CHARDA
Causal Hybrid Automata Recovery via Dynamic Analysis

Adam Summerville, Joseph Osborn, Michael Mateas
University of California, Santa Cruz
{asummerv, jcosborn} @ucsc.edu, michaelm@soe.ucsc.edu

Abstract

We propose and evaluate a new technique for learning hybrid automata automatically by observing the runtime behavior of a dynamical system. Working from a sequence of continuous state values and predicates about the environment, CHARDA recovers the distinct dynamic modes, learns a model for each mode from a given set of templates, and postulates *causal* guard conditions which trigger transitions between modes. Our main contribution is the use of information-theoretic measures (1) as a cost function for data segmentation and model selection to penalize over-fitting and (2) to determine the likely causes of each transition. CHARDA is easily extended with different classes of model templates, fitting methods, or predicates. In our experiments on a complex videogame character, CHARDA successfully discovers a reasonable over-approximation of the character's true behaviors. Our results also compare favorably against recent work in automatically learning probabilistic timed automata in an aircraft domain: CHARDA exactly learns the modes of these simpler automata.

1 Introduction

Hybrid automata (HAs) combine discrete finite state machines with continuous variables [Alur *et al.*, 1993]. These continuous variables are updated at different rates in different *states* (also called *modes*) according to state-specific *flow constraints*. *Transitions* between states may be *guarded* on conditions involving (classically) the continuous variables or other predicates, and these transitions may *update* continuous variables to new values instantaneously. States may also have associated *invariant* conditions; if an invariant is violated, the state immediately exits along one of its available transitions.

Hybrid automata are a convenient notation for many different dynamical systems, and have at least semi-decision algorithms for a variety of interesting properties (e.g. satisfiability of LTL formulae, general reachability, and existence of optimal control policies) [Alur *et al.*, 1995; Henzinger *et al.*, 1995; Henzinger and Kopke, 1999]. Learning (or recovering) HAs from existing systems yields convenient abstractions for human analysis and high-level automated planning; moreover, these abstractions can be refined, possibly automatically (via new data or experimentation).

In this work we present *CHARDA*, Causal Hybrid Automata Recovery via Dynamic Analysis, a non-parametric framework that learns an HA from observations of a dynamical system. CHARDA has two phases: mode identification and causal guard learning. We identify modes via a dynamic programming approach that segments the trace and finds switchpoints where the dynamics of the system change. Then CHARDA learns causal guard conditions for mode-to-mode transitions using information-theoretic measures.

CHARDA's segmentation requires no prior knowledge of the number of potential modes or the location of switchpoints, requiring only a set of potential model templates (e.g. $\dot{x} = a$ or $\dot{x} := a; \ddot{x} = b$, read respectively as constant velocity or constant acceleration b starting from a reset velocity value a). Although the models can take any form (so long as a likelihood function is available), here we use general linear models (multivariate linear regressions). CHARDA performs model selection and segmentation via a principled penalty function. In this work, we tried both the Bayesian Information Criterion (BIC) and Minimum Description Length (MDL), but CHARDA is also penalty-function-agnostic.

We demonstrate CHARDA in a novel domain: videogames, specifically *Super Mario Bros* (SMB). Games offer a unique set of challenges including non-physical dynamics and potentially very frequent mode transitions on the order of fractions of a second. As a domain, games lie somewhere between synthetic data and a physical robot or other cyber-physical system. Furthermore, games are interesting objects of analysis in their own right. In games specifically, CHARDA has some exciting applications:

- In the General VideoGame (GVG) playing domain, an AI could derive HA models for game entities and then do planning on this abstracted space without relying on a forward model [Perez-Liebana *et al.*, 2016]
- Model-checking/safety analysis of character automata without the overhead of manual modeling by human game designers [Smith *et al.*, 2009]
- Extracting features for quantitative comparative analysis between games or game rules [Fasterholdt *et al.*, 2016; Ho *et al.*, 2016]
- Automatic scraping of characters from existing games

for a character behavior corpus, which could then be used for analysis or procedural generation as game levels are already [Summerville *et al.*, 2016]

The rest of the paper is structured as follows. First, we discuss other approaches to learning dynamical system models and how CHARDA fits into the existing work here. We then briefly introduce the concrete domain of interest and explain CHARDA's design and implementation. Finally, we evaluate CHARDA in two domains: internally on the SMB domain, and externally in an aircraft tracking domain for comparison with another recent automaton learning algorithm.

2 Related Work

Hybrid automata are an attractive computational model for analysis, control synthesis, and estimation of real-world systems. The inclusion of discrete behavior makes them expressive enough to describe many dynamical systems of interest, and although many classes of hybrid automaton have strong undecidability results [Henzinger *et al.*, 1995] there are efficient semi-decision procedures to determine configuration reachability or equivalence between automata [Alur *et al.*, 1995]. Hybrid automata, suitably constrained, can also be directly implemented in software or hardware, with proofs about the model translating to the implemented system (given assumptions of e.g. component failure rates and latencies).

Despite the general undecidability of many HA properties, it is possible to constrain models or carefully choose semantics to obtain different analysis characteristics: discretizing time or variable values evades undecidability by approximating the true dynamics [Jha *et al.*, 2007]; keeping these continuous but constraining the allowed flow and guard conditions admits geometric analysis [Frehse, 2005]; and one can always merge states together to yield an over-approximation, producing smaller and simpler models. There are also composable variations of hybrid automata that admit compositional analysis [Alur *et al.*, 2003] as well as a logical axiomatization [Platzer, 2008], not to mention the body of tools and research that already exist for synthesizing control policies, ensuring safety, characterizing reachable areas, et cetera.

Given the desirable properties of this class of model, and the ready availability of tools for dealing with them, many researchers have explored automatically recovering these high-level models from real-world system behaviors. CHARDA shares motivations with HyBUTLA [Niggemann *et al.*, 2012], which also aimed to learn a complete automaton from observational data. HyBUTLA seems able to learn only acyclic hybrid automata, since it works by constructing a prefix acceptor tree of the modes for each observation episode and then merges compatible modes from the bottom up. Moreover, HyBUTLA assumes that the segmentation is given in advance and that all transitions happen due to individual discrete events, presumably from a relatively small set. The overall structure of both algorithms—split the observations into a number of intervals in which mode functions are fit, then merge redundant modes—is similar, but CHARDA learns a larger class of automata and does not require data to be pre-split into episodes or segments.

Santana *et al.* [2015] learned Probabilistic Hybrid Automata (PHA) from observation using Expectation-Maximization. At each stage of the EM algorithm a Support Vector Machine was trained to predict the probability of transitioning to a new mode. Unlike CHARDA, their work requires a priori knowledge about the number of modes.

The closest work to ours is that of Ly and Lipson[2012] which used Evolutionary Computation to perform clustered symbolic regression to find common modes with the Akaike Information Criterion uses to penalize model complexity. However, unlike CHARDA their work assumes *a priori* knowledge about the number of modes. Moreover, since their work assigns individual datapoints, not intervals, to a mode, their approach can only model stationary processes.

Several approaches have sought to learn models that describe dynamical systems' behavior. Hidden Markov Models [Baum and Petrie, 1966] learn probabilistic state transitions between a hidden state and the observed data. The Infinite HMM [Beal *et al.*, 2002] extends this to an unbounded number of states which assumes a Chinese Restaurant Process governs the state space. These approaches do not characterize guard *conditions*, but instead learn the *probability* of taking state transitions at each instant.

Data segmentation has a natural connection to automaton learning, and CHARDA uses an approach based on least squares regression [Bellman and Roth, 1969]. Model-based recursive partitioning [Zeileis *et al.*, 2008] is an alternative family of techniques which fits a model to the entire dataset and then iteratively and greedily splits that model until reaching a threshold quality level or split count. Unfortunately, each split is only locally optimal so there are no guarantees about global optimality. The Forget-Me-Not-Process [Milan *et al.*, 2016] finds a partitioning of time segments that allows for models to be repeated across different partitioned segments; however, it only works for stationary processes, i.e. distributions that do not change over time.

In terms of finding abstract models specifically of Nintendo games, we were inspired by Murphy's work [2016] in automatically determining physical properties of game characters. That project, like ours, examined runtime memory structures to determine where objects were; they further explored, through experimentation, causal linkages between arbitrary locations in RAM and the visual position of characters on the screen. These relations were used to drive other experiments, e.g. to discover whether game characters fell due to gravity or whether their movement was obstructed by particular types of game objects. In a sense, their work is an ad hoc property-based testing approach to learning which of a fixed set of properties holds. Our work requires less domain knowledge and captures the characters' behavior more precisely.

In the future we look forward to combining our more general approach with such knowledge-rich techniques to capture more complicated interactions between multiple agents and their environment. A recent publication by Summerville *et al.* [2017] similarly used games as their domain, attempting to find causal interactions shared by different entities, and we build on this approach for the causal guard learning.

3 Domain

CHARDA learns hybrid discrete/continuous behaviors of videogame characters or other agents whose inputs and movement behavior are observable. We obtain these inputs from an example playthrough of a game (e.g. SMB), assuming these inputs are representative of the character in question. Replaying this input sequence once through a software emulator of the game's hardware platform, we read out high-level features from the simulated graphics hardware and assemble those into distinct agents whose positions are tracked over time (we elide the details for space). Importantly, characters may pop in and out of existence, collide with fixed or moving obstacles of various types, or perform other arbitrary (often non-physical) behaviors. We can only observe characters' positions at a resolution of 1 pixel (a character is generally 8–32 pixels high); even then, the game world and our sensing are at a $\frac{1}{60}$-second fixed discrete time interval. All our position readings are therefore inaccurate by up to one spatial unit, and these errors naturally propagate to velocity and other calculations.

The input to our automaton learning process for a single entity is: sequences of discrete variable values that are possible control inputs (e.g. button presses), continuous variable values, and sets of predicates describing facts in external theories such as collision (e.g., the character was touching an object with appearance A at time t on one side or another). The goal is to go from that input data, presumed to be representative of the entity's "true" behaviors, to an abstraction suitable for planning or other purposes. This type of data is not hard to obtain for cyber-physical systems under the analyst's control or in cases where the possible causes for behavior change can be observed at some precision (even a probability distribution for these causes would suffice).

In this work, we look at learning a constrained class of hybrid automata from a combination of controlled (or at least witnessed) inputs and observed outputs. Specifically, though the learned automata may have any structure in terms of the number of modes and transitions, the modes may only have flows from a given set of model *templates*. In this specific work (and without loss of generality), every mode's flow condition is a specialization of $\ddot{x} = a, a \in \mathbb{R}$; moreover, all transitions leading into a given mode are forced to have the same update function, either $\dot{x} := n, n \in \mathbb{R}$ or the empty update. Finally, the set of guard conditions is currently assumed to be conjunctions of predicates from a given labeled set. Our causal learning component learns which of these predicates is most associated with the transitions, and prefers those predicates which are more strongly causal. There is no reason these guards could not also be learned as e.g. linear inequalities, since we know the set of modes and their active intervals at the time of cause assignment. Again, we focus here on learning reasonably small over-approximations of the true model: these can always be refined, but we don't want to exclude any witnessed behaviors.

4 Method

We break down the hybrid automaton learning process into two parts: Identifying modes and determining causes for transitions. Again, these algorithms operate over a sequence of continuous variable values and a sequence of sets of predicates describing the automaton's environment at each instant. We roughly follow the classic dynamic programming solution to the segmented least squares problem [Bellman and Roth, 1969] with a number of distinctions:

- Different model templates are considered for each segment, instead of a single least squares regression
- A principled penalty instead of a hand-chosen constant
- Merging segments if it results in a more optimal model.

4.1 Mode Identification

The mode identification process first requires the construction of all possible models for all possible sub-intervals. Let T be a table of model parameters with one entry for each interval i, j and model template m. Then we define T's entries as:

$$T[i, j, m] = \text{train}(m, d[i : j]) \quad \text{s.t.}$$
$$1 < i < j < n, m \in \mathcal{M}$$

where n is the number of potential switchpoints, $\mathcal{M}$ is the set of model templates, d is the dataset, and $T[i, j, m]$ is the model of template m trained on data from the interval of i to j. For this work our set of models are all multivariate regressions, but our approach is general enough to work with any approach that supports a likelihood function $\mathcal{L}(m|d)$.

The cost for a given model m for sub-interval i to j is therefore:

$$C[i, j, m] = -\log(\mathcal{L}(T[i, j, m]|d[i : j])) + \text{pen}(m, d[i : j])$$

given the penalty criterion pen. For this work we considered two penalties for model complexity. We wanted a principled measure for model complexity for the selection of a given sub-model for an interval, for when a break should occur (due to the inclusion of a switch point increasing model complexity), and for when a merging of modes should occur (due to the inherent fact that two similar but distinct modes are more complex than one mode). To that end we considered both the Bayesian Information Criterion (BIC) [Schwarz and others, 1978] and the Minimum Description Length (MDL) [Stine, 2004].

$$\text{BIC} = -\log(\mathcal{L}(m|d)) + \dim(m)\log(n)/2$$
$$\text{So: } \text{pen}_{\text{BIC}} = \dim(m)\log(n)/2$$

Where $\dim(m)$ is the number of parameters in model m and n is the number of datapoints in dataset d.

$$\text{MDL} = -\log(\mathcal{L}(m|d)) + \dim(m)(1 + \log(n)/2)$$
$$\text{So: } \text{pen}_{\text{MDL}} = \dim(m)(1 + \log(n)/2)$$

The two measures are very similar, being asymptotically the same, but differ in the constants applied to the penalty term. BIC assumes a Bayesian standpoint and determines which model from a set of models is the true model. It operates asymptotically as n trends to ∞, given a fixed loss for choosing the wrong model. MDL instead takes an information theoretic standpoint and assumes a spike-and-slab prior distribution for each parameter. Given that prior it takes approximately $(1 + \frac{\log(n)}{2})$ bits to encode the parameter, 1 bit for

whether the parameter $\theta = 0$ (i.e. is a slab) and $\frac{\log(n)}{2}$ bits to encode its value (i.e. if it is a spike).

For all segments that end at point j we find the optimal model and segmentation that leads to that point. $O[j]$ is the optimal cumulative cost of models across segments up to datapoint j.

$$O[j] = \begin{cases} 0 & \text{if } j = 0 \\ \text{argmin}_{0 \le i \le j, m} \; C(i, j, m) + O[i-1] & \text{if } j > 0 \end{cases}$$

We use dynamic programming to work backwards from the last switch point, finding the optimal sequence of segments that produces the optimal set of models,

After segmentation, the segments' models are merged if this will improve the overall attractiveness of the entire model, namely by reducing the number of parameters in the overall model by a large enough amount that the decrease in complexity is greater than the decrease in likelihood.

This is accomplished by constructing a new model from the data for segments s_1, s_2 concatenated to $d[s_1 s_2]$:

$$m_{s_1 s_2} = \text{argmin}_{m \in \mathcal{M}} \; \log \mathcal{L}(\text{train}(m, d[s_1 s_2]) | d[s_1 s_2])$$

The overall sequence of models is improved by the merging if the following inequality holds:

$$\begin{gathered} -\log(\mathcal{L}(m_{s_1 s_2} | d[s_1 s_2])) + \text{pen}(m_{s_1 s_2}) < \\ -\log(\mathcal{L}(O[s_1])) - \log(\mathcal{L}(O[s_2])) + \\ \text{pen}(O[s_1]) + \text{pen}(O[s_2]) \end{gathered}$$

4.2 Guard Learning

From these merged modes, causal guarded transitions between modes are learned by finding probabilistically likely conditions where the direction of causality is known. Our target domain comes with some advantages for ascribing causality, namely we have inputs supplied by a player and we can be sure of the direction of causality regarding them; however, any domain that allows for instrumentation of exogenous inputs can utilize our same methodology. Another potential source of causal transition guards in our domain is collisions between visible entities, of which, again, we can be sure of the direction of causality. We also look at endogenous variables as a last resort (and then mainly qualitatively), since causality is much harder to ascertain: for example, if we enter a mode with flow $\dot{y} = -4$ it could be that $\dot{y}$ is saturating at a terminal velocity, or it might be for some other reason.

For the SMB domain we consider the following set of predicates for guard condition learning:

- *Control* I (Pressed; Held; Released) — A change in the binary control input I — *Exogenous*
- *Collision* with X from direction Y — Collision with another entity, X, from a given direction Y — *Exogenous*
- 0*-in,* 0*-out by Sign* - A zero crossing or touching in velocity and its characteristics (e.g. from negative to positive, or vice versa) — *Endogenous*
- *Velocity Extremum* - $\dot{x} = \text{ext}_{\pm}(s)$ - the velocity is roughly equal to the extremum for a given mode s — *Endogenous*
- *Acceleration Sign* — $\ddot{x}$ has the sign -1, 0, or 1 — Endogenous
- *Velocity Sign* — $\dot{x}$ has the sign -1, 0, or 1 — *Endogenous*

The *Control* and *Collision* predicates are given priority as we can be sure of their direction of causality.

Summerville *et al.* used Normalized Pointwise Mutual Information (NPMI) to learn semantic information about game objects [2017], which led us to believe that we could determine transition guards using a similar technique. We calculate the NPMI of each transition from a predecessor mode to a successor mode with each predicate active during the predecessor mode. NPMI is a scaling of pointwise mutual information defined as:

$$\text{npmi}(x, y) = \frac{\text{pmi}(x, y)}{-\log(p(x, y))} = \frac{\log(\frac{p(x,y)}{p(x)p(y)})}{-\log(p(x, y))}$$

NPMI for two events is -1 when they never co-occur, 0 when independent, and 1 when they always co-occur. In this work we considered two different thresholds for NPMI, 0.9 for *universal* (present all, or nearly all, the times that transition is taken) events and 0.4 for *relevant* events. For example, to learn the cause for transitions from hypothetical mode **A** into mode **B**, we look at all time intervals where **A** is active, determine for each predicate how strongly correlated it is with the transition event $\mathbf{A} \Rightarrow \mathbf{B}$, and take all those passing a threshold to be causes. These correspond to conjuncts in the guard condition. Those correspondences which are high enough to be of interest but do not meet the threshold are called *relevant* and are possible disjuncts in the guard condition (assuming it has the form $c_1 \wedge \ldots \wedge c_i \wedge (d_1 \vee \ldots \vee d_j)$). If we have an exogenous explanation, we discard endogenous explanations.

We may have cases where out-transitions of a mode are non-deterministic: they have identical causes, or one's causes subsume another. In these situations the offending target modes are merged, one pair at a time, re-connecting edges as necessary until a fixpoint is reached. This merging greedily abstracts the true automaton, but in practice it seems to work well for domains like game characters whose discrete state changes are generally strongly tied to control inputs or collisions; future work will explore more sophisticated approaches to resolving non-determinism.

5 Evaluation

To evaluate our work we considered two domains: Aircraft Dynamics Modeling and Mario's Jump Dynamics from SMB.

We explore the use of CHARDA in aircraft modeling for a direct comparison with Santana *et al.* [2015]. Their approach used Expectation Maximization [Dempster *et al.*, 1977] to recover a hybrid automaton from observational data by iteratively refining an Interactive Multiple Model. Guard conditions were learned by applying support vector machines. As in Santana, we also include results for a Jump Markov Linear System (JMLS) which assumes Markovian transitions.

The aircraft model is given in two distinct scenarios: the first, "Lawnmower" (see Fig. 1), features an aircraft moving in a constant velocity for some period of time and then making a constant-rate turn to reverse heading, repeating this

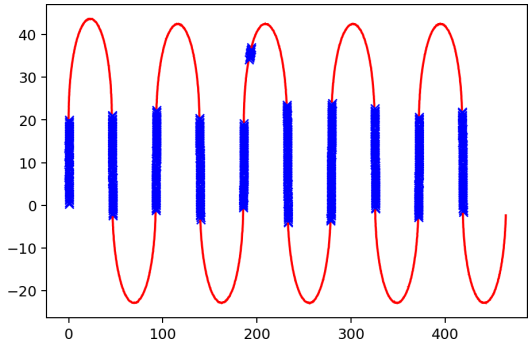

Figure 1: The Lawnmower data, as segmented by CHARDA with BIC. Beyond having slight errors on the beginning and end of the turns, there is one turn where it incorrectly reverts to a constant velocity in the middle. *Only the switch point detection portion of CHARDA is used.

pattern for some number of iterations. In the second scenario, "Random," the aircraft makes a given maneuver (either constant heading or constant turn) for 50 time steps and then changes to a random maneuver; this is repeated 17 times. We must note that this portion of our evaluation is only based on CHARDA's segmentation algorithm and does not employ transition guard learning. As the observational data offers no causal information indicating why a mode transition might be made, we do not learn any causal transition guards (which would simply overfit the given observations).

As in Santana's work we ran 32 trials and discarded the best and worst runs; the results are shown in Table 1. We see that for the Lawnmower domain that we outperform Santana *et al.*, but both are close enough to the ground truth that the difference is negligible. In the Random domain we outperform the prior work dramatically because our segmentation is not based on learning linear guards; we instead find an optimal segmentation based on model accuracy and complexity. We must note again that there are no real causes for why the aircraft changes maneuvers, so it is impossible to learn true causal guards. Santana *et al.* learn correlative guards for a given training instance, but their learned guards are not applicable to unseen data because they are tuned to that specific training instance (for example, if the aircraft's flight pattern was rotated or translated, all of their learned guards would be invalidated due to their training domain and linear nature). As such, we feel that it is only relevant to compare the segmentation portion of CHARDA to the prior work. CHARDA would be better-suited if the domain were framed as a control problem and the dataset contained features like operator controls and aircraft sensors.

For the Mario domain, we made no assumptions about the number of true modes and let the non-parametric nature of our approach attempt to recover the correct modes. This means that we are unable to compare to Santana *et al.* as it requires the number of modes a priori, so instead we compare our results to a manually-defined automaton based on human reverse-engineering of the game's program code [jdaster64, 2012] (see Fig. 2). We present the HAs learned by CHARDA in Figure 4. The Mario trace used for this work was 3772

Method	Data	Attribution Error
JMLS	Lawnmower	53.93%
PHA	Lawnmower	3.33%
CHARDA	Lawnmower	**2.75%***
JMLS	Random	58.91%
PHA	Random	63.2%
CHARDA	Random	**4.10%***

Table 1: Percentage of modes misattributed for CHARDA, PHA, and JMLS. The results shown here are only based on the segmentation portion, and do not include causal guard learning as there are no causal reasons for the mode transitions.

On Ground $\dot{y} = 0$ — Caused by Mario colliding with something solid from above

Jump(1,2,3) Three jumps with parameters:

- $\dot{y} := 4, \ddot{y} = -\frac{1}{8}$
- $\dot{y} := 4, \ddot{y} = -\frac{31}{256}$
- $\dot{y} := 5, \ddot{y} = -\frac{5}{32}$

Entered from **On Ground** when the **A button** is pressed and $|\dot{x}| < 1$, $1 \leq |\dot{x}| < 2.5$, or $2.5 < |\dot{x}|$, respectively

Release(1,2,3) $\dot{y} := min(\dot{y}, 3)$ — Entered from the respective **Jump** when the **A** button is released; $\ddot{y}$ same as respective **Jump**.

Fall(1,2,3) Falling at one of three rates: $\ddot{y} = -\frac{7}{16}$, $-\frac{3}{8}$, or $-\frac{9}{16}$; entered from the respective **Jump** or **Release** mode when the apex is reached ($\dot{y} \leq 0$)

Terminal Velocity(1,2,3) $\dot{y} = -4$ - Entered from **Fall** when $\dot{y} \leq -4$. The initial timestep in the **Terminal Velocity** state is actually $\dot{y} = -4 + \dot{y} - \lfloor \dot{y} \rfloor$ before being set to -4.

Bump(1,2,3) $\dot{y} := 0$ — Entered from a **Jump** or **Release** when Mario collides with something hard and solid from below; $\ddot{y}$ same as respective **Jump** or **Release**

SoftBump(1,2,3) $\dot{y} := 1 + \dot{y} - \lfloor \dot{y} \rfloor$ — Entered from a **Jump** or **Release** when Mario collides with something soft and solid from below; $\ddot{y}$ same as respective **Jump** or **Release**

Bounce(1,2,3) $\dot{y} := 4, \ddot{y} := a$ — Entered when Mario collides with an enemy from above; a is given by the respective **Jump**, **Release**, **Fall**, or **Terminal Velocity** state

Figure 2: The true HA for Mario's jump in *Super Mario Bros.* $:=$ represents the setting of a value on transition into the given mode, while $=$ represents a flow rate while within that mode.

frames in length, 63 seconds. The learned HAs are over-approximations of the true HA. Whereas the true HA has 3 separate jump modes based on the state of $\dot{x}$ at the time of transition, the learned HAs have only one such jump whose parameters are averages of the parameters of the true modes. Following from learning just one jump, CHARDA learns only a single falling mode. MDL does learn that releasing the **A** button while ascending leads to a different set of dynamics,

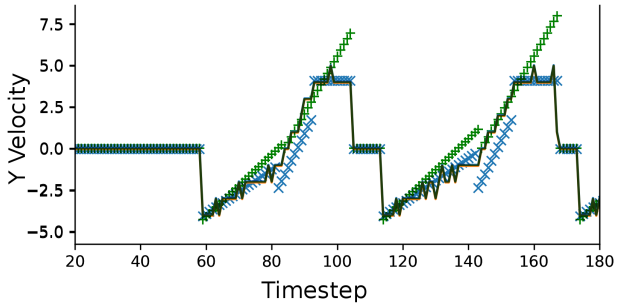

Figure 3: Modeled behavior using MDL criterion (Blue X) and BIC (Green +) vs true behavior (Black Line). MDL's largest error source is resetting to an specific value when the true behavior involves clamping to that value, whereas since BIC learns to transition at the 0 crossing it has a more accurate reset velocity. BIC does not learn the transition from **Falling** to **Terminal Velocity**. MDL has a Mean Absolute Error (MAE) of 0.522 while BIC has an MAE of 0.716.

but it considers this a change in gravity as opposed to a reset in velocity.

MDL produces the more faithful model of the true behavior, but is overzealous in its merging of the distinct jump mode chains into a single jump mode chain. As such, it only recovers 7 of the 22 modes; however, abstracting away the differences between the jump chains it learns 7 of 8 modes, only missing the distinction between hard bump and soft bump. A comparison of the modeled behaviors and the truth can be seen in figure 3.

6 Conclusion and Future Work

We have presented CHARDA, a novel combination of techniques (dynamic programming with a grounded penalty for data segmentation, causal relationship learning) that can recover hybrid automata from observations of a dynamical system. CHARDA outperforms an existing HA learning algorithm in data segmentation, and in a well-suited domain can find causal (not merely correlative) transition guards. We have also demonstrated CHARDA in a novel domain, videogames, that comes with an interesting set of challenges (short time durations, non-physical dynamics) and benefits (full access to all command inputs).

The use of a well-founded penalty criterion in conjunction with the dynamic programming approach is only one of many possible segmentation techniques, and it remains future work to test the general framework of *Segmentation + Guarded Transition* learning with other techniques. However, the biggest source of error in the learned HAs comes not from mistakes in segmentation, but rather from overzealous merging of modes. The learned parameters at segmentation in fact do describe modes in line with **Jump1** and **Jump3** (i.e. $\dot{y} = 4$ vs $\dot{y} = 5$), but these modes are merged together since it improves the overall learned model according to the criterion. It remains for future work to determine if there is a different principled way to learn these similar but distinct modes. It is also future work to incorporate techniques from other approaches, such as mode assignment via a Chinese Restaurant Process or the Forget-Me-Not Process, to pool modes at segmentation time instead of a post-segmentation merge process.

On Ground $\dot{y} = 0$ — Caused by Mario colliding with something solid from above

Jump $\dot{y} := [3.97, 4.10], \ddot{y} = [-0.140, -0.131]$ — Entered from **On Ground** when the **A button** is pressed

Release $\dot{y} := [2.10, 2.54], \ddot{y} = [-0.430, -0.384]$ — Entered from **Jump** when the **A button** is released

Fall $\dot{y} := 0, \ddot{y} = [-0.373, -0.359]$ — Entered from **Jump** or **Release** when the apex is reached

Bump $\dot{y} := [-1.85, -1.27], \ddot{y} = [-0.324, -0.238]$ — Entered from **Jump** when something solid is collided with from below

Bounce $\dot{y} := [3.51, 3.82], \ddot{y} = [-0.410, -0.378]$ — Entered from **Jump** when an enemy is collided with from above

Terminal Velocity $\dot{y} = [-4.15, -4.06]$ — Entered from **Jump** or **Fall**

(a) HA with MDL as the penalty.

On Ground $\dot{y} = 0$ — Caused by Mario colliding with something solid from above

Jump $\dot{y} := [4.19, 4.42], \ddot{y} = [-0.195, -0.181]$ — Entered from **On Ground** when the **A button** is pressed

Fall $\dot{y} := 0, \ddot{y} = [-0.356, -0.338]$ — Entered from **Jump** when the apex is reached

Bump $\dot{y} := [-2.37, -1.67], \ddot{y} = [-0.289, -0.188]$ — Entered from **Jump** when something solid is collided with from below

Bounce $\dot{y} := [3.52, 3.88], \ddot{y} = [-0.424, -0.391]$ — Entered from **Jump** when an enemy is collided with from above

Terminal Velocity $\dot{y} = [-4.16, -4.05]$ — Entered from **Jump** when the threshold of -4 is reached.

(b) HA with BIC used as the penalty

Figure 4: Learned Mario HAs. Parameters as 95% confidence intervals.

Beyond improving segmentation, there are also possible improvements to learning guarded transitions. Assuming we had perfect segmentation and mode assignment, we would still not be able to fully capture the guarded transitions of Mario given that our transitions do not have knowledge of Mario's horizontal velocity, nor are they able to learn transitions based on comparisons to arbitrary thresholds. In some domains, experimentation is possible: we might be able to control the dynamical system in question or to put it into situations where its behavior could be informative. We would like to explore this to improve the precision of our analysis, either by helping to split truly distinctive merged modes or by testing hypothesized guard conditions.

References

[Alur *et al.*, 1993] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*. Springer, 1993.

[Alur *et al.*, 1995] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 1995.

[Alur *et al.*, 2003] Rajeev Alur, Thao Dang, Joel Esposito, Yerang Hur, Franjo Ivancic, Vijay Kumar, P Mishra, GJ Pappas, and Oleg Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 2003.

[Baum and Petrie, 1966] Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 1966.

[Beal *et al.*, 2002] Matthew J Beal, Zoubin Ghahramani, and Carl Edward Rasmussen. The infinite hidden markov model. *Advances in neural information processing systems*, 2002.

[Bellman and Roth, 1969] Richard Bellman and Robert Roth. Curve fitting by segmented straight lines. *Journal of the American Statistical Association*, 1969.

[Dempster *et al.*, 1977] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1977.

[Fasterholdt *et al.*, 2016] Martin Fasterholdt, Martin Pichlmair, and Christoffer Holmgård. You say jump, i say how high? operationalising the game feel of jumping. 2016.

[Frehse, 2005] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *International workshop on hybrid systems: computation and control*. Springer, 2005.

[Henzinger and Kopke, 1999] Thomas A Henzinger and Peter W Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221(1):369–392, 1999.

[Henzinger *et al.*, 1995] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*. ACM, 1995.

[Ho *et al.*, 2016] Xavier Ho, Martin Tomitsch, and Tomasz Bednarz. Finding design influence within roguelike games. *Meaningful Play 2016 Conference Proceedings*, 2016.

[jdaster64, 2012] jdaster64. SMB physics spec. http://forums.mfgg.net/viewtopic.php?p=346301, 2012. Accessed: 2017-02-13.

[Jha *et al.*, 2007] Susmit Jha, Bryan A Brady, and Sanjit A Seshia. Symbolic reachability analysis of lazy linear hybrid automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2007.

[Ly and Lipson, 2012] Daniel L Ly and Hod Lipson. Learning symbolic representations of hybrid dynamical systems. *Journal of Machine Learning Research*, 13(Dec):3585–3618, 2012.

[Milan *et al.*, 2016] Kieran Milan, Joel Veness, James Kirkpatrick, Michael Bowling, Anna Koop, and Demis Hassabis. The forget-me-not process. In *Advances in Neural Information Processing Systems*, 2016.

[Murphy, 2016] Tom Murphy, VII. The glEnd() of Zelda. *SIGBOVIK*, April 2016.

[Niggemann *et al.*, 2012] Oliver Niggemann, Benno Stein, Asmir Vodencarevic, Alexander Maier, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. In *AAAI*, 2012.

[Perez-Liebana *et al.*, 2016] Diego Perez-Liebana, Spyridon Samothrakis, and Julian Togelius. General video game ai: Competition, challenges and opportunities. 2016.

[Platzer, 2008] André Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 2008.

[Santana *et al.*, 2015] Pedro Santana, Spencer Lane, Eric Timmons, Brian Williams, and Carlos Forster. Learning hybrid models with guarded transitions, 2015.

[Schwarz and others, 1978] Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 1978.

[Smith *et al.*, 2009] Adam M. Smith, Mark J. Nelson, and Michael Mateas. Prototyping games with BIPED. In *Proceedings of the Fifth AIIDE Conference*, 2009.

[Stine, 2004] Robert A Stine. Model selection using information theory and the mdl principle. *Sociological Methods & Research*, 2004.

[Summerville *et al.*, 2016] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontanón. The vglc: The video game level corpus. *arXiv preprint arXiv:1606.07487*, 2016.

[Summerville *et al.*, 2017] Adam Summerville, Morteza Behrooz, Michael Mateas, and Arnav Jhala. What does that ?-block do? learning latent causal affordances from mario play traces. *Proceedings of the first What's Next for AI in Games Workshop at AAAI 2017*, 2017.

[Zeileis *et al.*, 2008] Achim Zeileis, Torsten Hothorn, and Kurt Hornik. Model-based recursive partitioning. *Journal of Computational and Graphical Statistics*, 2008.