The Siren Song of Temporal Synthesis

Moshe Y. Vardi

Rice University
Verification

Model Checking:

- **Given**: Program $P$, Specification $\varphi$.
- **Task**: Check that $P$ satisfies $\varphi$

Success:

- **Algorithmic methods**: temporal specifications and finite-state programs.
- **Also**: Certain classes of infinite-state programs
- **Tools**: SMV, SPIN, SLAM, etc.
- **Impact** on industrial design practice is increasing.

Problems:

- Designing $P$ is hard and expensive.
- Redesigning $P$ when $P$ does not model $\varphi$ is hard and expensive.
Automated Design

Basic Idea:

- Start from spec $\varphi$, design $P$ s.t. $P$ satisfies $\varphi$.

  Advantage:
  - No verification
  - No re-design

- Derive $P$ from $\varphi$ algorithmically.

  Advantage:
  - No design

**In essence**: Declarative programming taken to the limit.

Harel, 2008: “Can Programming be Liberated, Period?”
Program Synthesis

The Basic Idea: “Mechanical translation of human-understandable task specifications to a program that is known to meet the specifications.”


- Prove realizability of function, e.g., \( (\forall x)(\exists y)(Pre(x) \rightarrow Post(x, y)) \)

- Extract program from realizability proof.

Classical vs. Temporal Synthesis:

- **Classical:** Synthesize transformational programs

- **Temporal:** Synthesize programs for ongoing computations (protocols, operating systems, controllers, robots, etc.)
Temporal Logic

Linear Temporal logic (LTL): logic of temporal sequences (Pnueli, 1977)

**Main feature**: time is implicit

- **next** $\varphi$: $\varphi$ holds in the next state.
- **eventually** $\varphi$: $\varphi$ holds eventually
- **always** $\varphi$: $\varphi$ holds from now on
- **$\varphi$ until** $\psi$: $\varphi$ holds until $\psi$ holds.

**Semantics**: over infinite traces

- $\pi, w \models next \varphi$ if $w \bullet \bullet \bullet \bullet \bullet \bullet \ldots$
- $\varphi \models until \psi$ if $w \bullet \bullet \bullet \bullet \bullet \bullet \bullet \ldots$

Examples

• always not (CS$_1$ and CS$_2$): mutual exclusion (safety)

• always (Request implies eventually Grant): liveness

• always (Request implies (Request until Grant)): liveness
Synthesis of Ongoing Programs

*Spec*: Temporal logic formulas

**Early 1980s**: Satisfiability approach (Wolper, Clarke+Emerson, 1981)

- *Given*: $\varphi$

- *Satisfiability*: Construct model $M$ of $\varphi$

- *Synthesis*: Extract $P$ from $M$.

**Example**: 

\[
\text{always } \ (\text{odd }\rightarrow\text{ next }\neg\text{odd}) \land \\
\text{always } \ (\neg\text{odd }\rightarrow\text{ next odd})
\]
Reactive Systems

**Reactivity**: Ongoing interaction with environment (Harel+Pnueli, 1985), e.g., hardware, operating systems, communication protocols, robots, etc. (also, *open systems*).

**Example**: Printer specification –

- \( J_i \) - job \( i \) submitted, \( P_i \) - job \( i \) printing.

- **Safety**: two jobs are not printing together
  
  always \( \neg (P_1 \land P_2) \)

- **Liveness**: every jobs is eventually printed
  
  always \( \land_{i=1}^{2} (J_i \rightarrow eventually P_i) \)
Satisfiability and Synthesis

**Specification Satisfiable?** Yes!

*Model* $M$: A single state where $J_1$, $J_2$, $P_1$, and $P_2$ are all false.

**Extract program from** $M$? No!

*Why?* Because $M$ handles only one input sequence.

- $J_1, J_2$: input variables, controlled by environment
- $P_1, P_2$: output variables, controlled by system

**Desired**: a system that handles *all* input sequences.

**Conclusion**: Satisfiability is *inadequate* for synthesis.
Realizability

\( I \): input variables
\( O \): output variables

Game:
- \textbf{System}: choose from \( 2^O \)
- \textbf{Env}: choose from \( 2^I \)

Infinite Play:
\( i_0, i_1, i_2, \ldots \)
\( 0_0, 0_1, 0_2, \ldots \)

Infinite Behavior: \( i_0 \cup o_0, i_1 \cup o_1, i_2 \cup o_2, \ldots \)

Win: Behavior satisfies spec.

Specifications: LTL formula on \( I \cup O \)

Strategy: Function \( f : (2^I)^* \rightarrow 2^O \)

\textbf{Realizability:} Abadi+Lamport+Wolper, 1989
Pnueli+Rosner, 1989
Existence of winning strategy for specification.

\textbf{Desideratum:} A \textit{universal} plan! \textbf{Why:} Autonomy!
Church’s Problem

Church, 1957: Realizability problem wrt specification expressed in MSO (monadic second-order theory of one successor function)

Büchi+Landweber, 1969:

- Realizability is decidable.
- If a winning strategy exists, then a finite-state winning strategy exists.
- Realizability algorithm produces finite-state strategy.


**Question**: LTL is subsumed by MSO, so what did Pnueli and Rosner do?

**Answer**: better algorithms!
Strategy Trees

**Infinite Tree:** $D^* (D - \text{directions})$

- **Root:** $\varepsilon$; **Children:** $xd, x \in D^*, d \in D$

**Labeled Infinite Tree:** $\tau: D^* \rightarrow \Sigma$

**Strategy:** $f: (2^I)^* \rightarrow 2^O$

*Rabin’s insight:* A strategy is a labeled tree with directions $D = 2^I$ and alphabet $\Sigma = 2^O$.

**Example:** $I = \{p\}, O = \{q\}$

Winning: Every branch satisfies spec.

**Rabin, 1972:** Finite-state automata on infinite trees
Emptiness of Tree Automata

\[ L(A) = \emptyset \]

Emptiness of Automata on Finite Trees: PTIME test (Doner, 1965)

Emptiness of Automata on Infinite Trees: Difficult

- Rabin, 1969: non-elementary
- Hossley+Rackoff, 1972: 2EXPTIME
- Rabin, 1972: EXPTIME
- Emerson, V.+Stockmeyer, 1985: In NP
- Emerson+Jutla, 1991: NP-complete
Rabin’s Realizability Algorithm

\textbf{REAL}(\phi):

\begin{itemize}
  \item Construct Rabin tree automaton \(A_\phi\) that accepts all winning strategy trees for spec \(\phi\).
  \item Check non-emptiness of \(A_\phi\).
  \item If nonempty, then we have realizability; extract strategy from non-emptiness witness.
\end{itemize}

\textbf{Complexity}: non-elementary

\textit{Reason}: \(A_\phi\) is of non-elementary size for spec \(\phi\) in MSO.
Post-1972 Developments

- **Pnueli, 1977**: Use LTL rather than MSO as spec language.

- **V. + Wolper, 1983**: Elementary (exponential) translation from LTL to automata.

- **Safra, 1988**: Doubly exponential construction of tree automata for strategy trees wrt LTL spec (using V. + Wolper).


- **Rosner, 1990**: Realizability is 2EXPTIME-complete.
Impractical! 2EXPTIME is a horrible complexity.

Response:

- 2EXPTIME is just worst-case complexity.
- 2EXPTIME lower bound implies a doubly exponential bound on the size of the smallest strategy; thus, hand design cannot do better in the worst case.

Real Challenge: very difficult algorithmics!
Classical AI Planning

Deterministic Finite Automaton (DFA)

\[ A = (\Sigma, S, s_0, \rho, F) \]

- **Alphabet:** \( \Sigma \)
- **States:** \( S \)
- **Initial state:** \( s_0 \in S \)
- **Transition function:** \( \rho : S \times \Sigma \rightarrow S \)
- **Accepting states:** \( F \subseteq S \)

**Input word:** \( a_0, a_1, \ldots, a_{n-1} \)  **Run:** \( s_0, s_1, \ldots, s_n \)

- \( s_{i+1} = \rho(s_i, a_i) \) for \( i \geq 0 \)

**Acceptance:** \( s_n \in F \).

**Planning Problem:** Find word leading from \( s_0 \) to \( F \).

- **Realizability:** \( L(A) \neq \emptyset \)
- **Program:** \( w \in L(A) \)
Dealing with Nondeterminism

**Nondeterministic Finite Automaton (NFA)**
\[ A = (\Sigma, S, s_0, \rho, F) \]
- **Alphabet**: \( \Sigma \)
- **States**: \( S \)
- **Initial state**: \( s_0 \in S \)
- **Transition function**: \( \rho : S \times \Sigma \rightarrow 2^S \)
- **Accepting states**: \( F \subseteq S \)

**Input word**: \( a_0, a_1, \ldots, a_{n-1} \)  
**Run**: \( s_0, s_1, \ldots, s_n \)

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

**Acceptance**: \( s_n \in F \).

**Planning Problem**: Find word leading from \( s_0 \) to \( F \).
- **Realizability**: \( L(A) \neq \emptyset \)
- **Program**: \( w \in L(A) \)
Automata on Infinite Words

Nondeterministic Büchi Automaton (NBW)

\[ A = (\Sigma, S, s_0, \rho, F) \]
- **Alphabet:** \( \Sigma \)
- **States:** \( S \)
- **Initial state:** \( s_0 \in S \)
- **Transition function:** \( \rho : S \times \Sigma \rightarrow 2^S \)
- **Accepting states:** \( F \subseteq S \)

**Input word:** \( a_0, a_1, \ldots \)

**Run:** \( s_0, s_1, \ldots \)
- \( s_{i+1} \in \rho(s_i, a_i) \) for \( i \geq 0 \)

**Acceptance:** \( F \) visited infinitely often

**Motivation:**
- characterizes \( \omega \)-regular languages
- equally expressive to MSO (Büchi 1962)
- more expressive than LTL
Examples

\[ ((0 + 1) * 1)^\omega : \]

\[ \begin{array}{c}
\emptyset \\
0 \\
1 \\
\end{array} \]

- infinitely many 1’s

\[ ((0 + 1) * 1)^\omega : \]

\[ \begin{array}{c}
\emptyset \\
0, 1 \\
1 \\
\end{array} \]

- finitely many 0’s
Infinitary Planning

Planning Problem: Given NBW $A = (\Sigma, S, s_0, \rho, F)$, find infinite word $w \in L(A)$

From Automata to Graphs: $G_A = (S, E_A)$,
$E_A = \{(s, t) : t \in \rho(s, a) \text{ for some } a \in \Sigma\}$.

Lemma: $L(A) \neq \emptyset$ iff there is a state $f \in F$ such that $G_A$ contains a path from $s_0$ to $f$ and a cycle from $f$ to itself.

Corollary: $L(A) \neq \emptyset$ iff there are finite words $u, v \in \Sigma^*$ such that $uv^\omega \in L(A)$.

Bonus: Finite-state program.

Synthesized Program: Do $u$ and then repeatedly do $v$. 
Temporal Logic vs. Büchi Automata

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

**The Compilation Theorem**: V.-Wolper, 1983

Given an LTL formula $\varphi$, one can construct an NBW $A_{\varphi}$ such that a computation $\sigma$ satisfies $\varphi$ if and only if $\sigma$ is accepted by $A_{\varphi}$. Furthermore, the size of $A_{\varphi}$ is at most exponential in the length of $\varphi$.

always eventually $p$:

$$
\begin{array}{c}
\xrightarrow{p} \\
\xleftarrow{p}
\end{array}
$$

– infinitely many $p$’s

eventually always $p$:

$$
\begin{array}{c}
\xrightarrow{p} \\
\xleftarrow{\overline{p}}, \overline{p}
\end{array}
$$

– finitely many $\overline{p}$’s
LTL Planning

- **Input**: LTL formula $\varphi$
- **Planning Problem**: Find word $w \models \varphi$
- **Realizability**: $\varphi$ is satisfiable.
- **Solution**: Solve infinitary planning with $A_{\varphi}$
Synthesis of Reactive Systems

**Game Semantics:** view an open system $S$ as playing a game with an adversarial environment $E$, with the specifications being the winning condition.

**DFA Games:**
- $S$ choose output value $a \in \Sigma$
- $E$ choose input value $b \in \Delta$
- **Round:** $S$ and $E$ set their values
- **Play:** word in $(\Sigma \times \Delta)^*$
- **Specification:** DFA $A$ over the alphabet $\Sigma \times \Delta$
- $S$ wins when play is accepted by $A$.

**Realizability and Synthesis:**
- **Strategy** for $S$ – $\tau : \Delta^* \to \Sigma$
- **Realizability** – exists *winning* strategy for $S$
- **Synthesis** – obtain such winning strategy.
Solving DFA Games

\[ A = (\Sigma \times \Delta, S, s_0, \rho, F) \]

Define \( \text{win}_i(A) \subseteq S \) inductively:

- \( \text{win}_0(A) = F \)
- \( \text{win}_{i+1}(A) = \text{win}_i(A) \cup \{ s : (\exists a \in \Sigma)(\forall b \in \Delta) \rho(s, (a, b)) \in \text{win}_i(A) \} \)

**Lemma:** \( S \) wins the \( A \) game iff \( s_0 \in \text{win}_\infty(A) \).

**Bottom Line:** *linear-time*, least-fixpoint algorithm for DFA realizability. What about synthesis?
**Transducers**

**Transducer**: a finite-state representation of a strategy—deterministic automaton with output

\[ T = (\Delta, \Sigma, Q, q_0, \alpha, \beta) \]

- \( \Delta \): input alphabet
- \( \Sigma \): output alphabet
- \( Q \): states
- \( q_0 \): initial state
- \( \alpha : S \times \Delta \rightarrow S \): transition function
- \( \beta : S \rightarrow \Sigma \): output function

**Key Observation**: A transducer representing a winning strategy can be extracted from \( win_0(A), win_1(A), \ldots \)
Reachability Games

**Game Graphs:** $G = (V_0, V_1, E, v_s, W)$
- $E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$
- $v_s$: start node
- $W \subseteq V_0 \cup V_1$: winning set
- Player 0 moves from $V_0$, Player 1 moves from $V_1$.
- Player 0 wins: reach $W$.

**Fact:** Reachability games can be solved in *linear time* – least fixpoint algorithm

**Consequence:** realizability and synthesis
NFA Games

NFA Games:
• $S$ choose output value $a \in \Sigma$
• $E$ choose input value $b \in \Delta$
• Round: $S$ and $E$ set their variables
• Play: word in $(\Sigma \times \Delta)^*$
• Specification: NFA $A$ over the alphabet $\Sigma \times \Delta$
• $S$ wins when play is accepted by $A$.

Solving NFA Games: Basic mismatch between nondeterminism and strategic behavior.

• Nondeterministic automata have perfect foresight.
• Strategies have no foresight.

Conclusion: Determinize $A$ and then solve.
NBW Games

**NBW Games:**
- $S$ choose output value $a \in \Sigma$
- $E$ choose input value $b \in \Delta$
- **Round:** $S$ and $E$ set their variables
- **Play:** infinite word in $(\Sigma \times \Delta)^\omega$
- **Specification:** NBW $A$ over the alphabet $\Sigma \times \Delta$
- $S$ wins when infinite play is accepted by by $A$.

**Resolving the mismatch:** Determinize $A$

**LTL Games:**
- **Specification:** LTL formula $\varphi$
- **Solution:** Construct $A_\varphi$ and determinize.

**History:**
- Church, 1957: problem posed (for MSO)
- Büchi-Landweber, 1969: decidability shown
- Rabin, 1972: solution via tree automata
Determinization

Key Fact (Landweber, 1969): Nondeterministic Büchi automata are more expressive than deterministic Büchi automata.

Example: \((0 + 1)^*1^\omega\):

![Diagram of Büchi automaton]

– finitely many 0’s

McNaughton, 1966: NBW can be determinized using more general acceptance condition – blow-up is doubly exponential.
**Parity Automata**

---

**Deterministic Parity Automata (DPW)**

\[ A = (\Sigma, S, s_0, \rho, \mathcal{F}) \]

- \( \mathcal{F} = (F_1, F_2, \ldots, F_k) \) - partition of \( S \).
- **Parity index**: \( k \)
- **Acceptance**: Least \( i \) such that \( F_i \) is visited infinitely often is even.

**Example**: \((0 + 1)^*1^\omega\)

\[
\begin{array}{c}
\ell \\
0 \\
\hline
\end{array}
\quad \begin{array}{c}
1 \\
\hline
\end{array}
\quad \begin{array}{c}
r \\
1 \\
\hline
\end{array}
\]

– finitely many 0’s

**Parity condition**: \((\{\ell\}, \{r\})\)

---

**Safra, 1988**: NBW with \( n \) states can be translated to DPW with \( n^{O(n)} \) states and index \( O(n) \).
Parity Games

Game Graphs: \( G = (V_0, V_1, E, v_s, W) \)
- \( E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0) \)
- \( v_s \): start node
- \( W \subseteq V_0 \cup V_1 \): winning set
- Player 0 moves from \( V_0 \),
  Player 1 moves from \( V_1 \).
- \( W = (W_1, W_2, \ldots, W_k) \) – partition of \( V_0 \cup V_1 \)
- Play 0 wins: least \( i \) such that \( W_i \) is visited infinitely often is even.

Solving Parity Games: complexity
- Jurzinski, 1998: \( \text{UP} \cap \text{co-UP} \)
- Schewe, 2007: \( O(n^{k/3}) \)
- Calude et al., 2017: Quasi-PTIME

Open Question: In PTIME?
Algorithm for LTL Synthesis:

- Convert specification $\varphi$ to NBW $A_\varphi$ (exponential blow-up)
- Convert NBW $A_\varphi$ to DPW $A^d_\varphi$ (exponential blow-up)
- Solve parity game for $A^d_\varphi$ (exponential)

Pnueli-Rosner, 1989: LTL realizability and synthesis is 2EXPTIME-complete.

- **Transducer**: finite-state program with doubly exponentially many states (exponentially many state variables)
Theory, Experiment, and Practice

Automata-Theoretic Approach in Practice:

- Mona: MSO on finite words
- Linear-Time Model Checking: LTL on infinite words

Experiments with Automata-Theoretic Approach:

- Symbolic decision procedure for CTL (Marrero 2005)
- Symbolic synthesis using NBT (Wallmeier-Hütten-Thomas 2003)

Why LTL synthesis is so hard?

- *NBW determinization is hard in practice:* from 9-state NBW to 1,059,057-state DRW (Althoff-Thomas-Wallmeier 2005)
- *NBW determinization is hard in practice:* no symbolic algorithms
- Parity games are hard in practice!

2EXPTIME: Need not be an insurmountable problem, but algorithmics is very challenging!
Solution 1: General Reactivity (1)

Piterman-Pnueli-Sa’ar, 2006: Limit LTL specification:

\[
(\text{AlwaysEventually } P) \rightarrow (\text{AlwaysEventually } Q)
\]

**Pros:**

- Cubic game solvability (wrt game size)
- Tools, e.g., *Slugs*
- Broad applicability – popular in robotics

**Cons:** low expressiveness!
Solution 2: $\text{LTL}_f$ – Finite-Horizon LTL

**Crux:** [De Giacomo+V., 2013]
- Full syntax of LTL
- Interpreted over *finite* traces

**Example:** Always Eventually $p$ – $p$ must hold at last point of trace.

**Algorithmic Ideas** [De Giacomo+V., 2015]
- If $\varphi$ is an $\text{LTL}_f$ formula, then it can be translated (w. 2exp blow-up) to DFA.
- Synthesis via DFA games

**Implementation** [Zhu-Tabajara-Li-Pu-V., 2017]:
- Translate $\varphi$ to FOL, and use MONA to translate to BDD-based *Symbolic DFA*.
- Solve DFA game symbolically
- Open Tool: *Syft*
Performance Comparison

![Graph showing performance comparison between Syft and Acacia+ for different lengths of the formula. The x-axis represents the length of the formula (1 to 5), and the y-axis represents the number of solved cases. The graph includes bars for both Syft and Acacia+ for each length of the formula.]
Discussion

Question: Can we hope to reduce a 2EXPTIME-complete approach to practice?

Answer:

- Worst-case analysis is pessimistic.
  - Mona solves nonelementary problems.
  - SAT-solvers solve huge NP-complete problems.
  - Model checkers solve PSPACE-complete problems.
  - Doubly exponential lower bound for program size.

- We need algorithms that blow up only on hard instances

- More algorithmic engineering is needed.
AI vs SE

Some Crossfertilization:

- From planning to verification: *bounded model checking*

- From verification to planning: *BDDs, temporal goals*

More collaboration needed!

- Where does one get comprehensive specification?
- Can system learn from experience?
- What about humans in the loop?