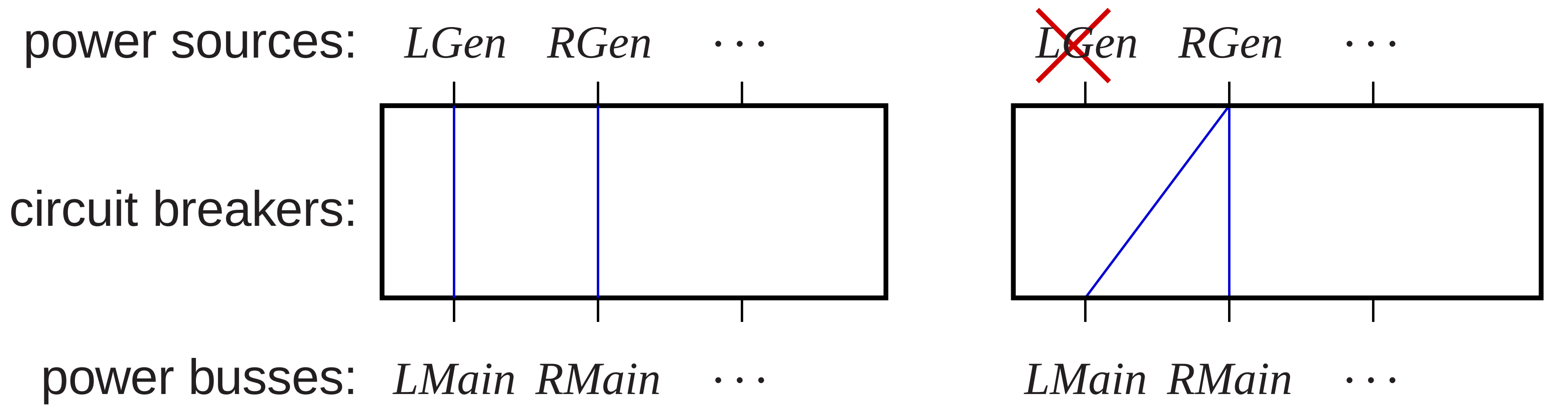
- Domain-dependent properties:
  - Descent inhibition:
    $\mathrm{AG}(Alt < 1000 \rightarrow \neg Descend)$
  - Output agreement:
    $\mathrm{AG}\neg(GoalRate \geq 0 \wedge Descend)$

EPD System

Electrical Power Distribution system used on Boeing 777.

- Distribute power from power sources to power busses via circuit breakers.
- Tolerate failures in power sources and circuit breakers.
- Prototype specification for research purposes.
- Exercised extensively in simulation.

Failure Handling



Analysis of EPD System

Joint work with David Jones and William Warner of Boeing. [ICSE 99]

- 90 Boolean variables, 10^{27} states.
- Fault tolerance
  - $\mathsf{AG}(\mathit{NoFailures} \rightarrow (\mathit{LMain} \wedge \mathit{RMain} \wedge \mathit{LBackup} \wedge \mathit{RBackup}))$.
  - $\mathsf{AG}(\mathit{AtMostOneFailure} \rightarrow (\mathit{LMain} \wedge \mathit{RMain}))$.
  - $\mathsf{AG}(\mathit{AtMostTwoFailures} \rightarrow (\mathit{LBackup} \vee \mathit{RBackup}))$.
- Found modeling errors and logical flaws.

Not as complex as TCAS II, but initial analysis failed.

Issues/Lessons

- BDDs can't handle complicated arithmetic.
  - Abstract
  - Bound and discretize
    - Not sound, but it's ok.
  - Combine with a constraint solver.
- Domain expertise is essential.
  - For domain-specific properties
  - For abstraction
    - But, again, doesn't need to be sound and complete.

Issues/Lessons (cont'd)

- Can help understand interactions among components.
- Forward vs. backward search
  - Lots of open questions.
  - For us, backward can be much better than forward.
- Synchronization affects efficiency.
- Can exploit high-level knowledge to do optimizations.
  - Can be much more efficient than using model checker as a black box.

SMC vs. Theorem Proving

Similarity: Pre is essentially the dual of WP.

Some key differences:

SMC	Theorem Proving
finite-state	no assumption
can be automated	need user guidance
efficient representations	readable representations
counterexamples (if false)	inspiring proofs (if true)

- MC is more useful because most systems are buggy!
- In MC, you gain confidence in correctness thru experiments.
- Much current work on *infinite-state* SMC.