Temporal Synthesis:
From Infinite to Finite Horizon

Moshe Y. Vardi

Rice University

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
- *Liveness*: every job is eventually printed

Crux: Behavioral requirements over infinite traces rather than input/output requirements

Temporal Logic

Linear Temporal logic (LTL): logic of temporal sequences (Pnueli, 1977) – *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: over infinite traces (non-termination)

- $\pi, w \models$ *next* φ if w $\bullet \longrightarrow \underset{\varphi}{\bullet} \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \ldots$
- $\pi, w \models \varphi$ *until* ψ if w $\underset{\varphi}{\bullet} \longrightarrow \underset{\varphi}{\bullet} \longrightarrow \underset{\varphi}{\bullet} \longrightarrow \underset{\psi}{\bullet} \longrightarrow \bullet \ldots$

Examples

- always not (CS_1 and CS_2): mutual exclusion (safety)
- always (Request implies eventually Grant): liveness
- always (Request implies (Request until ($\neg$Request$\vee$ Grant))): liveness
- always ((always eventually Request) implies eventually Grant): liveness

Printer

Example: Printer specification – J_i - job i submitted, P_i - job i printing.

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

- *Liveness*: every jobs is eventually printed
  $always \;\; \bigwedge_{j=1}^{2}(J_i \rightarrow eventually \; P_i)$

Verification

Model Checking:
- *Given*: Program P, Specification φ.
- *Task*: Check that P satisfies φ

- *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.

Challenges:

- Designing P is hard and expensive.
- Redesigning P when P does not satisfy φ is hard and expensive.

Automated Design

Basic Idea:

- Start from spec φ, design P s.t. P satisfies φ.
  Advantage: No verification, no re-design
- Derive P from φ 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."

Deductive Approach (Green, 1969, Waldinger and Lee, 1969, Manna and Waldinger, 1980): 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 input/output programs
- *Temporal*: Synthesize programs for *ongoing computations* (of reactive systems)

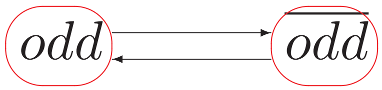
Synthesis of Ongoing Programs

Spec: Temporal logic formulas

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

- *Given*: φ
- *Satisfiability*: Construct model M of φ
- *Synthesis*: Extract P from M.

Example: $always\ \ (odd \rightarrow next\ \neg odd) \wedge$
$always\ \ (\neg odd \rightarrow next\ odd)$



Satisfiability and Synthesis

Print-Server 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: *System*: choose from 2^O, *Env*: choose from 2^I

Infinite Play: (i_0, o_0), (i_1, o_1), (i_2, o_2), $\ldots$

Win: Behavior satisfies spec $\varphi(I, O)$

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

Realizability: [Abadi+Lamport+Wolper, 1989 Pnueli+Rosner, 1989a]: Existence of winning strategy for specification.

Desideratum: A *universal* plan! **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 – *synthesis*.

Rabin, 1972: Simpler solution via Rabin tree automata.

> **Question**: LTL is subsumed by MSO, so what did Pnueli and Rosner do?
> **Answer**: better algorithms!

Strategy Trees

Infinite Tree: D^* (D - directions)

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

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

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

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

Rabin, 1972: Finite-state automata on infinite trees

Emptiness of Tree Automata

Emptiness: $L(A) = \emptyset$

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

Emptiness of Rabin 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

REAL(φ):

- Construct Rabin tree automaton A_φ that accepts all winning strategy trees for spec φ.
- Check non-emptiness of A_φ.
- If nonempty, then we have realizability; extract strategy from non-emptiness witness.

Complexity: non-elementary

Reason: A_φ is of non-elementary size for spec φ 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+Pnueli, 1989: 2EXPTIME realizability algorithm wrt LTL spec (using Safra).
- Rosner, 1990: Realizability is 2EXPTIME-complete.

Standard Critique

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.

2EXPTIME: Need not be an insurmountable problem, but algorithmics is *very challenging*!

Automata on Infinite Words

Nondeterministic Büchi Automaton on Words (NBW)
$A = (\Sigma, S, s_0, \rho, F)$

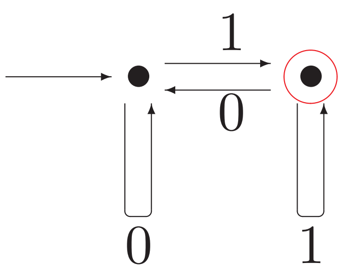
- *Alphabet*: Σ
- *States*: S
- *Initial state*: $s_0 \in S$
- *Transition function*: $\rho : S \times \Sigma \to 2^S$
- *Accepting states*: $F \subseteq S$

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: (1) characterizes ω-*regular* languages; (2) equally expressive to MSO (Büchi 1962); (3) more expressive than LTL

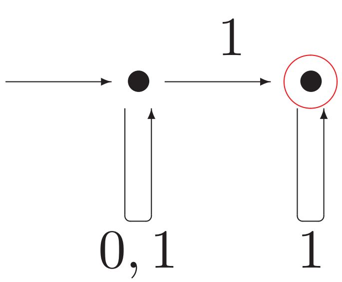
Examples

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



– infinitely many 1's

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



– finitely many 0's

Temporal Logic vs. Büchi Automata

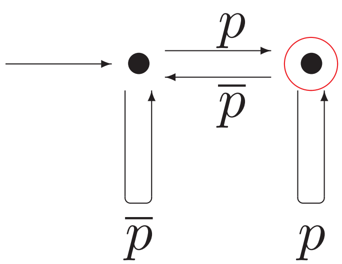
Paradigm: Compile high-level logical specifications into low-level finite-state language

The Compilation Theorem: V.-Wolper, 1983

Given an LTL formula φ, one can construct an NBW 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.

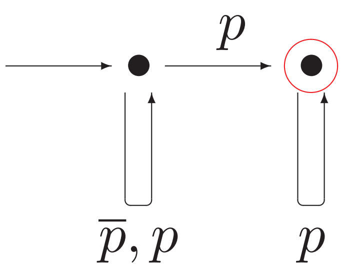
Temporal Logic vs. Büchi Automata: Examples

always eventually p:



– infinitely many p's

eventually always p:



– finitely many $\overline{p}$'s

Realizability 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.

A Mismatch:

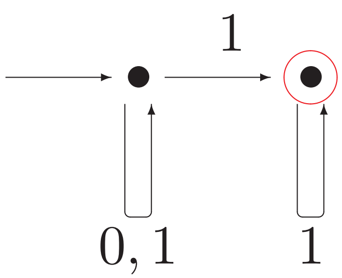
- Nondeterministic automata have "foresight".
- Strategies do not have foresight.

Solution: Determinize A

Determinization

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

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



– 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.

Safra, 1988: NBW with n states can be translated to DPW with $n^{O(n)}$ states and parity index $O(n)$.

Parity Games

Game Graphs: $G = (V_0, V_1, E, v_s, \mathcal{W})$
- $E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$
- 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.
- $\mathcal{W} = (W_1, W_2, \ldots, W_k)$ – 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:

- Calude et al., 2017: Quasi-PTIME

Open Question: In PTIME?

LTL Synthesis

Algorithm for LTL Synthesis:

- Convert specification φ to NBW A_φ (exponential blow-up)
- Convert NBW A_φ to DPW A^d_φ (exponential blow-up)
- Solve parity game for A^d_φ (quasi-polytime)

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

- *Transducer*: finite-state program with doubly exponentially many states

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)

LTL Synthesis

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!

Esparza-Kretnsky-Sickert, 2020: Direct translation from LTL to ω-automata – but still hard!

Solution: General Reactivity (1)

Piterman-Pnueli-Sa'ar, 2006: Limit LTL specification: $(AlwaysEventually\ \ P) \rightarrow (AlwaysEventually\ Q)$

Pros:

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

Cons: low expressiveness!

A New Approach: Finite-Horizon Reasoning

De Giacomo–V., 2013: LTL_f – LTL on finite traces

Example: Always Eventually p – p holds at final state of trace.

Motivation: AI planning, robot task-motion planning, business processes

Remark: Note popularity of *Co-safe* LTL [Kupferman&V., 2001].

Earlier Work: fLTL [Baier&McIlraith, 2006], FLTL [Bauer, Leucker, Schallhart, 2010]

Cons: 2EXPTIME-complete, **Pros**: Easier algorithmics

Classical AI Planning

Deterministic Finite Automaton (DFA)

$A = (\Sigma, S, s_0, \rho, F)$

- *Alphabet*: Σ
- *States*: S
- *Initial state*: $s_0 \in S$
- *Transition function*: $\rho : S \times \Sigma \to S$
- *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*: $w \in L(A)$

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

Universal Planning

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 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 $win_i(A) \subseteq S$ inductively:

- $win_0(A) = F$
- $win_{i+1}(A) = win_i(A) \cup$
  $\{s : (\exists a \in \Sigma)(\forall b \in \Delta)\rho(s, (a, b)) \in win_i(A)\}$

Lemma: S wins the A game iff $s_0 \in win_\infty(A)$.

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

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

- Δ: input alphabet
- Σ: output alphabet
- Q: states
- q_0: initial state
- $\alpha : S \times \Delta \to S$: transition function
- $\beta : S \to \Sigma$: output function

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

Algorithmic Ideas

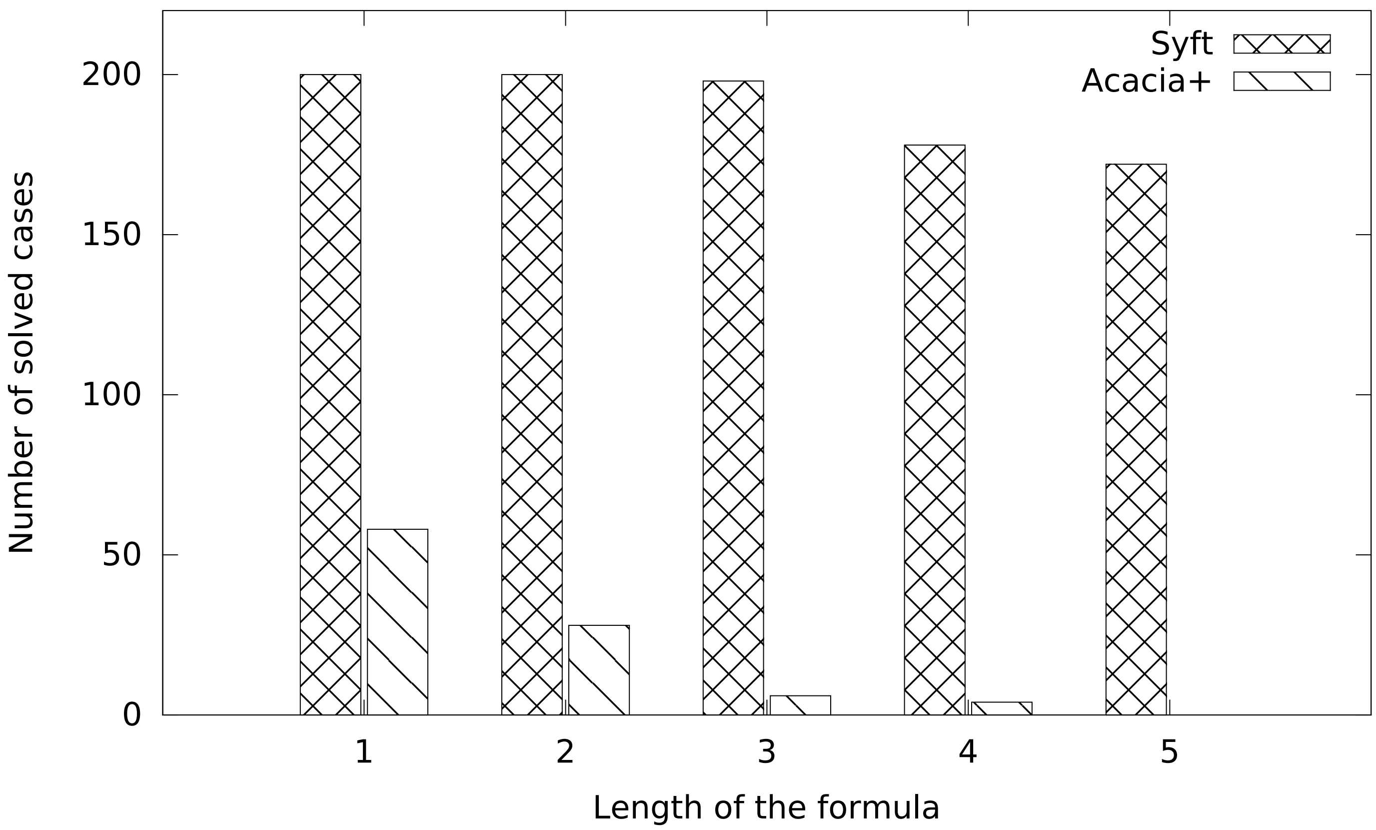
Observations

- If φ is an LTL$_f$ formula, then it can be translated to DFA [Buechi, Elgot, Trakhtenbrot, 1960-62, Kamp, 1968].
- Solve DFA games for realizability and synthesis [De Giacomo &V., 2015]

Implementation [Zhu-Tabajara-Li-Pu-V., 2017]:

- Translate φ to FOL, and use MONA to translate to BDD-based *Symbolic DFA*.
- Solve DFA game symbolically
- Open Tool: *Syft*

Performance Comparison



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.
  - We need algorithms that blow up only on hard instances!
  - More algorithmic research needed!
- **Question** Can "DFA Technology" be used beyond LTL$_f$?

Application: Safety LTL

Normal Form for Safety Temporal Properties:

> Zhu-Tabajara-Li-Pu-V., 2017: Limit LTL specification: No *Until* or *Eventually*

Example: Replace $Always(Snd \rightarrow EventuallyRcv)$ by $Always(Snd \rightarrow NextNextNextRcv)$

Pros:

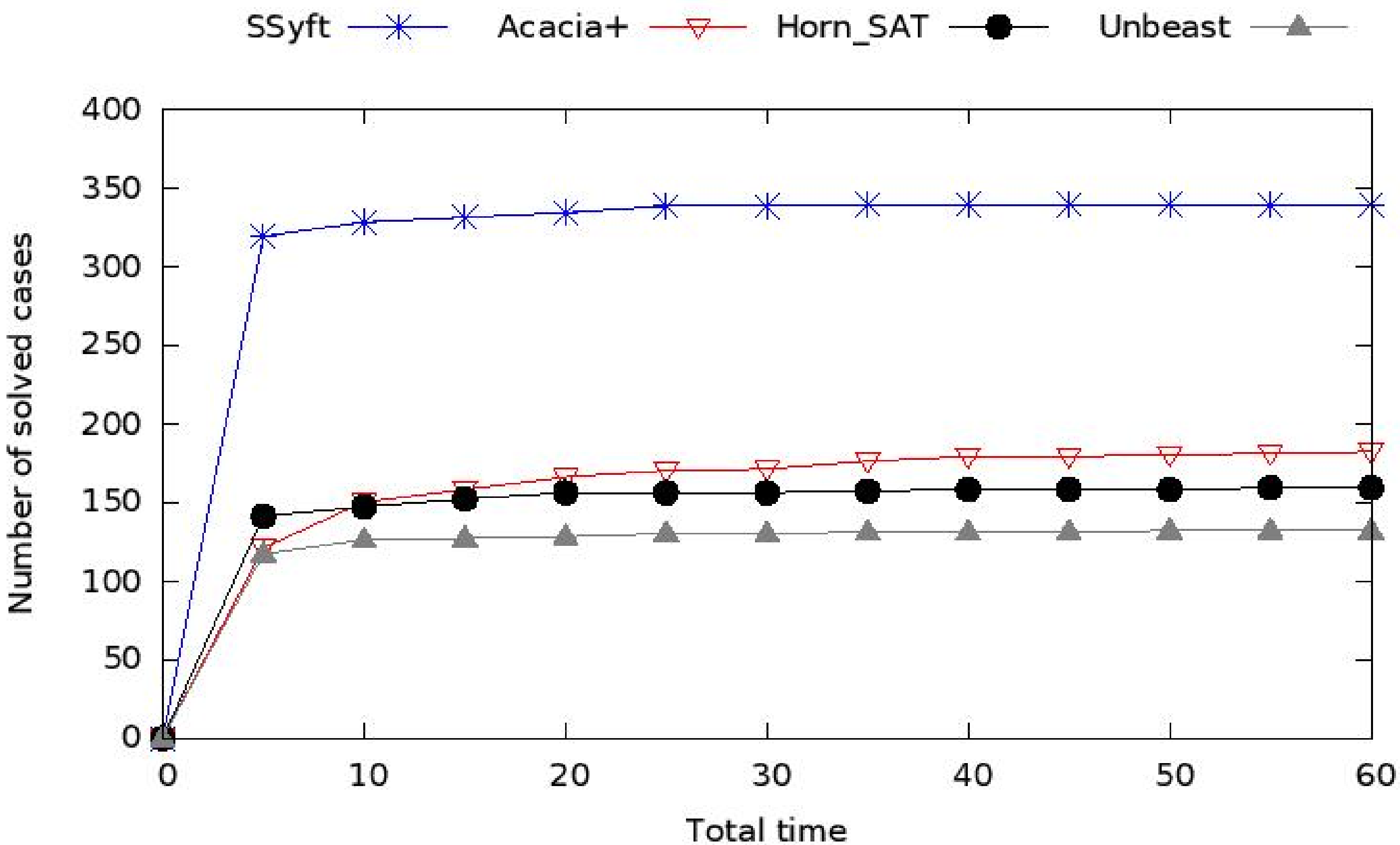
- Linear game solvability (wrt game size)
- Tools, e.g., *Ssyft*

Algorithmic Ideas

Observations

- If φ is a Safety LTL formula, then $\neg\varphi$ is a co-safety formula, which can be translated to DFA [Kupferman+V., 2000].
- Solve for adversary in DFA game.
- Translate $\neg\varphi$ to FOL, and use MONA to translate to BDD-based *Symbolic DFA*.
- Solve DFA game symbolically.

Performance Comparison



Best of all Possible Worlds

Two success stories in temporal synthesis

- GR(1)
- Finite-horizon synthesis

Question: Can the two approaches be combined?

Comment: Finite-horizon goals may require infinite-horizon assumptions, e.g., $EventuallyReceived$ may require $AlwaysEventuallyChannelUp$.

De Giacomo, Di Stasio, Tabajara, V., S. Zhu, IJCAI'21: *Finite-Trace and Generalized-Reactivity Specifications in Temporal Synthesis* – use finite-horizon techniques to construct game arena and solve GR(1) games on that arena – *outperforms LTL synthesis*

In Conclusion

The Siren Song of Temporal Synthesis

- **LTL Synthesis**: a seductive idea
- **But**: horrible complexity and challenging algorithmics

Conclusion: high price for infinite-horizon semantics

Life Wisdom: You can run further by running slower. Time to consider finite-horizon temporal reasoning!