Exhibiting the behaviour of time-delayed systems via an extension to qualitative simulation

Miguel, Ian; Shen, Qiang

Published in:
IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)

DOI:
10.1109/TSMCA.2005.843387

Publication date:
2005

Citation for published version (APA):
Miguel, I., & Shen, Q. (2005). Exhibiting the behaviour of time-delayed systems via an extension to qualitative simulation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 298-305. https://doi.org/10.1109/TSMCA.2005.843387

Exhibiting the Behavior of Time-Delayed Systems via an Extension to Qualitative Simulation

Ian Miguel and Qiang Shen

***Abstract*—This paper presents an extension to qualitative simulation that enables a qualitative reasoning system to support variables that exhibit delayed reactions to their constraining functions. Information stored in the previous levels of the behavior tree is retrieved and used to constrain multiple delayed variables and to capture the time-delay behavior of the system. The extension is applicable to qualitative simulators that generate time-stamped behaviors. In particular, this is implemented and integrated with the existing fuzzy qualitative simulation algorithm. Results of an example application of this extended algorithm are provided.**

***Index Terms*—Qualitative fuzzy simulation, qualitative reasoning, time delays.**

I. Introduction

A standard technique used to control industrial plants is to choose a measured variable and maintain the required value of this variable through a process of measurement, comparison, and adjustment. A time delay between a disturbance in the plant and the effects of this disturbance showing in the behavior of a measured variable presents an important problem to systems control. This is because the longer the delay is, the further the plant may have deviated from the designed conditions and hence the harder it becomes for the control system to regulate the measured variable.

A solution to this problem is to determine how a time-delayed system will behave, subject to a certain initial condition through computer simulation, so that appropriate control actions can be taken. The system may be simulated numerically via a differential equation model, but this is only possible when the system parameters are precisely known. Further, such a simulation could only use real number values to specify the initial state and parameters; it is far more useful to be able to simulate a whole *range* of values. An interval could be discretized, such that regular points along it are simulated numerically. However, this introduces two problems: first, how to guarantee that all possible behavior is simulated if the process exhibits nonlinear characteristics, and second, how to interpret the behavior at the boundaries between neighboring intervals.

Qualitative reasoning [1]–[3] has already proven successful in modeling complex processes, where a numerical solution is difficult or infeasible to obtain, (e.g., [4]–[6]). Here, the system behaviors are simulated nonnumerically, with each variable taking *symbolic* values. Hence, a single qualitative variable represents many quantitative behavior possibilities, significantly reducing the complexity of the simulation. In most cases, system descriptions are not required to be complete. Current qualitative reasoning research concentrates on simulating time-invariant behavior, however. Most existing qualitative simulation methods [7], [8] do not deal with systems that exhibit a time-delay, such as $y(t_i) = f(x(t_{i-\Delta T}))$. The CA-EN simulator [9] may support time-delayed behavior, but requires explicit causal interaction information between the variables and parameters of a system. CA-EN also uses a different underlying simulation mechanism from that used by the simulators supported by the current proposed extension.

This paper, based on initial work presented in [10], shows how a qualitative simulation algorithm may be extended to support such systems. Although developed with the fuzzy qualitative simulation algorithm (FuSim) [8] in mind, which was previously reported in this journal, the extension is generally applicable to simulators (e.g. [7]) which enable an estimation of the durations of generated qualitative states and whose underlying simulation mechanism involves first producing all possible behaviors and then filtering out those which conflict with the system description or model.

It is worth noting that this paper addresses the use of qualitative structural and behavioral models to perform semisymbolic simulation in an explicit manner. This should not be confused with the work carried out in the area of fuzzy logic control. Although very successful in practical applications, fuzzy control approaches generally follow a different knowledge representation scheme, where physical systems are typically described using fuzzy relational models or production rules [11], [12]. Also, the work on system behavior identification through phase-plane analysis and discrete sampling, as used in conventional and fuzzy-control engineering, forms a complementary contribution to that made herein.

However, this paper has, to a large extent, conceptual resemblance with the findings as reported in [13]–[16]. For example, the approach developed in this paper and the work on fuzzy inductive reasoning (FIR) [13], [14] are both essentially intended to model the process of human understanding of complex physical systems. Yet FIR works with the assumption that humans analyze system behavior on the basis of analogies with similar processes, via implementing the reasoning systems with pattern recognition techniques. The present research focuses on how system behavior may be exhibited via qualitative physical considerations, mimicking humans' ability to make inferences on the basis of coarse and structured knowledge. This forms a sharp contrast with the underlying approach taken in FIR. In addition, the present work has a specific task of modeling time-delay behaviors, extending the simulation power of algorithms akin to FuSim, which the other relevant work does not focus on.

The rest of this paper is arranged as follows. Section II gives an overview of the FuSim algorithm as a basis for the proposed improvements. It also highlights the data structure, the behavior tree, that is fundamental to this approach to qualitative time-delay simulation. Section III describes exactly how FuSim may be updated to support time-delayed system variables, detailing the changes that must be made to the algorithm. Section IV provides experimental results and analysis. Finally, Section V concludes the paper.

II. Background

To be self-contained, a brief review of the FuSim algorithm is given here. Also, an outline of the problem of (qualitative) time-delays is provided.

A. Overview of FuSim

Most approaches to qualitative simulation developed in qualitative reasoning adopt the generate-and-test methodology in producing behavioral simulation of physical systems. That is, system behaviors are qualitatively exhibited by first generating all possible descriptions, based on the assumed continuity and differentiability of the variables, and then removing those behavioral descriptions that are inconsistent

Manuscript received August 2, 1999; revised November 20, 2002 and March 24, 2004. This work was supported in part by the UK-EPSRC under Grant 97305803. This paper was recommended by Associate Editor W. Pedrycz.

I. Miguel is with the Artificial Intelligence Group, Department of Computer Science, University of York, York YO10 5DD, U.K. (e-mail: ianm@cs.york.ac.uk).

Q. Shen is with the Department of Computer Science, University of Wales, Aberystwyth SY23 3DB, U.K. (e-mail: qqs@aber.ac.uk).

Digital Object Identifier 10.1109/TSMCA.2005.843387

with the given system model, which is represented as a set of qualitative differential equations. A qualitative differential equation imposes restriction over the possible values of the variables involved in it, where each variable takes values from a finite quantity space [1]–[3]. Such an equation is hereafter generally referred to as a qualitative constraint.

The FuSim algorithm is a relative of the qualitative simulation algorithm, QSIM [3]. It models a physical system using the structural description of its behavior, defined by a set of fuzzy constraints between the system variables. In place of conventional qualitative constraints, which each involve at most three arguments, the following generic constraint format is used:

$$\mathrm{LHS} \cap \mathrm{RHS}.$$

Both sides of the intersection may be arbitrarily complex, removing the requirement for several strict fuzzy constraints connected by pseudovariables, which are artificially introduced for modeling purposes, to represent a single more complex constraint. Here, the constraint is satisfied if there is a (fuzzy) intersection between the left- and right-hand sides (LHS and RHS, respectively) [17]. This allows a significant reduction in spurious behaviors otherwise generated due to qualitative ambiguities [18].

As per QSIM, a system variable is described in terms of its qualitative state, which is in turn described by a pair of its qualitative magnitude and qualitative rate of change. However, FuSim makes use of the theory of fuzzy sets to discretize the representation of system variables, as opposed to the alternating point and interval method used by QSIM. This helps ease the difficulty in representing the behavior at boundaries between adjacent states. In particular, both the magnitude and the rate of change of the system variables range over an arbitrarily discretized fuzzy quantity space [8].

An outline of FuSim follows in order that subsequent changes to the algorithm may be related to the original design. Given a structural description consisting of a set of constraints over the system variables and an initial system state, the objective is to produce a set of possible behaviors of the system. The first step is to calculate the possible next states of each variable by applying a set of rules which dictate, given the current magnitude and derivative components of a variable, what values these components may take in the next time-step. The next step is to generate a set of state-tuples for each constraint that consist of the cross product of the possible next states of those variables involved in that particular constraint. The resultant sets of tuples are checked for consistency using a standard constraint satisfaction technique [19]. This involves two substeps: a) *Self-Consistency Filtering*: Restriction is imposed over the set of tuples associated with each constraint such that the remainder all satisfy that constraint; and b) *Pairwise Filtering*: Tuples that are inconsistent between constraints that share a common variable are removed.

A set of potential next system states are then generated by recombining the remaining tuples, and are in turn restricted by the application of global filters such as energy conservation [20] and phase trajectory nonintersection [21]. The simulation algorithm must then process each of the remaining potential next system states in the same way. The output of this process is a behavior tree: each node corresponds to a single system state, and each branch corresponds to a distinct possible behavior pattern for the system (see Fig. 1).

B. Time Delays

It is useful at this stage to examine a method that could be used to implement a time delay in a conventional numerical simulation. Consider the system

$$y(t_i) = x(t_{i-2}).$$

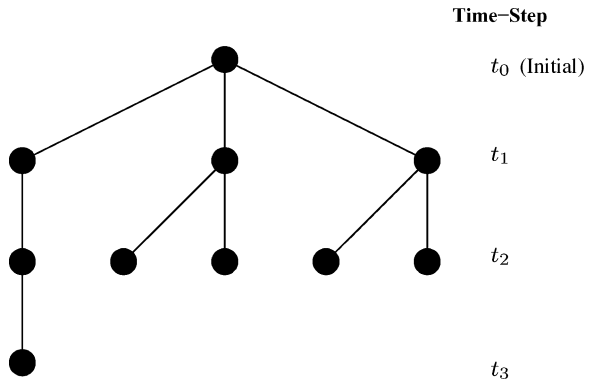

Fig. 1. Behavior tree.

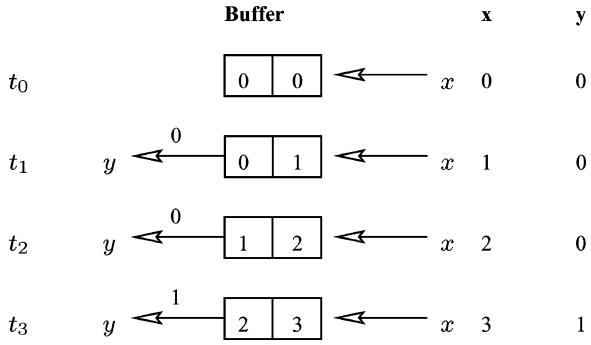

Fig. 2. Buffer mechanism.

Clearly, the value of y depends on that of x from two time steps ago. A typical method for modeling this within an iterative numerical simulation is to use a buffer. The buffer holds previous values of x, which may be retrieved in order to update y, as shown in Fig. 2.

Initially, the buffer is filled with the current value of y, so that y cannot change before the requisite number of time-steps have elapsed. At each iteration y takes its value from the leftmost element of the buffer. The entire contents of the buffer are then shifted left, and the current value of x is stored in the rightmost element. Therefore, to model d steps of delay, a buffer of d elements is required. The expression used to calculate y may be arbitrarily complex, as long as the result is stored in this way.

This buffer-based method is relatively simple for numerical simulation since just one value has to be stored at each time-step. However, as the behavior tree is generated, FuSim must process multiple possible behavior patterns, creating a much more complex scenario (though FuSim will produce a unique behavior for the above extremely simple case). Although qualitative time-delay may be supported via spurious dummy variables, this would have the undesirable effect of an increase in the branching factor of the behavior tree [17]. In the worst case, each dummy variable multiplies the branching factor by six, the number of transition rules FuSim uses to generate successor states for each variable.

1) Qualitative Time Delay: It is possible to use information stored in the behavior tree to implement a qualitative time-delay mechanism. Such information can be an estimation of the durations of the states or simply a symbolic time-stamped representation of the states. From this, a general way of looking at the buffer mechanism becomes to check whether it effectively enables the calculation of the delayed variable from information available d steps previously. If at least d previous levels of the behavior tree are stored and retrievable, the required information *is* available. The working assumption made here is that the delay of any variable remains constant throughout the simulation

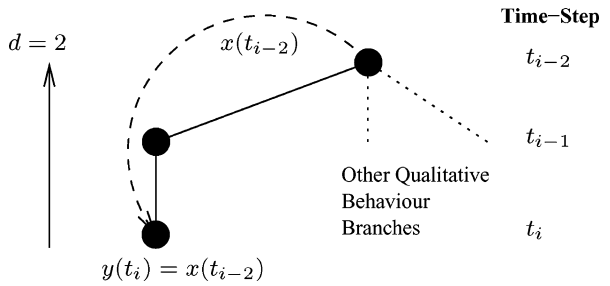

Fig. 3. Retrieving variable state information from previous levels of the behavior tree.

process, i.e., the system being modeled is time-invariant. This is a common presumption adopted by many modeling approaches since the behavior of physical systems can often be arbitrarily approximated by time-invariant descriptions. The delay of each variable can be specified precisely in the same way as for conventional nondelayed qualitative models. This may be done by abstracting existing numerical models (if available), or by any other knowledge acquisition techniques suitable for qualitative modeling [22].

In so doing, a delayed variable may be constrained by retrieving the states of the system variables involved from d levels higher up the same behavior branch. Fig. 3 shows this process for the example system. In order to constrain y, the algorithm searches up the behavior branch $d = 2$ levels to retrieve the value of x stored at that time-step in order to constrain y.

As per the buffer, the initial d steps of simulation must be treated as a special case for a variable delayed by d time-steps. To prevent a delayed variable from exhibiting a response to the other variables before sufficient time-steps have elapsed, it is constrained using the state information at t_0 for d time-steps. This is analagous to the prefilling of the buffer described above.

III. Extending FuSim

To implement the proposed improvements, changes must be made to the standard FuSim algorithm. These changes are detailed below.

A. Representation of Time Delay

Consider a simple system which is numerically represented as follows:

$$x(t_{i-2}) + y(t_i) = z(t_{i-2}).$$

This constraint conveys information about the current state of the delayed variable, y, as a function of the other two variables' states two sampling time-steps ago. To model this type of system, the *delay-indicator, delay(.)* is introduced such that the equivalent representation for FuSim is as follows, where nondelayed variables have an implicit delay indicator of 0:

$$x + y \cap z$$
$$\text{delay}(y) = 2.$$

The delay indicator is set with reference to a fixed time frame (e.g., the system internal clock or the sample rate in performing model-based reasoning tasks). FuSim recognizes when a constraint contains a delayed variable and, to constrain it, retrieves the state information for the other variables from the behavior tree as per Fig. 3. It is important to note that the state information of the other variables is retrieved for the sole purpose of constraining the delayed variable. The current states of variables x and z can only be expressed in terms of $y(t_{i+2})$, which at this stage is not yet determined. Therefore, FuSim enforces constraints backward in time from the most delayed variable(s).

The situation becomes more complex when multiple delayed variables are involved. Consider the following system, as represented numerically and in a fuzzy constraint format with appropriate delay indicators:

$$x(t_{i-1}) + y(t_i) = z(t_{i-2})$$
$$x + y \cap z$$
$$\text{delay}(x) = 1$$
$$\text{delay}(y) = 2$$

where x and y are delayed by one and two time-steps, respectively, with respect to the current state of z. Despite the differing amounts of delay present, this constraint conveys information about the *current* state of the most delayed variable, y, only. It can be satisfied by constraining y using variable state information retrieved from the appropriate level of the behavior tree. The level is computed on a variable by variable basis, by noting the difference in delay indicators between a particular variable and the most delayed variable.

Fig. 4 shows the interaction of the variables in this system, describing their state transitions at each time-step as new predictions are triggered. Note that i, the number of possible immediate next states, will differ for different simulation methods; for FuSim it is at most six (as determined by the transition rules which govern the generation of successor states from the current one, and which are derived from versions of the intermediate value and mean value theorems [8], [23]). Since FuSim enforces consistency backward in time, it is not possible to constrain the current states of variables x and z. Here, these variables proceed to all states allowed by the transition rules to maintain completeness. In a more complex system, it is likely that these variables would be involved in other constraints, and so would be further constrained. When the constraint is enforced back from the current state of y, any states of x and z that do not satisfy the constraint for at least one of the potential values of y at the current time-step will not be used to create potential system states. This process removes invalid values from further consideration as soon as possible.

It is now possible to present a method of supporting a system containing multiple delayed variables with differing amounts of delay. Given a constraint over a set V of variables, it is necessary to establish the set D of most-delayed constrained variables with a delay indicator of d for each of its elements, and the set $D' = V - D$, which contains variables delayed to a lesser extent (including those without any delay, i.e., a delay-indicator of 0). D is the set of variables whose current states are constrained by interaction with each other and with previous states of those variables in the set D'. In the case of the single constraint of the previous example, $V = \{x, y, z\}$, $D = \{y\}$ with $d = 2$ and $D' = \{x, z\}$.

The nondelayed version of FuSim is a special case of this; for each constraint, D' is empty (i.e. $V = D$) and $d = 0$. The variables are constrained as usual using current state transition information. This is also possible if $V = D$, but $d > 0$. This might be the case if two equally delayed variables had a specific relation to each other as well as to other system variables.

In the general case where D' is not empty for a given constraint, for each element $u \in D'$, retrieve state information from the level of the tree obtained by offsetting the current level by the difference between d and the delay indicator of u. Fig. 5 shows this process, where $\text{delay}(u') = 1$, $\text{delay}(u'') = 2$, and $\text{delay}(u''') = 4$.

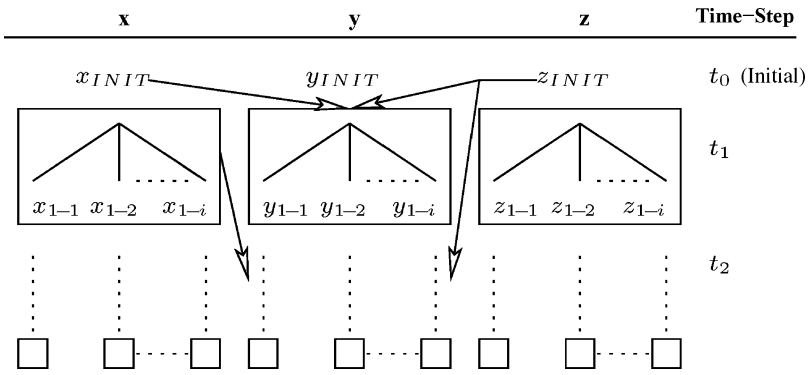

Fig. 4. Interaction between time-delayed variables.

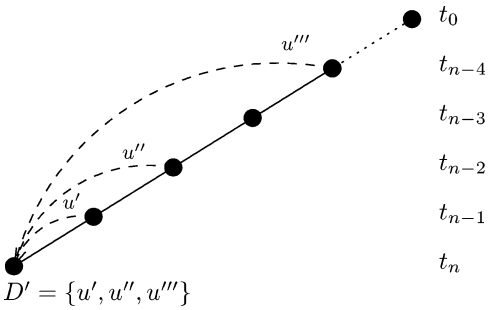

Fig. 5. Retrieving variable state information for the variables in D' from multiple levels of the behavior tree.

TABLE I
PARTIAL BEHAVIOUR, TIME-STEPS t_0 AND t_1

Time-step	Variable	Magnitude	Derivative
0	x	$P-Medium$	$Zero$
	y	$Zero$	$N-Medium$
	z	$N-Medium$	$Zero$
1	x	$P-Medium$	$Zero$
	y	$N-Small$	$N-Medium$
	z	$N-Medium$	$P-Small$

B. Extended Simulation Algorithm

To illustrate how the FuSim algorithm has been modified to support delayed variables, the following system will be used. (A summary of the updated algorithm will be given at the end of this section.) There are three system variables: x, y and z with the following constraints:

$$\dot{x} \cap y$$
$$\dot{y} \cap z$$
$$-x \cap z$$
$$\text{delay}(x) = 2.$$

The magnitude and derivative of each variable are taken from a fuzzy quantity space which consists of the following quantities: $\{N-\text{Top}, N-\text{Large}, N-\text{Medium}, N-\text{Small}, \text{Zero}, P-\text{Small}, P-\text{Medium}, P-\text{Large}, P-\text{Top}\}$. A partial behavioral description is presented in Table I.

Delayed variables must be treated differently according to whether a sufficient number of time-steps have elapsed for them to exhibit a reaction to the changes in the other variables. Throughout this *initial period*, a delayed variable is constrained via the state information at t_0.

The calculation of possible state transitions for each system variable is unchanged. Consider the example system at time-step t_1, as presented in Table I. The algorithm now computes system states for time-step t_2. First, the transition rules are used to produce the potential states for each variable, as presented in Table II.

TABLE II
POSSIBLE STATES FOR EACH VARIABLE AT TIME-STEP t_2

x {Mag, Deriv}
$\{P-Small, N-Small\}$
$\{P-Medium, P-Small\}$
$\{P-Medium, Zero\}$
$\{P-Medium, N-Small\}$
$\{P-Large, P-Small\}$
y {Mag, Deriv}
$\{N-Medium, N-Small\}$
$\{N-Medium, N-Medium\}$
$\{N-Medium, N-Large\}$
$\{N-Small, N-Small\}$
$\{N-Small, N-Medium\}$
$\{N-Small, N-Large\}$
z {Mag, Deriv}
$\{N-Small, P-Small\}$
$\{N-Small, Zero\}$
$\{N-Medium, P-Small\}$
$\{N-Medium, Zero\}$

The next stage is to construct a set of state-tuples for each constraint. Each set is generated from the cross-product of potential states (for the most delayed variables of D) and information retrieved from the behavior tree (for the less- and none-delayed variables in D'), depending on the particular constraint structure. Table III presents the tuples for constraints $\dot{x} \cap y$ and $-x \cap z$. Constraint $\dot{y} \cap z$ is omitted for clarity, since it begins with 24 tuples (with six possible states for y and four for z at the same time). Note how the state information for y and z is retrieved from time-step t_0 (see Table I) for the constraints presented in Table III, since x has a delayed response of two time-steps.

TABLE III
TUPLES GENERATED FOR TIME-STEP t_2, COMBINING POSSIBLE NEXT STATES AND RETRIEVED STATES

Constraint: $\dot{x} \cap y$	
x {Mag, Deriv} (t_2)	y {Mag, Deriv} (t_0)
$\{P-Small, N-Small\}$	$\{Zero, N-Medium\}$
$\{P-Medium, P-Small\}$	$\{Zero, N-Medium\}$
$\{P-Medium, Zero\}$	$\{Zero, N-Medium\}$
$\{P-Medium, N-Small\}$	$\{Zero, N-Medium\}$
$\{P-Large, P-Small\}$	$\{Zero, N-Medium\}$
Constraint: $-x \cap z$	
x {Mag, Deriv} (t_2)	z {Mag, Deriv} (t_0)
$\{P-Small, N-Small\}$	$\{N-Medium, Zero\}$
$\{P-Medium, P-Small\}$	$\{N-Medium, Zero\}$
$\{P-Medium, Zero\}$	$\{N-Medium, Zero\}$
$\{P-Medium, N-Small\}$	$\{N-Medium, Zero\}$
$\{P-Large, P-Small\}$	$\{N-Medium, Zero\}$

TABLE IV
TUPLES FOR TIME-STEP t_2 FOLLOWING SELF-CONSISTENCY FILTERING

Constraint: $\dot{x} \cap y$	
x {Mag, Deriv} (t_2)	y {Mag, Deriv} (t_0)
$\{P-Medium, Zero\}$	$\{Zero, N-Medium\}$
Constraint: $-x \cap z$	
x {Mag, Deriv} (t_2)	z {Mag, Deriv} (t_0)
$\{P-Medium, P-Small\}$	$\{N-Medium, Zero\}$
$\{P-Medium, Zero\}$	$\{N-Medium, Zero\}$
$\{P-Medium, N-Small\}$	$\{N-Medium, Zero\}$

TABLE V
TUPLES FOR TIME-STEP t_2 FOLLOWING PAIRWISE-CONSISTENCY FILTERING

Constraint: $\dot{x} \cap y$	
x {Mag, Deriv} (t_2)	y {Mag, Deriv} (t_0)
$\{P-Medium, Zero\}$	$\{Zero, N-Medium\}$
Constraint: $-x \cap z$	
x {Mag, Deriv} (t_2)	z {Mag, Deriv} (t_0)
$\{P-Medium, Zero\}$	$\{N-Medium, Zero\}$

Self-consistency filtering is applied as usual to all constraints: the tuples associated with each constraint are restricted to those which satisfy the constraint. Table IV shows the restricted tuples for each constraint. Again, constraint $\dot{y} \cap z$ is omitted for clarity. However, self-consistency filtering reduces the number of tuples for this constraint threefold, from 24 to 8.

Pairwise consistency filtering must be dealt with more carefully. It is designed to remove tuples that are inconsistent between pairs of constraints that share a common variable. If this process is applied to a mixture of delayed and nondelayed variables, inconsistent results will be obtained: the variable state information for the set D' is there solely to constrain the variables in the set D (of a given constraint). Since this information is retrieved from previous levels of the behavior tree, it is alien to the current state and should be ignored by a pairwise filter. Hence, the pairwise filter uses just the variables in the most delayed variables sets to make adjacency calculations (for shared variables) between constraints. With reference to the current example, the pairwise filter will only operate upon constraints $\dot{x} \cap y$ and $-x \cap z$, since they are adjacent via x. This is shown in Table V.

A similar situation arises when complete system states are generated from the remaining tuples. A system state consists of a combination of a potential current state of each variable, and so should not contain any variable state information retrieved from previous time-steps. In addition, as noted previously, a variable not in the most-delayed set D of any constraint may only be constrained backward at a subsequent time-step. Hence, system states are generated using a combination of the state information of variables in set D of each constraint and, for the remaining system variables, of the potential next states as computed using the transition rules (derived from assumed continuity and differentiability [8]). Table VI shows the potential next system states for the example system.

The simulation algorithm can proceed as normal from this point in applying global filters to further reduce the number of potential states. Since the extension proposed herein follows the original generate-and-test strategy in producing future states from the initial, only inconsistent states are removed. Hence, the original properties of FuSim such as soundness and completeness remain.

1) Algorithm Summary: In summary, the extended fuzzy qualitative simulation algorithm can be described as follows. Following a triggered prediction, for each valid system state the algorithm does.

1) **State Transition**. Generate potential next states using the original state transition rules.
2) **State Tuple Construction**. For each constraint, say constraint i:
   a) For each variable in the less-delayed set, D_i', retrieve the variable state from the appropriate level of the behavior tree. If the variables in the most-delayed variable set D_i are within their *initial period*, state information should be retrieved from level 0.
   b) Form the cross product of the retrieved state information with the possible next states of the variables in D_i.
3) **Self-Consistency Filtering**. Apply to the tuples associated with each constraint as usual.
4) **Pairwise-Consistency Filtering**. Apply between each pair of constraints j and k that share a variable, such that the shared variable is in the most delayed variable sets, D_j and D_k of the two constraints.
5) **System State Generation**. Combine variable state information in the most delayed variable set of each constraint with state transition information of any remaining system variables.
6) **Global Filtering**. Apply to each generated system state as usual.
7) **Iteration of the above for each remaining system state**.

2) Space Complexity: As noted in Section III-A, the maximum number of next states for a single variable is six in FuSim. The support of delayed variables does not change the branching factor. The extra cost stems from the need to store a number of levels of the tree in order to retrieve information to constrain delayed variables. Assuming the behavior tree is explored in a breadth-first manner, the space required will grow with the current search depth, i.e., the time-step. Given a maximum delay of d, and a system composed of n variables, the worst-case space required at time-step t is expressed as

$$\sum_{i=t-d}^{t-1} 6^{\text{in}}.$$

Depending on the system constraints, it is normally not necessary to store the entire system state, but only the states of those variables whose information needs to be retrieved to constrain delayed variables.

When used within a model-based reasoning task that synchronously tracks the real behavior at a constant sample rate, the complexity can be reduced drastically. This is because tracking, also referred to as measurement interpretation, is the process of using observations to follow a behavioral path through the behavior graph generated by the simulator [24]. For model-based applications such as control and diagnosis on which qualitative reasoning techniques are typically focused, synchronous tracking is a must [7], [25]–[27]. That is, comparisons between real observations and the simulated behavior must be made coherently at the same system state.

In particular, synchronous tracking works as follows, with the assumption that the system being modeled is deterministic and time-invariant (i.e., there is only one unique true behavior underlying the real system with respect to a given input): Treat the current observation $\mathrm{OBS}(t_0)$ as the initial state and the time t_0 that the observation is made as the initial temporal point for prediction. For the next observation, $\mathrm{OBS}(t_1)$, generate possible simulated behaviors from $\mathrm{OBS}(t_0)$ and compare these with $\mathrm{OBS}(t_1)$. Each new observation $\mathrm{OBS}(t_i)$, $i > 0$ *triggers* the simulator to produce further predictions, PREDS_{t_i}, from the currently matched state. A comparison is made between $\mathrm{OBS}(t_{i+1})$ and $\mathrm{PREDS}_{t_{i+1}}$. All behavior branches which do not match are terminated, thereby saving space. The model being used is still valid if at least one predicted behavior matches the current observation. Further simulation continues from the behaviors that contain the matched state.

TABLE VI
POTENTIAL SYSTEM STATES FOR TIME-STEP t_2

x {Mag, Deriv}	y {Mag, Deriv}	z {Mag, Deriv}
$\{P-Medium, Zero\}$	$\{N-Medium, N-Small\}$	$\{N-Small, Zero\}$
$\{P-Medium, Zero\}$	$\{N-Medium, N-Small\}$	$\{N-Small, P-Small\}$
$\{P-Medium, Zero\}$	$\{N-Medium, N-Medium\}$	$\{N-Medium, Zero\}$
$\{P-Medium, Zero\}$	$\{N-Medium, N-Medium\}$	$\{N-Medium, P-Small\}$
$\{P-Medium, Zero\}$	$\{N-Small, N-Small\}$	$\{N-Small, Zero\}$
$\{P-Medium, Zero\}$	$\{N-Small, N-Small\}$	$\{N-Small, P-Small\}$
$\{P-Medium, Zero\}$	$\{N-Small, N-Medium\}$	$\{N-Medium, Zero\}$
$\{P-Medium, Zero\}$	$\{N-Small, N-Medium\}$	$\{N-Medium, P-Small\}$

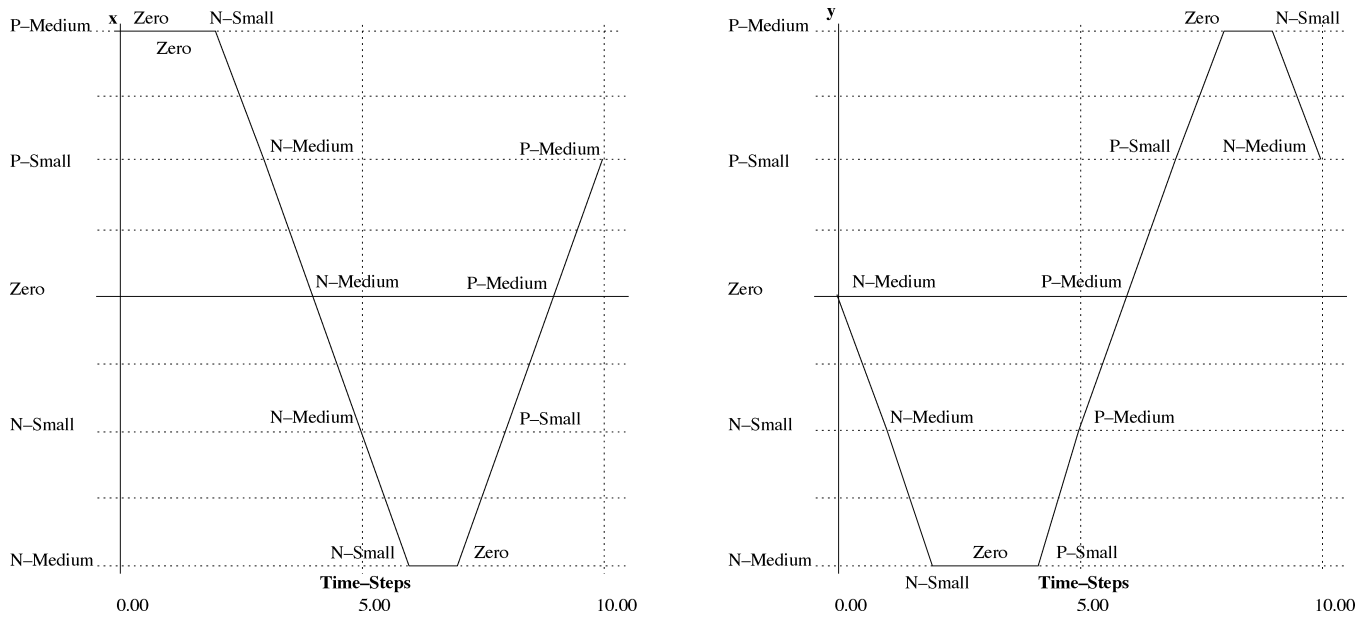

Fig. 6. Behavior pattern of the example system.

IV. EXPERIMENTAL RESULTS

The extended FuSim algorithm has been applied to a number of systems. As an example, this section will concentrate on discussing the results from a second-order system model which resembles a class of physical systems that involve variables which exhibit delayed periodic oscillation behavior. For instance, the same model can be used to represent an resistance–inductance–capacitance (*RLC*) circuit in electronic engineering and a coolant system in manufacture engineering [28].

The system model used extends that given in Section III-B by asserting that variable y also has a delayed response of one time-step. This gives the structural description

$$\begin{aligned} \dot{x} \cap y \\ \dot{y} \cap z \\ -x \cap z \\ \text{delay}(x) = 2 \\ \text{delay}(y) = 1. \end{aligned}$$

The initial system state is as presented in Table I (time-step t_0).

This system was simulated for ten sampling time-steps, starting from the initial state. At each time-step, a new prediction was triggered. A behavior of the delayed variables x and y is shown in Fig. 6. The behavior graph is annotated with the value of the derivative component.

Since x is delayed by two time-steps, it is initially constrained via the state information at t_0, forcing it to remain at $\{P-\text{Medium}, \text{Zero}\}$ during time-step t_1, as can be seen. At time-step t_2, x exhibits a reaction to y at t_1 and z at t_0. In the behavior pattern shown, x is constrained to equal $\{P-\text{Medium}, N-\text{Small}\}$, which satisfies the constraint $\dot{x} \cap y$, since the magnitude of y at time-step t_1 is $N-\text{Small}$, and $-x \cap z$, since the magnitude of z at t_0 is $N-\text{Medium}$. Over the simulated time-steps, x is constrained to a delayed periodic oscillation, compared to a similar system that does not exhibit time-delayed behavior as presented in Fig. 7.

The behavior of y is also presented in Fig. 6. Contrary to what might be expected, there is no valid behavior pattern where y remains at $\{\text{Zero}, N-\text{Medium}\}$ at t_1. This is because Zero is a fuzzy-real [8], and FuSim's state transition rules preclude a variable with a fuzzy-real magnitude from remaining unchanged unless its derivative component is also $Zero$. In this case, y is equal to $\{N-\text{Small}, N-\text{Medium}\}$ at t_1, satisfying constraint $\dot{y} \cap z$, since the magnitude of z is $N-\text{Medium}$ at t_0. Again, the behavior of y in Fig. 6 is clearly a delayed version of that presented in Fig. 7.

Note that a potential application scenario of this modeling and simulation example may be explained in the context of qualitative model-referenced systems control [29], where a qualitative model is required to predict, say, future behavior of a coolant system under regulation. The model used here and the qualitative behavior derived from it may then be interpreted as how the coolant loop is designed to function. That is, under normal conditions, the temperature of the coolant is designed to delay by one time-step in response to the flow rate of the coolant within the loop, which is itself also one-step delayed with respect to the setting of the power supply to the pump. If the model-based prediction is in conflict with the observation, certain control actions will be

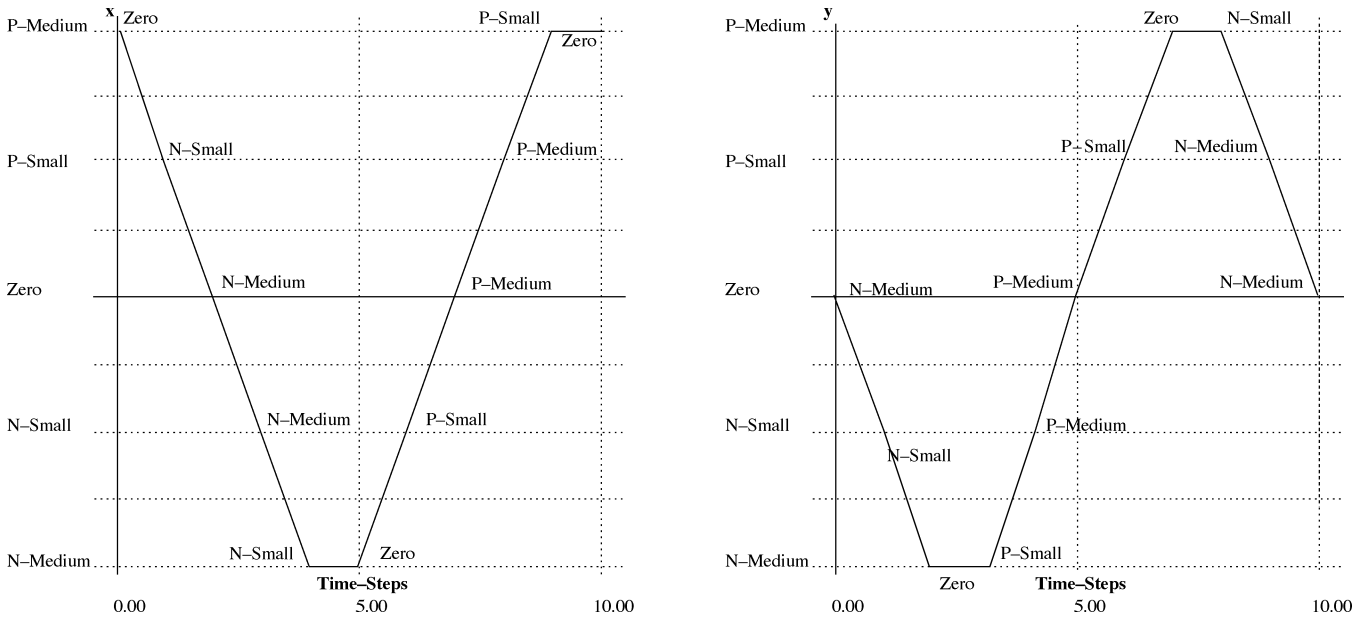

Fig. 7. Equivalent behavior pattern of example system with no delay.

taken to regulate the loop's behavior. As reflected by Figs. 6 and 7, the reference provided by the prediction that is generated by the present model differs significantly from that simulated by a nondelay model. Without the ability of simulating delayed systems, wrong control actions may therefore incur.

As noted in the previous section, the maximum branching factor of the behavior tree is unaffected by the presence of delay. However, the average branching factor does increase with the number of delayed variables, since the constraints involving the delayed variables are processed "backward" only. In the present example, this has the effect of increasing the total size of the behavior tree (assuming a breadth-first exploration) to thousands, rather than hundreds, of nodes. However, it is necessary to generate the behavior tree to depth eight to exhibit the delayed behavior due to the delayed response of x and y, whereas a depth of six is sufficient for the nondelayed version. Furthermore, as argued earlier, if the model simulated is to be used in performing a model-based reasoning task, this complexity can be significantly reduced.

V. Conclusion

The extension to FuSim presented in this paper is a novel approach to qualitatively simulating time-delayed behavior. The proposed work does not require explicit causal dependencies between system variables (as required by CA-EN [9]) and builds on the conventional generate and test simulation paradigm. It allows a qualitative model-based reasoning system to deal with system variables that exhibit different delays in their time-invariant responses to the constraining functions. The resulting algorithm works by constraining multiple delayed variables via retrieving state information from the behavior tree at a depth according to the magnitude of each delay.

The results to date have been very encouraging. It does remain, however, to apply the algorithm to a more varied and complex set of problems. It may be that for a large delay time and a complex structural description, storing such a large section of the behavior tree will become impractical. In this case, it would be useful to develop a scheme whereby only the variable states that are actually going to be retrieved (as opposed to the entire system state) are stored. However, this is largely an implementation issue, requiring little change in the modified FuSim algorithm itself.

Acknowledgment

The authors are grateful to J. Ponton and R. Banares-Alcantara of the School of Engineering and Electronics, University of Edinburgh, Edinburgh, U.K., for their contribution, whilst taking full responsibility for the views expressed in this paper. The authors would also like to thank the anonymous referees for their constructive comments which were very helpful in revising this paper.

References

[1] B. Williams and J. de Kleer, "Special volume on qualitative reasoning about physical systems II," *Artif. Intell.*, vol. 51, no. 1–3, pp. 1–473, 1991.

[2] D. Bobrow, "Special volume on qualitative reasoning about physical systems," *Artif. Intell.*, vol. 24, no. 1–3, pp. 1–491, 1984.

[3] B. Kuipers, *Qualitative Reasoning: Modeling and Simulation With Incomplete Knowledge*. Cambridge, MA: MIT Press, 1994.

[4] A. Capelo, L. Ironi, and S. Tentoni, "The need for qualitative reasoning in automated modeling: a case study," in *Proc. 10th Int. Workshop Qualitative Reasoning*, 1996, pp. 32–39.

[5] P. Mosterman and G. Biswas, "Diagnosis of continuous valued systems in transient operating systems," *IEEE Trans. Syst., Man, Cybern.*, vol. 29, no. 6, pp. 554–565, Jun. 1999.

[6] N. Snooke and C. Price, "Challenges for qualitative electrical reasoning in automotive circuit simulation," in *Proc. 11th Int. Workshop Qualitative Reasoning*, 1997, pp. 175–180.

[7] D. Berleant and B. Kuipers, "Qualitative and quantitative simulation: bridging the gap," *Artif. Intell.*, vol. 95, no. 2, pp. 215–255, 1997.

[8] Q. Shen and R. Leitch, "Fuzzy qualitative simulation," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 4, pp. 1038–1061, Apr. 1993.

[9] K. Bousson and L. Trave-Massuyes, "Fuzzy causal simulation in process engineering," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, 1993, pp. 1536–1541.

[10] I. Miguel and Q. Shen, "Extending qualitative modeling for simulation of time-delayed behavior," in *Proc. 12th Int. Workshop Qualitative Reasoning*, 1998, pp. 161–166.

[11] E. Cox, *The Fuzzy Systems Handbook: A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems*. New York: Academic, 1994.

[12] R. Leitch and Q. Shen, "Qualitativeness does not imply fuzziness," in *Proc. 3rd IEEE Int. Conf. Fuzzy Syst.*, 1994, pp. 1257–1262.

[13] F. Cellier, "General system problem solving paradigm for qualitative modeling, qualitative simulation modeling and analysis," in *Qualitative Simulation Modeling and Analysis*, P. Fishwick and P. Luker, Eds. New York: Springer-Verlag, 1991, pp. 51–71.

[14] F. Cellier, A. Nebot, F. Mugica, and A. de Albornoz, "Combined qualitative/quantitative simulation models of continuous-time processes using fuzzy inductive reasoning techniques," *Int. J. Gen. Syst.*, vol. 24, no. 1–2, pp. 95–116, 1996.
[15] P. Fishwick and P. Luker, *Qualitative Simulation Modeling and Analysis*. New York: Springer-Verlag, 1991.
[16] G. Klir, "Aspects of uncertainty in qualitative systems modeling," in *Qualitative Simulation Modeling and Analysis*, P. Fishwick and P. Luker, Eds. New York: Springer-Verlag, 1991, pp. 24–50.
[17] S. Case, Q. Shen, R. Banares-Alcantara, and J. Ponton, "Detecting inverse responses in chemical processes with qualitative simulation," in *Proc. 11th Int. Workshop Qualitative Reasoning*, 1997, pp. 249–255.
[18] P. Struss, "Mathematical aspects of qualitative reasoning," *Artif. Intell. Eng.*, vol. 3, pp. 156–169, 1988.
[19] I. Miguel and Q. Shen, "Solution techniques for constraint satisfaction problems (part 1: Foundations and part 2: Advanced techniques)," *Artif. Intell. Rev.*, vol. 15, no. 4, pp. 231–245, 269–293, 2001.
[20] P. Fouche and B. Kuipers, "Reasoning about energy in qualitative reasoning.," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 1, pp. 47–63, Jan. 1992.
[21] P. Struss, "Global filters for qualitative behaviors," in *Proc. 7th Nat. Conf. Artif. Intell.*, 1988, pp. 275–279.
[22] J. Keppens and Q. Shen, "On compositional modeling," *Knowledge Eng. Rev.*, vol. 16, no. 2, pp. 157–200, 2001.
[23] B. Williams, "The use of continuity in a qualitative physics," in *Proc. 3rd Nat. Conf. Artif. Intell.*, 1984, pp. 350–354.
[24] K. Forbus, "Interpreting measurements of physical systems," in *Proc. 5th Nat. Conf. Artif. Intell.*, 1986, pp. 113–117.
[25] D. Dvorak and B. Kuipers, "Process monitoring and diagnosis," *IEEE Expert*, vol. 6, no. 3, pp. 67–74, 1991.
[26] E. Manders, G. Biswas, P. Mosterman, L. Barford, and J. Barnett, "Signal interpretation for monitoring and diagnosis: a cooling system testbed," *IEEE Trans. Instrum. Meas.*, vol. 49, no. 3, pp. 503–509, Mar. 2000.
[27] Q. Shen and R. Leitch, "Diagnosing continuous systems with qualitative dynamic models," *Artif. Intell. Eng.*, vol. 9, no. 2, pp. 107–125, 1995.
[28] R. Rosenberg and D. Karnopp, *Introduction to Physical System Dynamics*. New York: McGraw-Hill, 1983.
[29] U. Keller, "Qualitative model reference adaptive control (QMRAC)," Ph.D. dissertation, Dept. Comput. Elect. Eng., Heriot Watt Univ., Edinburgh, UK, 1999.

Multiagent Immediate Incremental View Maintenance for Data Warehouses

Gary C. H. Yeung and William A. Gruver

***Abstract*—Data warehouse systems typically designate downtime for view maintenance, ranging from tens of minutes to hours depending on the system size. In this paper, we develop a multiagent system that achieves immediate incremental view maintenance (IIVM) for continuous updating of data warehouse views. We describe an IIVM system that processes updates as transactions are executed at the underlying data sources to eliminate view maintenance downtime for the data warehouse—a crucial requirement for internet applications. The use of a multiagent framework provides considerable process speed improvement when compared with other IIVM systems. Since agents are used to delegate the data sources and warehouse views, it is easy to reorganize the components of the system. Through the use of cooperative agents, the data consistency of IIVM can be easily maintained. The test results from this research show that the proposed system increases the availability of the data warehouse while preserving a stringent requirement on data consistency.**

***Index Terms*—Data warehouse, multiagent, view maintenance.**

I. Introduction

View maintenance requires updating the materialized views in a database system as changes are made to the underlying data. View maintenance is well understood in conventional transaction database systems. Most prior research in view maintenance has been concerned with deferred incremental view maintenance (DIVM), in which updates to the underlying data are logged and applied to modify the materialized views collectively during system downtime. DIVM, the primary view maintenance method adopted by most commercial data warehousing products, assumes that the data warehouse has a convenient system downtime. For data warehouses that provide global access, however, downtime may not be acceptable. The time required for view maintenance requirement is a major limitation on the size of a data warehouse.

Data warehouse views may also be updated by the use of immediate incremental view maintenance (IIVM). In this technique, changes to the underlying data are applied immediately and individually to the materialized views. Potential data inconsistency due to asynchronous messaging, however, constrains its usage in commercial systems. In this study, we propose a multiagent framework for performing IIVM in data warehousing with parallel processing.

Hanson [1] conducted one of the earliest studies of DIVM in which differential tables were maintained on base tables that contain the suspended updates that have not been applied to the database state. Colby [2] applied base logs and differential tables for periodic update of views, a concept that is similar to taking snapshots from every state of change in the base tables. Mumick [3] improved Colby's method by storing changes of base tables in a Summary-Delta table in which the information is updated to the summary tables during off-hours. Gupta [4] proposed a counting algorithm to keep track on the order of updates for each tuple in a view so that updates can be applied at the correct data warehouse states. Hull [5] decomposed an integrated view

Manuscript received December 3, 2001; revised December 2, 2003 and March 17, 2004. This paper was recommended by Associate Editor Y. Pan.

The authors are with the Intelligent Robotics and Manufacturing Systems Laboratory, School of Engineering Science, Simon Fraser University, Burnaby, V5A 1S6 BC, Canada (e-mail: gruver@cs.sfu.ca).

Digital Object Identifier 10.1109/TSMCA.2005.843385