Synthesis of Reactive Programs with Structured Latent State
Ria Das, Joshua B. Tenenbaum, Armando Solar-Lezama, Zenna Tavares
MIT CSAIL

Motivation

Objective: To inductively synthesize functional reactive programs, i.e. from a finite data sequence.

Why does it matter? Reactive settings are plentiful in the real world (e.g. a robot or self-driving car operating on the street and updating its time-varying environment model, or a child learning how a video game works by watching for some time), but existing techniques do not learn these programs from data. Standard reactive synthesis inputs a logical formula and outputs an automaton. Programs are often more useful representations than automata, because large numbers of automaton states can be abstractly expressed in compact programs.

Why is it hard? While programs scale better and are more useful, standard methods for functional program synthesis cannot synthesize time-varying latent state, the core element of reactive settings. Precisely, functional synthesis expects its inputs and outputs to be fully observed, but both the inputs and outputs are partially observed in a reactive setting.

Our Solution: How can we inductively synthesize programs with time-varying latent state? Our approach is to integrate functional and (inductive) automata synthesis. We first try to synthesize the program using functional synthesis; if this fails, automata synthesis generates new latent state that then enables functional synthesis to succeed.

Methodology: We instantiate our algorithm in the domain of time-varying, interactive, Atari-like grid worlds, and write programs using a language called Autumn. An Autumn program defines object and latent (integer) variables, and describes grid-world dynamics using statements of the form on event update, where update changes a variable’s value. Given a observed sequence of grid frames and user actions, we seek the program in the Autumn language that generates the observations. Concisely, we want to learn (latent) variables and on-clauses.

Running Example: In the Mario program (top-center column), the agent (red) moves around with arrow keys and collects coins (gold). If the agent has collected a positive number of coins, on a click event, a bullet (black) is released upwards, and the agent’s coin count is decremented. The number of collected coins is not displayed anywhere on the grid at any time, so the only way to write an Autumn program for Mario is to define a latent or invisible variable that tracks the number of coins.

The AutumnSynth Algorithm: An Overview

INPUT
Sequence of grid frames and (not shown) user actions.

PERCEPTION & TRACKING
Objects are parsed from frames and mapped to objects in subseq. frames.

UPDATE FUNCTION SYNTHESIS
Each object-object mapping is described with an Autumn expr. (update function).

EVENT SYNTHESIS
For each distinct update function in an object type, a trigger event is sought (i.e. one that is true at exactly the update func’s times).

AUTOMATA SYNTHESIS
When no matching event is found, new latent state (in the form of an automaton) is made so a matching event can be written. Here, we have named the latent variable numCoins.

OUTPUT
Autumn program that creates observations.

Example Synthesized Automata

Water Plug: Clicking on an empty square adds a colored square. The square color depends on the last of the three leftmost buttons clicked.

Paint: Inspired by MSFT Paint. Clicking an empty square adds a colored square, and the five different colors may be cycled through by pressing up.