Synthesis of Causal Reactive Programs with Structured Latent State

Ria Das[1], Joshua B. Tenenbaum[1], Armando Solar-Lezama[1], Zenna Tavares[2]

[1]MIT, [2]Columbia University

Motivation

Background: Without being told the rules, children can figure out how a new toy or video game works after just minutes of observation. This flexible, data-efficient ability to discover causal models has yet to be replicated in a machine. Existing methods fall short: CGMs cannot concisely express this complexity, and deep learned models are too data-hungry.

Approach: We frame the problem of discovering a causal model from observed data as one of *program synthesis*. We focus on the domain of time-varying, Atari-like grid worlds, and represent causal models using a language called AUTUMN. Discovering the causal structure underlying an observation sequence is equivalent to identifying the *program* in the AUTUMN language that generates the observations.

Challenge: How do we learn the time-varying latent state in a causal program? For example, in the Mario program (top-center column), the agent (red) moves around with arrow keys and can collect 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. Existing methods cannot do this.

Solution: Learn latent state via automata learning.

The Autumn Language

The AUTUMN language was designed to concisely express a rich variety of causal mechanisms in interactive grid worlds. Variables are defined as streams of values over time, via the syntax `var = init expr1 next expr2`, and a variable's previous value can be accessed using the primitive `prev`. For example, the agent object in Mario is defined with `agent = init (Agent (Position 15 7)) next moveDownNoCollision (prev agent)`, indicating that at every time, the agent should move down unless it would collide with another object. The default `next` behavior for a variable may be overridden using *on-clauses*, which are expressed as

```
on event
  update_function,
```

where `update_function` is a variable update of the form `var = expr`, and `event` is a Boolean which, when true, triggers the override.

The AutumnSynth Algorithm: An Overview

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

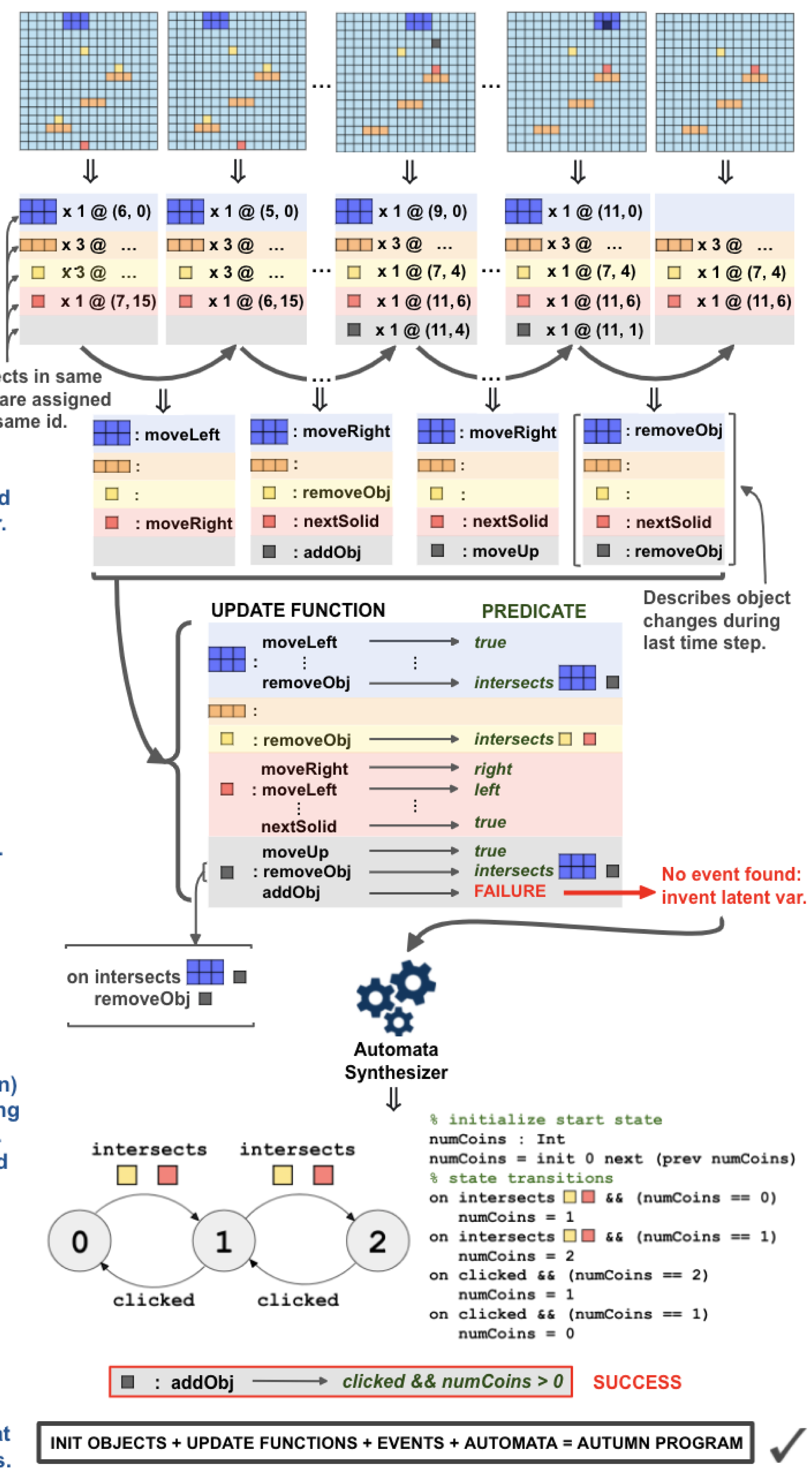
PERCEPTION & TRACKING
Objects are parsed from frames and mapped to objects in subseq. frames.

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 Autumn Programs

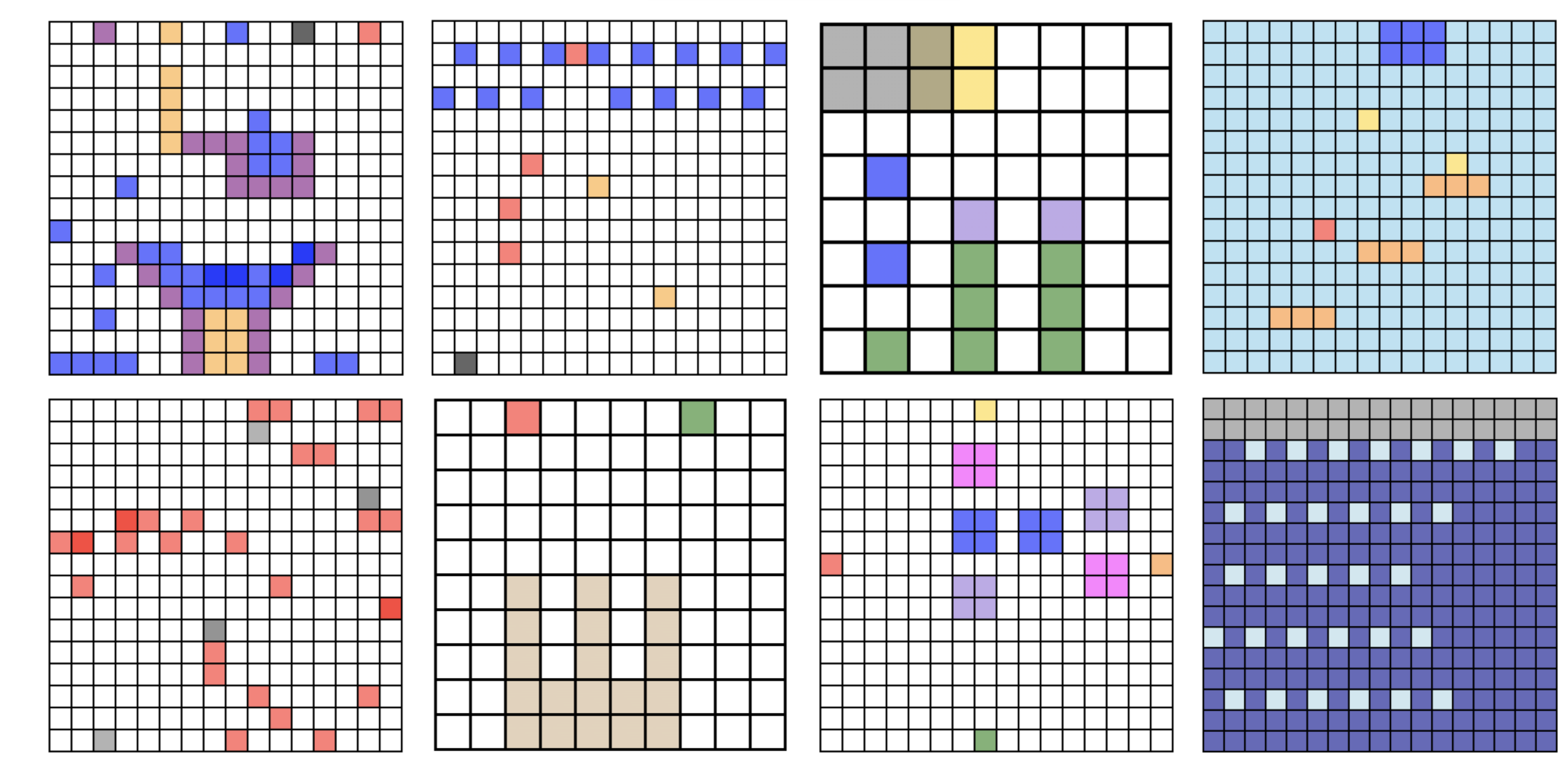
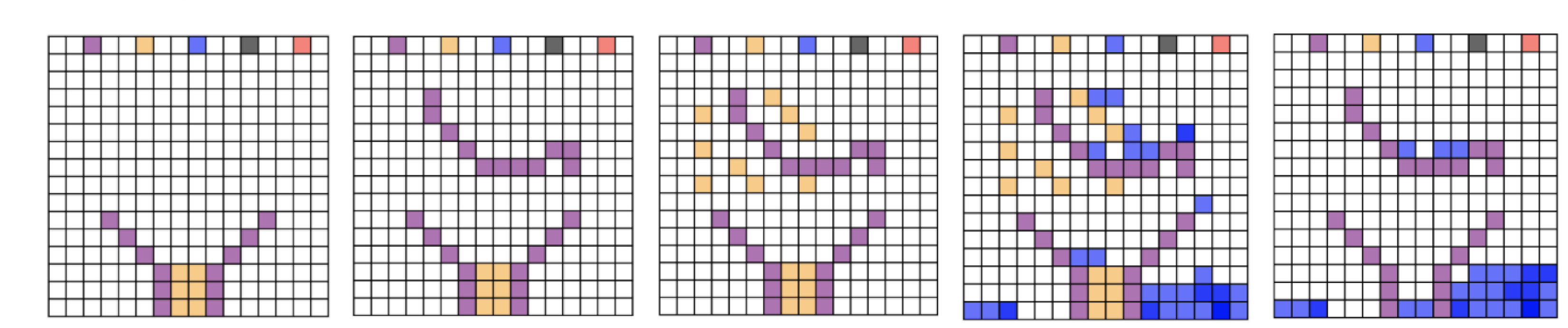
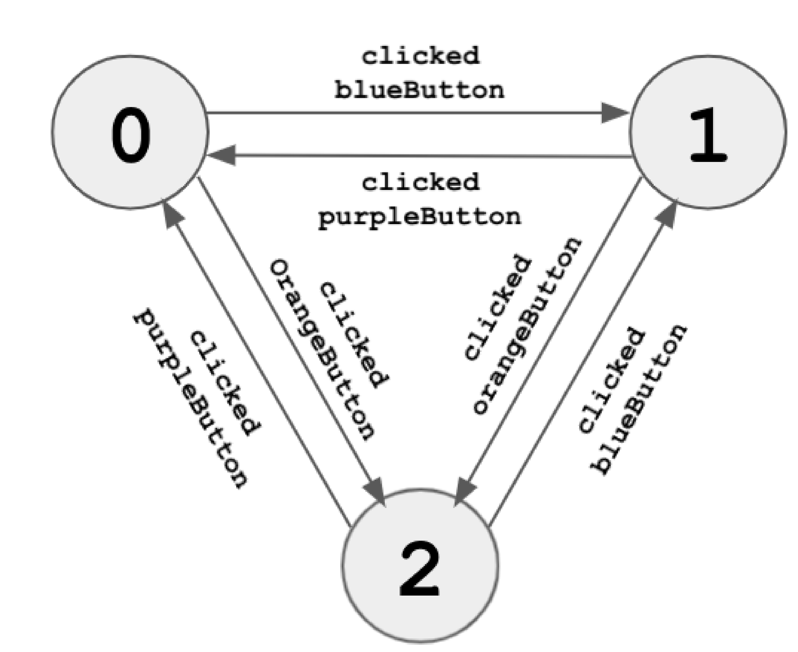


Figure 1: A collection of AUTUMN programs. Clockwise from top-left: Water interacting with a sink; a clone of Space Invaders; plants growing under sunlight and water; a distillation of Mario; snow falling under different wind conditions; an abstract, alternative gravity simulation; a sandcastle that erodes with water; and ants foraging for food.

Example Synthesized Automata

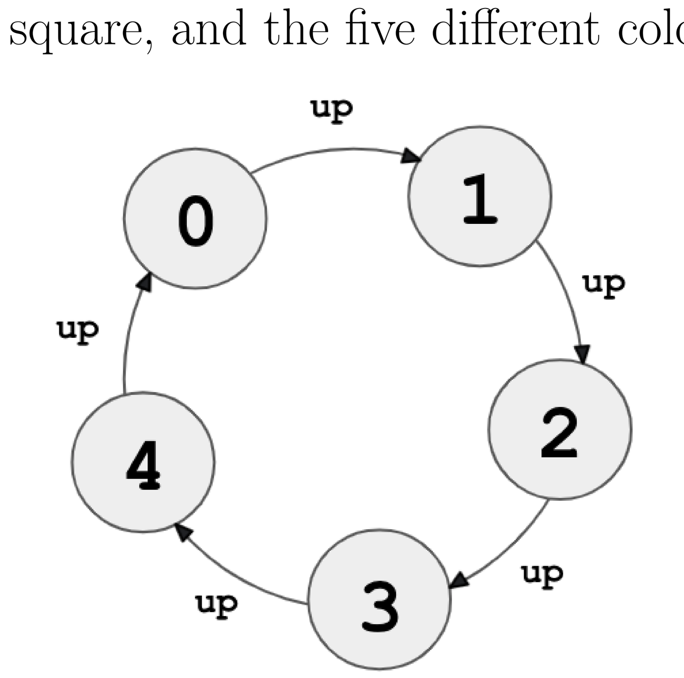
Water Plug: Clicking on an empty square adds a colored square. The square color depends on the last of the three leftmost buttons clicked.






```
// initialize start state
active : Int
active = init 0 next (prev active)
// state transitions
on clicked blueButton
  active = 1
on clicked orangeButton
  active = 2
on clicked purpleButton
  active = 0
```

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.




```
% initialize start state
activeColor : Int
activeColor = init 0 next (prev activeColor)
% state transitions
on up && (activeColor == 0)
  activeColor = 1
on up && (activeColor == 1)
  activeColor = 2
on up && (activeColor == 2)
  activeColor = 3
on up && (activeColor == 3)
  activeColor = 4
on up && (activeColor == 4)
  activeColor = 0
```