Course ITI8531: Software Synthesis and Verification

Lecture 11: Overview of Software Synthesis Approaches and Preliminaries of Temporal Synthesis

Spring 2019

Leonidas Tsiopoulos

Lecture slides are partly based on presentations by Moshe Vardi [8] and Nir Piterman [9]

Programming and Verification

- Verification with Model Checking:
  - *Given* a program (model) *P* and a specification φ, *check* that *P* satisfies φ.
  - Success:
    - Usage is increasing with several available model checkers based on advanced algorithmic methods.
  - Issues:
    - Designing *P* is *hard* and *expensive*.
    - Redesigning *P* when *P* does not satisfy φ is *hard* and *expensive*.
- Verification with Theorem Proving:
  - Similar issues as above.
- Alternative solution:
  - *Start* from specification φ and *synthesize* *P* such that *P* satisfies φ.

Synthesis – What it is about?

- The main motivation is that „*if we can verify why not go directly from specification to correct-by-construction systems by synthesis*?“
- Various approaches exist:
  - Deductive approach where first the realizability of a function is proved and then the program is extracted from the proof.
  - Computational approach where a transformational program is synthesized to produce a result on termination in terms of input and output relations.
    - By some called *classical* approach or just *program synthesis*.
  - Reactive/Temporal approach where programs are synthesized for *ongoing* computations (protocols, operating systems, robot controllers, etc).
- The focus of this course in on the reactive approach.

Program Synthesis

- Program Synthesis is the task of discovering an executable program from user intent expressed in the form of some constraints.

- Main challenge: Establishing a proper synergy between the human and the synthesizer is fundamental to the success of synthesis.

Deductive Synthesis

- Synthesis of systems which allow the user to provide insight directly into the synthesizer

- Program can be extracted from a constructive proof of the satisfiability of a specification.
  - E.g., $(\forall x)(\exists y)(Pre(x) \rightarrow Post(x, y))$

- Powerful technique but demands a high level of expertise.
- Tools available: KIDS, NuPRL, ...

Reactive Synthesis

- *„if we can verify why not go **directly** from specification to correct-by-construction systems by synthesis*?“
- Now we are in 2019 and still ***„directly“*** involves several transformational steps in the background no matter the underlying approach.
- Old topic started already in 1960s by Church.
  - Given a circuit interface specification partitioned to inputs and outputs and a behavioral specification in first order logic, determine if there is an automaton that realizes the specification. If the specification is realizable, construct an implementing automaton.

History on reactive synthesis - 1

- Problem as defined by Church [1].
- Büchi and Landweber define *two-player games* of infinite duration [2].
- Rabin introduces automata on infinite trees generalizing Büchi's work on ω-automata to trees [3].
  - Simpler solution via Rabin tree automata.

History on reactive synthesis - 2

- Pnueli in 1977 [4] proposed to use Linear Temporal Logic (LTL) rather than MSO (monadic second-order theory of one successor function) as specification language for less complexity.
- Vardi, Wolper and Sistla in 1983 [5] showed that the translation from LTL to automata is of *elementary* (exponential) complexity.
- Safra in 1988 [6] showed that the construction of tree automata for strategy trees wrt LTL specification is doubly exponential (using [5]).
    - Procedure for the *determinization* of Rabin automata.
- In 1989 Pnueli and Rosner [7] established LTL *realizability* to be 2EXPTIME complete using Safra's approach.
    - Very high complexity when determinizing non-deterministic automata!
    - Caused halting of research in this field for many years.

History on reactive synthesis - 3

- From beginning of 2000s this research topic was brought back to the scene with several approaches offering „Safraless“ solutions to avoid the very expensive determinisation step and also better algorithms working on „symbolic“ representation of the state space.

- The focus of this part of the course will be on one such approach implemented with the tool Acacia+.

System specification: satisfiability vs realizability

- Satisfiability: Exists some behavior that satisfies the specification.
- Realizability: Exists system that **implements** the specification and it **must work for all inputs** (controlled by the environment) to the system.

Example on satisfiability

- **Example**: Printer specification
- J_i - job i submitted, P_i - job i printing, $i \in \{1,2\}$.
- *Safety* property: two jobs are not printing together - $always$ $\neg(P_1 \wedge P_2)$
- *Liveness* property: every job is eventually printed
    - $always \bigwedge_{i=1}^{2}(J_i \rightarrow eventually\ P_i)$
- Is specification satisfiable? Yes!
- Model *M*: A single state where J_i , P_i are all *false.*
- Can we extract a program from *M*? No!
- Why? *M* handles only one input sequence.
- J_i are inputs controlled by the environment. We need a system that handles all input sequences.
- Only satisfiability is not enough for synthesis!

Formal context for synthesis

- A *specification* will be in LTL over *input* and *output propositions*.
- A system will be an *automaton with output*.
- Input and output are combined to create a *sequence* of assignments to propositions.
- All possible *infinite paths* over the automaton should *satisfy* the specification.

LTL preliminaries

- Formal language which extends the propositional Boolean logic.

- Variables: atomic propositions, e.g., *p* and *q.*

- A set of atomic propositions partitioned to inputs and outputs denoting the basic *facts* about a system and its environment.

- Usual Boolean operators are allowed, e.g., $p \longrightarrow q$ ($\neg p \lor$ q) is an LTL formula, but it refers to the *first element* of an infinite sequence.

LTL preliminaries – Operators and formulae

Temporal operators

- **G**: **g**lobally (always), e.g., **G**($p \rightarrow q$) means "in each element of the sequence, $p \rightarrow q$ holds".
- **F**: in the **f**uture, e.g., **F**($p \rightarrow q$) means "for some element of the sequence, $p \rightarrow q$ holds".
- **X**: on the ne**x**t step, e.g., **X**($p \rightarrow q$) means "$p \rightarrow q$ holds for the second element of the sequence".
- **U**: **u**ntil, e.g., p **U** q means "q must happen at some step, and the sequence must satisfy p until (non-inclusive) q happens".
- Linear Temporal Logic **formulae** are constructed as follows:
  - $\varphi ::= p \parallel \varphi \wedge \varphi \parallel \neg\varphi \parallel \varphi_1 \mathbf{U} \varphi_2 \parallel \ldots$

LTL Semantics

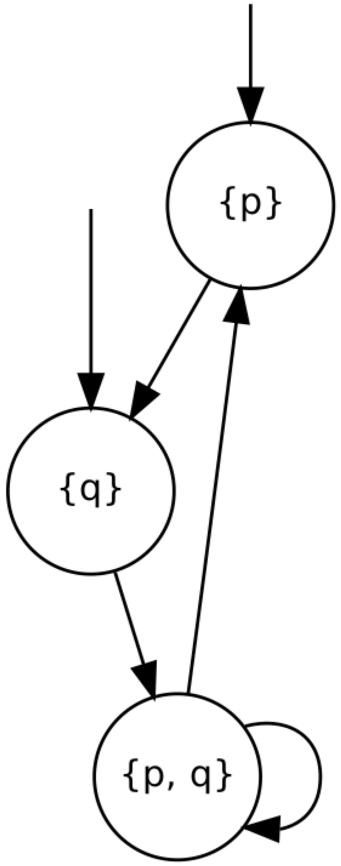
- A *model* for an LTL formula is an *infinite sequence* $\sigma = \sigma_0, \sigma_1,$ … with a *designated location* $j \geq 0$.
- Each letter σ_i is a set of propositions true at time i.
- Formula φ *holds* over *sequence* σ in location $j \geq 0$, denoted $(\sigma, j) \vDash \varphi$, if:
  - If φ is a proposition $(\sigma, j) \vDash \varphi \Leftrightarrow \varphi \in \sigma_j$
  - $(\sigma, j) \vDash \neg\varphi \Leftrightarrow (\sigma, j) \ |\neq \ \varphi$
  - $(\sigma, j) \vDash (\varphi_1 \vee \varphi_2) \Leftrightarrow (\sigma, j) \vDash \varphi_1$ **or** $(\sigma, j) \vDash \varphi_2$
  - $(\sigma, j) \vDash \boldsymbol{\varphi_1} \, \mathbf{U} \boldsymbol{\varphi_2} \Leftrightarrow \exists k \geq j \,.\, (\sigma, k) \vDash \varphi_2$ **and** $\forall j \leq l < k \,.\, (\sigma, l) \vDash \varphi_1$
  - **…**

LTL and verification

- Kripke structure M satisfies LTL formula φ (written: $M \vDash \varphi$), if **all paths** in M which **start** in M's **initial states** satisfy φ.

- Which of these LTL formulae are satisfied by the Kripke structure on the right?

- $\varphi_1 = \mathbf{G}p$
- $\varphi_2 = \mathbf{F}(\neg p \wedge \neg q)$
- $\varphi_3 = p\ \mathbf{U}\ (\neg p \wedge \neg q)$
- $\varphi_4 = \mathbf{GF}(p \wedge q)$



- Only φ_4

Automata

- Systems with *discrete states*.
- Formally, A = ⟨Σ, Q, δ, q_0 ⟩, where
  - Σ – a finite input alphabet.
  - Q– a finite set of states.
  - δ: Q × Σ ⟶ 2^Q – a transition function associating with state and an input letter a set of successor states.
  - q_0– an initial state.
- An input word w = σ_0, σ_1, … is a sequence of letters from Σ.
- A run r = q_0, q_1, … over w is a sequence of states starting from q_0 such that for every i ≥ 0 we have $q_{i+1} \in \delta(q_i, \sigma_i)$.
- An automaton is deterministic if for every $q \in Q$ and $\sigma \in \Sigma$ we have

  $|\delta(q, \sigma)| \leq 1$.
- Several variations exist: Rabin, Büchi, ”Safety”, Uppaal Timed Automata, ...

Games for Synthesis – Why?

- We need to synthesize a system that **implements** the specification and it **must work for all inputs** (controlled by the environment) to the system.
- Controlling so that *uncontrollable* events do not lead to damage.
- This can be a *two-player game*.
- **Realizability** with regard to games: Existence of *winning strategy* for the system in a game *against* the environment.
    - Addressed (rather) efficiently by Pnueli and Rosner [7] providing better algorithms (based on μ-calculus and least fixpoint) compared to previous approaches.
    - But still these approaches were based on Safra's highly complex determinizing step.

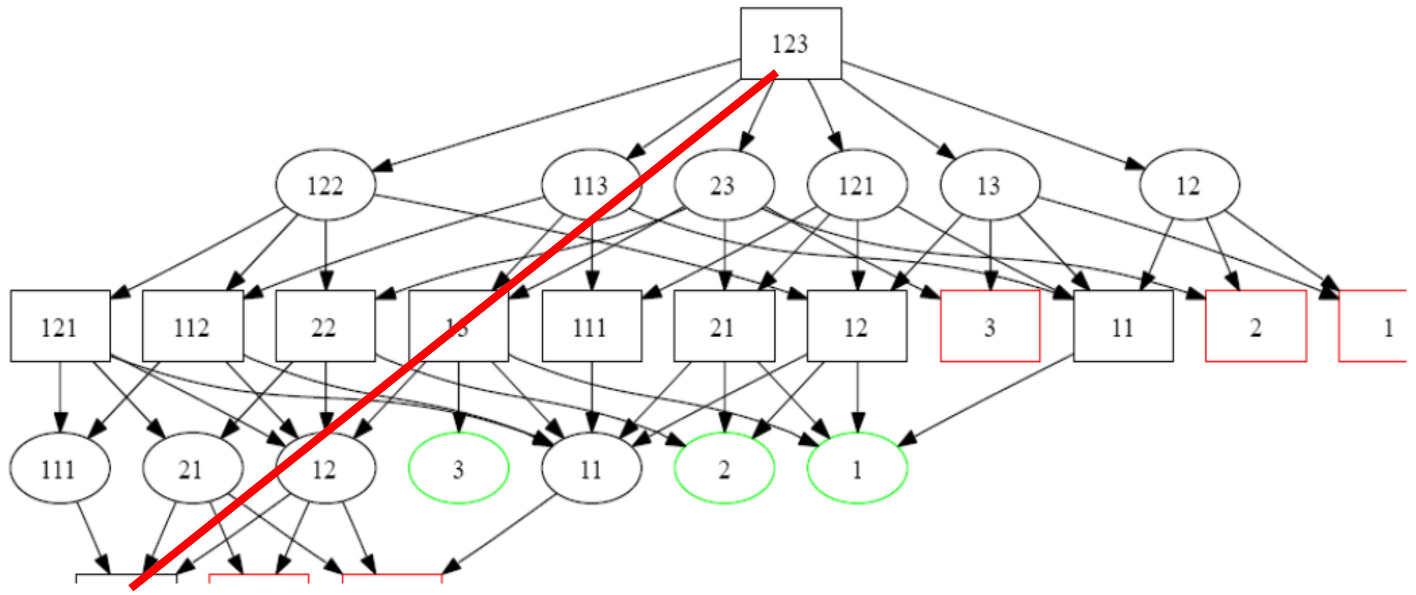
Games

- Formally, a game is $G = \langle V, V_0, V_1, E, \alpha \rangle$, where
  - V is a set of nodes.
  - V_0 and V_1 form a partition of V. V_0 concern the System and V_1 the Environment.
  - $E \subseteq V \times V$ is a set of edges.
- A *play* is $\pi = v_0, v_1, \ldots$
  - α is a *set of winning plays*.
- A *strategy* for player i is a function $f_i : V^* \cdot V_i \rightarrow V$ such that (v, $f_i(w \cdot v)) \in E$.
- A play $\pi = v_0, v_1, \ldots$ is compatible with f_i if for every $j \geq 0$ such that $v_j \in V_i$ we have $v_{j+1} = f_i\,(v_0 \cdots v_j\,)$.
- A strategy for player 0 is *winning* if every play compatible with it is in α. A strategy for player 1 is winning if every play compatible with it is not in α.
- A node v is won by player i if she has a winning strategy for all plays starting from v.

A play of a game

▭	Environment
⬭	System

[redacted]



Games - Realizability and Synthesis

- *Realizability*: Exists winning strategy for System.
- *Synthesis*: Obtain such winning strategy. How?
  - With a Transducer - Moore Machine (also Mealy depending on approach).
- Formally, T = $\langle \Delta, \Sigma, Q, q_0, \alpha, \beta \rangle$, where
  - Δ – input alphabet
  - Σ – output alphabet
  - Q – states
  - q_0 – initial state
  - $\alpha : Q \times \Delta \rightarrow Q$ – transition function
  - $\beta : Q \rightarrow \Sigma$ – output function.
- A transducer representing a winning strategy can be extracted from the winning states of the system after solving the game.
  - Details in the next lectures.

Game types for synthesis

- Safety games
  - Avoiding the „bad“ state of the safety automaton.
- Reachability games – *dual* to safety games.
  - Trying to reach a target state.
- Büchi games
  - Accepting states from which system can force returning to an *accepting* state *infinitely often*.
  - Almost the same as for reachability games.

Plan for next lecture

- Short overview of some state-of-the-art synthesis approaches based on LTL or fragments of it.
  - Generalised Reactivity (1) fragment of LTL and reactive synthesis, by Pnueli, Piterman and others.
  - Safety LTL and synthesis by Tabayara and Vardi.
  - Symbolic bounded synthesis by Ehlers.
  - Acacia+ LTL synthesis – Will learn the details of it in the rest of this course.
  - Lightweight comparison between them.
- The rest of the lecture will be dedicated to LTL transformation to automata and solving of the relevant games for these approaches and generating the winning strategy for the synthesised system.
  - Emphasis on Acacia+ approach. http://lit2.ulb.ac.be/acaciaplus/

References

- 1) A. Church, Logic, Arithmetic and Automata, Journal of Symbolic Logic, 29 (4), 1964.
- 2) J. R. Büchi, L. H. Landweber, Solving Sequential Conditions by Finite-State Strategies, Transactions of the AMS, Vol. 138 (Apr. 1969).
- 3) M. O. Rabin, Automata on Infinite Objects and Church's Problem, American Mathematical Society, Boston, MA, USA, 1972.
- 4) A. Pnueli. The temporal logic of programs. In Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS '77). IEEE Computer Society, Washington, DC, USA, 46-57. DOI: https://doi.org/10.1109/SFCS.1977.32, 1977.
- 5) P. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In Proc. 24th IEEE Symposium on Foundations of Computer Science, pages 185–194, Tucson, 1983.
- 6) S. Safra, On the complexity of ω-automata. In: Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press (1988).
- 7) A. Pnueli and R. Rosner. On the synthesis of a reactive module. In Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '89) ACM, NY, USA, 179-190. DOI=http://dx.doi.org/10.1145/75277.75293, 1989
- 8) M. Y. Vardi, The Siren Song of Temporal Synthesis
  - http://www.dis.uniroma1.it/~kr18actions/slides/invitedMosheVardi.pdf
- 9) N. Piterman, Games and Synthesis, EATCS Young Researchers School, 2014
  - http://eatcs-school.fi.muni.cz/_media/piterman.pdf