Extending Qualitative Modelling for Simulation of Time-Delayed Behaviour

Ian Miguel and **Qiang Shen**
Department of Artificial Intelligence
University of Edinburgh, UK
{ianm, qiangs}@dai.ed.ac.uk

Abstract

This paper presents an extension to qualitative simulation that enables a QR system to support variables that exhibit delayed reactions to their constraining functions. Through a process of synchronised tracking, information stored in the previous levels of the behaviour tree can be retrieved and used to constrain multiple delayed variables. It is demonstrated that the extended simulation mechanism is powerful enough to capture the time-delay behaviour observed in actual physical systems. Experimental testing is described and results are provided.

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 behaviour 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.

The problem is to determine how a time-delayed system will behave subject to a certain initial condition, 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 discretised such that regular points along it are simulated numerically. However, this introduces two problems: first, how to guarantee that all possible behaviour is simulated if the process exhibits non-linear characteristics, and second, how to interpret the behaviour at the boundaries between neighbouring intervals.

Qualitative reasoning (QR) has already proven successful in modelling complex processes where a numerical solution is difficult or infeasible to obtain, (e.g. (Snooke & Price 1997), (Capelo *et al* 1996)). Here, the system behaviours are simulated non-numerically, with each variable taking *symbolic* values. In most cases, system descriptions are not required to be complete. Current QR research concentrates on simulating time-invariant behaviour, however. Existing qualitative simulation methods are unable to deal with a system that exhibits a time-delay, such as $y(t) = f(x(t - \Delta T))$.

This paper shows how a qualitative simulation algorithm may be extended to support such systems. The behaviour simulator employed to predict a system's behaviour must produce (at least estimates of) the durations associated with respective system states, enabling a synchronous tracking of the evolution of the observations with the simulated behaviours. This prohibits the use of traditional qualitative simulation algorithms as they do not provide such durations. Advances in qualitative simulation allow temporal information to be computed with the generation of system states (e.g. (Shen & Leitch 1993), (Berleant & Kuipers 1997)). Although developed with the fuzzy qualitative simulation algorithm (FuSim) (Shen & Leitch 1993) in mind, the extension should be generally applicable to simulators that enable synchronous tracking.

The rest of this paper is arranged as follows. The next section gives an overview of the FuSim algorithm as a basis for the proposed improvements. It also highlights the data structure, the behaviour tree, that is fundamental to this approach to qualitative time-delay simulation. Section 3 describes exactly how FuSim may be updated to support time-delayed system variables, detailing the changes that must be made to the algorithm. Section 4 provides experimental results and analysis. Finally, section 5 concludes the paper and describes important future work.

FuSim and Time Delays

The FuSim algorithm is a relative of QSIM (Kuipers 1994). It uses a structural description of the physical system defined by a set of constraints between the system variables. In place of strict (fuzzy) constraints, which each involve at most three arguments, the following constraint format is used (Case *et al.* 1997):

$$LHS \cap RHS$$

Both sides of the intersection may be arbitrarily complex, removing the requirement for several strict fuzzy constraints connected by pseudo-variables to represent a single more complex constraint. In this case, the constraint is satisfied if there is a (fuzzy) intersection between the left- and right-hand sides.

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 discretise the representation of system variables, as opposed to the alternating point and interval method used by QSIM. Both the magnitude and the rate of change range over an arbitrarily discretised fuzzy quantity space.

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 behaviours 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 (Miguel & Shen 1998). This involves two sub-steps: 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 re-combining the remaining tuples, and are in turn restricted by the application of global filters such as energy conservation (Fouche & Kuipers 1992). 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 behaviour tree: each node corresponds to a single system state, and each branch corresponds to a distinct possible behaviour pattern for the system.

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:

$$y(t) = x(t-2)$$

Clearly, the value of y depends on that of x from two time-steps ago. A typical method for modelling 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 figure 1.

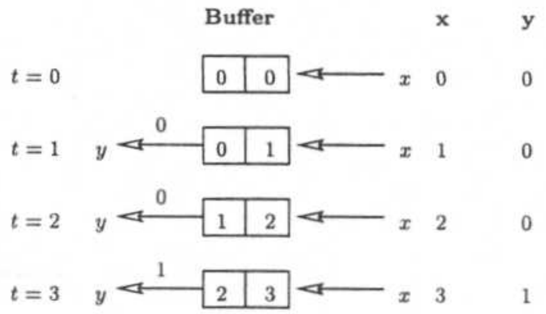

Figure 1: The Buffer Mechanism

Initially, the buffer is filled with the current value of y so that any change does not filter through to it 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. It should be clear that, 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 behaviour tree is generated, FuSim must process multiple possible behaviour patterns due to qualitative ambiguities, creating a much more complex scenario (though FuSim will produce a unique behaviour for the above extremely simple case).

It is possible, nevertheless, to use information stored in the behaviour tree to implement a qualitative time-delay mechanism if the simulator is to be used within a model-based reasoning task that works by synchronously tracking the real behaviour, where explicit observation sampling time-steps are available. This requirement is not as restrictive as it sounds. For model-based reasoning applications such as control and diagnosis, synchronous tracking is a must. Comparisons between real observations and the simulated behaviour must be made coherently at the same system state. Given a matched system state, its originally estimated temporal duration can be reset to the underlying sampling time (typically represented using a sampling time-step count) at which the state matched the observation. The following presentation will assume that the time-steps associated with a simulated behaviour are those reset to reflect the observation time indices.

A general way of looking at the buffer mechanism is to see that it is effectively enables the calculation of the delayed variable from information available d steps previously. If at least d previous levels of the behaviour tree are stored and retrievable, the required information *is* available. Hence, a delayed variable may be constrained by retrieving the states of the system variables involved from d levels higher up the same behaviour branch. Figure 2 shows this process for the example system. In order to constrain y, the algorithm searches up the behaviour branch $d = 2$ levels to retrieve the value of x stored at that time-step in order to constrain y.

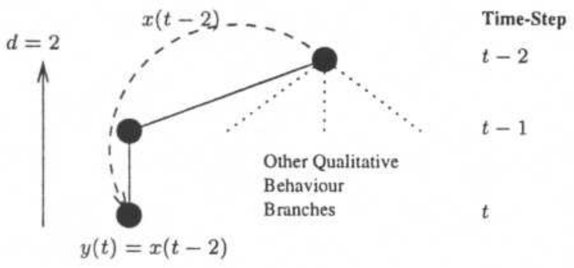

Figure 2: Retrieving Variable State Information from Previous Levels of the Behaviour Tree

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 exhibiting a response before sufficient time-steps have elapsed, its state must be held constant for d time-steps.

Extending FuSim

To implement the proposed improvements, changes must be made to the standard FuSim algorithm. These changes, detailed below, implement a unified framework to deal with both delayed and non-delayed variables.

Representation of Time Delay

Consider a simple system which is numerically represented as follows:

$$x(t-2) + y(t) = z(t-2)$$

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 non-delayed variables have an implicit delay indicator of 0.

$$\begin{aligned} x + y \quad &\cap \quad z \\ delay(y) \quad &= \quad 2 \end{aligned}$$

Hence, FuSim recognises when a constraint contains a delayed variable and, to constrain it, retrieves the state information for the other variables from the behaviour tree as per figure 2. 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 $y(t+2)$, which at this stage is not yet determined. Therefore FuSim enforces constraints backwards 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:

$$x(t-1) + y(t) = z(t-2)$$

$$\begin{aligned} x + y \quad &\cap \quad z \\ delay(x) \quad &= \quad 1 \\ delay(y) \quad &= \quad 2 \end{aligned}$$

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 behaviour 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.

Figure 3 shows the interaction of the variables in this system, describing their state transitions at each time-step. Note that i will differ for different simulation methods; for FuSim it is at most 6. Since FuSim enforces consistency backwards 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.

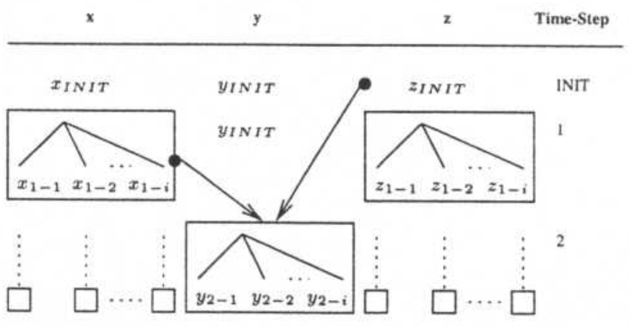
x
y
z
Time-Step
x_{INIT}
y_{INIT}
z_{INIT}
INIT
y_{INIT}
1
x_{1-1} x_{1-2} x_{1-i}
z_{1-1} z_{1-2} z_{1-i}
2
y_{2-1} y_{2-2} y_{2-i}

Figure 3: Interaction between Time-Delayed Variables

It is now possible to present a unified 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 with a delay-indicator of

Variable	Magnitude	Derivative
X	$Zero$	$Zero$
Y	$Zero$	$Zero$

Table 1: Example: Initial System State

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 non-delayed 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. Figure 4 shows this process, where $delay(u_1) = 1$, $delay(u_2) = 2$, and $delay(u_3) = 4$.

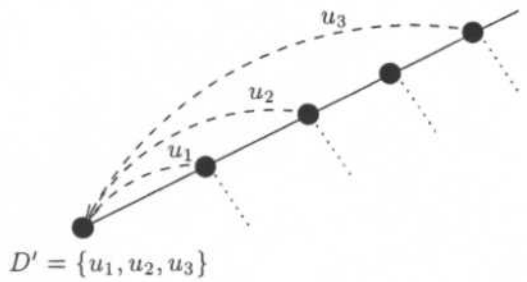

Figure 4: Retrieving Variable State Information for the Variables in D' from Multiple Levels of the Behaviour Tree

The Extended Simulation Algorithm

To illustrate how the underlying FuSim algorithm may be changed 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 two system variables: x, and y (the delayed variable) with the following structural description. The initial state of this system is shown in table 1.

$$\begin{aligned} \dot{x} &\cap x \\ y &\cap x \\ \dot{y} &\cap \dot{x} \\ delay(y) &= 2 \end{aligned}$$

At each stage of the algorithm, delayed variables must be treated differently according to whether a sufficient number of time-steps have elapsed for them to exhibit a change from their initial state. Throughout

Variable	Magnitude	Derivative
X	$P-Small$	$P-Small$
Y	$Zero$	$Zero$

Table 2: Example: Current State Following Time-Step 1

X {Mag, Deriv}	Y {Mag, Deriv}
$\{P-Medium, P-Medium\}$	$\{Zero, Zero\}$
$\{P-Medium, P-Small\}$	$\{P-Small, P-Small\}$
$\{P-Medium, Zero\}$	
$\{P-Small, P-Medium\}$	
$\{P-Small, P-Small\}$	
$\{P-Small, Zero\}$	

Table 3: Example: Possible Next States for Each Variable Following Time-Step 1

this *initial period*, the state of a delayed variable is held constant at its initial position. Hence, in the example system, y is held constant at $\{Zero, Zero\}$ during time-steps 0 and 1. The Process of tuple generation and constraint-filtering is unnecessary for a constraint whose most-delayed variable set, D, has yet to pass through this initial period. This is because the states of these variables are in effect strictly constrained to remain the same. The following assumes that all delayed variables have passed through their initial periods.

The calculation of possible state transitions for each system variable is unchanged. Consider the example system at time-step 1, where the algorithm is in the process of computing system states for time-step 2: the current state is as shown in table 2, and y is to have passed out of its initial period. The transition rules produce the potential next states for each variable as shown in table 3. For simplicity, the magnitude of both variables is restricted to the quantity $Zero$ and above.

The next stage is to construct a set of state-tuples for each constraint. Each set will typically be generated from a mixture of potential next states (for the most delayed variables of D) and information retrieved from the behaviour tree (for the less delayed variables of D'), depending on the particular constraint structure. In this example, since the first constraint $\dot{x} \cap x$ constrains just the variable x, it has associated with it just the possible next states of x, as shown in table 3 above. Table 4 shows those tuples associated with constraints $y \cap x$ and $\dot{y} \cap \dot{x}$. These two constraints have an identical tuple list initially, since they constrain the same variables. The state of x is not taken from the set of its possible next states, but is retrieved from the behaviour tree. In this case, the variable x had the state $\{Zero, Zero\}$ at time-step 0, as given in table 1.

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 5 shows the restricted tuples for each constraint. Pairwise consistency filtering must be dealt with more carefully. It is designed to remove tuples that are inconsistent

X {Mag, Deriv} (t=0)	Y {Mag, Deriv} (t=2)
{Zero,Zero}	*{P − Small,P − Small}*
{Zero,Zero}	*{Zero,Zero}*

Table 4: Example: Tuples Generated for both Constraints: $y \cap x$ and $\dot{y} \cap \dot{x}$

Constraint	X {Mag, Deriv}	Y {Mag, Deriv}
$\dot{x} \cap x$	*{P − Small, P − Small}* *{P − Medium, P − Medium}*	
$y \cap x$	*{Zero,Zero}*	*{Zero,Zero}*
$\dot{y} \cap \dot{x}$	*{Zero,Zero}*	*{Zero,Zero}*

Table 5: Example: Tuples after Self-Consistency Filtering

between pairs of constraints that share a common variable. If this process is applied to a mixture of delayed and non-delayed 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 behaviour 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 table 5 for the current example, the pairwise filter will only operate upon constraints two and three, since they are adjacent via y.

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 backwards 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. Table 6 shows the potential next system states for the example system. The simulation algorithm can proceed as normal from this point in applying global filters and processing the remaining system states.

In summary, the extended fuzzy qualitative simulation algorithm can be described as follows. Given an initial system state, the algorithm does:

X {Mag, Deriv}	Y {Mag, Deriv}
{P − Small,P − Small}	*{Zero,Zero}*
{P − Medium,P − Medium}	*{Zero,Zero}*

Table 6: Example: Potential Next System States

- **State Transition.** Hold constant any variable still within its *initial period*. For each other variable, generate potential next states using the original transition rules.
- **State Tuple Construction.** For each constraint, say constraint i:
  - If the variables in the most-delayed variable set D_i are within their *initial period*, no tuples need to be generated (variables in set D_i are constrained to remain the same).
  - Else, generate a set of n-tuples, where n is the number of all the variables involved in the constraint. To do this, for each variable in the less-delayed set, D'_i, retrieve the variable state from the appropriate level of the behaviour tree and form the cross product with the possible next states of the variables in D_i.
- **Self-Consistency Filtering.** Apply to the tuples associated with each constraint as usual.
- **Pairwise-Consistency Filtering.** Apply between pairs 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.
- **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.
- **Global Filtering.** Apply to each generated system state as usual.
- **Iteration of the above for each remaining system state.**

Experimental Results

The extended FuSim algorithm has been applied to a number of systems. Due to space limitations, this section will concentrate on discussing the results from a second-order system which resembles a class of physical systems that exhibit delayed periodic oscillation behaviour. For instance, the same model can be used to represent a RLC circuit in electronic engineering and a coolant system in manufacture engineering.

Three variables are used to simulate the system: x, y, and z. The reaction of x to y and z is delayed by three time-steps. The magnitude and derivative of each variable are taken from a fuzzy quantity space which consists of the following quantities: $\{N-Top,\ N-Large,\ N-Medium,\ N-Small,\ Zero,\ P-Small,\ P-Medium,\ P-Large,\ P-Top\}$. The initial system state is shown in table 7. The structural description of the system is composed of three constraints, as well as the delay indicator for x.

$$\begin{aligned} \dot{x} &\cap y \\ \dot{y} &\cap z \\ z &\cap -x \\ delay(x) &= 3 \end{aligned}$$

Variable	Magnitude	Derivative
X	$P - Medium$	$Zero$
Y	$Zero$	$N - Medium$
Z	$N - Medium$	$Zero$

Table 7: Initial State of the Example System

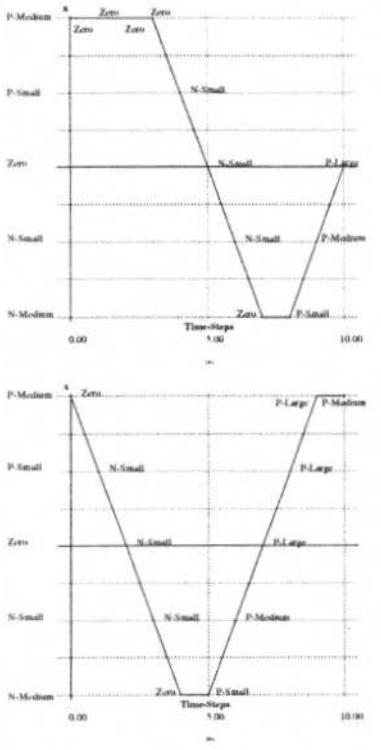

Figure 5: Delayed vs. Non-Delayed Behaviour Patterns

This system was simulated for a total of ten sampling time-steps, starting from the initial state. The behaviour of the delayed variable, x, is shown in figure 5.a. The behaviour graph is annotated with the value of the derivative component at each time-step. Since x is delayed by three time-steps, it is held constant at $\{P - Medium, Zero\}$ during this initial period, as can be seen. At time-step 3, sufficient time has elapsed for x to exhibit a reaction to y and z at $t = 0$. It is therefore constrained to remain equal to $\{P - Medium, Zero\}$. This is because $P - Medium = -(N - Medium)$, where $N - Medium$ is the magnitude of z at $t = 0$, and $Zero$ is the magnitude of y at $t = 0$, thus satisfying the constraint. In the ensuing time-steps, it can clearly be seen that x is constrained to a delayed periodic oscillation, compared to a similar system that does not exhibit time-delayed behaviour as shown in figure 5.b.

Conclusion and Future Work

The extension to FuSim outlined in this paper is a novel approach to qualitatively simulating time-delayed behaviour when used in a process of synchronous tracking. It allows a QR system to deal with system variables that exhibit different delays in their responses to their constraining functions. The algorithm works by constraining multiple delayed variables via retrieving state information from the behaviour 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 and to examine whether this approach possesses the properties of soundness and completeness (though there has not been any indication thus far that such properties may be lost due to the present extension). It may be that for a large delay time and a complex structural description, storing such a large section of the behaviour 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.

Acknowledgement: The authors are grateful to Professor Jack Ponton and Dr. Rene Banares-Alcantara, of the Department of Chemical Engineering, University of Edinburgh, for their contribution, whilst taking full responsibility for the views expressed in this paper.

References

Berleant, D., and Kuipers, B. 1997. Qualitative and quantitative simulation: Bridging the gap. *Artificial Intelligence* 95:215–255.

Capelo, A.; Ironi, L.; and Tentoni, S. 1996. The need for qualitative reasoning in automate modeling: A case study. *Proceedings of the 10th International Workshop on Qualitative Reasoning* 32–39.

Case, S.; Shen, Q.; Banares-Alcantara, R.; and Ponton, J. 1997. Detecting inverse responses in chemical processes with qualitative simulation. *Proceedings of the 11th International Workshop on Qualitative Reasoning* 249–255.

Fouche, P., and Kuipers, B. 1992. Reasoning about energy in qualitative reasoning. *IEEE Transactions on Systems, Man, and Cybernetics* 22(1):47–63.

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

Miguel, I., and Shen, Q. 1998. Solution techniques for constraint satisfaction problems. *Submitted for Journal Publication.*

Shen, Q., and Leitch, R. 1993. Fuzzy qualitative simulation. *IEEE Transactions on Systems, Man, and Cybernetics* 23(4):1038–1061.

Snooke, N., and Price, C. 1997. Challenges for qualitative electrical reasoning in automotive circuit simulation. *Proceedings of the 11th International Workshop on Qualitative Reasoning* 175–180.