Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

ABSTRACTION MECHANISMS IN DISCRETE-EVENT INDUCTIVE MODELING

Hessam S. Sarjoughian

AI & Simulation Research Group
Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721-0104, U.S.A.
hessam@ece.arizona.edu
http://www-ais.ece.arizona.edu/~hessam

Bernard P. Zeigler

AI & Simulation Research Group
Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721-0104, U.S.A.
zeigler@ece.arizona.edu
http://www-ais.ece.arizona.edu/~zeigler

ABSTRACT

The power of abstraction lies in its ability to deal with "lack" of knowledge. In this regard, success in modeling and simulation rests on discovering useful abstractions that can support objectives of modeling. In our treatment, we refer to "data abstraction" as opposed to "structure simplification" since we consider a system's behavior rather than its structure. A system's behavior can be represented as time-varying input/output segments. Given the behavior of a causal, time-invariant system, we define some basic *abstraction mechanisms* to support inductive modeling. The basis for these abstraction mechanisms are a set of *general assumptions* which allow *consistent* abstraction of IO segments. Then, given these assumptions and non-monotonic reasoning paradigm, capable of handling them, we try to tackle the fundamental problem of insufficient knowledge in the realm of inductive modeling. In this way, by making useful abstractions, we can predict a system's unobserved behavior according to a well-defined framework of discrete-event inductive modeling.

1 INTRODUCTION

Vast amounts of observed data from various systems are continually collected with the expectation that they will assist us in understanding their underlying structure and behavior. An extensive body of research has been devoted to finding ways to predict a system's future behavior based on its previously observed behavior — Biermann and Feldman (1972), Zeigler (1976), Klir (1985), Michalski, Carbonell, and Mitchell (1986), Grossberg (1988), Cellier (1991), Omlin, Thornber, and Giles (1996) are some efforts. Here we do not discuss abstraction in deductive or abductive modeling paradigms. Hobbs and Moore (1985), Weld (1992), Kuipers (1994), and Fishwick (1995), among others, discuss abstraction in these settings.

In Sarjoughian (1995a) a Discrete-event Inductive Reasoner (DIR) has been developed based on the concepts from systems theory, Wymore (1993), and non-monotonic reasoning, McCarthy (1990), Davis (1990), Ginsberg (1993). While the former provides us simple means to formulate a well-defined structure representing a system's observed behavior, the latter supports the more powerful means necessary to reason with aforementioned structures in the absence of complete knowledge. Supposing that a causal, dynamic, time-invariant system's finite observed behavior can be partitioned into IO segments, we identify a well-defined set of assumptions which support prediction of the unobserved behavior of a system. An example would be when we have an unobserved input segment for which we would like to find its output segment. The problem is trivial if the input segment and its output segment are in the repository of observed IO segments. However, if no such input segment is available, it becomes essential to abstract certain features of the candidate input segment so that it is "equivalent" to one of the observed input segments for which we know its corresponding output segment.

To employ abstractions, Giunchiglia and Walsh (1992), and homomorphism, Wymore (1993), it becomes necessary to make assumptions (and thus non-monotonic reasoning) based on which unobserved IO segments can be predicted.

In our discussion, we do not argue the DIR's foundations nor its methodology. Moreover, we exclude the presentation of the overall architecture and the implementation of the DIR. Instead, we show the underlying mechanisms of a part which corresponds to defining abstraction mechanisms and how they would facilitate well-defined prediction of IO segments. Here, it suffices to say that the discrete-event inductive reasoner is comprised of a repository of observed IO segments, an inference engine and a logic-based truth maintenance system. Forbus and de Kleer (1993) describe and discuss thoroughly logic-based truth maintenance systems. For more discussion of the above and details of what follows refer to Sarjoughian (1995a), Sarjoughian and Zeigler (1995b).

2 ITERATIVE IO FUNCTION OBSERVATION STRUCTURE

Given input generator segments (Ω_G) and output generator segments (Ψ_G), the associated input/output generator segment set $IOspace_G$ is defined as:

$$\begin{aligned} IOspace_G = \; & \{(\omega_G,\psi_G) \mid (\omega_G,\psi_G) \in IOspace, \\ & \omega_G \in \Omega_G, \psi_G \in \Psi_G, \\ & IOspace_G \subseteq IOspace, \\ & dom(\omega_G) = dom(\psi_G)\} \end{aligned}$$

where $IOspace$ is a collection of IO trajectories (ω is an input trajectory and ψ is an output trajectory):

$$\begin{aligned} IOspace = \{ & (\omega,\psi) \mid (\omega,\psi) \in (X,T) \times (Y,T), \\ & dom(\omega) = dom(\psi).\} \end{aligned}$$

Then, given an IOFO specification, Zeigler (1976), an *iterative IOFO specification* for a causal, time-invariant IO function observation structure can be defined as:

$$G_F = \langle T, X, S_i, Y, IOspace_G, F_G, \gamma_G \rangle$$

where

T	time base
X	input value set
Y	output value set
S_i	set of initial states
$IOspace_G$	causal, time-invariant input/output segment generator set
F_G	IO function generator set
γ_G	final state hypothesizer

with the following constraints:

$$F_G : S_i \longrightarrow \text{partial } IOspace_G,$$
$$\gamma_G : S_i \times IOspace_G \longrightarrow S_i.$$

The interpretation of the function γ_G is that it maps a given initial state and an input/output segment pair into another initial state.

To obtain an output segment ψ_G for an input segment $\omega_G \in \Omega_G$, an initial state $s_i \in S_i$, and a given data set F_G, the *output segment matching function* is defined as:

$$\eta_G : S_i \times \Omega_G \times F_G \longrightarrow \Psi_G.$$

where the elements of F_G are pairs of IO generator segments.

The interpretation of the above function is that if there exists $s_i \in S_i$ and $f : s_i \mapsto (\omega_G, \psi_G) \in F_G$ then

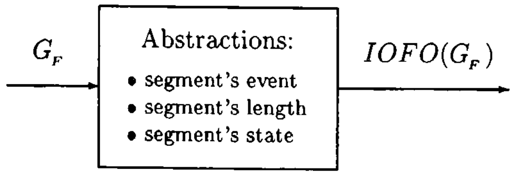

Figure 1: Role of Assumptions in Deriving $IOFO(G_F)$ From G_F.

$\eta_G : (s_i, \omega_G, f) = \psi_G$. For example, using the output segment matching function η_G and the special form of the *final state hypothesizer function* $\gamma_G : \Psi_G \to S_f$, the final state can be determined by consecutive application of γ_G followed by η_G. That is, $\gamma_G(\eta_G(s_i,\omega,f)) = \gamma_G(\psi) = s_f$ or using the general form of the final state hypothesizer, $\gamma_G : S_i \times IOspace_G^+ \to S_i$, we have the general form for combining η_G and γ_G:

$$\gamma_G(s_i,(\omega,\eta_G(s_i,\omega,f))) = \gamma_G(s_i,(\omega,\psi)) = s_i'.$$

This iterative IOFO specification can be specialized into Discrete-event IOFO specification (DEVF).

3 ASSUMPTION-BASED ITERATIVE IOFO SPECIFICATION

Once we have iterative IOFO, its *free* iterative specification is $IOFO(G_F) = \langle T, X, S_i, Y, IOspace_G^+, F_G^+ \rangle$, where F_G^+ is the set of all nonempty finite concatenations of elements of F_G. Likewise, $IOspace_G^+$ is constructed from $IOspace_G$.

However, if F_G does not represent a system's complete IO behavior, then it may be necessary to *compose* trajectories from segments, some of which have to be predicted based on some belief set which we denote as an *assumption set* (see Figure 1). To compose two segments ω_1 and ω_2, they may be concatenated as $\omega_2 \circ \omega_1$. However, unless the final state of ω_2 is the same as the initial state of the ω_1, we are forced to make $\omega_2 \circ \omega_1$ a hypothesis. That is, ignoring a mismatch between the final state of ω_2 and the initial state of the ω_1 results in a hypothesized trajectory. Given an input segment, we define the assumption set to allow abstractions on *length, event* and *state*. In this setting, the specification based on the assumption set is called *assumption-based iterative IOFO*.

For example, suppose we have two input segments $\omega_{i,j}, \omega_{k,\ell} \in \Omega_G$ (where they are contiguous w.r.t. to time) and the initial state s_i is associated with $\omega_{i,j}$. Given the extended output segment matching function,

$$\eta_G^* : S_i \times \Omega_G^+ \times F_G^+ \longrightarrow \Psi_G^+,$$

defined by:

$$\eta_G^*(s_i, \omega_{i,j} \circ \omega_{k,\ell} \circ \ldots \circ \omega_{n,q}, f_i, f_k, \ldots, f_n) = \eta_G((\ldots \gamma_G(\eta_G(\gamma_G(\eta_G(s_i, \omega_{i,j}, f_i)), \omega_{k,\ell}, f_k)), \ldots), \omega_{n,q}, f_n)$$

we have:

$$\begin{aligned}
&(\omega_{i,j}, \psi_{i,j}) \in F_G(s_i),\\
&(\omega_{k,\ell}, \psi_{k,\ell}) \in F_G(s_k),\\
&f_i : s_i \mapsto (\omega_{i,j}, \psi_{i,j}) \in F_G,\\
&f_k : s_k \mapsto (\omega_{k,\ell}, \psi_{k,\ell}) \in F_G, \text{ and}\\
&\eta_G^*(s_i, \omega_{i,j} \circ \omega_{k,\ell}, f_i, f_k) = \psi_{i,j} \circ \psi_{k,\ell}.
\end{aligned}$$

In the above composition (i.e., $\psi_{i,j} \circ \psi_{k,\ell}$), the final state of $(\omega_{i,j}, \psi_{i,j})$ is the same as the initial state of $(\omega_{k,\ell}, \psi_{k,\ell})$. If a composite output segment $\psi_{i,j} \circ \psi_{k,\ell}$ can be generated by applying η_G and γ_G on individual segments, then we have:

$$\eta_G(\gamma_G(\eta_G(s_i, \omega_{i,j}, f_i)), \omega_{k,\ell}, f_k) = \psi_{i,j} \circ \psi_{k,\ell}.$$

What this says is that if $\gamma_G(\eta_G(s_i, \omega_{i,j}, f_i)) = \gamma_G(\psi_{i,j}) = s_k'$ and the initial state associated with $\omega_{k,\ell}$ is s_k, then $s_k' = s_k$ must be satisfied in order for η_G^* to hold. In the next section, we discuss how to overcome the restrictiveness of *equality* that is required in composing two IO segments can be used to so that $s_k'' = s_k$.

3.1 IO Segment Pair Types

The IO function generator set F_G is simply a database containing pairs, each comprised of an initial state associated with an IO segment pair. That is,

$$\begin{aligned}
&F_G : S_i \longrightarrow \text{partial } IOspace_G,\\
&f = (s, g) \in F_G \qquad \text{where} \qquad g = (\omega, \psi).
\end{aligned}$$

In this form, no final state is assigned to any IO segment pair. Instead, the quasi-state identification function γ_G is specified in G_F to hypothesize about them. For any particular set of final states, we can suppose that every IO segment has both an initial state s_i and a final state s_f (i.e., $(s_i, s_f, (\omega, \psi))$). We now begin with a classification of all possible ways in which an input segment might be represented.

Let us denote an input segment as (s_i, ω), an output segment as (s_f, ψ), and an IO segment pair as $((s_i, \omega), (s_f, \psi))$. An input segment's representation, without considering its initial and final states for now, can be categorized into several types depending on whether events occur at one or both of its initial and final time-points; likewise, for output segments. For example, suppose we have an input segment ω and an output segment ψ with duration dt. We associate t_i with the initial time-point of the input/output segment pair and t_f with its final time-point. All segments are assumed to be of the discrete-event type since we confine our discussion to discrete-event systems. Thus,

- Input segment type 1:

$$\omega(t) = null\_event \qquad \text{for} \qquad t_i \le t \le t_f$$

- Input segment type 2:

$$\omega(t) = \begin{cases} input\_event & \text{for} \quad t = t_i \\ null\_event & \text{for} \quad t_i < t \le t_f \end{cases}$$

- Input segment type 3:

$$\omega(t) = \begin{cases} null\_event & \text{for} \quad t_i \le t < t_f \\ input\_event & \text{for} \quad t = t_f \end{cases}$$

- Input segment type 4:

$$\omega(t) = \begin{cases} input\_event & \text{for} \quad t = t_i \\ null\_event & \text{for} \quad t_i < t < t_f \\ input\_event & \text{for} \quad t = t_f \end{cases}$$

Likewise, output segments are of the same 4 types. Then, considering input and output segments together, we require that each conform to one of the following 4 types out of all 16 possible combinations of IO segment pairs.

- IO segment type 1:

$$\omega(t) = null\_event \qquad \text{for} \qquad t_i \le t \le t_f$$

$$\psi(t) = null\_event \qquad \text{for} \qquad t_i \le t \le t_f$$

- IO segment type 2:

$$\omega(t) = \begin{cases} input\_event & \text{for} \quad t = t_i \\ null\_event & \text{for} \quad t_i < t \le t_f \end{cases}$$

$$\psi(t) = null\_event \qquad \text{for} \qquad t_i \le t \le t_f$$

- IO segment type 3:

$$\omega(t) = null\_event \qquad \text{for} \qquad t_i \le t \le t_f$$

$$\psi(t) = \begin{cases} null\_event & \text{for} \quad t_i \le t < t_f \\ output\_event & \text{for} \quad t = t_f \end{cases}$$

- IO segment type 4:

$$\omega(t) = \begin{cases} input\_event & \text{for} \quad t = t_i \\ null\_event & \text{for} \quad t_i < t \le t_f \end{cases}$$

$$\psi(t) = \begin{cases} null\_event & \text{for} \quad t_i \le t < t_f \\ output\_event & \text{for} \quad t = t_f \end{cases}$$

The rationale for this restriction is the following: The trajectories for single-input, single-output systems as well as a class of multi-input multi-output systems, can be partitioned in several ways. Clearly, IO trajectories for a system ought to be partitioned with respect to one another. If there are no time-points at which both an input event and output event occur, then the partitioning of IO trajectories results in IO segments having the above proper types. This is not a restriction *per se* since inductive reasoning should operate on models having coarse granularity. A trajectory may be partitioned at time-points where either an input event or an output event occurs. Thus, partition points (time instants along a trajectory where an event occurs) should occur only at time-points where either an input event or an output event occurs, exclusively.

Having decided on the proper representations for input and output segments, we can determine the representation of *Complete IO segments.* That is, any IO trajectory pair (partitioned according to assumption set-I) would result in IO segment pairs represented as:

$$((s_i,(x_{val},dt)), \quad (s_f,(y_{val},dt)))$$

where dt is the duration of IO segments and $x_{val} \in \{nil, input\_event\}$ and $y_{val} \in \{nil, output\_event\}$.

Hence we can reformulate the earlier specification of G_F in terms of $((s_i,(x_{val},dt)), \ (s_f,(y_{val},dt)))$ as:

$$\widehat{G_F} = \langle T, X, S, Y, IOspace_G, \widehat{F_G} \rangle \qquad \text{where}$$

T	time base
X	input value set
Y	output value set
S	set of states
$IOspace_G$	time-invariant IO segment generator set
$\widehat{F_G}$	partial IO function generator set

and

$$\widehat{F_G} : \quad S_i \times PJN(IOspace_G, 1) \rightarrow S_f \times PJN(IOspace_G, 2)$$
such that $S_i \subseteq S$, and $S_f \subseteq S$ or

$$\widehat{F_G} = \{((s_i,(x_{val},dt)), \ (s_f,(y_{val},dt))) \\ s_i, s_f \in S, \\ (x_{val},dt) \in PJN(IOspace_G, 1), \\ (y_{val},dt) \in PJN(IOspace_G, 2)\}.$$

3.2 Input Segments Equivalence

The purpose behind the iterative IOFO specification is to support predictability. Given *candidate input segment*, the above assumption set is to be used to reason about IO segments contained in $\widehat{F_G}$ such that its output segment can be predicted. An unobserved input segment is called a candidate input segment. A *concrete* input segment (or IO segment), however, refers to one that is observed. We need to be specific about what the assumption set is, and what it entails given $\widehat{G_F}$.

Suppose we are given an input trajectory partitioned into a finite number of sequential segments and for which we would like to find its corresponding output trajectory. For example, suppose one of its input segments is $(s_i', (x_{val}', dt'))$, and there exists an input/output segment $((s_i,(x_{val},dt)),(s_f,(y_{val},dt))) \in \widehat{F_G}$. If the input segment $(s_i', (x_{val}', dt')$ is *equal* to the input segment $(s_i,(x_{val},dt))$ (i.e., $x_{val} = x_{val}'$, $dt = dt'$, and $s_i = s_i'$), then it is trivial to determine its corresponding output segment.

However, the hope of composing the output trajectory is dashed if no equal input segment can be found in the database. For instance, given the three sequential input segments $(s_1', (x_1', dt_1'))$, $(s_2', (x_2', dt_2'))$, and $(s_3', (x_3', dt_3'))$, no output segments will be found for the second or third input segments if no output segment can be found for the first. That is, given the candidate input segment $(s_1', (x_1', dt_1')) \notin PJN(\widehat{F_G}, 1)$, (i.e., there exists no IO segment pair $((s_i,(x_{val},dt))$, $(s_f,(y_{val},dt))) \in \widehat{F_G}$ in the database, such that $x_{val} = x_1'$, $dt = dt_1'$, and $s_i = s_1'$), then no output trajectory can be obtained.

The notion of equality, of course, is *too strong* for inductive modeling. It becomes imperative to speak of *equivalence* instead. (Our usage of the term equivalence is different from the one used in Gill (1962) where various notions of (deductive) equivalence are defined for finite-state memory machines.) Otherwise, we have to limit our claims of prediction to trajectories that can be composed from the IO segments found in $\widehat{F_G}$ only. It is impossible to find in the database equal input segments for all imaginable new input segments. This underlies the need for defining equivalence between two input segments. We use the

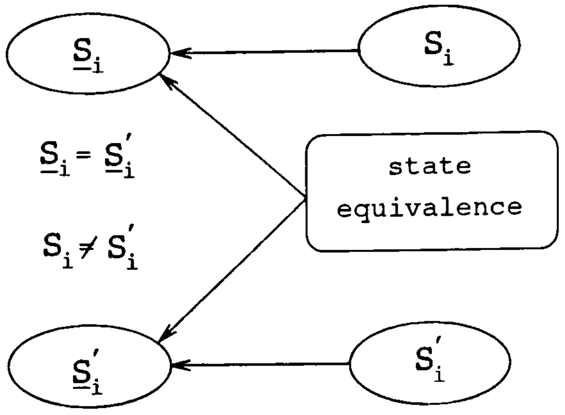

Figure 2: Abstraction of Two Concrete States Into Their Respective Abstracted States

term equivalence to indicate that, even though two input segments are not equal, we can *think* of them as being equal.

Given input segments $(s'_i, \omega') \notin PJN(\widehat{F_G}, 1)$ and $(s_i, \omega) \in PJN(\widehat{F_G}, 1)$, when they can be *considered* to be equal? That is., when they can be called *equivalent*. Given the two input segments $(s_i, (x_{val}, dt))$ and $(s'_i, (x'_{val}, dt'))$, three primitive types of equivalences are possible. They are based on *length-equivalence*, *input-equivalence*, and *state-equivalence*. Each equivalence type ignores the inequality in one of three aspects: length (or duration), initial state, or input.

In our earlier example, we had $(s_i, (x_{val}, dt))$ and $(s'_i, (x'_{val}, dt'))$, where $s'_i \neq s_i$, $x'_{val} = x_{val}$, and $dt' = dt$. Consequently, we could assume state-equivalence in order to consider these two segments as being equivalent. How can this formally be accomplished? One approach is to turn the inequality $s'_i \neq s_i$ to an equality between their respective abstracted states $\underline{s}_i$ and $\underline{s}'_i$. Now, we treat the state-equivalence between s_i and s'_i as an *assumption*. Then, we can use $s'_i \neq s_i$ (fact) along with the state-equivalence assumption (belief) to construct two abstract states $\underline{s}_i$ and $\underline{s}'_i$ from the concrete states s_i and s'_i where $\underline{s}'_i = \underline{s}_i$ (refer to Figure 2). The term *fact* is restricted in the sense that its truth value is fixed and cannot be subjected to revision. The term *belief*, however, may change its truth value. Another difference between these is that, whereas a belief can be converted to a fact, the converse is not true. Every piece of data is either a fact or a belief, exclusively.

Hence the use of the state-equivalence is substantiated by treating the inequality of two states as a fact while using the state-equivalence assumption between them as a belief. Having two abstracted states $\underline{s}'_i = \underline{s}_i$, as well as two concrete input segments where $(s_i, (x_{val}, dt)) \neq (s'_i, (x'_{val}, dt'))$, then the abstract candidate input segment $(\underline{s}'_i, (x'_{val}, dt'))$ and the abstract observed input segment $(\underline{s}_i, (x_{val}, dt))$ are equal (see Figure 3). That is

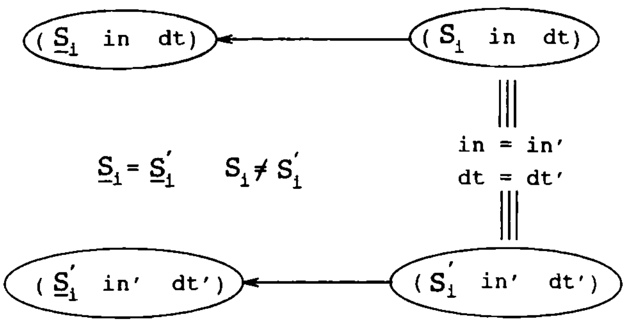

Figure 3: Abstraction of Two Input Segments Into Their Respective Abstracted Input Segments

$$(\underline{s}_i, (x_{val}, dt)) = (\underline{s}'_i, (x'_{val}, dt')).$$

Now, with well-defined semantics, we can simply use the state-equivalence knowing that in fact we are using inequality of s_i and s'_i together with the state-equivalence assumption. Thus, we may say that the two concrete input segments are equivalent.

Definition 1 *Two input segments* $(s_i, (x_{val}, dt))$ *and* $(s'_i, (x'_{val}, dt'))$ *are called:*

1. *length-equivalent iff*

$$x_{val} = x'_{val}, \quad \underline{dt} = \underline{dt}', \quad s_i = s'_i;$$

2. *input-equivalent iff*

$$\underline{x}_{val} = \underline{x}'_{val}, \quad dt = dt', \quad s_i = s'_i,$$

3. *state-equivalent iff*

$$x_{val} = x'_{val}, \quad dt = dt', \quad \underline{s}_i = \underline{s}'_i$$

where $\underline{dt}, \underline{dt}', \underline{x}_{val}, \underline{x}'_{val}, \underline{s}_i, \underline{s}'_i$ *are abstractions of* $dt, dt', x_{val}, x'_{val}, s_i, s'_i$*, respectively.*

That is, despite the presence of an inequality in each of the above equivalences, the two input segments are *believed* to be equal. Hence, given two input segments, either the *length*, the *input*, or the *state* can be ignored in terms of their corresponding equivalences. Various combinations of the above equivalences comprise one form of the assumption set.

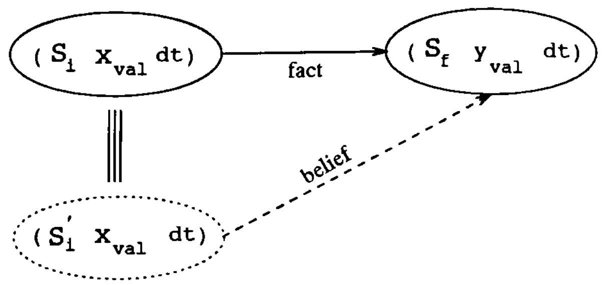

Figure 4: Constructing an Unobserved IO Segment

3.3 Predicting IO segments

In the above example, state-equivalence is supported by abstraction of states. In general, we have:

$$\begin{aligned}&(s_i, (x_{val}, dt)) \equiv (s_i', (x_{val}', dt'))\\ &\quad s_i \neq s_i',\ \ \underline{s_i} = \underline{s_i'};\\ &\quad dt \neq dt',\ \ \underline{dt} = \underline{dt'};\\ &\quad x_{val} \neq x_{val}',\ \ \underline{x}_{val} = \underline{x}_{val}'\end{aligned}$$

where one or more of the inequalities may be used in generating equivalence between two unequal input segments.

Although the concrete candidate input segment $(s_i', (x_{val}', dt')) \notin \widehat{F_G}$, we have seen its abstraction can be equal to the abstraction of an observed input segment. The equality between these two abstractions can be used to construct a new unobserved IO segment (cf. Figure 4) with the concrete candidate input segment and the output segment of the observed IO segment. That is, we can construct (predict)

$$((s_i', (x_{val}', dt')),\quad (s_f, (y_{val}, dt))).$$

To predict an output trajectory for an input trajectory, it is essential to partition the input trajectory into candidate input segments. The assumption on a segment's length is devised to partition an input trajectory using one of three possibilities. We have specialized length-equivalence to be one of *longest, exact,* and *all.* The first choice, *exact,* is used when any candidate input segment's length must be equal to one of the input segments from the repository. The second choice, *longest,* prefers the candidate input segments with the longest length which match input segments from the repository. The third choice, *all,* is defined to allow candidate input segments of any length.

To deal with incorrect hypothesized IO segments, the machinery of the non-monotonic reasoning provides well-defined means to retract such hypotheses once there exists support to do so.

4 AN EXAMPLE

Suppose we have observed two IO trajectories from a FIFO (first-in-first-out) discipline queue (refer to Figure 5). Then the discrete-event inductive reasoner is able to predict an output trajectory, given the candidate input trajectory. To generate the predicted output trajectory, an assumption set which is comprised of two assumptions is chosen. The first prescribes how the input trajectory is to be partitioned with respect to the IO segments in the repository. The second allows the inequality of the input events between a candidate input segment and an input segment from the repository of available IO segments. The predicted output trajectory is not completely correct. (Other possibilities exist based on alternative assumptions.) Instead of predicting output event c to occur at time 3, it is predicted to occur at time 2. This is due to length abstraction. Also, the output event is predicted as b instead of a, due to input event abstraction. When there was no matching input segment in the repository, (i.e., input event c and duration 1, and input event a and duration 7), it was inevitable that assumptions would be made when predicting output segments. Without making assumptions, it would have been impossible to predict anything given the observed IO segments one and two.

5 RELATED WORK/CONCLUSIONS

The importance of abstraction is undisputed in modeling and simulation. The inductive modeling research programs that we are aware of are based on techniques from artificial intelligence, probability theory in many different forms, neural networks, and genetic algorithms. Of these, our approach falls into the artificial intelligence arena in that it allows the introduction and manipulation of abstractions. We showed that once a well-defined structure is available, with the aid of powerful non-monotonic reasoning paradigm, abstractions can be formalized and manipulated to deal effectively with one of the fundamental issues in inductive modeling — qualification problem.

None of the other inductive modeling methodologies can support making explicit assumptions about a system's critical features such as state, inputs, and outputs. In an earlier effort, Dietterich (1984), given some observed behavior of a system, discussed how partial theories can be constructed and revised. However, this work lacks a framework. Hence, it provided no canonical representation for capturing a system's observed behavior. Also it did not establish the important role of non-monotonic reasoning in inductive

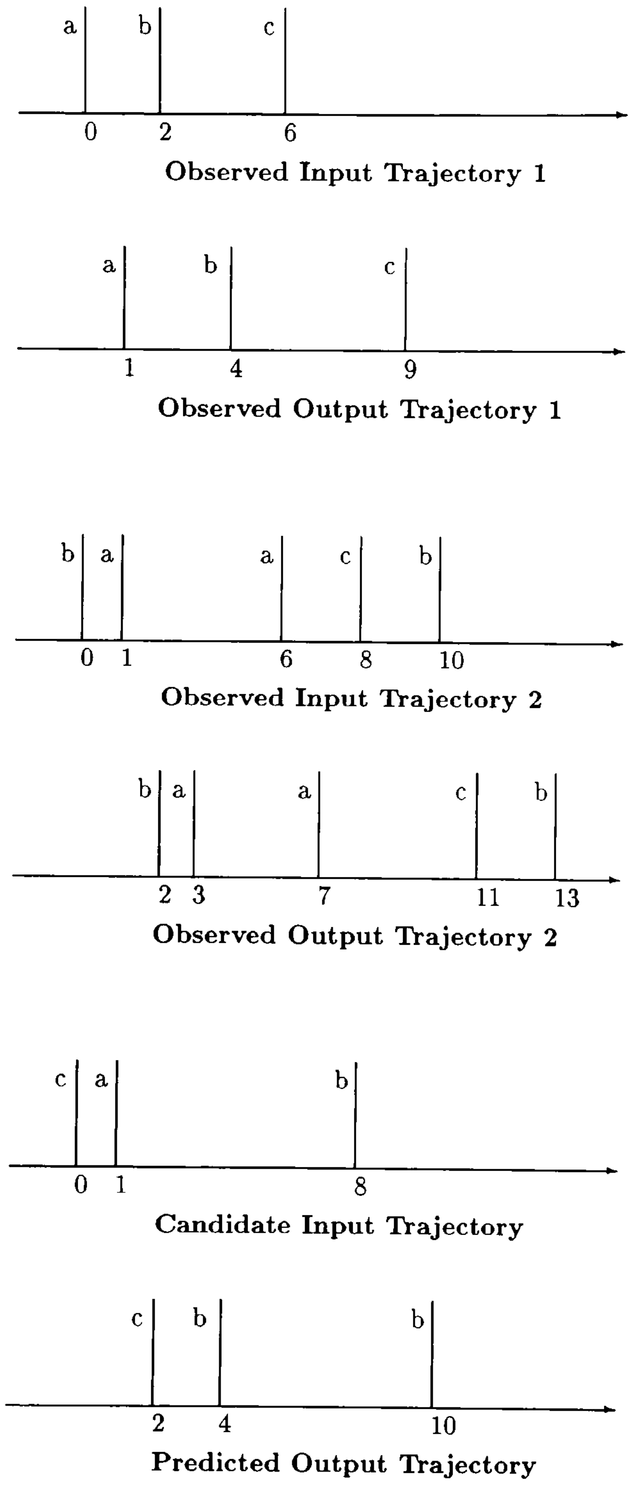
a b c
0 2 6
Observed Input Trajectory 1
a b c
1 4 9
Observed Output Trajectory 1
b a a c b
0 1 6 8 10
Observed Input Trajectory 2
b a a c b
2 3 7 11 13
Observed Output Trajectory 2
c a b
0 1 8
Candidate Input Trajectory
c b b
2 4 10
Predicted Output Trajectory

Figure 5: FIFO Discipline Example

modeling. Consequently, no abstraction mechanisms were introduced and integrated to support explicit reasoning with the observed data based on abstraction (assumptions).

Our concern is with causal/time-invariant dynamic systems. Other research efforts in inductive modeling that have focused on such systems are the works of Biermann and Feldman (1972), Dietterich (1984), Klir (1985), and Cellier (1991) although none of these takes into account the role of abstraction as we have.

In this light, our approach to inductive modeling differs fundamentally from those that are based on probability theory, neural networks, as well as other artificial approaches. We cannot engage in a discussion of AI-based inductive modeling approaches. However, it suffices to indicate that artificial intelligence approaches either represent states implicitly or contain no state variables; or just deal with static systems.

In conclusions, DEVS inductive modeling framework has been developed as part of the DEVS framework, Zeigler (1976, 1984, 1990), with the objective of laying a foundation based on concepts from systems theory and artificial intelligence. It supports abstractions representation and reasoning with them such that lack of data can be explicitly dealt with. The predictably of a system's behavior can benefit greatly by employing useful and effective abstractions. Our work discussed here suggests research in several directions: applying DIR to other types of systems and domains, studying issues related to handling large amounts of data, supporting the use of domain-specific knowledge, and formally characterizing abstraction (i.e., introduction of assumptions and their manipulation) in more depth.

REFERENCES

Biermann A. W. and J. A. Feldman, 1972. On the Synthesis of Finite-State Machines from Samples of Their Behavior, *IEEE Transactions on Computers*, Vol. 21, pp. 592–597.

Cellier F. E., 1991. *Continuous System Modeling*, Springer-Verlag, New York.

Davis E., 1990. *Representation of Commonsense Knowledge*, Morgan Kaufmann.

Dietterich T. G., 1984. Learning about Systems that Contain State Variables, *AAAI*, pp. 96–100.

Fishwick P. A., 1995. *Simulation Model Design and Execution: Building Digital Worlds*, Prentice Hall.

Forbus K. D. and J. de Kleer, 1993. *Building Problem Solvers*, MIT Press.

Genesereth M. and N. J. Nilsson, 1987. *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann.

Gill A., 1962. *Introduction to the Theory of Finite-State Machines*, McGraw-Hill, New York.

Ginsberg M. L., 1993. *Essentials of Artificial Intelligence*, Morgan Kaufmann.

Giunchiglia F. and T. Walsh, 1992. A Theory of Abstraction, *Artificial Intelligence*, Vol. 57, pp. 323–389.

Grossberg S., 1988. *Neural Networks and Natural Intelligence*, MIT Press, Cambridge.

Hobbs J. R. and R. C. Moore (editors), 1985. *Formal Theories of the Commonsense World*, Ablex Publishing Corporation.

Klir G. J., 1985. *Architecture of Systems Problem Solver*, Plenum Press.

Kuipers B. J., 1994. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*, MIT Press.

McCarthy J., 1990. *Formalizing Common Sense: Collected Papers of John McCarthy on Commonsense Reasoning*, edited by V. Lifschitz, Ablex Publishing Corporation.

Michalski R. S., J. G. Carbonell and T. M. Mitchell (editors), 1986. *Machine Learning: An Artificial Approach*, Vol. 2, Morgan Kaufmann.

Omlin C. W., K. K. Thornber, and C. L. Giles, 1996. Fuzzy Finite-state Automata can be Deterministically Encoded into Recurrent Neural Networks, Tech. Report UMIACS-96-12, University of Maryland.

Sarjoughian H. S., 1995a. Inductive Modeling of Discrete-event Systems: A TMS-based Non-monotonic Reasoning Approach, Ph.D. Thesis, University of Arizona, Department of Electrical and Computer Engineering.

Sarjoughian H. S. and B. P. Zeigler, 1995b. Inductive Modeling: A Framework Marrying Systems Theory and Non-monotonic Reasoning, P. Antsaklis, et. al. (Editors), *Hybrid Systems II*, LNCS 999, pp. 417-435, Springer Verlag.

Weld D., 1992. Reasoning About Model Accuracy, *Artificial Intelligence*, Vol. 56, pp. 255-300.

Wymore W. A., 1993. *Model-based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*, CRC, Boca Raton.

Zeigler B. P., 1976. *Theory of Modeling and Simulation*, John Wiley and Sons.

Zeigler B. P., 1984. *Multi-Facetted Modelling and Simulation*, Academic Press.

Zeigler B. P., 1990. *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*, Academic Press.

AUTHOR BIOGRAPHIES

HESSAM S. SARJOUGHIAN earned his M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Arizona in 1989 and 1995. His interests are Modeling and Simulation Methodologies, Artificial Intelligence, Software Engineering, and Intelligent Control with applications in engineered/natural systems. Presently, he is working on a Computer-Aided Business Engineering project funded by the Armstrong Research Laboratory, US Air Force. He has worked at Allied-Signal Aerospace company and International Business Machine.

BERNARD P. ZEIGLER is a professor of Electrical and Computer Engineering at the University of Arizona, Tucson. He has published over two hundred journal and conference articles on modeling and simulation, knowledge-based systems, and high-autonomy systems. His books include *Theory of Modelling and Simulation* (Wiley, 1976), *Multifaceted Modelling of Discrete Event Simulation* (Academic Press, 1984), and *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems* (Academic Press, 1990). He has a new book which will appear in 1996 to be published by Springer-Verlag, NY: "Objects and Systems". Zeigler's research has been supported by federal agencies including NSF, NASA, USAF, and the US Army, as well as industrial sponsors including Siemens, McDonnell Douglas, and Motorola. He was elected as Fellow of the IEEE for his innovative work in discrete event modelling theory. In 1995, Zeigler served on a National Research Council committee to suggest directions for information technology in the 21st Century US Army and is currently a member of an NRC committee given a similar task by the US Navy, focusing on modelling and simulation. Zeigler is currently editor-in-chief of the Transactions of The Society for Computer Simulation. He received his B. Eng. Physics from McGill University, 1962, M.S.E.E. from MIT, 1964, and Ph.D. from the University of Michigan in 1968.