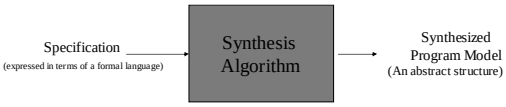
CS 5090: Software Fault Tolerance

Automated Program Synthesis

Ali Ebnenasir
Department of Computer Science
Michigan Technological University

What is automatic program synthesis?



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

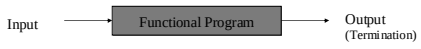
Question:
Does there exist an algorithm to design a program from its specification?

Benefit: Correct by construction

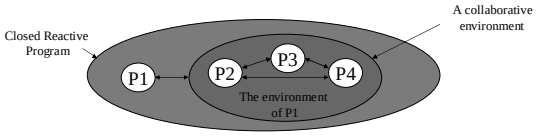
MichiganTech

Programs

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



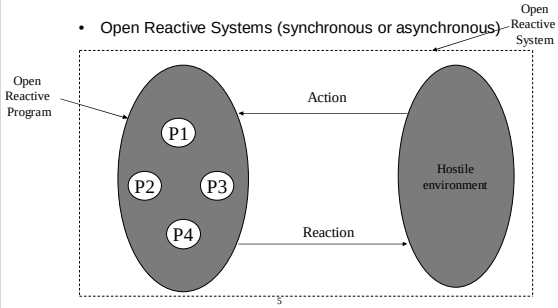
Closed Reactive: Non-Terminating Behaviors
(e.g., a group of processes competing for their critical section)



MichiganTech

Programs (Cont'd)

- Open Reactive Systems (synchronous or asynchronous)



MichiganTech

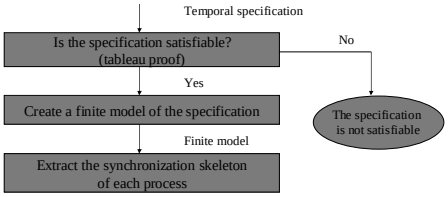
Outline

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

MichiganTech

Model-Theoretic Approach

- Automatic synthesis of closed reactive programs



7

MichiganTech

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]

8

MichiganTech

Model-Theoretic Approach - Example

- Mutual Exclusion problem [Emerson&Clarke 1982]
- Problem Specification (i = 1, 2):
  1. Start state; $NCS_1 \wedge NCS_2$.
  2. Mutual exclusion; $AG(\neg(CS_1 \wedge CS_2))$.
  3. Progress; $AG(TRY_i \Rightarrow AFCS_i)$.
- Invariants:

$$AG(NCS_i \vee TRY_i \vee CS_i),$$
$$AG(NCS_i \Rightarrow \neg(TRY_i \vee CS_i)),$$
$$AG(TRY_i \Rightarrow \neg(NCS_i \vee CS_i)),$$
$$AG(CS_i \Rightarrow \neg(TRY_i \vee NCS_i)).$$

Model-Theoretic Approach - Example

- Structural Specification:

$$(AG(NCS_i \Rightarrow (AX_iTRY_i \wedge EX_iTRY_i)))$$

- No process interferes with the transitions of the other process

$$AG(NCS_i \Rightarrow AX_jNCS_i),$$
$$AG(TRY_i \Rightarrow AX_jTRY_i)),$$
$$AG(CS_i \Rightarrow AX_jCS_i)).$$

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:

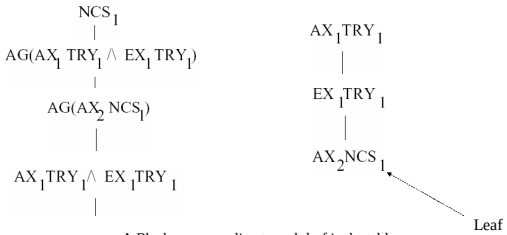
$\alpha = f \wedge g$	$\alpha_1 = f$	$\alpha_2 = g$
$\alpha = AGg$	$\alpha_1 = g$	$\alpha_2 = AXAGg$
$\alpha = EGg$	$\alpha_1 = g$	$\alpha_2 = EXEGg$

- Disjunctive rules

$\beta = f \vee g$	$\beta_1 = f$	$\beta_2 = g$
$\beta = AFg$	$\beta_1 = g$	$\beta_2 = AXAFg$
$\beta = EFg$	$\beta_1 = g$	$\beta_2 = EXEFg$

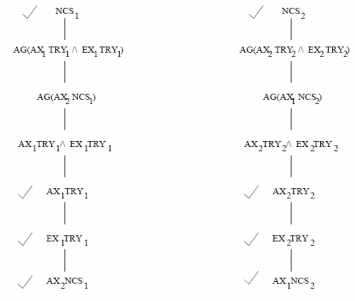
Model-Theoretic Approach – Example (continued)

- A tableau for the initial node



A Block corresponding to each leaf in the tableau

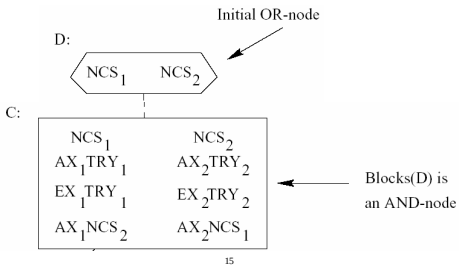
Model-Theoretic Approach – Example (continued)



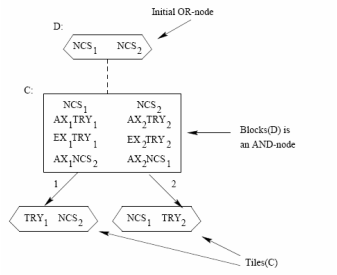
The reduced tableau for NCS_1 and NCS_2.

Model-Theoretic Approach – Example (continued)

- Blocks($NCS_1 \wedge NCS_2$)



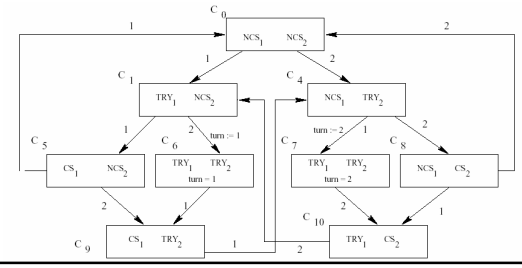
Model-Theoretic Approach – Example (continued)



The *Blocks* of the initial OR-node and the *Tiles* of each *Block*.

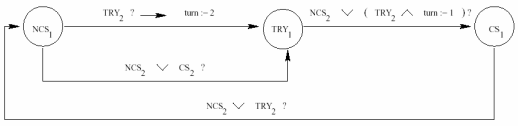
Model-Theoretic Approach – Example (continued)

3. Build the tableau
4. Extract a finite model



Model-Theoretic Approach – Example (continued)

5. Extract the synchronization skeleton of each process



MichiganTech

Model-Theoretic Approach – Some References

- E. Allen Emerson, Edmund M. Clarke: Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. Sci. Comput. Program. 2(3): 241-266 (1982)
- Paul C. Attie, E. Allen Emerson: Synthesis of Concurrent Systems with Many Similar Processes. ACM Trans. Program. Lang. Syst. 20(1): 51-115 (1998)
- Paul C. Attie, E. Allen Emerson: Synthesis of concurrent programs for an atomic read/write model of computation. ACM Trans. Program. Lang. Syst. 23(2): 187-242 (2001)
- Zohar Manna, Pierre Wolper: Synthesis of Communicating Processes from Temporal Logic Specifications. ACM Trans. Program. Lang. Syst. 6(1): 68-93 (1984)

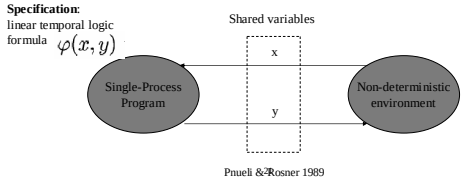
MichiganTech

Automata-Theoretic Approach

MichiganTech

Automata-Theoretic Approach

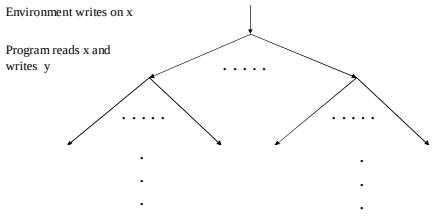
- Automatic synthesis of open reactive programs
  - Shared-variable synchronous open reactive programs
  - Shared-variable asynchronous open reactive programs



Pnueli & Rosner 1989

Automata-Theoretic Approach – Basic Concepts

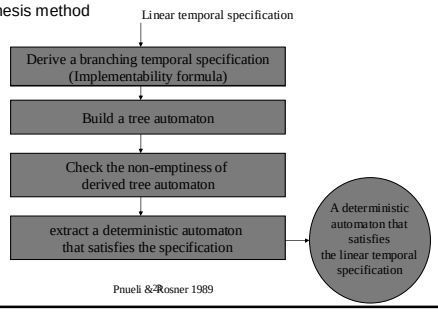
- Open reactive system computations



Pnueli & Rosner 1989

Automata-Theoretic Approach – Synthesis Method

- Synthesis method



Pnueli & Rosner 1989

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

- Amir Pnueli, Roni Rosner: On the Synthesis of an Asynchronous Reactive Module. ICALP 1989: 652-671
- Amir Pnueli, Roni Rosner: On the Synthesis of a Reactive Module. POPL 1989: 179-190
- Amir Pnueli, Roni Rosner: Distributed Reactive Systems Are Hard to Synthesize FOCS 1990: 746-757
- Orna Kupferman, Moshe Y. Vardi: Synthesizing Distributed Systems. LICS 2001