Synthesis from Components

Moshe Y. Vardi

Rice University

What Good is Model Checking?

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.

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.)

Linear 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.

- $\pi, w \models next\ \varphi$ if w 

- $\pi, w \models \varphi\ until\ \psi$ 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

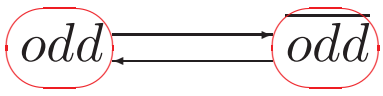
Synthesis of Ongoing Programs

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



Reactive Systems

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

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 is receptive to *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
Dill, 1989, Pnueli+Rosner, 1989
Existence of winning strategy for system.

Synthesis: Pnueli+Rosner, 1989
Extraction of winning strategy for system.

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!

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).
- Pnueli+Rosner, 1989: 2EXPTIME realizability algorithm wrt LTL spec (using Safra).
- Rosner, 1990: Realizability is 2EXPTIME-complete.

Synthesis from Components

Basic Intuition: [Lustig+V., 2009]

- In practice, systems are typically not built from scratch; rather, they are constructed from existing components.
- Can we automate "construction from components"?

Setup:

- *Library* $L = \{C_1, \ldots, C_k\}$ of *component types*.
- *Linear temporal specification*: φ

Problem: Construct a finite system S that satisfies φ by composing components that are *instances* of the component types in L.

Question: What are components? How do you compose them?

Dataflow Synthesis from Components

Setup:

- *Components*: multi-input multi-output transducers e.g., sequential circuits
- *Dataflow composition*: connect input and output channels, e.g., connect sequential circuits

Theorem: [Lustig+V.,2009]
Dataflow synthesis from components is undecidable.

Crux:

- Number of component instances not bounded, a priori.
- Cell of Turing-machine tape can be viewed as a transducer, connected to cells to its left and right.

Transducers

Transducers: Canonical model of reactive computation

$T = (\Sigma, \Delta, S, s_0, \rho, \gamma)$

- Σ: input alphabet
- $\Delta = 2^{Prop}$: output alphabet
- S: state set
- $s_0 \in S$: initial state
- $\rho : S \times \Sigma \to S$: transition function
- $\gamma : S \to \Delta$: output function

Input word: $w = a_0, a_1, \ldots$
Run: $s_0, s_1, \ldots$
- $s_{i+1} \in \rho(s_i, a_i)$ for $i \geq 0$

Output: $out(T, w) = \gamma(s_0), \gamma(s_1), \ldots$

Controlflow Synthesis from Components

Setup:

- *Components*: single-input single-output transducers with *exit states*, e.g., software module
- *Controlflow composition*: upon arrival at an exit state, continue at the start state of another component e.g., *goto* – composer *chooses* target of branch.

Synthesis Problem: Given temporal formula φ and component library L, decide if satisfying composition exists and construct it.

Theorem: [Lustig+V.,2009]
Controlflow synthesis from components is 2EXPTIME-complete.

Crux:

- Consider general (possible infinite) composition trees, that is, unfoldings of compositions
- Use automata to check that all possible computations wrt composition satisfy φ
- Show that if general composition exists then finite composition exists.

Controlflow Synthesis from Probabilistic Components

Probabilistic Transducers: Canonical model of probabilistic reactive computation

$T = (\Sigma, \Delta, S, s_0, \rho, \gamma)$

- Σ: input alphabet
- Δ: output alphabet
- S: state set
- $s_0 \in S$: initial state
- $\rho : S \times \Sigma \to dist(S)$: transition function
  - $dist(S)$: probability distributions over S
- $\gamma : S \to \Delta$: output function

Semantics: Given an input word $w \in \Sigma^\omega$, the probabilistic transducer T induces a probability distribution $out(T, w)$ on Δ^ω.

Probabilistic Temporal Semantics

Recall: An LTL formula φ is interpreted over a trace in $(2^{Prop})^\omega$

Temporal Correctness:

- *Deterministic Correctness*: for all $w \in \Sigma^\omega$, we have that $out(T, w)$ satisfies φ.

- *Probabilistic Correctness*: for all $w \in \Sigma^\omega$, we have that $out(T, w)$ satisfies φ *almost surely* – i.e., *with probability one*.

  – Need to show that $models(\varphi)$ is measurable.

Synthesis from Probabilistic Components

Setup:

- *Components*: probabilistic components with *exit states*
- *Controlflow composition*: from exit state continue to another component (composer's *choice*)

Synthesis Problem: Given temporal formula φ and component library L, decide if there is a composition that satisfies φ and construct it.

Theorem: [Lustig+Nain+V.,2011]
Controlflow synthesis from probabilistic components is decidable.

Deterministic vs. Probabilistic Synthesis

Open Questions:

1. For deterministic synthesis, there exists a composition iff there exist a *finite* composition.

   This is open for probabilistic composition.

2. Deterministic synthesis is 2EXPTIME-complete.

   For probabilistic synthesis, best known upper bound is 4EXPTIME.