Model Checking as A Reachability Problem

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 the year 2020*.

But: Will the current *business model* for the semiconductor industry be viable until 2020?

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: $w_0, w_1, w_2, \ldots$

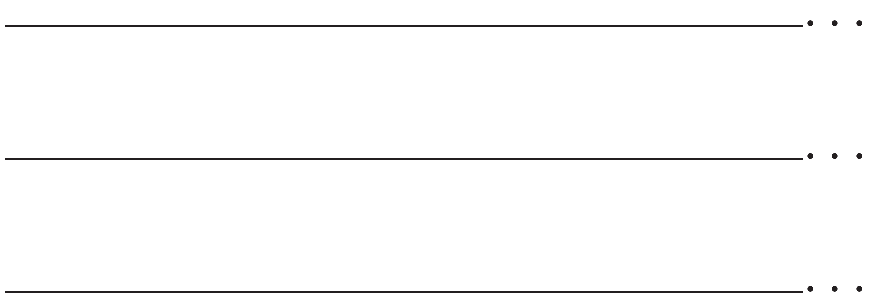
- $w_0 \in I$
- $(w_i, w_{i+1}) \in E$ for $i = 0, 1, \ldots$

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.

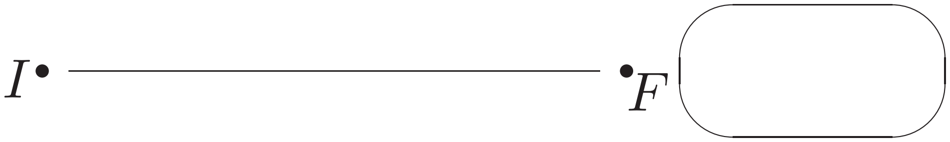


Algorithmic Foundations

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 (find a reachable SCC with fair states)

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

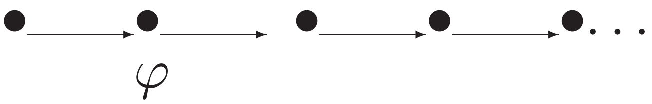
Linear Temporal logic (LTL): logic of temporal sequences (Pnueli'77)

Main feature: time is implicit

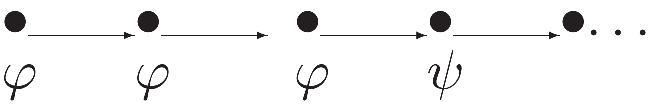
- *next* ϕ: ϕ holds in the next state.
- *eventually* ϕ: ϕ holds eventually
- *always* ϕ: ϕ holds from now on
- ϕ *until* ψ: ϕ holds until ψ holds.

Semantics

- $\pi, w \models$ *next* φ if w



- $\pi, w \models \varphi$ *until* ψ 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

Automata on Finite Words

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

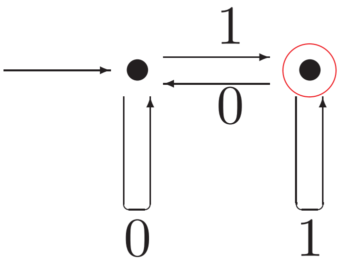
- *Alphabet*: Σ
- *States*: S
- *Initial states*: $S_0 \subseteq S$
- *Transition function*: $\rho : S \times \Sigma \to 2^S$
- *Accepting states*: $F \subseteq S$

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

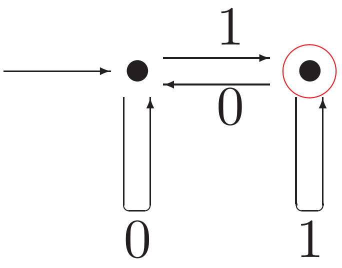
- *Alphabet*: Σ
- *States*: S
- *Initial states*: $S_0 \subseteq S$
- *Transition function*: $\rho : S \times \Sigma \rightarrow 2^S$
- *Accepting states*: $F \subseteq S$

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



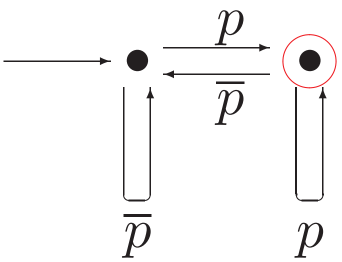
– infinitely many 1's

Temporal Logic vs. Automata

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

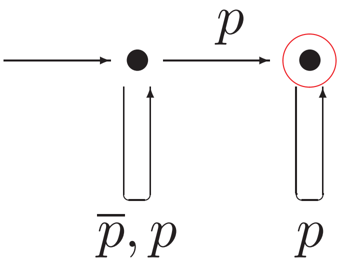
The Compilation Theorem: V.&Wolper, 1983
Given an LTL formula ϕ, one can construct an automaton A_ϕ such that a computation σ satisfies ϕ if and only if σ is accepted by A_ϕ. Furthermore, the size of A_ϕ is at most exponential in the length of ϕ.

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 states.

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

Catching Bugs with A Lasso

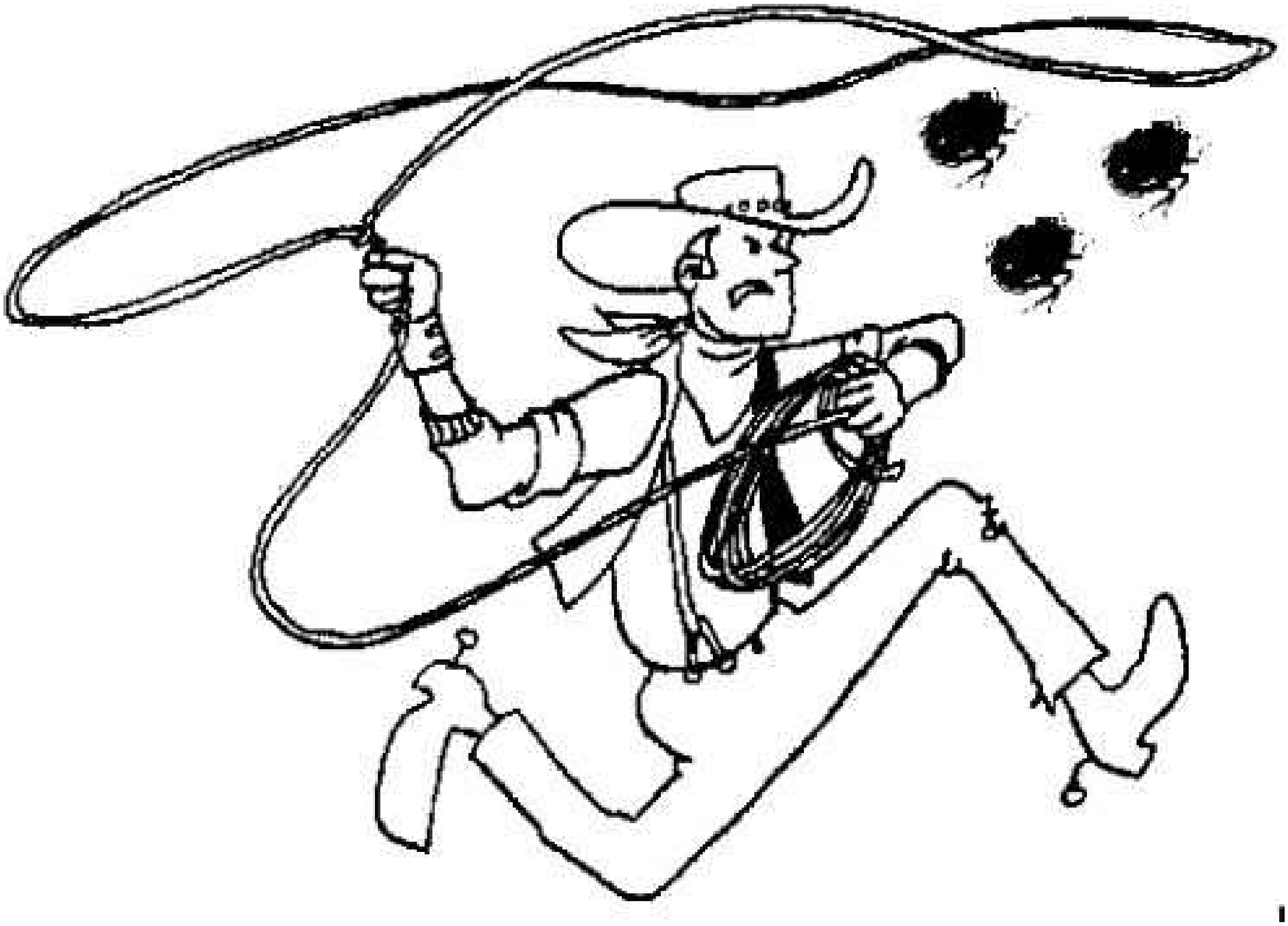


Figure 1: Ashutosh’s blog, November 23, 2005

State of The Art: 1996

Two LTL model checkers: **Spin**, **Cadence SMV**.

Spin: Explicit-State Model Checker

- *Automata Generation*: GPVW'95 (optimized version of VW)
- *Lasso Detection*: nested depth-first search–(*NDFS*) (CVWY'90)

SMV: Symbolic (BDD-based) Model Checker

- *Automata Generation*: CGH'94 (optimized symbolic version of VW)
- *Lasso Detection*: nested fixpoints–*NF* (EL'86)

Lasso Detection:

- *NDFS*: one DFS to find reachable accepting states, second DFS to find cycle from accepting states.

- *NF*: inner fixpoint to find states that can reach accepting states, outer fixpoint to delete states that cannot reach accepting states.

Symbolic Model Checking

Basic idea:

- Encodes states as bit vectors
- Represent set of states symbolically
- Represent transitions symbolically
- Reason symbolically

Example: 3-bit counter

- *Variables*: v_0, v_1, v_2
- *Transition relation*: $R(v_0, v_1, v_2, v'_0, v'_1, v'_2)$
  - $v'_0 \Leftrightarrow \neg v_0$
  - $v'_1 \Leftrightarrow v_0 \oplus v_1$
  - $v'_2 \Leftrightarrow (v_0 \wedge v_1) \oplus v_2$

That Was Then, This Is Now

Summary: We know more, but we are more confused!

Many Issues:

- Automata generation
- Deterministic vs. nondeterministic automata
- Explicit and symbolic lasso-detection algorithms
- SAT-based algorithms
- Büchi properties

Bottom Line: No simple recipe for superior performance!

Automata Generation

History:

- VW’83: exponential translation.
- GPVW’95: demand-driven state generation, avoid exponential blowup in many cases.
- DGV’99: light-weight Boolean reasoning to avoid redundant states.
- Cou’99: accepting conditions on transitions, BDDs for Boolean reasoning.
- SB’00,EH’00: pre-generation rewriting, post-generation minimization.
- V’94, GO’01: alternating automata as intermediate step
- GL’02,Thi’02,Fri’03,ST’03: more optimizations.

Question: “Mirror, mirror, on the wall, Who in this land is fastest of all?”

Who Is The Fastest?

Difficult to Say!

- Papers focus on minimizing automata size, but size is just a proxy. What about model checking time and memory? (Exc., ST'03.)

- Tools often return incorrect answers! (Best tool: *SPOT*)

- No tool can handle the formula

$$\begin{gathered}((GFp_0 \rightarrow GFp_1)\&(GFp_2 \rightarrow GFp_0)\& \\ (GFp_3 \rightarrow GFp_2)\&(GFp_4 \rightarrow GFp_2)\& \\ (GFp_5 \rightarrow GFp_3)\&(GFp_6 \rightarrow GF(p_5 \vee p_4))\& \\ (GFp_7 \rightarrow GFp_6)\&(GFp_1 \rightarrow GFp_7)) \rightarrow GFp_8\end{gathered}$$

Specialized tool generates 1281 states!

- Which is better: Büchi automata or *generalized* Büchi automata? It is automata generation vs. model checking.

- LTL is weak, theoretically and practically! What about industrial languages such as PSL?

Note: BDDs are essentially deterministic automata. BDD tools can handle BDDs with *millions* of nodes!

Comparison on Counter Formulas

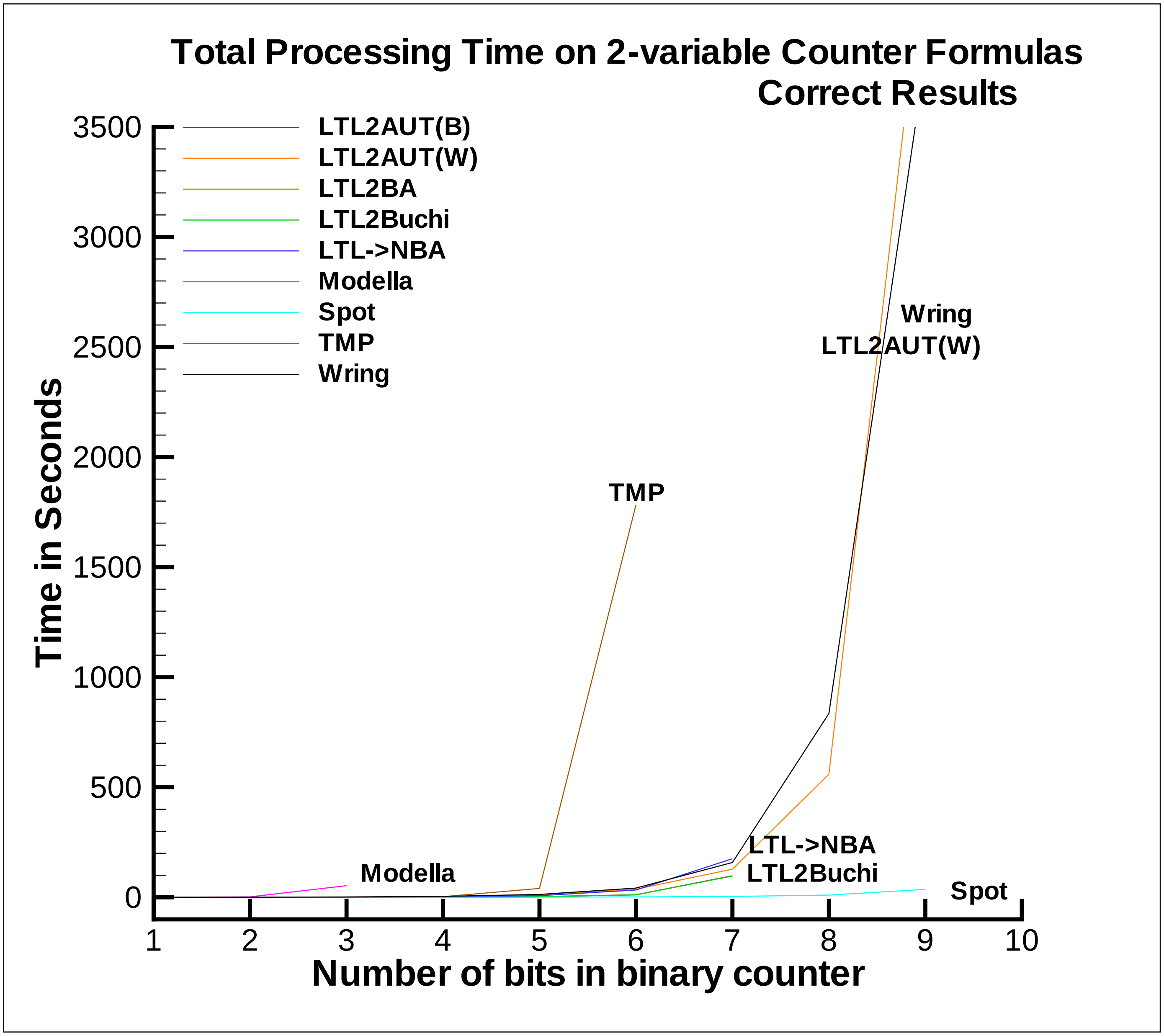


Figure 2: Translators' comparison, by K. Rozier

Comparison on Random Formulas

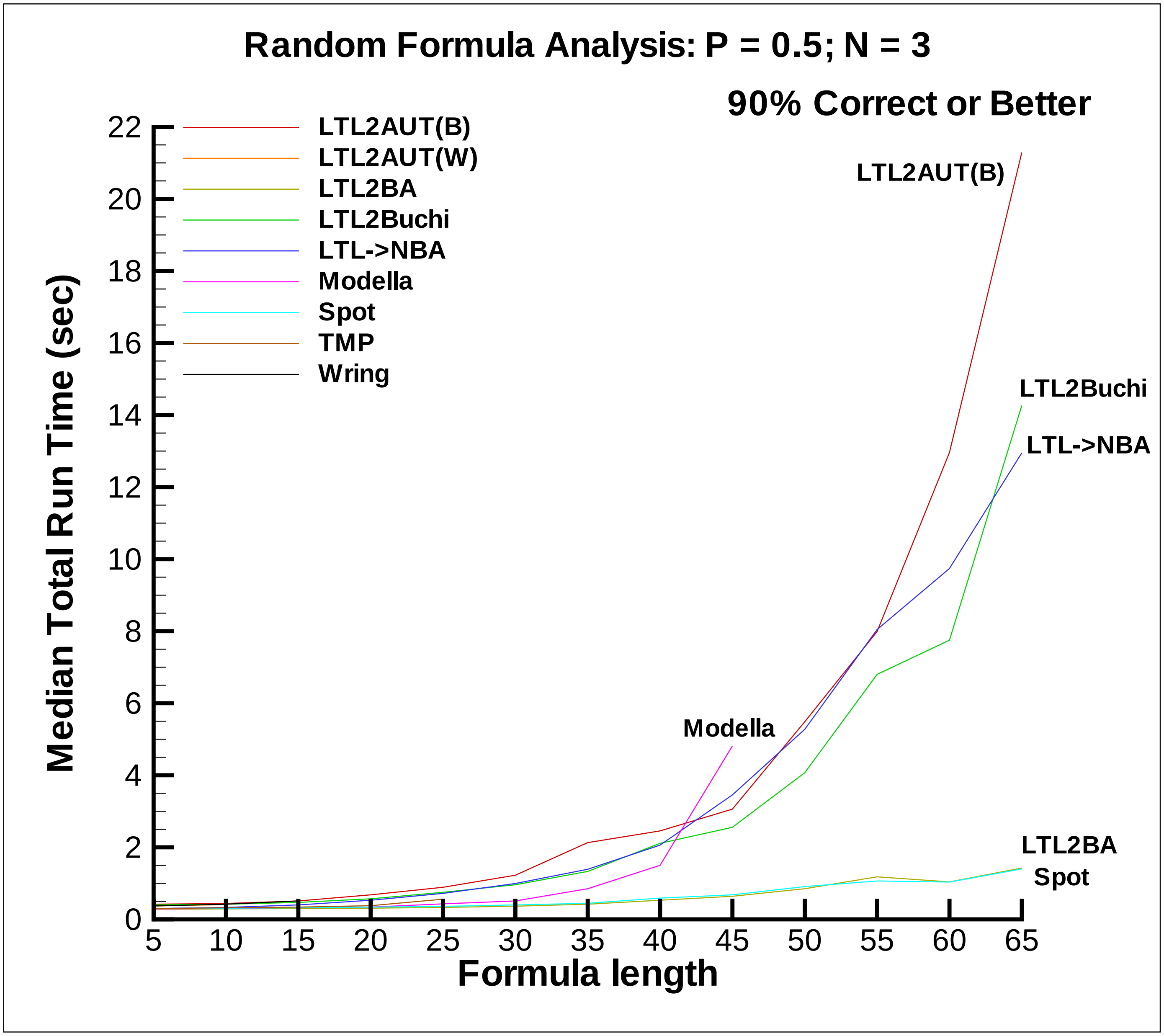


Figure 3: Translators’ comparison, by K. Rozier

Temporal Logic: From Theory to Practice

- Pnueli, 1977: focus on ongoing behavior, rather than input/output behavior – *LTL*
- Intel Design Technology, 2001: LTL augmented with regular expressions, multiple clocks and resets – *ForSpec*
- IBM Haifa Research Lab, 2001: CTL augmented with regular expressions – *Sugar*
- IEEE Standard, 2005: LTL augmented with regular expressions, multiple clocks and resets – *PSL*
- IEEE Standard, 2005 LTL augmented with regular expressions, multiple clocks and resets – *SVA*

Today: Support by many CAD companies for both PSL and SVA – major industrial application of Büchi automata!

Is Determinism Bad?

Key Observation: Most properties are *safety* properties, i.e., cycles of lassos not needed.

- KV'99: Replace NBA by NFA, use simpler model checking algs (NDFS→DFS/BFS, NF→F)

Surprise: Not used by tools other than VIS.

Furthermore: Should we use NFA or DFA?

- DFA can be exponentially larger,
- but search space is smaller!

AEFKV'05: For SAT-based model checking, DFA are better than NFA.

- *Reason*: SAT solver searches for a trace, but not for accepting automaton run.

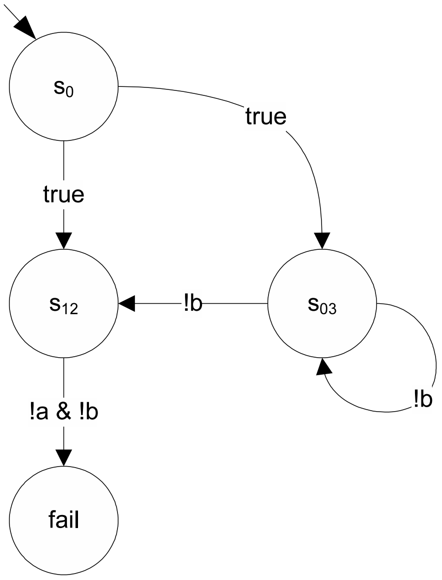
From LTL to DFA

Problem: Blowup is double exponential! (KV’98)

Solution: Represent DFA symbolically! (AEFKV’05).

Example: $next\ a\ wuntil\ next\ b$.

Explicit NFA:



DFA in Verilog

```
reg s0, s12, s03;
wire fail, sysclk;
assign fail = s12 && !a & !b;
initial begin
    s0 = 1’b1; s12 = 1’b0; s03 = 1’b0;
end
always @(posedge sysclk) begin
    s0   <= 1’b0;
    s12   <= s0 || !b && s03;
    s03 <= s0 || s03 && !b;
end
```

Size of Symbolic DFA: Linear in size of explicit NFA.

Explicit Lasso Detection

NDFS:

- Improvements by GH'93 and HPY'96: early termination, hash table, partial-order reduction (implemented in *Spin*)

- Improvement by SE'04: early termination with less auxiliary memory (not implemented in *Spin*)

A Competing Algorithm: SCC decomposition (Cou99, GH'04)

> **Question**: "Mirror, mirror, on the wall, Who in this land is fastest of all?"

It Depends! SE'04, CDP'05

- *NDFS* can use bit-state hashing, can handle very large state spaces.

- SCC decomposition is better for main-memory execution.

- Cou99 and GH'04 each has some merits.

Fair Termination

Fair Transition System: $M = (W, I, E, F, \pi)$

- W: state set (not necessarily finite)
- $I \subseteq W$: initial state set
- $E \subseteq W^2$: transition relation
- $F \subseteq W$: fair state set
- π: observation function

Fair path: infinite path in M that visits F infinitely often.

Fair termination: no fair path in M from I

- Checking livelock can be reduced to fair termination.
- Model checking LTL properties can be reduced to fair termination.

Note: On finite fair transition systems fair termination is the dual of lasso detection.

Fair-Termination Checking

$M = (W, I, E, F, \pi)$

Definition: Let $X, Y \subseteq W$. $until(X, Y)$ is the set of states in X that can properly reach Y, while staying in X.

EC'80: characterization of fair termination

```
$Q \leftarrow W$
while change do
        $Q \leftarrow Q \cap until(Q, Q \cap F)$
endwhile
return $(I \cap Q = \emptyset)$
```

Intuition: Repeatedly delete states that cannot be on a fair path because they cannot properly reach F event once.

EL'86: quadratic algorithm for fair termination – *NF*.

BCMDH'90: can be implemented by means of BDDs.

NF vs. OWCTY

FFKVY'01: *OWCTY*

```
$Q \leftarrow W$
while change do
    while change do
        $Q \leftarrow Q \cap pre(Q)$
    endwhile
    $Q \leftarrow Q \cap until(Q, Q \cap F)$
endwhile
return $(I \cap Q = \emptyset)$
```

Intuition: Dead-end states cannot lie on a fair path.

Question: "Mirror, mirror, on the wall, Who in this land is fastest of all?"

- FFKVY'01: *OWCTY* can be linear, when *NF* is quadratic.
- SRB'02: *OWCTY* may incur linear overhead over *NF*.

Bottom Line: Inconclusive!

Breaking The Quadratic Barrier

Note: Both *NF* and *OWCTY* may involve a $O(n^2)$ number of symbolic operations.

Question: Can we do better?

- *Lockstep*: $O(n \log n)$ symbolic operations. (BGS'00)
- *SCC–Find*: $O(n)$ symbolic operations. (GPP'03)

Theory vs. Practice:

- RBS'00: *Lockstep* is not better than *NF*.
- No experimental evaluation of *SCC-Find*.

Hybrid Approach: Explicit Automata, Symbolic Systems

Basic Intuition:

- Systems are typically large–represent them symbolically.

- Automata are typically small–represent them explicitly.

Property-Driven Partitioning:

- System states–W, automaton states–Q
- Product states–$W \times Q$
- Partition $P \subseteq W \times Q$ into

$P_q = \{w : (w, q) \in P$, $q \in Q$

Applicability: all symbolic algorithms

- Replace single BDD by array of BDDs

Effectiveness: can be exponentially faster than standard symbolic algorithms (STV'05).

SAT-Based Algorithms

Bounded Model Checking: Is a bad state reachable in k steps? (BCCZ'00)

$$I(\mathbf{X}) \wedge TR(\mathbf{X}, \mathbf{X}) \wedge \ldots \wedge TR(\mathbf{X}k - 1, \mathbf{X}) \wedge B(\mathbf{X})$$

Question: How to encode LTL property?

Many Answers: CRS'04, LBHJ'05

Basic weakness: Ignore work on LTL translation.

- Treat automata as graphs.
- But nodes have "inner structure" – they are sets of subformulas.

Also: Different approaches to represent lassos.

- Add cycle variables (LBHJ'05)
- Reduce liveness to safety (BAS'02)

Question: Is there fastest method?

Büchi Properties

Motivation: Use Büchi automata to specify desired behavior, e.g., *COSPAN*.

The following are equivalent:

- M satisfies A
- $L(M) \subseteq L(A)$
- $L(M) \cap (\Sigma^\omega - L(A)) = \emptyset$
- $L(M) \cap L(A^c) = \emptyset$
- $L(M \times A^c) = \emptyset$

Complementation: $L(A^c) = \Sigma^\omega - L(A)$

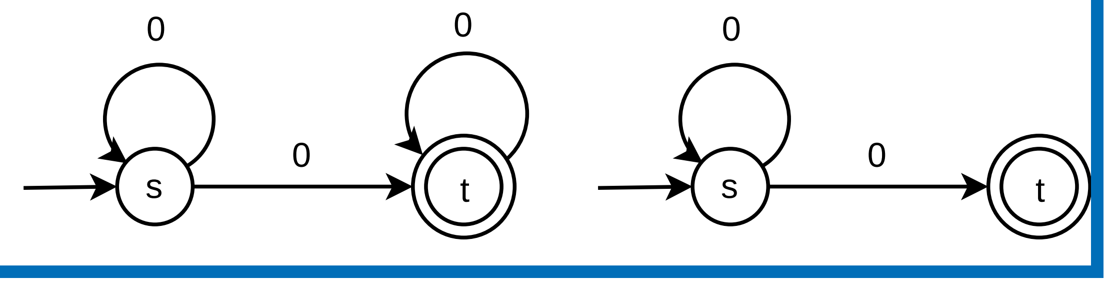
Known: Büchi complementation is hard!

- *COSPAN* requires property automata to be deterministic.

Recall: NFA complementation is exponential (subset construction), but we can complement NFAs with hundreds of states, in spite of exponential blowup (TV'05).

Büchi Complementation

Problem: subset construction fails!



History

- Büchi'62: doubly exponential construction.
- SVW'85: 2^{16n^2} upper bound
- Saf'88: n^{2n} upper bound
- Mic'88: $(n/e)^n$ lower bound
- KV'97: $(6n)^n$ upper bound
- GKSV'03: optimized implementation of KV'97
- FKV'04: $(0.97n)^n$ upper bound
- Yan'06: $(0.76n)^n$ lower bound
- Schewe'09: $(0.76n)^n$ upper bound

Question: Have we reached practicality?

Complementation of Random Automata

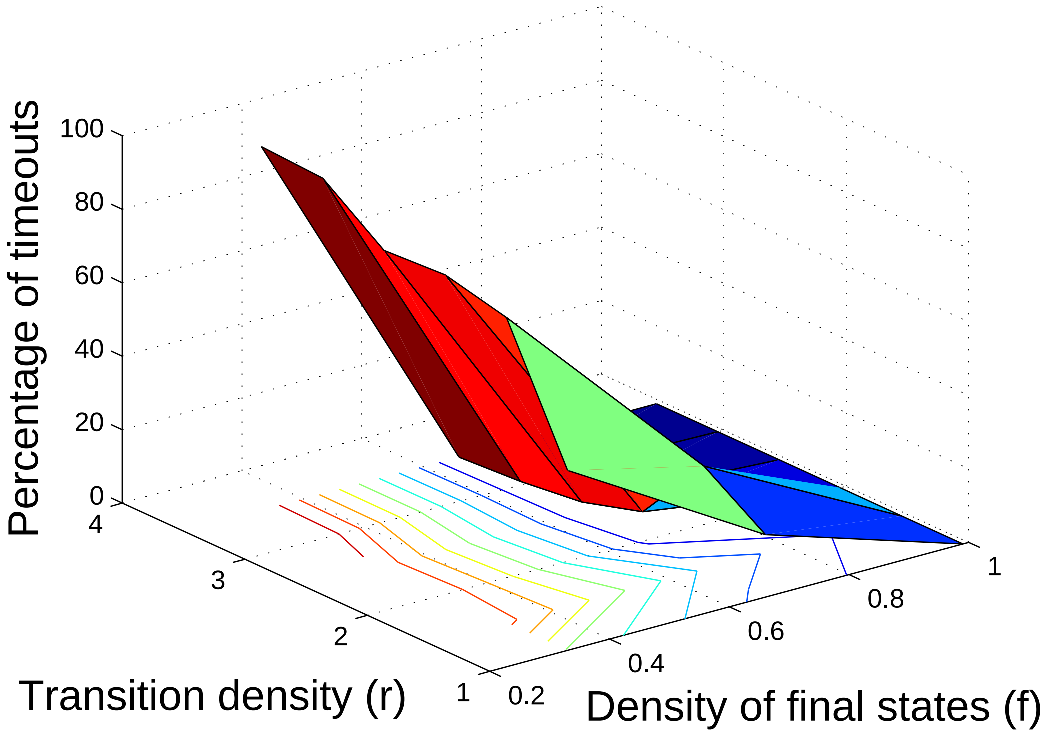


Figure 4: Wring timeouts, by D. Tabakov

Timeout: 3600 sec.
States: Six!

Recent improvements: TV'07,DR'07 (up to 30 states for difficult automata)

Summary

History:

- It took 10 years from conception to implementation.
- Much progress in the following 10 years, leading to industrial adoption.

Challenge:

- Many algorithms.
- Relative merits not always clear.
- Probably no "best" algorithm.

Advocated Approach:

- Abandon "winner-takes-all" approach.
- Borrow from AI *a portfolio approach to algorithm selection*, in which we match algorithms to problem instances.
- E.g., adapt algorithm to property (BRS'99).