Automata-Theoretic Model Checking Revisited

Moshe Y. Vardi

Rice University

Engines of Progress:
Semiconductor Technology

Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Result: Cost of memory and MIPS dropped roughly six orders of magnitude (10^6) over the last 40 years.

Semiconductor industry 10-year outlook: *there is no physical barrier to the transistor effect in silicon being the principal element in the semiconductor industry to for the next 10 years*.

But: Will the current business model for the semiconductor industry be viable for the next 10 years?

A Major Challenge: design productivity crisis

- complexity growth rate: 60% per year
- Productivity growth rate: 20% per year

Critical need: better design tools

Design Verification

A watershed event: Pentium FDIV bug, 1995

- Bug would result in occasional inaccuracies when doing floating-point arithmetic.
- Eventually Intel promised to replace all Pentiums with the fixed chip.
- Cost to Intel: $500M.

Verification methodology:

- *Traditional*: simulation on carefully chosen test sequences
- *New*: formal verification of entire state space

Formal Verification

- **Theorem proving**: formally prove that hardware is correct
  - requires a large number of expert users
  - application cycle slower than design cycle
- **Model checking**:

> uncommonly effective debugging tool
> - a systematic exploration of the design state space
> - good at catching difficult "corner cases"

Designs are Labeled Graphs

Key Idea: Designs can be represented as transition systems (finite-state machines)

Transition System: $M = (W, I, E, F, \pi)$
- W: states
- $I \subseteq W$: initial states
- $E \subseteq W \times W$: transition relation
- $F \subseteq W$: fair states
- $\pi : W \rightarrow Powerset(Prop)$: Observation function

Fairness: An assumption of “reasonableness” – restrict attention to computations that visit F infinitely often, e.g., “the channel will be up infinitely often”.

Runs and Computations

Run: $w_0, w_1, w_2, \ldots$

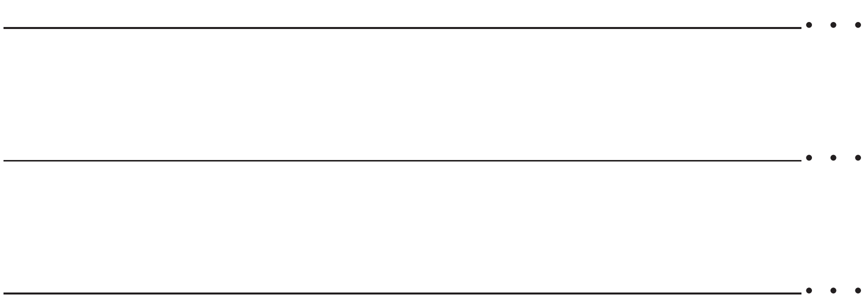
- $w_0 \in I$
- $(w_i, w_{i+1}) \in E$ for $i = 0, 1, \ldots$

Computation: $\pi(w_0), \pi(w_1), \pi(w_2), \ldots$

- $L(M)$: set of computations of M

Verification: System M satisfies specification ϕ –

- all computations in $L(M)$ satisfy ϕ.

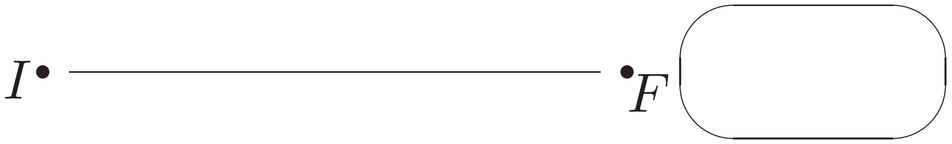


Algorithmic Foundations

Basic Graph-Theoretic Problems:

- *Reachability*: Is there a *finite* path from I to F?

- *Fair Reachability*: Is there an *infinite* path from I that goes through F infinitely often.



Note: These paths may correspond to error traces.

- *Deadlock*: A finite path from I to a state in which both $write_1$ and $write_2$ holds.
- *Livelock*: An infinite path from I along which snd holds infinitely often, but rcv never holds.

Computational Complexity

Complexity: Linear time

- *Reachability*: breadth-first search or depth-first search
- *Fair Reachability*: depth-first search

The fundamental problem of model checking: the *state-explosion* problem – from 10^{20} states and beyond.

The critical breakthrough: symbolic model checking

Specifications

Specification: properties of computations.

Examples:

- “No two processes can be in the critical section at the same time.” – *safety*
- “Every request is eventually granted.” – *liveness*
- “Every continuous request is eventually granted.” – *liveness*
- “Every repeated request is eventually granted.” – *liveness*

Temporal Logic

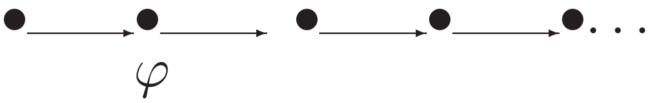
Linear Temporal logic (LTL): logic of temporal sequences (Pnueli'77)

Main feature: time is implicit

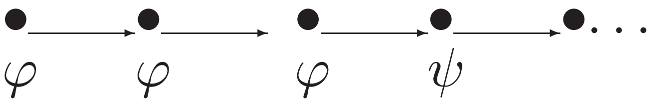
- *next* ϕ: ϕ holds in the next state.
- *eventually* ϕ: ϕ holds eventually
- *always* ϕ: ϕ holds from now on
- ϕ *until* ψ: ϕ holds until ψ holds.

Semantics

- $\pi, w \models next\ \varphi$ if w



- $\pi, w \models \varphi until\ \psi$ if w



Examples

- always not (CS$_1$ and CS$_2$): mutual exclusion (safety)
- always (Request implies eventually Grant): liveness
- always (Request implies (Request until Grant)): liveness
- always (always eventually Request) implies eventually Grant: liveness

Industrial standards based on LTL: PSL and SVA.

Automata on Finite Words

Nondeterministic Automata (NFA): $A = (\Sigma, S, S_0, \rho, F)$

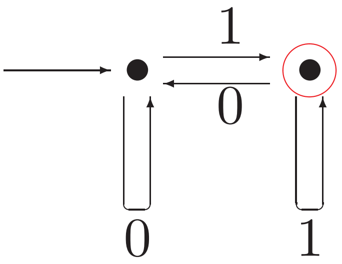
- *Alphabet*: Σ
- *States*: S
- *Initial states*: $S_0 \subseteq S$
- *Transition function*: $\rho : S \times \Sigma \to 2^S$
- *Accepting states*: $F \subseteq S$

Input word: $a_0, a_1, \ldots, a_{n-1}$

Run: $s_0, s_1, \ldots, s_n$

- $s_0 \in S_0$
- $s_{i+1} \in \rho(s_i, a_i)$ for $i \geq 0$

Acceptance: $s_n \in F$.



– ends with 1's

Automata on Infinite Words

Nondeterministic Büchi Automaton (NBA): $A = (\Sigma, S, S_0, \rho, F)$

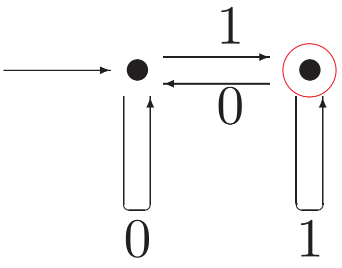
- *Alphabet*: Σ
- *States*: S
- *Initial states*: $S_0 \subseteq S$
- *Transition function*: $\rho : S \times \Sigma \rightarrow 2^S$
- *Accepting states*: $F \subseteq S$

Input word: $a_0, a_1, \ldots$

Run: $s_0, s_1, \ldots$

- $s_0 \in S_0$
- $s_{i+1} \in \rho(s_i, a_i)$ for $i \geq 0$

Acceptance: F visited infinitely often



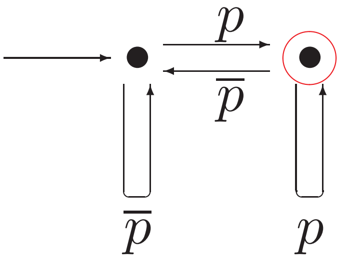
– infinitely many 1's

Temporal Logic vs. Automata

Paradigm: Compile high-level logical specifications into low-level finite-state language

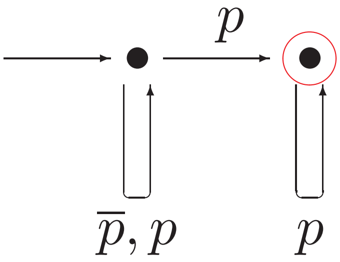
The Compilation Theorem: V.&Wolper, 1983
Given an LTL formula ϕ, one can construct an automaton A_ϕ such that a computation σ satisfies ϕ if and only if σ is accepted by A_ϕ. Furthermore, the size of A_ϕ is at most exponential in the length of ϕ.

always eventually p:



– infinitely many p's

eventually always p:



– finitely many $\overline{p}$'s

Model Checking

The following are equivalent:

- M satisfies ϕ
- all computations in $L(M)$ satisfy ϕ
- $L(M) \subseteq L(A_\phi)$
- $L(M) \cap \overline{L(A_\phi)} = \emptyset$
- $L(M) \cap L(A_{\neg\phi}) = \emptyset$
- $L(M \times A_{\neg\phi}) = \emptyset$

In practice: To check that M satisfies ϕ, compose M with $A_{\neg\phi}$ and check whether the composite system has a reachable (fair) path, that is, a reachable SCC with an accepting state.

Intuition: $A_{\neg\phi}$ is a "watchdog" for "bad" behaviors. A reachable (fair) path means a bad behavior.

Catching Bugs with A Lasso