

# Relational Sequential Inference with Reliable Observations

**Alan Fern and Robert Givan**  
Electrical and Computer Engineering  
Purdue University

{afern, givan}@purdue.edu

November 26, 2003

## **Abstract**

We present a trainable sequential-inference technique for processes with large state and observation spaces and relational structure. Our method assumes “reliable observations”, i.e., that each process state persists long enough to be reliably inferred from the observations it generates. We introduce the idea of a “state-inference function” (from observation sequences to underlying hidden states) for representing knowledge about a process and develop an efficient sequential-inference algorithm, utilizing this function, that is correct for processes that generate reliable observations consistent with the state-inference function. We describe a logical representation for state-inference functions in relational domains and give a corresponding supervised learning algorithm. Empirical results, in relational video interpretation, show that the resulting trainable system provides significantly improved accuracy and speed relative to a variety of recent, hand-coded, non-trainable systems.

# 1 Introduction

We consider processes with hidden state that produce sequences of noisy observations. By watching the observations, our task is to infer the underlying state sequence of the process. We are interested in problems with enormous numbers of possible states and observations that are represented in relationally factored form (with sets of relational atoms such as  $\text{ON}(\text{block}_1, \text{block}_2)$ ). In such large problems, general-purpose modeling approaches that fail to make strong structural assumptions are typically intractable.

In place of more familiar independence assumptions (e.g. Markov modeling), our inference approach exploits an assumption that the process generates “reliable” observations. By this we mean that hidden states persist for multiple observations, and that the sequence of observations generated while remaining in a single hidden state reliably determines the state—i.e., no other hidden state is likely to generate that sequence of observations. The assumption of reliable observations appears to hold well in our video-interpretation application domain. Many video frames pass while the underlying interpretation remains fixed, and we are able to infer (using a “state inference function”) that underlying interpretation from the sequence of frames.

We introduce the idea of using a state-inference function to represent knowledge about a process. We provide an efficient sequential-inference algorithm that is correct, assuming reliable observations and a correct state-inference function. Our inference algorithm is not tied to a particular representation of states, observations, or state-inference function. Thus, we first describe the problem setup (Section 2) and inference method (Section 3) for arbitrary sets of observations and states. When these sets are relationally represented, our method provides general-purpose, relational, sequential inference, given reliable observations.

Although our inference technique is independent of the representation, representation is relevant because the required state-inference function is generally unavailable and must be learned automatically. To facilitate learning, the states and observations must be represented in some factored form, and below we describe a (familiar) general relational representation (in Section 4). We then describe a logic-based representation for relational state-inference functions and provide a corresponding supervised learning algorithm (in Section 5). Finally, we give promising experimental results for that relational representation (in Section 6) in our (noisy) video-interpretation application.

We note that probabilistic modeling and inference is a popular approach

to dealing with noisy problems such as ours; however, here we do not use probabilistic methods. Related probabilistic modeling work is applicable only with unclear extensions (see Section 7). Here, we present a simple, logical-constraint-based approach with surprising robustness to noise.

## 2 Problem Setup

A sequential process is a triple  $(\mathcal{O}, \mathcal{S}, \mathcal{P})$ , where the *observation space*  $\mathcal{O}$  and *state space*  $\mathcal{S}$  are arbitrary disjoint sets that contain all possible observations and states, respectively.  $\mathcal{P}$  is a probability distribution over  $(\mathcal{O} \times \mathcal{S})^*$ , i.e., the space of finite sequences constructed from members of  $\mathcal{O} \times \mathcal{S}$ . We can extract from each such sequence a pair of an *observation sequence* (*o-sequence*) and a *state sequence* (*s-sequence*), and we often treat  $\mathcal{P}$  as assigning probabilities to such pairs. We say state  $s$  *generates* o-sequence  $o_1, \dots, o_k$  in state-observation sequence  $P$ , if  $(s, o_1), \dots, (s, o_k)$  is a subsequence of  $P$ , and also that  $o_1, \dots, o_k$  is *generated by consecutive states*  $s_1$  and  $s_2$  in sequence  $P$  if, for some  $t$ , the sequence  $(s_1, o_1), \dots, (s_1, o_t), (s_2, o_{t+1}), \dots, (s_2, o_k)$  is a subsequence of  $P$ .

Sequential inference is the problem of mapping an o-sequence to the most likely (hidden) s-sequence. In our large structured problems, the o-sequence typically determines the s-sequence, so we simplify this goal to finding the single possible s-sequence. Here, we do not require a model of  $\mathcal{P}$ , rather we use supervised learning, needing only to sample a training set of sequences of state/observation pairs from  $\mathcal{P}$ .

Our method leverages an assumption that hidden states persist for many observation steps. We also assume that the utility of an inferred state sequence primarily derives from the sequence of *distinct* states (with consecutive repetitions removed), rather than whether it also identifies the exact state-transition points. This assumption holds in our video-interpretation domain, where the exact location of transition points is often ambiguous (as judged by a human observer) and unimportant. For example, in Figure 1, it will typically be unimportant exactly which frame is considered to be the transition. We therefore consider a state-sequence label to be accurate if it agrees on the sequence of distinct states. Let  $\text{COMPRESS}(S)$  denote the sequence that is derived from  $S$  by removing its consecutive repetitions. For example,  $\text{COMPRESS}(a; a; a; b; b; a; c; c) = a; b; a; c$ . We consider  $\text{COMPRESS}(S)$  to be an accurate label for the sequence  $S$ .

**Example 1.** *In our experimental video-interpretation domain, the process corresponds to a hand playing with a set of blocks. Figure 1 shows key video frames from a sequence where the hand picks up a red block from a green*

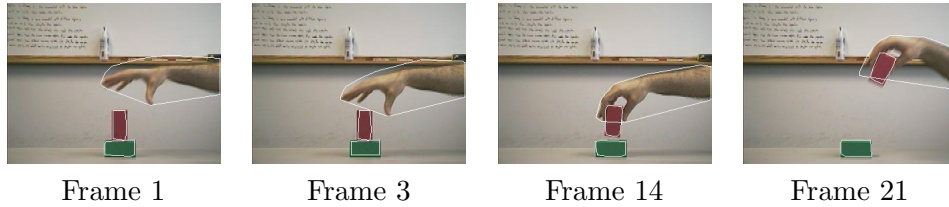


Figure 1: Key frames in a video segment showing a hand picking up a red block from a green block. The object tracker’s output is shown by the polygons. The video segment has two distinct force-dynamic states given by:  $\{\text{GROUNDED}(\text{HAND}), \text{GROUNDED}(\text{GREEN}), \text{CONTACTS}(\text{GREEN}, \text{RED})\}$  (frames 1 and 3) and  $\{\text{GROUNDED}(\text{HAND}), \text{GROUNDED}(\text{GREEN}), \text{ATTACHED}(\text{HAND}, \text{RED})\}$  (frames 14 and 20). The transition occurs between frames 3 and 14. See Example 3 regarding the predicates GROUNDED, CONTACTS, and ATTACHED.

*block. Our goal is to observe the video and infer the underlying force-dynamic states, describing the support relations among the objects. The figure caption describes the single state transition. States are represented as sets of force-dynamic facts, such as ATTACHED(HAND, RED). Observations are represented as sets of low-level numeric facts, such as DISTANCE(GREEN, RED, 3), that are easily derived from an object tracker’s noisy output (shown by the polygons in the figure). The state and observation sets are large, with roughly  $2^{35}$  states for a three-block scene with one hand.*

### 3 Sequential Inference with Reliable Observations

A simple approach to the sequential inference problem is to assume that each observation uniquely determines the state generating it. We could then use the training data to learn a (possibly non-trivial) observation-state mapping that can reconstruct a hidden s-sequence from a given o-sequence. However, this assumption is quite strong, and, empirically, does not hold in our video-interpretation domain, due to noise and natural ambiguity near force-dynamic transitions. Instead, we make a much weaker assumption sufficient for robust performance: we assume reliable observations, as defined below.

**Deinition 1 (Defining Sequence).** *For process  $\mathcal{P}$ , an o-sequence  $O$  is a defining sequence for state  $s$  if: (1)  $s$  generates  $O$  in some sequence drawn from  $\mathcal{P}$ , and (2) no other state generates  $O$  in any sequence drawn from  $\mathcal{P}$ .*

**Deinition 2 (Reliable Observations).** *Process  $\mathcal{P}$  has reliable observations with redundancy  $r$  if, in each sequence drawn from  $\mathcal{P}$ , each state  $s$  persists long enough to generate an observation sequence that can be divided into at least  $r$  consecutive defining sequences for  $s$ .*

Let  $\mathcal{RO}_r$  denote the set of processes having reliable observations with

redundancy at least  $r$ . Later we show that our inference technique is correct for processes in  $\mathcal{RO}_2$ . In practice, processes with rare violations of this assumption also admit our techniques. The reliable observations assumption is intuitively nearly met by our video-interpretation domain, where semantic scene properties persist for enough video frames to be inferred.

Reliable observations (with redundancy at least 1) imply that the maximal-length observation sequence generated by a state, any time that state occurs, cannot be generated by any other state. Thus, under reliable observations, there exists a mapping from “long enough” o-sequences generated by single states to the unique states likely to generate them. A *state-inference function*  $\sigma$  is, then, a mapping from  $\mathcal{O}^*$  to  $\mathcal{S} \cup \{\perp\}$ . We say that  $\sigma$  is *correct* if, for each  $O \in \mathcal{O}^*$ ,  $\sigma(O)$  is a state capable of generating  $O$  under  $\mathcal{P}$ , and  $\sigma(O) = \perp$  exactly when no single state is capable of generating  $O$  under  $\mathcal{P}$ . Our inference algorithm assumes we have a nearly correct state-inference function (particularly, correct for “long enough” observation sequences), which we provide for our application by machine learning (see Section 5). A correct  $\sigma$  is *monotone*, returning  $\perp$  for a sequence whenever it returns  $\perp$  for any subsequence.

Given an o-sequence  $O$ , if we are somehow told which subsequences of  $O$  were generated by single states, then we can apply a correct state-inference function to each such subsequence to correctly infer the underlying s-sequence. However, in practice, we are not given this information. Nevertheless, under reliable observations we are able to infer this subsequence information with sufficient accuracy by detecting state transitions. To see how, note that, by definition, no single state can generate an o-sequence that contains defining sequences for two distinct states. This implies that a correct state-inference function “detects transitions” by returning  $\perp$  on a “long enough” o-sequence generated by consecutive states—in particular, long enough to include a defining sequence from each state. This property leads to a greedy algorithm for constructing a compressed s-sequence.

The *forward-greedy-merge* (FGM) algorithm, Figure 2, applies a state-inference function  $\sigma$  to increasing prefixes of o-sequence  $O$ , until locating the shortest prefix  $o_1, \dots, o_k$  for which  $\sigma(o_1, \dots, o_k)$  is  $\perp$ . For correct  $\sigma$ , and  $O$  drawn from  $\mathcal{P}$ ,  $k \geq 2$ . If  $k = 1$ , then  $o_1$  has been incorrectly labeled “impossible”, and the algorithm returns “fail”. Otherwise, FGM adds  $\sigma(o_1, \dots, o_{k-1})$  to the inferred s-sequence and recursively processes the remaining suffix of  $O$ . The number of calls to  $\sigma$  is linear in  $|O|$ . In our application,  $\sigma$  runs in poly-time in its input size, and thus the overall inference process is poly-time.

In practice, we handle the case that FGM returns “fail” due to an incorrect  $\sigma$  by pre-processing the observation sequence to remove all single observations identified incorrectly as “impossible” by  $\sigma$ . We give empirical results with and without this *sequence-cleaning*, showing that very few observations are removed, but that such removal improves performance.

**Example 2.** *The FGM algorithm is inspired by imagining a viewer watching a (noisy) video very slowly, analyzing each frame consciously. Any given frame may not provide enough information to reconstruct the scene semantics (the hidden state). Each new frame provides more information about the current scene, which the viewer adds to the saved partial knowledge. Only when something contradictory to the currently inferred scene is noticed does the viewer assume that a state transition has occurred. At that point, whatever has been inferred about the previous “current state” is taken to completely describe that state. Our technique does not (yet) reason about connections between distinct, adjacent states, beyond detecting transitions—such reasoning may be needed in some domains but, empirically, not in ours.*

FGM is not guaranteed to be correct for all processes in  $\mathcal{RO}_1$ . However, for processes in  $\mathcal{RO}_2$ , FGM can detect state transitions accurately enough to correctly infer the underlying compressed state sequence.<sup>1</sup>

**Proposition 1.** *Let process  $\mathcal{P}$  be in  $\mathcal{RO}_2$  and  $\sigma$  be a correct state-inference function for  $\mathcal{P}$ . For any state and observation sequences  $S$  and  $O$  drawn together from  $\mathcal{P}$ ,  $FGM(O, \sigma)$  returns  $COMPRESS(S)$ .*

**Proof:** (Sketch) Let  $COMPRESS(S) = s_1, s_2, \dots, s_n$  and  $O_i$  denote the maximal o-sequence in  $O$  that was generated by  $s_i$  (assume w.l.o.g. that no two  $s_i$  are the same state). Note that  $O = O_1; O_2; \dots; O_n$ , where “;” indicates concatenation. Since  $\mathcal{P}$  has a redundancy of at least two, each  $O_i$  can be written  $O_i = O'_i; O''_i$ , where  $O'_i$  and  $O''_i$  are both defining sequences for state

```

FGM( $O, \sigma$ )
  Input: Obs. seq.  $O = (o_1, \dots, o_n)$ ,
         State-inference function  $\sigma$ 
  Output: Compressed state seq. or “fail”
  if  $O = \text{NULL}$  then return NULL
  if  $\sigma(o_1) = \perp$  then return “fail”
   $k \leftarrow 2$ 
  while  $(\sigma(o_1, \dots, o_k) \neq \perp) \ \&\& \ (k \leq n)$ 
     $k \leftarrow k + 1$ 
   $S' \leftarrow FGM((o_k, \dots, o_n), \sigma)$ 
  if  $S' = \text{“fail”}$ 
    then return “fail”
  else return  $\sigma(o_1, \dots, o_{k-1}) \mid S'$ 

```

Figure 2. Pseudo-code for forward-greedy-merge. “ $x|y$ ” is the list  $y$  with  $x$  at the front.

<sup>1</sup>While our algorithm outputs only  $COMPRESS(S)$ , FGM does infer state transition points ( $k$  after the **while** loop) that can be used to construct an estimate of  $S$ , if desired.

ATTACHED( $x, y$ )	$x$ supports $y$ by attachment	GROUNDING( $x$ )	support of $x$ is unknown
CONTACTS( $x, y$ )	$x$ supports $y$ by contact		

---

DIRECTION( $x, d$ )	$x$ moving in direction $d$	SPEED( $x, s$ )	$x$ 's speed is $s$
ELEVATION( $x, e$ )	$x$ 's elevation is $e$	MORPH( $x, c$ )	$x$ 's shape-change factor
DISTANCE( $x, y, d$ )	distance between $x$ and $y$	$\Delta$ DIST( $x, y, dd$ )	change in distance
COMPASS( $x, y, c$ )	compass direction of $y$ to $x$	ANGLE( $x, y, a$ )	angle between $x$ and $y$

Table 1: Force-dynamic state predicates  $R_s$  (top) and observation predicates  $R_o$  (bottom) for our application.

$s_i$ . To complete the proof, prove by induction on  $k$  that, for  $1 \leq k \leq n$ ,  $\text{FGM}(O_1; \dots; O_k, \sigma) = s_1, s_2, \dots, s_k$  and  $s_k$  was “produced” by applying  $\sigma$  (in the last line of FGM in Figure 2) to a suffix of  $O_k$  that includes  $O_k''$ . The key proof step notes that applying  $\sigma$  to such a suffix, with  $O_{k+1}'$  concatenated on the end, yields  $\perp$ . Thus, state transitions will be detected.  $\square$

Without reliable observations or a correct state-inference function, FGM does not guarantee correct inference. However, FGM does solve an intuitively appealing optimization problem, finding a state sequence allowed by  $\sigma$  with the fewest possible state transitions. More formally, a *partition* of an observation sequence  $O$  is a sequence  $(Q_1, \dots, Q_k)$  of non-empty subsequences of  $O$  such that  $Q_1; \dots; Q_k = O$ . We say that  $\sigma$  *allows* a state sequence if that sequence is  $(\sigma(Q_1), \dots, \sigma(Q_k))$  for some partition. We prefer fewer transitions both because we have an inertial bias and because longer o-sequences yield more reliable state inference (we assumed  $\sigma$  is correct for “long” o-sequences).

**Proposition 2.** *When  $\sigma$  is monotone,  $\text{FGM}(O, \sigma)$  is a minimal-length state sequence allowed by  $\sigma$  for  $O$ , or there is no allowed state sequence.*

**Proof:** (Sketch) Prove, by induction on  $|O|$ , that for any observation sequence  $O'$ , with suffix  $O$ , any state sequence allowed by  $\sigma$  for  $O'$  is at least as long as  $\text{FGM}(O, \sigma)$ .  $\square$

## 4 Relational State and Observation Spaces

We say that a process  $(\mathcal{O}, \mathcal{S}, \mathcal{P})$  is *relational* when  $\mathcal{O}$  and  $\mathcal{S}$  are given by specifying a domain set of objects  $D$ , a set of observation predicates  $R_o$ , and a set of state predicates  $R_s$ . An *observation fact* (*state fact*) is a predicate symbol in  $R_o$  ( $R_s$ ) applied to the appropriate number of objects from  $D$ . For example, a state fact might be  $\text{ON}(a, b)$  where “on” is in  $R_s$  and  $a$  and  $b$  are objects in  $D$ . Observations are taken to be finite sets of observation facts and  $\mathcal{O}$  contains all such sets, likewise the states are taken to be finite sets of state facts with  $\mathcal{S}$  containing all such sets.

**Example 3.** *Our video-interpretation application involves inferring the sequence of force-dynamic states in videos of a hand playing with blocks.*

The domain of objects  $D$ , contains all hands and blocks that may eventually enter the visual field, along with the real numbers. To describe the state and observation spaces, there are three force-dynamic state predicates<sup>2</sup> and eight observation predicates, shown in Table 1. The movie in Figure 1 contains two distinct force-dynamic states given by the state-fact sets shown in the caption.<sup>3</sup> The object tracker places convex polygons around each object in the visual field, and the observations are low-level numeric features of these polygons and polygon pairs. For each video frame, an observation is the set of observation facts calculated by computing the numeric argument of each predicate for all objects and object pairs.

## 5 Learning a Relational State Inference Function

For relational processes, a state-inference function maps relational o-sequences to relational states (or  $\perp$ ). Learning such a function corresponds to the difficult problem of multiple-predicate learning [2] from the area of inductive logic programming (ILP) [7]. In order to achieve robust and “example efficient” learning, below we introduce a representation for relational state-inference functions, based on DATALOG [14], that leverages problem structure found in our application domain and others like it. We then use an off-the-shelf ILP system, CLAUDIEN [1], to learn the required DATALOG program.

**Representation.** A *DATALOG* program consists of a set of “if <body> then <head>” rules, built up from logical atoms over available predicates. Here, the available predicates are the observation and state predicates along with the comparison predicates  $\leq$  and  $\neq$ . A *logical atom* is a predicate applied to the appropriate number of variables and/or numeric constants. The rule <body> is a conjunction of logical atoms, and the <head> is either  $\perp$  or a logical atom whose variables appear in the body.

We will consider two types of rules for defining state-inference functions. First, *o-rules* allow only observation predicates in the body and state predicates in the head, and can derive state facts from observations. For example, if  $\text{DISTANCE}(x, y, d) \wedge (d \leq 5) \wedge \text{SPEED}(y, s) \wedge (6 \leq s)$  then  $\text{ATTACHED}(x, y)$  is an o-rule. Second, *s-constraints* are rules that do not involve observation predicates (the head may involve  $\perp$ ). These rules place logical con-

---

<sup>2</sup>Our predicates vary somewhat from Siskind [12], e.g., “attached” implies support.

<sup>3</sup>Notice that  $\text{GROUNDED}(\text{GREEN})$  is in both states, even though it seems to be supported by the table. This is because the object tracker does not recognize the table as an object, and thus the table is an “unknown” source of support for the the green block.



straints on sets of state facts and can detect sets of facts that do not belong to any state (i.e. sets that violate some constraint). For example, (if  $\text{ATTACHED}(x, y) \wedge \text{CONTACTS}(x, y)$  then  $\perp$ ) is an s-constraint that says  $x$  cannot support  $y$  by both contact and attachment.<sup>4</sup>

Any way of replacing the variables in a rule (consistently) with objects and/or numbers gives an *instance* of that rule. *Applying* a rule to a premise set of state and observation facts produces new assertions, in the usual way: for each instance of the rule with the instance body true, relative to the premise set, the instance head is produced as an assertion. For example, if we are given the observation  $\{\text{DISTANCE}(\text{GREEN}, \text{RED}, 3), \dots, \text{SPEED}(\text{RED}, 10)\}$ , then the above o-rule will assert  $\text{ATTACHED}(\text{GREEN}, \text{RED})$ . Given a rule set  $R$  and premise set  $Q$ , the *one step consequence operator*  $\tau_R(Q)$  computes the union of all rule assertions for  $Q$ . We inductively define  $\tau_R^i(Q) = \tau_R(\tau_R^{i-1}(Q))$ , where  $\tau_R^0(Q) = Q$ , and let  $\tau_R^*(Q)$  denote the union over all  $i$  of  $\tau_R^i(Q)$ .

A DATALOG program  $\Sigma = \Sigma_o \cup \Sigma_s$ , with o-rules  $\Sigma_o$  and s-constraints  $\Sigma_s$ , defines a state inference function  $\sigma$  as follows. The *result set*  $\Sigma(O)$  for an observation sequence  $O = (o_1, \dots, o_n)$  is calculated by computing the o-rule assertions for each  $o_i$  and then iteratively applying the s-constraints to the union of the assertions. Formally we have  $\Sigma(O) = \tau_{\Sigma_s}^*(\bigcup_i \tau_{\Sigma_o}(o_i))$ .<sup>5</sup> Finally, we define  $\sigma(O)$  to be  $\perp$  if  $\perp \in \Sigma(O)$ , and to be  $\Sigma(O)$ , otherwise.

This DATALOG representation for state-inference functions is motivated by two observations about our application. First, although we are unable to learn o-rules that accurately map *single* observations to *all* of the underlying state facts (due to noise and ambiguity), we are able to learn nearly sound o-rules (i.e. rules that rarely produce false assertions) that assert *some* of the underlying state facts for single observations. Intuitively, the rules only assert the “most obviously true” state facts for a given observation. Typically, for states in our application, each state fact is “obviously true” in at least one of the observations a state generates. Thus, unioning o-rule assertions across observations (as done above) typically yields exactly the true state facts.

The second observation about our application domain is that the union of facts from distinct consecutive states do not correspond to any actual state, i.e. the state facts are inconsistent. Given s-constraints to detect such inconsistent fact sets, the above computation can detect when an input

---

<sup>4</sup>We could also consider rules with both observation and state predicates in the body, resulting in more expensive learning. Such rules were not needed for our application.

<sup>5</sup>The iteration of  $\Sigma_s$  is required since s-constraints can be recursive. However, the iteration will always terminate since there are only a finite number of possible state facts.

observation sequence was (most likely) not generated by a single state.

**Example 4.** *As an illustration of when  $\sigma$  will return  $\perp$ , assume that  $\Sigma$  includes the rule (if  $\text{ATTACHED}(x, y) \wedge \text{CONTACTS}(z, y)$  then  $\perp$ ), representing the constraint that no object is supported via both contact and attachment. Let  $O$  be the o-sequence from the video in Figure 1, which is generated by two distinct force-dynamic states. We expect that, for some frame during the first force-dynamic state (e.g., frame 1 or frame 3), the rules will be able to assert  $\text{CONTACTS}(\text{GREEN}, \text{RED})$ , and that, for some frame in the second state (e.g., frame 20), the rules will assert  $\text{ATTACHED}(\text{HAND}, \text{RED})$ . Given these assertions, the above rule will assert  $\perp$  and thus  $\sigma(O) = \perp$ , which signals that  $O$  did not arise from a single state according to  $\sigma$ .*

**Learning.** We use CLAUDIEN to search for the most general o-rules and s-constraints that agree with all of the state-observation pairs in the training set.<sup>6</sup> Here, a rule  $r_1$  is *more general* than  $r_2$  if any assertion produced by  $r_2$  can also be produced by  $r_1$ , for all premise sets. For our domain, CLAUDIEN typically produces a large, redundant ruleset (with 300-400 rules).

Motivated by Occam’s Razor (i.e. reducing overfitting) and the fact that small rulesets are cheaper to apply, we prune to find a smaller, but “practically equivalent”, subset of the CLAUDIEN-generated o-rules.<sup>7</sup> Let  $\Delta$  be a set of o-sequences (typically the training data o-sequences). We consider two rulesets  $\Sigma$  and  $\Sigma'$  to be *FGM-equivalent on  $\Delta$*  (written  $\Sigma =_{\Delta} \Sigma'$ ) if for any  $O$  in  $\Delta$ , we have  $\text{FGM}(O, \sigma) = \text{FGM}(O, \sigma')$ , where  $\sigma$  and  $\sigma'$  are the state-inference functions defined by  $\Sigma$  and  $\Sigma'$  respectively.

```

Prune-Ruleset( $\Sigma_s, \Sigma_o, \Delta$ )
 $\Sigma' \leftarrow \Sigma_s$ 
while  $\Sigma_s \cup \Sigma_o \neq_{\Delta} \Sigma'$ 
     $r \leftarrow \arg \max_{r \in \Sigma_o} \mathcal{C}(\Sigma' \cup \{r\}, \Delta)$ 
     $\Sigma' \leftarrow \Sigma \cup \{r\}$ 
return  $\Sigma'$ 

```

Figure 3. Pruning Routine.

Given the CLAUDIEN ruleset  $\Sigma = \Sigma_o \cup \Sigma_s$ , where  $\Sigma_o$  and  $\Sigma_s$  are the o-rules and s-constraints, we use a heuristic method to find a smaller  $\Sigma'$  that is FGM-equivalent. We define the *coverage*  $\mathcal{C}(\Sigma, \Delta)$  of  $\Sigma$  to be the sum, over all *individual observations*  $o$  in  $\Delta$ , of  $|\Sigma(o)|$ . This measure rewards rule sets that assert true state facts more frequently. We start with  $\Sigma' = \Sigma_s$  and add o-rules greedily, according to coverage, until FGM equivalence is achieved. We show pseudo-code for our pruning method in Figure 3. In our application, pruning reduces error by over 50%, indicating significant pre-pruning overfitting.

<sup>6</sup>A rule agrees with a state-observation pair when the rule produces no new assertions on the premise set given by the union of the state and observation fact sets.

<sup>7</sup>One could also consider pruning s-constraints, though unnecessary for our application.

## 6 Experimental Results

We evaluate our techniques by applying them to force-dynamic state inference. The LEONARD system [11] uses inferred force-dynamic states to recognize visual events from video-camera input—a simple example of an event is “a hand picking up a block”, as depicted in Figure 1. LEONARD is distinctive in its use of force-dynamic properties for event recognition, which Siskind argues is more semantically grounded (and thus more generally accurate) for many event types than motion profile analysis. LEONARD uses a hand-crafted force-dynamic inference technique (details in [12]), based on kinematic physics that was shown to correctly infer force-dynamic relations for approximately 80% of the  $\sim 10,000$  video frames in a test corpus. A large part of the inaccuracy stems from noise in the object tracker’s output, including, for example, variable strength “jitter” and more serious errors such as “object teleportation”. Our original motivation for this work was to develop a robust trainable system to replace and improve the accuracy and speed of LEONARD’s reconstruction of force-dynamic state. We note that, in improving these features, we have dropped the kinematic-physics approach to the problem (among other things), which may have ramifications yet to be explored in either system by evaluation on a much wider variety of data. **Procedure**<sup>8</sup>. We use the same 210 videos (and the same object tracker output) that were used to demonstrate LEONARD [12]. The videos depict a hand playing with up to 3 blocks and are divided into 7 different event types (30 movies each), which vary in complexity from a simple pick-up to assembling towers. From the tracker output of each video, we can construct the corresponding relational observation sequence as described in Example 3.

We hand-labeled 3 randomly selected videos from each event type with the human-judged force-dynamic state, yielding 21 training videos in total.<sup>9</sup> We labeled the other 189 videos with their compressed s-sequence only (the output of COMPRESS), as that is the labeling our algorithm produces. This compressed label, in fact, depends only on the event type.

We drew three training sets of 7, 14, and 21 videos from the training instances, drawing equally from each event type in each set, and learn state-inference functions  $\sigma_7$ ,  $\sigma_{14}$ , and  $\sigma_{21}$ , respectively. For each state-inference function and each test-video observation sequence, we inferred a force-dynamic state sequence using the FGM inference algorithm, both with and without

---

<sup>8</sup>All data along with detailed descriptions are available at the first author’s web site.

<sup>9</sup>Learning appears insensitive to variation in human labeling at ambiguous transitions.

the pre-processing sequence cleaning described in Section 3.

We compare our results with LEONARD. We note, however, that the goals of the LEONARD project and our work are quite different. LEONARD is an attempt to create a general, force-dynamic interpretation system, whereas our approach represents a trainable sequential inference technique that can be tuned to the class of videos exhibited by the training data. This comparison is analogous to work showing that learned domain-specific language parsers [13] outperform general-purpose language parsers within the trained domain.

We hand-designed programs for force-dynamic inference aimed at the class of movies in our corpus. HACK<sub>1</sub> was designed after examining only the size-14 training videos. HACK<sub>2</sub> was an improvement found by examining the errors HACK<sub>1</sub> made *on the test data*—a form of cheating.

**Whole-Movie Performance.** The second column of Table 2 shows the percentage of test videos labeled incorrectly. The first three rows are for FGM with the learned state-inference functions, both with and without sequence cleaning (the latter in parentheses). The final three rows show LEONARD and our hand-constructed programs. We see that, as more training data is included, the performance of FGM improves. With only a relatively small set of training data, FGM achieves a 3% error rate when sequence cleaning is used. Sequence cleaning significantly improves the FGM performance, though less than 0.5% of the observations were removed. Comparing to our hand-coded systems, FGM always outperforms HACK<sub>1</sub> and is comparable to HACK<sub>2</sub> for the larger training sets. So, our system is able to learn an inference system that is on par with a significant, even cheating, attempt to hand-code a solution.

**Per-Frame Performance.** The poor performance of LEONARD relative to whole-movie error does not properly reflect its ability. Although LEONARD rarely computes the exact true compressed state sequence, it does correctly label most individual observations with the correct force-dynamic state. The evaluation measure used in [12] considered the inferred state sequence as a multi-set, and then calculated the percentage of the multi-set members that did not appear in the correct state labeling (so state order does not affect the error). The first column of Table 2 shows this error measure for LEONARD and FGM. Under this measure, LEONARD labels over 80% of the frames

Table 2: Test error. Parentheses indicate no sequence cleaning.

	Percent Error	
	Frame	Video
$\sigma_7$	0.8	15 (23)
$\sigma_{14}$	0.1	6 (16)
$\sigma_{21}$	5e-4	3 (13)
LEONARD	16.4	100
HACK <sub>1</sub>	—	33
HACK <sub>2</sub>	—	9

correctly. FGM, however, significantly outperforms LEONARD.

We also compare inference time on the 210 videos for each method, all implemented in Scheme and running on the same machine. Frame rates were 1 per second for LEONARD, 3.8 per second for FGM (with  $\sigma_7$ ,  $\sigma_{14}$ , or  $\sigma_{21}$ ), and 5.3 per second for either HACK<sub>1</sub> or HACK<sub>2</sub>. So, FGM is about 4 times faster than LEONARD, but 28% slower than our hand-constructed domain-specific programs. Importantly, 90% of FGM’s runtime was spent computing the observation predicates from the tracker output. FGM runs at frame rate (30 frames/second) when given the observation predicates. We believe these predicates can also be computed at frame rate with a C implementation.

## 7 Related Work

Sliding-window techniques [4] label each observation using a fixed-size local window of observations and possibly previous classifications. These techniques leverage a stronger assumption than reliable observations due to the fixed window size. Finding a good state-inference function is problematic here because of the ambiguity at transitions—FGM can be viewed as varying the window size to avoid the ambiguities at transition points.

Other work, e.g. [6, 8], uses observation-subsequence classifiers to construct optimization problems. Each classification assigns a measure of “good fit” (with the classified subsequence) to each state, and then an optimization problem is solved to select a state sequence. These methods have assumed a small explicit state space, and generalization to our problem is unclear.

Probabilistic modeling is a widely preferred approach to achieving noise robustness, but is not straightforward to apply to our problem. Most techniques, such as HMMs [9] and conditional random fields [5], have traditionally assumed small “explicit” state spaces, and extensions such as dynamic Bayesian networks [3] require a fixed number of state variables. In our problem, the number of state factors can vary based on the number of objects.

However, it is not our intention to argue against the use of probabilistic models for problems such as ours. In fact, the recently proposed dynamic probabilistic relational model [10] is a candidate, as it explicitly factors states in terms of objects. Use of these models for our problem would require development of corresponding learning methods. In addition, exact inference in such models is typically intractable, and various approximate inference techniques would need to be considered. Instead, we give a simple logic-based approach, to both the learning and inference problems, that achieves good robustness to noise without using probabilistic modeling.

## References

- [1] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
- [2] L. De Raedt, N. Lavrac, and S. Dzeroski. Multiple predicate learning. In *IJCAI*, 1993.
- [3] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
- [4] Thomas G. Dietterich. Machine learning for sequential data: A review. In *Proceedings of the Fourth International Workshop on Statistical Techniques in Pattern Recognition*, 2002.
- [5] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.
- [6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [7] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
- [8] V. Punyakanok and D. Roth. The use of classifiers in sequential inference. In *NIPS*, 2000.
- [9] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [10] S. Sanghai, P. Domingos, and D. Weld. Dynamic probabilistic relational models. In *IJCAI'03*, 2003.
- [11] J. Siskind. Grounding lexical semantics of verbs in visual perception using force dynamics and event logic. *JAIR*, 15:31–90, 2001.
- [12] J. Siskind. Reconstructing force-dynamic models from video sequences. *AIJ*, to appear.

- [13] Lappoon R. Tang and Raymond J. Mooney. Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In *Joint Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 2000.
- [14] J. Ullman. *Principles of Database and Knowledge-Base Systems*. CS Press, 1988.