What is automatic program synthesis?

- Specification (expressed in terms of a formal language)
- Synthesis Algorithm
- Synthesized Program Model (An abstract structure)

Why automatic program synthesis?

- Verification (especially model checking):
  - Checking a property with respect to a program model
  - Develop a program model
  - Check the model with respect to specified properties
  - Drawback: verification after the design

Question:
Does there exist an algorithm to design a program from its specification?

Benefit: Correct by construction
Programs

Functional: Terminating behavior (e.g., square root)

Input  →  Functional Program  →  Output
(Termination)

Closed Reactive: Non-Terminating Behaviors
(e.g., a group of processes competing for their critical section)

Closed Reactive Program

P1  P2  P3  P4

The environment of P1

A collaborative environment

Programs (Cont’d)

- Open Reactive Systems (synchronous or asynchronous)

Open Reactive System

P1  P2  P3  P4

Reaction

Hostile environment

Outline

- Program Synthesis Methods
  - Model-Theoretic
  - Automata-Theoretic
  - Calculational
Model-Theoretic Approach

- Automatic synthesis of closed reactive programs

Temporal specification

- Is the specification satisfiable?
  - (tableau proof)

  Yes

  No

- Create a finite model of the specification

- Extract the synchronization skeleton of each process

Model-Theoretic Approach - Issues

- Specification Language
  - Variants of propositional temporal logic

- Program Model
  - Shared memory [Emerson&Clarke 1982]
  - Message passing [Manna&Wolper 1984]

- Decision procedure
  - Tableau-based proof

- Complexity
  - Exponential in the length of the specification

- Distribution
  - Decomposition of test-and-set actions (high atomicity) into atomic (low atomicity) read/write actions [Emerson&Clarke 2001]

Model-Theoretic Approach - Example

- Mutual Exclusion problem [Emerson&Clarke 1982]

- Problem Specification ($i = 1, 2$):
  1. Start state: $NCS_i \land NCS_2$.
  2. Mutual exclusion: $AG(\neg(CS_i \land CS_2))$.
  3. Progress: $AG(TRY_i \Rightarrow AFCS_i)$.

- Invariants:
  $AG(NCS_i \lor TRY_i \land CS_i)$,
  $AG(NCS_i \Rightarrow \neg(TRY_i \land CS_i))$,
  $AG(TRY_i \Rightarrow \neg(NCS_i \land CS_i))$,
  $AG(CS_i \Rightarrow \neg(TRY_i \lor NCS_i))$. 
Model-Theoretic Approach - Example

• Structural Specification:

\[ AG(NCS_i \Rightarrow (AX_{i}TRY_i \land EX_{i}TRY_i)) \]

• No process interferes with the transitions of the other process

\[ AG(NCS_i \Rightarrow AX_{i}NCS_i), \]
\[ AG(TRY_i \Rightarrow AX_{i}TRY_i)), \]
\[ AG(CS_i \Rightarrow AX_{i}CS_i)). \]

Model-Theoretic Approach – Example (continued)

• Synthesis method
  – Build a tableau proof of the spec.
  – Extract a finite model
  – Extract the synchronization skeleton of each process

Model-Theoretic Approach – Example (Reduction Rules)

- Conjunctive rules:
  \[ \alpha = f \land g \quad \alpha_1 = f \quad \alpha_2 = g \]
  \[ \alpha = AG \quad \alpha_1 = g \quad \alpha_2 = AXAG \]
  \[ \alpha = EG \quad \alpha_1 = g \quad \alpha_2 = EXEG \]

- Disjunctive rules
  \[ \beta = f \lor g \quad \beta_1 = f \quad \beta_2 = g \]
  \[ \beta = AF \quad \beta_1 = g \quad \beta_2 = AXAF \]
  \[ \beta = EF \quad \beta_1 = g \quad \beta_2 = EXEF \]
Model-Theoretic Approach – Example (continued)

• A tableau for the initial node

\[
\begin{align*}
\text{NCS}_1 \\
\text{AG}(\text{AX}_1 \text{TRY}_1 \land \text{EX}_1 \text{TRY}_1) \\
\text{AG}(\text{AX}_2 \text{NCS}_1) \\
\text{AX}_1 \text{TRY}_1 \land \text{EX}_1 \text{TRY}_1
\end{align*}
\]

A Block corresponding to each leaf in the tableau

Model-Theoretic Approach – Example (continued)

The reduced tableau for NCS_1 and NCS_2.

Model-Theoretic Approach – Example (continued)

• Blocks (NCS_1, NCS_2)

D:

Initial OR-node

C:

NCS_1

AX_1 TRY_1

EX_1 TRY_1

AX_1 NCS_2

NCS_2

AX_2 TRY_2

EX_2 TRY_2

AX_2 NCS_1

Blocks(D) is an AND-node
Model-Theoretic Approach – Example (continued)

3. Build the tableau
4. Extract a finite model
5. Extract the synchronization skeleton of each process
Model-Theoretic Approach – Some References


Automata-Theoretic Approach

- Automatic synthesis of open reactive programs
  - Shared-variable synchronous open reactive programs
  - Shared-variable asynchronous open reactive programs

\[ \phi(x, y) \]

Specification: linear temporal logic formula
Automata-Theoretic Approach – Basic Concepts

- Open reactive system computations

Automata-Theoretic Approach – Synthesis Method

- Synthesis method
  - Derive a branching temporal specification (Implementability formula)
  - Build a tree automaton
  - Check the non-emptiness of derived tree automaton
  - Extract a deterministic automaton that satisfies the linear temporal specification

Automata-Theoretic Approach – Issues

- Specification language
  - Temporal formula → Tree automaton
- Program model
  - Single-process program
  - Concurrent processes on a distributed architecture
- Synthesis method
  - A sequence of conversions on the specification
  - Reducing the synthesis problem to non-emptiness checking problem
- Complexity
  - Single-process program: Doubly exponential in the length of specification
  - Distributed programs with arbitrary architecture: Undecidable
  [Pnueli & Rosner 1990]
Automata-Theoretic Approach – Some References

- Orna Kupferman, Moshe Y. Vardi: Synthesizing Distributed Systems. LICS 2001