Reactive Synthesis

Moshe Y. Vardi

Rice University
Synthesis = Automated Design

Basic Idea:

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

  Advantage:
  - No verification
  - No re-design

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

  Advantage:
  - No design

In essence: Declarative programming taken to the limit.
Synthesis of Ongoing Programs

**Specs:** Temporal logic formulas

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

- **Given:** \( \varphi \)
- **Satisfiability:** Construct \( M \models \varphi \)
- **Synthesis:** Extract \( P \) from \( M \).

**Example:**
\[
\text{always}(\text{odd} \to \text{next } \neg \text{odd}) \land \\
\text{always}(\neg \text{odd} \to \text{next odd})
\]
Reactivity: Interaction with environment (Harel+Pnueli, 1985)

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

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

- Liveness: every jobs is eventually printed
  always $\land_{j=1}^{2}(J_i \rightarrow \text{eventually } P_i)$
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 $f : (2^I)^* \rightarrow 2^O$

Pnueli+Rosner, 1989:

- **Realizability**: Existence of winning strategy for specification.
- **Synthesis**: Constructing a winning strategy for specification.
Strategy Trees

**Infinite Tree**: $D^*$ (D - 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**: $I = \{p\}$, $O = \{q\}$

---

[Diagram of a labeled infinite tree]

**Winning**: Every branch satisfies spec.
Rabin 1972’s Realizability Algorithm

- Construct Rabin tree automaton $A_\varphi$ that accepts all winning strategy trees for spec $\varphi$.
- Check non-emptiness of $A_\varphi$.
- If nonempty, then we have realizability; extract strategy from non-emptiness witness.
Post-1972 Developments

- Pnueli, 1977: Use LTL as spec language.

- Vardi+Wolper, 1983: Exponential translation from LTL to automata.


- 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 yet ready for practical implementation.
• Complete specification is difficult.
• Realistic systems are built from existing components.

Response: More research needed!

• Better algorithms
• Incremental algorithms – write spec incrementally
• Synthesis from components