Revisiting Synthesis of GR(1) Specifications

Uri Klein & Amir Pnueli

Courant Institute of Mathematical Sciences, NYU

Haifa Verification Conference, October 2010

What Is Synthesis?

- Rather than implement and then test/verify, why not have systems that are correct by construction?
- Is our specification implementable at all? Is it consistent?
- An automatic transformation:

- What language?
- Restrictions?
- Expressive?

- How?
- Efficient?
- Doable?

- Efficient?

The LTL Synthesis Problem

- Given an interface specification:
  - A set of input variables X
  - A set of output variables Y
- Given a Linear Temporal Logic formula $\varphi(X,Y)$

- Is $\varphi(X,Y)$ realizable? (Realizability≠Satisfiability)
- If so, synthesize a system that realizes $\varphi(X,Y)$

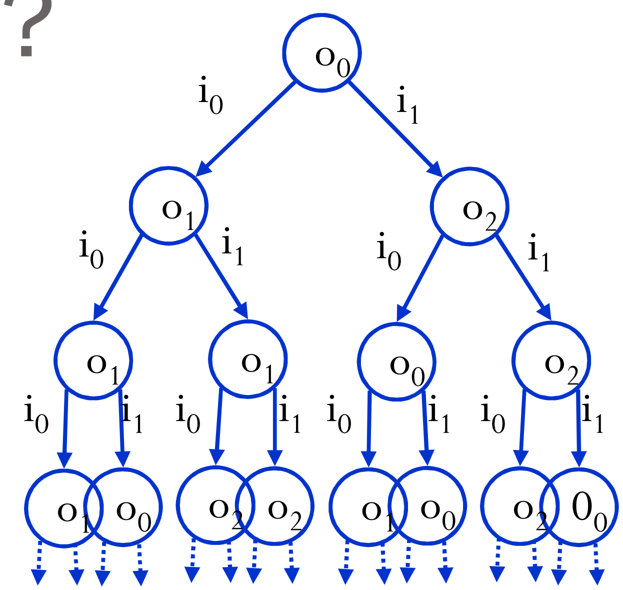
Essentially, convert the relation $\varphi(X,Y)$
to a causal function $f : history(X) \rightarrow Y$

Two Historical Approaches (1969)

1. Reducing the problem to emptiness of tree-automata
2. Solving a two-player game

Unfortunately, in '89, Pnueli & Rosner proved a doubly-exponential lower bound to the problem of realizability

Why Tree Automata?



- What is a system?

  $f : history(X) \rightarrow Y$

- Embed in a tree: nodes are labeled by system actions, edges labeled by all possible environment actions
- A tree automaton can check that all paths of a tree satisfy the specification
- Finally, check the emptiness of the tree automaton or, check for the existence of a tree

But…

- LTL realizability is 2EXPTIME-hard
- It includes determinization of Büchi automata as a subroutine
- It includes solving a Rabin/parity game

What Could Be Done for Realizability?

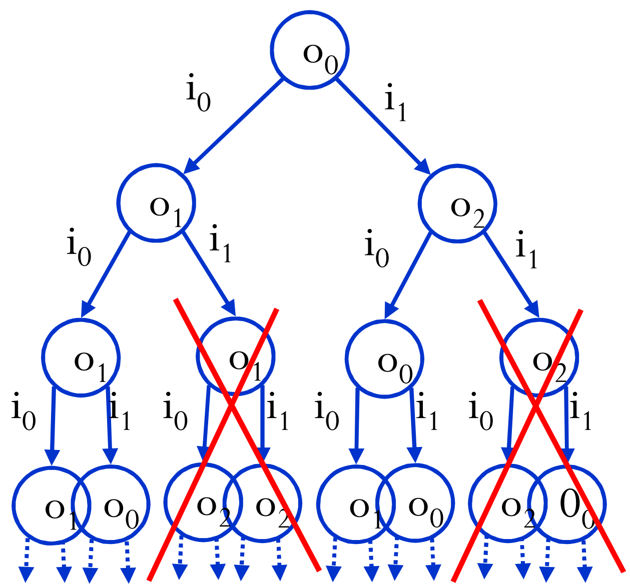
- Restricted sets of LTL specifications:
  - Asarin et al., '98: $O(N^2)$ for $\Box p; \Diamond p; \Box\Diamond p; \Diamond\Box p$ (N is size of the state-space)
  - Alur & La Torre, '04: Boolean combinations of $\Box p; \Diamond p$
  - Piterman et al., '06: $O(N^2)$ for GR(1) formulae, i.e., $\Box\Diamond p_1 \wedge \ldots \wedge \Box\Diamond p_m \to \Box\Diamond q_1 \wedge \ldots \wedge \Box\Diamond q_n$. This is a very expressive set
- Restricting the specifications, makes the search for trees easier

More on Piterman et al. (*The Syntactic Reduction*)

- In fact, it is claimed that realizability of formulae of the form $I_e \wedge \square S_e \wedge L_e \rightarrow I_s \wedge \square S_s \wedge L_s$ could be reduced to the existence of *safe trees* that satisfy GR(1) winning conditions, where
  - I_α is an initial condition, a Boolean formula
  - S_α is a Boolean transition relation between 'current' and 'next' states. $\square S_\alpha$ is, therefore, a safety property
  - L_α is liveness property such that $L_e \rightarrow L_s$ is a GR(1) winning condition
  - $\alpha = e$ is for the environment, $\alpha = s$ for the system

Safe Trees

- Using the components of the specification, safe trees include only paths that comply with I_α and with $\Box S_\alpha$



- The separation to components allows to search for smaller, pruned, trees, which is faster
- If a tree exists, it could be used to extract a synthesis

The Syntactic Reduction is Useful

- Was used, for the first time, to generate an actual circuit for AMBA's Advanced High-Performance bus (Bloem et al., '07)
- Was used as part of the production of several robot controllers (Kress-Gazit et al, '07, '08, '09;…)
- Synthesis from LSC specifications (Kugler et al., '09)
- This approach was extended to handle more LTL formulae (Sohail & Somenzi, '09; Sohail et al., '08)
- Was used within a method for the synthesis of asynchronous systems (Pnueli & Klein, '09)

Unfortunately, It Is Incomplete

- Roveri et al. (2006), came up with this specification:

  $\Box(\neg x') \wedge \Box\Diamond(x = y) \rightarrow \Box(x' = y') \wedge \Box\Diamond y$

- x here is an input controlled by the environment, y an output
- Using the syntactic reduction it is reported unrealizable
- However, it is clearly realizable (why?)

- This is a false negative!

When Do We Get False Negatives?

- Let us go back to the previous example:

$$\square(\neg x') \wedge \square\Diamond(x = y) \rightarrow \square(x' = y') \wedge \square\Diamond y$$

- The only way for the system to satisfy the specification is to violate its own safety requirement
- This results in a falsification of the assumption
- No such safe tree exists!

- The syntactic reduction misses this → a false negative

Recap

- The syntactic reduction was claimed to check existence of trees for specifications of the form:

  $\text{imp} : I_e \wedge \Box S_e \wedge L_e \rightarrow I_s \wedge \Box S_s \wedge L_s$

- It uses a syntactic analysis of the formula to limit the search of trees, extracting for each player a triplet $\langle I_\alpha, S_\alpha, L_\alpha \rangle$:
  - An initial condition
  - A safety property
  - A liveness property
- Changing the syntax of the specification, changes the algorithm's output!

The Syntactic Reduction Solves A Different Problem

- Theorem: Using the same syntactic analysis, this is the actual realizability problem that is solved:

$$\text{sep} : (I_e \rightarrow I_s) \wedge \left(I_e \rightarrow \Box\left(\left(\boxminus S_e\right) \rightarrow S_s\right)\right) \wedge \left(\left(I_e \wedge \Box S_e\right) \rightarrow \left(L_e \rightarrow L_s\right)\right)$$

- This is no longer an assume/guarantee formula
- Theorem: The syntactic reduction, however, is sound and synthesizes correct systems (no false positives since ‘sep’ implies ‘imp’)

Well-Separation of Environments

- Such cases are avoided with well-separated environments:

  > Definition - *Well Separated Environments*: cannot be forced to violate their own requirements, and can always continue with an infinite computation

- Theorem: The syntactic reduction is sound & complete for well separated environments
  - No more false negatives
  - Synthesized systems are correct (despite difference between imp & sep)
- Well separation is independent of systems

But How Can One Find Out?

- Luckily, well separation could be tested efficiently by the syntactic reduction itself
- Consider the following specification

$$I_e \wedge \Box S_e \wedge L_e \rightarrow T \wedge \Box T \wedge \Box \Diamond F$$

  - No limit on the system's transitions (T) allow to consider safe trees for all possible behaviors of the environment
  - Limiting the system's liveness (F), forces it to search for ways to falsify the environment's specification

- If no safe tree exist here, the environment is well separated
- For such specifications, 'sep' & 'imp' are equi-realizable

Can We Handle the General Case?

- Yes! If the system has no initial condition & safety property, all of its behaviors would exist in some safe trees
- The solution is to disguise these two components as system liveness
- This is done via temporal testers:
  - Require that $\boxminus(\circleddash \mathrm{T} \vee I_s)$ holds infinitely often
  - Require that $\boxminus S_s$ holds infinitely often

The Syntactic Reduction Is Now Both Sound & Complete

- Theorem: Using the temporal testers, the syntactic reduction can handle correctly even non-well-separated environments
- However, this comes with some price:
  - The resulting system is bigger and less 'readable'
  - It takes more time to compute
- The simple solution:
  - Check first for well-separation
  - Use testers only if not well-separated

Possible Future Work

Well-separation seems too strong a demand. What if a system could win both 'fair' and 'dirty' play? Is there a tighter definition?

Summary

- Synthesis of LTL is prohibitively expensive
- For GR(1) specifications, it is doable efficiently, and useful
- The existing algorithm was incomplete. It does not compute what it was supposed to & generates false negatives
- We explained when it fails, and proved what it actually computes
- We defined a class that contains all 'bad' specifications, and showed how to classify them efficiently
- We suggested a not-too-costly reduction to handle such 'bad' specifications soundly

Thank You