From Verification to Synthesis

Moshe Y. Vardi

Rice University

Verification

Model Checking:

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

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 practices is increasing.

Problems:

- Designing P is hard and expensive.
- Redesigning P when $P \not\models \varphi$ is hard and expensive.

Automated Design

Basic Idea:

- Start from spec φ, design P such that $P \models \varphi$.

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

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

- $\pi, w \models$ *next* φ if w $$\begin{tikzcd} \bullet \arrow[r] & \underset{\varphi}{\bullet} \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \ldots \end{tikzcd}$$
- $\pi, w \models \varphi$ *until* ψ if w $$\begin{tikzcd} \underset{\varphi}{\bullet} \arrow[r] & \underset{\varphi}{\bullet} \arrow[r] & \underset{\varphi}{\bullet} \arrow[r] & \underset{\psi}{\bullet} \arrow[r] & \bullet \ldots \end{tikzcd}$$

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

Specs: Temporal logic formulas

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

- *Given*: φ
- *Satisfiability*: Construct $M \models \varphi$
- *Synthesis*: Extract P from M.

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

$$\begin{tikzcd} odd \arrow[r, shift left] & \overline{odd} \arrow[l, shift left] \end{tikzcd}$$

Reactive Systems

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

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

- *Safety*: two jobs are not printed together
  $always\ \ \neg(P_1 \wedge P_2)$
- *Liveness*: every jobs is eventually printed
  $always\ \ \bigwedge_{j=1}^{2}(J_i \rightarrow eventually\ P_i)$

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:

- *System*: choose from 2^O
- *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 $\models$ spec

Specifications: LTL formula on $I \cup O$

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

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

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.

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)

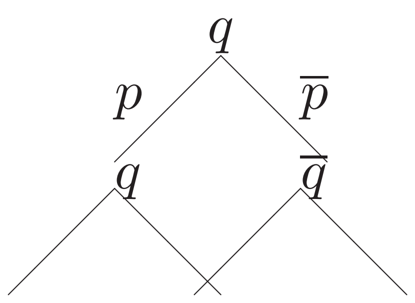
- *Root*: ε
- *Children*: xd, $x \in D^*$, $d \in D$

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 $\Sigma = 2^O$.

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



Winning: Every branch satisfies spec.

Rabin Automata on Infinite k-ary Trees

$A = (\Sigma, S, S_0, \rho, \alpha)$

- Σ: finite alphabet
- S: finite state set
- $S_0 \subseteq S$: initial state set
- ρ: transition function
  - $\rho : S \times \Sigma \to 2^{S^k}$
- α: acceptance condition
  - $\alpha = \{(G_1, B_1), \ldots, (G_l, B_l)\}$, $G_i, B_i \subseteq S$
  - *Acceptance*: along every branch, for some $(G_i, B_i) \in \alpha$, G_i is visited infinitely often, and B_i is visited finitely often.

Emptiness of Tree Automata

Emptiness: $L(A) = \emptyset$

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

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.

Real Critique

- Algorithmics not ready for practical implementation.
- Complete specification is difficult.

Response: More research needed!

- Better algorithms
- Incremental algorithms – write spec incrementally

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*: $w \in L(A)$

Dealing with Nondeterminism

Nondeterministic Finite Automaton (NFA)
$A = (\Sigma, S, s_0, \rho, F)$

- *Alphabet*: Σ
- *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*: 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 $i \geq 0$

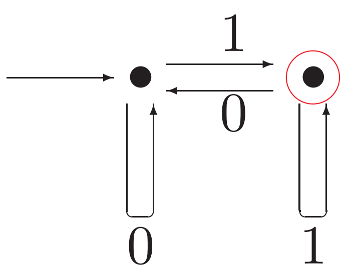
Acceptance: F visited infinitely often

Motivation:

- characterizes ω-*regular* languages
- equally expressive to MSO (Büchi 1962)
- more expressive than LTL

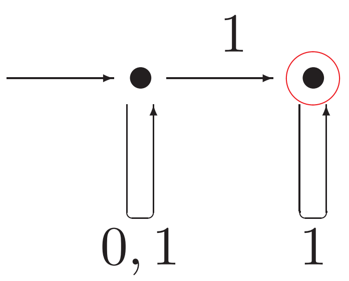
Examples

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



– infinitely many 1's

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



– 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: $G_A = (S, E_A)$,
$E_A = \{(s,t) : t \in \rho(s,a)$ for some $a \in \Sigma\}$.
Lemma: $L(A) \neq \emptyset$ iff there is a 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.

Dealing with Negative Specifications

Deterministic Automata:

- *Input* DFA $A = (\Sigma, S, s_0, \rho, F)$
- *Planning Problem*: Find word $w \not\in L(A)$.
- *Realizability*: $L(A) \neq \Sigma^*$
- *Solution*: Solve classical planning with complementary DFA $A^c = (\Sigma, S, s_0, \rho, S - F)$.

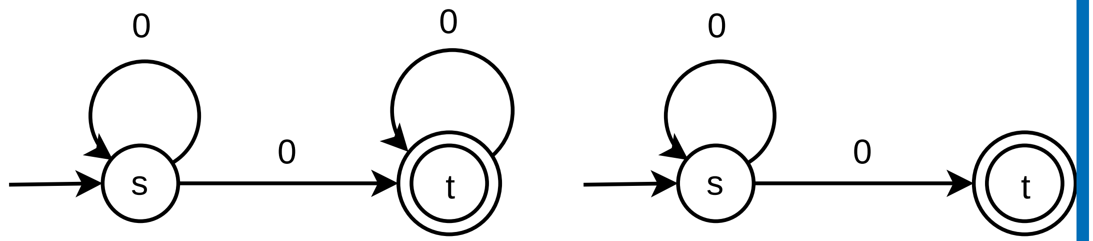
Nondeterministic Automata:

- *Input* NFA $A = (\Sigma, S, s_0, \rho, F)$
- *Planning Problem*: Find word $w \not\in L(A)$.
- *Realizability*: $L(A) \neq \Sigma^*$
- *Solution*: Solve classical planning with complementary DFA $A^c = (\Sigma, 2^S, \{s_0\}, \rho^c, F^c)$.
  - $\rho^c(P, a) = \bigcup_{s \in P} \rho(s, a)$
  - $F^c = \{P : P \cap F = \emptyset\}$

Planning with Complemented Büchi Automata

- *Input*: NBW $A = (\Sigma, S, s_0, \rho, F)$
- *Planning Problem*: Find infinite word $w \notin L(A)$.
- *Realizability*: $L(A) \neq \Sigma^\omega$
- *Solution*: Solve infinitary planning with complementary NBW A^c:
  - $L(A^c) = \Sigma^\omega - L(A)$

Problem: subset construction fails!



- $\Sigma = \{0\}$
- $S = \{s, t\}$
- $S_0 = \{s\}$
- $F = \{t\}$
- $\rho_1(s, 0) = \rho_2(s, 0) = \{s, t\}$
- $\rho_1(t, 0) = \{t\}$, $\rho_2(t, 0) = \emptyset$
- $\rho(\{s\}, 0) = \rho(\{s, t\}, 0) = \{s, t\}$

Büchi Complementation

History:

- Büchi, 1962: Doubly exponential complementation.
- Sistla-V.-Wolper 1985: Büchi's construction can be implemented with exponential blow-up (16^{n^2}).
- Safra, 1988: $n^{O(n)}$
- Michel, 1988: $n! \approx (n/e)^n$ lower bound.
- Kupferman-V., 1997: $(6n)^n$ upper bound.
- Friedgut-Kupferman-V., 2004: $(0.97n)^n$ upper bound.
- Yan, 2005: $(0.76n)^n$ lower bound.
- Schewe, 2009: $(0.76n)^n$ upper bound.

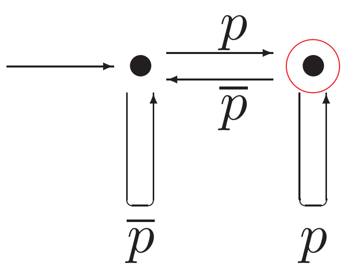
Temporal Logic vs. Büchi Automata

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

The Compilation Theorem: V.-Wolper, 1983

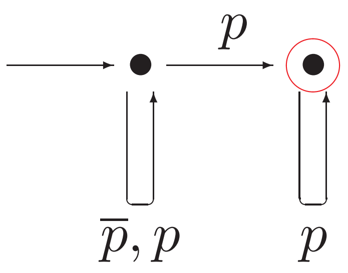
Given an LTL formula φ, one can construct an NBW A_φ such that a computation σ satisfies φ if and only if σ is accepted by A_φ. Furthermore, the size of A_φ is at most exponential in the length of φ.

always eventually p:



– infinitely many p's

eventually always p:



– finitely many $\overline{p}$'s

LTL Planning

Positive Direction:

- *Input* LTL formula φ
- *Planning Problem*: Find word $w \models \varphi$
- *Realizability*: φ is satisfiable.
- *Solution*: Solve infinitary planning with A_φ

Negative Direction:

- *Input* LTL formula φ
- *Planning Problem*: Find word $w \not\models \varphi$
- *Realizability*: $\neg\varphi$ is satisfiable.
- *Solution*: Solve infinitary planning with $A_{\neg\varphi}$

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 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
- $\alpha : S \times \Delta \rightarrow S$: 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 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 φ
- *Solution*: Construct A_φ and determinize.

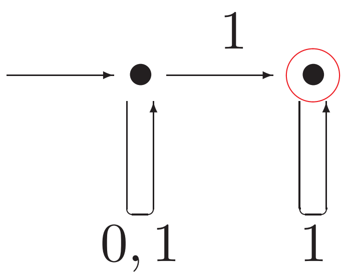
History:

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



– 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: $(0+1)^*1^\omega$

$$\begin{tikzcd} {} \arrow[r] & \ell \arrow[r, shift left, "1"] \arrow[loop below, "0"] & r \arrow[l, shift left, "0"] \arrow[loop below, "1"] \end{tikzcd}$$

– 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: $G = (V_0, V_1, E, v_s, \mathcal{W})$

- $E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$
- 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: complexity

- Jurdzinski, 1998: UP∩co-UP
- Jurdzinski, 2000: $n^{O(k)}$
- Jurdzinski+Petterson+Zwick, 2000: $n^{O(\sqrt{n})}$

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_φ^d (exponential blow-up)
- Solve parity game for A_φ^d (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 no serious implementation of LTL synthesis?

- *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
- lack of incremental algorithms

2EXPTIME: Should not be an insurmountable problem!

Better Algorithm: A Safraless Approach

Kupferman-V., 2005:

- Limit search to strategy trees that are generated by transducers of bounded size
  - Existence of bounded-size transducers follows from the Safraful approach
- Construct recurrence games that are generated by bounded-size transducers
- Solve recurrence games

Crux: focus on subset of strategies

- No determinization
- No parity games

Recurrence 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: *infinitely many* visits to W.

Fact: Recurence games can be solved in quadratic time– greatest fixpoint of reachability.

Consequence: reachability and synthesis.

Restricted specifications

General Reactivity 1 (GR-1):

- *Basic Initiation Assertion* (BIA): p_i
- *Basic Safety Assertion* (BSA): $always\ q_i$
- *Basic Fairness Assertion* (BFA): $always\ eventually\ r_i$
- *Basic Assertion* (BA): $BIA_0 \wedge BSA_0 \wedge \bigwedge_i BFA_i$
- *GR-1*: $(BA_1 \rightarrow BA_2)$.

Piterman-Pnueli-Sa'ar, 2006:

- Formalism is "sufficiently expressive to provide complete specifications of many designs"
- Synthesis can be solved in exponential time.

Bloem-Galler-Jobstmann-Piterman-Pnueli-Weiglhofer, 2007: "We generate compact circuits and we show their practicality by synthesizing a generalized buffer and an arbiter for ARM's AMBA AHB bus from specifications given in PSL. These are the first industrial examples that have been synthesized automatically from their specifications."

Incremental Synthesis

Basic Weakness of Synthesis: full specifications required to get started – unrealistic!

- Specifications evolve!

Incremental Synthesis: Suppose we synthesized programs for specifications φ and ψ, can we get programs for $\varphi \wedge \psi$ *without* starting from scratch.

Kupferman-Piterman-V., 2006: Use realizability proofs for φ and ψ as starting point for realizability testing and synthesis for $\varphi \wedge \psi$.

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
- New approaches are promising.
- More algorithmic engineering is needed.