In *The Eleventh International Workshop on Qualitative Reasoning (QR-97)*
Cortona, Italy, June 1997.

# Dynamic Chatter Abstraction: A scalable technique for avoiding irrelevant distinctions during qualitative simulation

**Daniel J. Clancy and Benjamin Kuipers**
Department of Computer Sciences
University of Texas at Austin, Texas 78712
clancy@cs.utexas.edu and kuipers@cs.utexas.edu

## Abstract

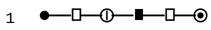
One of the major factors hindering the use of qualitative simulation techniques to reason about the behavior of complex dynamical systems is intractable branching due to a phenomenon called chatter. Chatter occurs when a variable's direction of change is constrained only by continuity within a region of the state space. This results in intractable, potentially infinite branching within the behavioral description due to irrelevant distinctions in the direction of change. Dynamic chatter abstraction provides a general purpose, scalable solution that abstracts chattering regions of the state space into a single state within the behavioral description. Chattering regions are identified via a dynamic analysis of the model and the current qualitative state using knowledge of the inference capability of the simulation algorithm.

The algorithm is described along with an empirical evaluation that compares dynamic chatter abstraction to previous solutions and demonstrates that it eliminates all instances of chatter without over-abstracting. This technique is used to simulate models that previously could not be simulated. Eliminating the problem of chatter significantly extends the range of problems that can be addressed using qualitative simulation techniques.

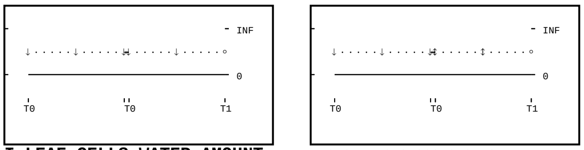
## Introduction

A variety of techniques have been developed that utilize qualitative simulation to reason about the behavior of an imprecisely defined dynamical system to perform tasks such as monitoring, diagnosis and design (Weld and de Kleer 1990). Applying these techniques to complex, real–world systems, however, is often hindered by the complexity of the simulation and the behavioral representation generated. Traditionally, qualitative simulation (De Kleer and Brown 1984; Forbus 1984; Kuipers 1994) is performed at a single level of detail highlighting a fixed set of distinctions. At times, some of these distinctions may be irrelevant to the current task and thus needlessly increase the complexity of the simulation. One of the major sources of irrelevant distinctions is a phenomenon called *chatter* (Kuipers 1994). Chatter occurs when the derivative of a variable is constrained only by continuity within a region of the state space. Within the simulation, the variable is free to alternate between increasing, steady, and decreasing with individual behaviors generated for different orderings of these values. The distinctions between these behaviors provide no additional information; however, they result in intractable branching and a potentially infinite simulation. The severity of the problem depends upon the number of unrelated variables chattering within a particular region of the state space. As the size of a model increases, the number of chattering variables often grows as the model becomes more loosely constrained. The elimination of chatter from a qualitative simulation increases the range of models that can be tractably simulated thus allowing qualitative simulation to be applied to tasks such as monitoring, diagnosis, design and tutoring (see figure 1).

While a number of techniques have been proposed to eliminate chatter, chatter box abstraction (Clancy and Kuipers 1993; 1997a) is the only one that provides a general solution that can eliminate all instances of chatter. Chatter box abstraction explores the potentially chattering region of the state space via a recursive call to the simulation algorithm that is restricted to the chattering region of the state space The complexity of this simulation, however, is exponential in the number of chattering variables. *Dynamic chatter abstraction* provides a scalable solution that avoids the need to recursively call the simulation algorithm by identifying the chattering variables through a dynamic analysis of the model and the current state. Dynamic chatter abstraction provides a more focused search of the potentially chattering region of the state space by exploiting knowledge about the type of inferences that the simulation algorithm can make. This paper presents the dynamic chatter abstraction algorithm and provides an empirical evaluation with respect to chatter box abstraction.




(a) Behavior tree.




I.LEAF-CELLS.WATER.AMOUNT I.STOMATES.CROSS-SECTIONAL-AREA

**(b) Behavior plots.**

Rickel and Porter (1994) use qualitative simulation to answer prediction questions within the domain of plant physiology. The *TRIPEL* algorithm automatically generates a qualitative model from a large–scale botany knowledge base (Porter et. al. 1988) in response to a user query. Answers are generated based upon the results of the simulation. Many of their models, however, exhibit a great deal of chatter. Thus, their system requires an efficient technique to automatically eliminate chatter.

- In this example, the following question is presented to the system:
  "What happens to the stomates when the leaves lose water?"
- Simulation of the simplest model generated by TRIPEL results in a single behavior (a) in which the cross sectional area of the stomates decreases as the leaves loose water (b).
- Dynamic chatter abstraction identifies a total of 7 chattering variables. States that exhibit chatter are represented by a box within the behavior tree and a double arrow in the behavior plots. Note that following the second time–point, the cross sectional area of the stomates begins to chatter. This is because the model generated is unable to represent relevant order of magnitude information after this point. The information provided up to this point, however, is sufficient to answer the question in the scenario tested.
- Without some form of automated chatter elimination, this model results in an intractable simulation that generates hundreds of behaviors and the query cannot be answered. While chatter box abstraction can be used to simulate this model, dynamic chatter abstraction offers a factor of 10 speed up in simulation time. For some of the models generated by TRIPEL, chatter box abstraction is simply unable to complete the simulation due to resource limitations.

Figure 1: Using qualitative simulation to answer prediction questions.

## Chatter and Qualitative Simulation

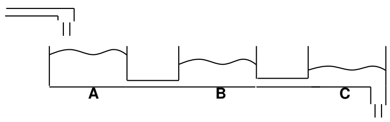
Dynamic chatter abstraction has been developed as an extension to the the QSIM simulation algorithm (Kuipers 1994). QSIM describes the behavior of a dynamical system via a tree of alternating time–interval and time–point states called a *behavior tree.* Each state specifies a qualitative magnitude (qmag) and a direction of change (qdir) for each variable within the model. The direction of change corresponds to the sign of the variable's derivative and can be increasing (`inc`), decreasing (`dec`) or steady (`std`). The qualitative magnitude is either a landmark value or an interval between two landmarks defined upon a totally ordered quantity space of landmark values. Landmarks may be defined during the simulation to represent critical points identified within the simulation. A branch occurs within the behavioral description whenever there is insufficient information within the model to identify a unique successor to a given state. Two types of branches can occur:

**Definition 1 (Event branch)**
*An* event *occurs when a variable reaches a landmark or becomes steady (i.e. its derivative becomes zero). An* event branch *occurs when there are multiple events following a time–interval state whose ordering is unconstrained by the model.*

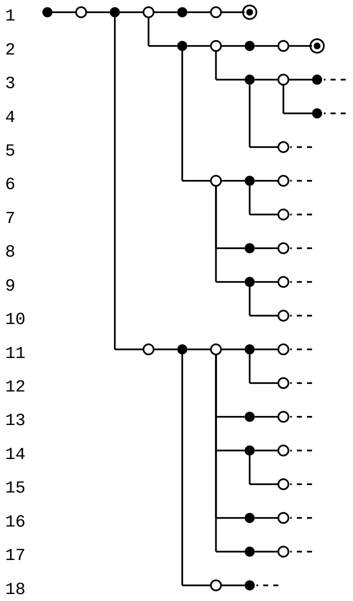
**Definition 2 (Chatter branch)** *A* chatter branch *occurs following time–point $t_i$ if there exists a variable $v$ that is currently steady whose direction of change is constrained only by continuity. A three way branch occurs depending upon whether the variable is increasing, decreasing or steady in the interval $(t_i, t_{i+1})$ (see figure 2).*

All branches within a qualitative simulation can be classified in one of these two categories. Event branches often represent relevant qualitative distinctions. Chatter branching, however, provides no additional information to the behavioral description and thus can be eliminated. The model decomposition and simulation (DecSIM) algorithm (Clancy and Kuipers 1997b) reduces irrelevant event branching by partitioning the model into loosely coupled components and simulating the components independently to eliminate distinctions between unrelated variables.

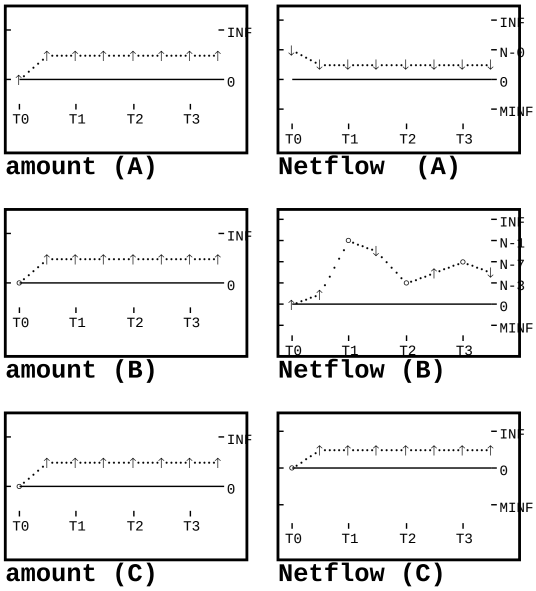
Chatter branching is problematic due to the repetitive nature of the phenomenon. Since the variable's direction of change is unconstrained, the variable is free to become steady again after the chatter branch and the process repeats itself (see figure 2). While some behaviors exit the unconstrained region of the state space, others will continue cycling between different values for the direction of change for an arbi-



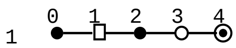

(a) **W-tube**



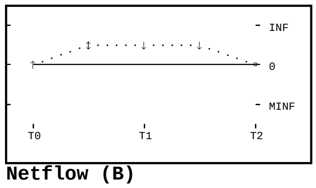

(b) **Behavior tree exhibiting chatter.**




(c) **Behavior plots exhibiting chatter.**




(d) Behavior tree using dynamic chatter abstraction



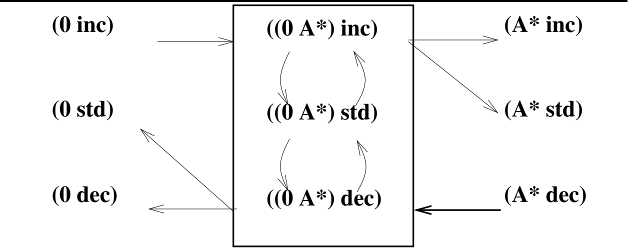

(e) Dynamic chatter abstraction behavior plot.

- In a qualitative model of three tanks arranged in sequence connected by tubes (a), *NetflowB(t)* = *InflowB(t)* - *OutflowB(t)*. The derivative of `NetflowB` is constrained only by continuity within the time–interval following the initial state.

- The simulation branches on all possible trajectories of *NetflowB(t)* while all other variables are completely uniform. The simulation is infinite in nature due to chatter and must be halted at an arbitrary state limit. A single behavior (c) from the behavior tree (b) is displayed demonstrating the unconstrained movement of *NetflowB(t)*. Higher order derivative information is used to eliminate chatter in a number of variables, however, it is unable to eliminate chatter in `NetflowB`. Ignore qdirs over-abstracts and generates an unconstrained behavior tree.

- Dynamic chatter abstraction generates a single behavior by abstracting the chattering region of the state space into a single qualitative state (d & e). If higher order derivative information is not used, a total of 7 variables exhibit chatter. With higher order derivative information only `NetflowB` chatters.

Figure 3: Intractable branching due to chatter in the simulation of a W tube.




- Qualitative value transitions consistent with continuity within the closed interval `(0 A*)` are identified by arcs within the figure.
- The boxed area denotes the chattering region of the state space (i.e. a chatter box). Once the system enters this region of the state space, the chattering variable can continue to cycle within the box.
- If landmarks are introduced, the simulation can remain within the boxed region for an infinite number of states since additional qualitative distinctions are introduced whenever the variable becomes steady.

Figure 2: Possible qualitative value transitions for a QSIM variable.

trary number of qualitative states resulting in an infinite simulation. While one variable chatters, relevant distinctions may occur in the qualitative value of other variables within the model. The irrelevant distinctions resulting from chatter branching must be eliminated while retaining the distinctions in other non–chattering variables. Figure 3 demonstrates the affects of chatter when simulating a model of three connected tanks.

Chatter branching is particularly difficult to eliminate due to the propagation of chatter through the model. Once one variable, $v_1$, begins to chatter and its derivative changes sign, it is possible that the derivative of another variable, $v_2$, that is related to $v_1$ will also become unconstrained, so $v_2$ begins to chatter. This process can repeat itself resulting in a large number of chattering variables within one region of the state space called a *chatter box*.

**Definition 3 (Chatter box)** *For a model with a set of variables $V$, a region of the state space is defined as a* chatter box *with respect to a set of variables $V_c$ if the derivatives of variables in $V_c$ are unconstrained with respect to variables in $V - V_c$.*

A state exits a chatter box when a non–chattering variable changes value or when a chattering variable changes qualitative magnitude and enters into a constrained region of the state space. It is possible for two chatter boxes to be adjacent if a non–chattering variable changes value while another variable continues to chatter.

The phenomenon of chatter becomes more complicated if a landmark exists within the unconstrained region of the state space and the magnitude of the variable is unconstrained around this landmark. This phenomenon is called *landmark chatter* and it results in changes in both the magnitude and the direction of change for the chattering variable. A special case of landmark chatter, called *chatter around zero*, occurs when a variable and its derivative are represented explicitly within a model and both of them exhibit chatter. In this case, the derivative variable will chatter around zero as its integral chatters.

While chattering behaviors may describe a real behavior of the system (Kuipers et. al. 1991), a disjunctive enumeration of all possible combinations of values provides no information. Furthermore, this enumeration obscures other distinctions within the description and can result in an infinite simulation. Eliminating this source of distinctions is essential if qualitative simulation is to be used to reason about complex dynamical systems.

## Previous Solutions

Three previous methods have been developed for eliminating chatter within a QSIM behavior tree simulation. The *higher order derivative* (HOD) (Kuipers et. al. 1991) technique uses the second and third order derivatives of the chattering variables to determine a unique direction of change for unconstrained variables and eliminate spurious behaviors. When possible, higher–order derivatives expressions are derived via algebraic manipulation of the constraints; otherwise, the operator must specify these expressions within the model. This technique, however, is not effective when an ambiguous evaluation of the HOD expression results or when an expression cannot be derived. In addition, the automatic derivation of an HOD expression places additional assumptions on the monotonic functions described by $M^+$ and $M^-$ constraints.

*Ignore qdirs* (Fouché and Kuipers, 1991) eliminates chatter by ignoring distinctions in a variable's direction of change throughout the simulation. This technique requires the modeler to identify the chattering variables prior to the simulation. For many models, a variable's direction of change is ambiguous only within certain regions of the state space. In other regions of the state space, direction of change information may be relevant in constraining the behavior of the system. Thus, ignore qdirs may result in over–abstraction and introduce spurious behaviors. In addition, ignore qdirs may prevent the application HOD constraints in other variables.

*Chatter box abstraction* (Clancy and Kuipers 1993; 1997a) was the first general solution to completely solve the problem of chatter. Chatter box abstraction explores the potentially chattering region of the state space via a recursive call to the simulation algorithm called a *focused envisionment.* The results of the envisionment are analyzed to identify chattering variables and the chattering region of the state space is abstracted into a single qualitative state within the main behavioral description. Successors to this abstract state are identified through an analysis of the focused envisionment graph. By recursively calling the simulation algorithm, chatter box abstraction exploits the inference capabilities already contained within the algorithm. This facilitates the integration of chatter box abstraction with other extensions to the simulation algorithm and lends itself to a straight–forward proof of the ability of chatter box abstraction to eliminate all instances of chatter without over–abstracting. The algorithm, however, explores the *entire* chattering region of the state space via simulation. The number of states within this region is exponential in the number of chattering variables. Thus, the solution does not scale up as the size of a model grows and the number of unconstrained variables increases.

DeCoste (DeCoste 1994) addresses chatter when simulating Qualitative Process Theory models (Forbus 1984) simply by ignoring distinctions in the direction of change when a unique value cannot be inferred via constraint propagation. He does not address the problem of landmark chatter nor does he provide an evaluation of this approach. In addition he does not discuss cases where constraint propagation is too weak of an inference mechanism to infer a unique direction of change.

## Identifying the set of chattering variables

For each time–interval state $S$, to determine if a variable $v$ can chatter before a non–chattering distinction occurs two questions must be answered:

**Consistency** - Is there a consistent state in which $v$ is free to chatter?

**Reachability** - Can this state be reached from $S$ only through changes occuring in other chattering variables? Such a state is called *chatter–reachable.*

# The Dynamic Chatter Abstraction Algorithm

Dynamic chatter abstraction uses an abstraction techniques similar to chatter box abstraction; however, as opposed to exploring the chattering region via simulation, it uses an understanding of the restrictions that are asserted by each constraint within the model along with the current qualitative state to determine if the derivative of a variable is constrained. For example, in the W-tube model the direction of change for `NetflowB` is restricted only by the constraint `NetflowB` + `Flow-BC` = `Flow-AB`. In the time–interval following the initial state, all three variables are increasing. Thus, in this state the qdir of `NetflowB` is unconstrained and `NetflowB` is free to chatter.

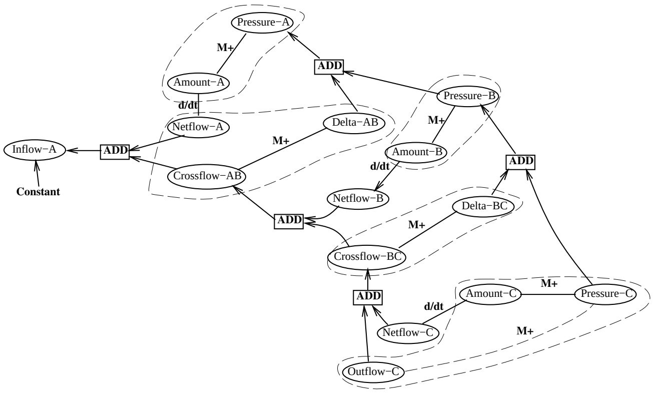
Dynamic chatter abstraction, however, must reason not only about the qualitative values contained within the current state, but also about how these values change as variables begin to chatter. Once `NetflowB` chatters, its derivative can change sign and the above addition constraint no longer restricts the derivative of `Flow-AB`. If `Flow-AB` is not prevented from chattering by other constraints, it is free to chatter and must be identified as such within the current chatter box.[1]

Once the chattering variables are identified, an abstract, time–interval state describing the chattering region of the state space is created and inserted into the behavior tree. Successors of this abstract state are computed through an extension of the QSIM successor generation algorithm.

The conditions under which a variable can chatter with respect to the current state are defined using a *chatter dependency graph.* Two types of nodes exist within the chatter dependency graph: equivalency nodes and dependency nodes. An *equivalency node* is created for each set of chatter equivalent variables within the model. The variables within the model are partitioned into chatter equivalency classes using an extension of an algorithm presented in (Kuipers et. al. 1991) (see figure 4). Equivalency nodes are connected through a directed, AND-OR subgraph of intermediate *dependency nodes.* The AND-OR subgraph can be viewed as a predicate that specifies the conditions under which the variables in an equivalency node are free to chatter. The variables within a node chatter within the current region of the state space if and only if there exists a consistent, chatter-reachable state that satisfies the predicate specified by the and–or subgraph.

**Building the chatter dependency graph** The conditions specified within the and–or subgraph are defined with respect to changes that must occur in the current qualitative state. For example, in the W-tube model `NetflowB` + `Flow-BC` = `Flow-AB` and all three variables are increasing following the initial state. Thus, in the current state `Flow-AB` is restricted from chattering; however, if either the deriva-

[1] Higher order derivative constraints can be applied to restrict chatter in all of the variables except `NetflowB`. While presenting the algorithm, this information is not used.




Constraints within the model are used to partition variables into chatter equivalency classes such that if one variable within a class chatters, then all of the variables will chatter.

- The figure provides a graphical representation of the W-tube model. Binary constraints are represented by labeled arcs while multi-variate constraints are represented by nodes.
- The dotted lines identify the chatter equivalence classes.
- Two variables are included within the same chatter equivalence class if they are the only two non-constant variables within a constraint. Thus, variables related by monotonic function constraints are contained within the same class and `NetflowA` and `Crossflow-AB` are also included within the same class since `Inflow-A` is constant.

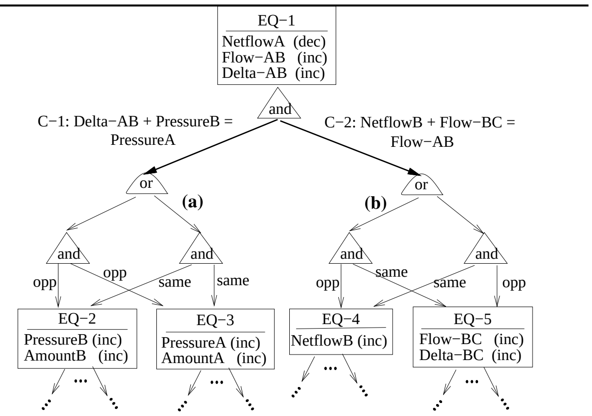
Figure 4: Chatter equivalency classes.

tive of `NetflowB` changes sign while `Flow-BC` remains the same or vice-a-versa, then `Flow-AB` is free to chatter. Figure 5 describes the structure of the AND-OR subgraph using the W-tube example.

To construct the AND-OR subgraph extending from an equivalency node $EQ_{source}$, each constraint containing a variable within $EQ_{source}$ is analyzed to determine the conditions under which the constraint fails to restrict the direction of change of the variables within $EQ_{source}$. These conditions are specified via labeled arcs within the AND-OR subgraph. Table 1 describes the possible label types.

**Evaluating the dependency graph** The dependency graph evaluation algorithm categorizes chatter equivalency classes as `chattering`, `non-chattering`, or `chatter-unknown`. The algorithm iterates through the equivalency classes moving them from the `chatter-unknown` category into one of the other two categories. Certain chatter equivalency classes can be identified as non–chattering throughout the simulation. For example, if one of the variables is constrained by a `constant` constraint, then none of the variables in the equivalency class can chatter.

To determine if the variables within an equivalency node $EQ$ will chatter, the algorithm attempts to form




For each time–interval state, a chatter dependency graph is created. The qdir for each variable in the state being analyzed is displayed within the node. The AND-OR subgraph extending from a node $EQ_{source}$ has the following structure:

- The root of the sub-graph is an `AND` node that has a link pointing to an `OR` node for each constraint relating a variable within $EQ_{source}$ to other variables within the model that can potentially restrict a variable from chattering. In the W-tube, there are two such constraints for $EQ_1$.
- Each `OR` node corresponds to a constraint $C$. A child is created for each set of qualitative value assignments for the variables within $C$ that allows the variables in $EQ_{source}$ to chatter.
- The children of these final conjunctive nodes are the equivalency nodes for the other variables within $C$. The arcs are labeled with information about how the qualitative values of these variables must change if the variables in $EQ_{source}$ are to chatter.
- For example, $C_1$ restricts the variables in $EQ_1$ from chattering if and only if the qdirs in $EQ_2$ and $EQ_3$ either both remain the same or both change. Note that the labels are dependent on the placement of a variable within a constraint.

Figure 5: Chatter dependency graph for the W-tube

a state that satisfies all of the conditions within the and–or subgraph extending from $EQ$. It begins by instantiated a partial state description with the current qualitative magnitude and direction of change for the variables within the node and all variables currently classified as `non-chattering`. This state is also instantiated with qualitative magnitude information for variables that cannot chatter around zero.[2] Then a backtracking algorithm is used in an attempt to identify a chatter–reachable state that satisfies the AND-OR sub-graph extending from $EQ$. Information is added to the partial state when another equivalency

[2] It is assumed that only variables whose integral is represented within the model will chatter around zero. If other variables chatter around zero, it is likely that the model is highly unconstrained.

node is encountered. The information added depends upon the label of the link pointing to the node. If the search encounters an equivalency node $EQ'$ whose direction of change is required to change (i.e. the arc is labeled with `opp`), then $EQ'$ must be classified as `chattering` before the required information can be added to the partial state description. This ensures that the state is chatter–reachable. The evaluation algorithm is called recursively if $EQ'$ is still classified as `chatter-unknown`. A cycle within the recursive calling sequence is treated as a dead end condition and the algorithm backtracks. A cycle between nodes $EQ$ and $EQ'$ means that along the path traversed, $EQ$ depends upon $EQ'$ to chatter first while $EQ'$ depends upon $EQ$ to chatter first. Thus, neither one can chatter[3]. This condition, however, does not necessarily prevent these nodes from chattering since there might be another set of qualitative value assignments (i.e. another path in the dependency graph) that allows one of these nodes to chatter. Once one chatters, the other may be free to chatter as well.

If a partial state is created satisfying the AND-OR subgraph, the QSIM constraint satisfaction algorithm is used to ensure that the variable assignments are consistent with all of the constraints within the model. If they are, then the equivalency node is classified as `chattering`. If a chatter–reachable, consistent state cannot be identified, then the equivalency node is classified as `non-chattering`.

**W-tube example** Figure 6 describes the evaluation of a portion of the dependency graph for the W-tube example. Without using HOD information, evaluation of the entire dependency graph results in a total of 7 chattering variables. This is consistent with the results of a standard QSIM simulation without any form of chatter elimination.

An extension to the dynamic chatter abstraction algorithm has been developed to handle higher order derivative constraints. These constraints must be handled differently since they are structured slightly differently than the standard QSIM constraints. When this extension is applied, the only variable that chatters is `NetflowB`.

### Abstract state creation and successor generation

Once the set of chattering variables is identified, the algorithm creates an abstract state with an abstracted qdir of (`inc std dec`) for each of the chattering variables providing a conjunctive list of values for the qdir

[3] Remember that the `opp` label requires variables in the destination node to change sign *before* the source node can chatter.

within the abstracted time–interval. The QSIM state successor algorithm has been extended to handle such abstract states.

If the qualitative value of a variable over a time–interval is $\langle (l_j \ l_{j+1}) \ (inc \ std \ dec)\rangle$, then at the ensuing time–point state the following values are consistent with continuity:

$$\langle (l_j l_{j+1})(inc\ std\ dec)\rangle$$
$$\langle l_j (std\ dec)\rangle$$
$$\langle l_{j+1} (inc\ std)\rangle$$

Thus, a time–point state can also have an abstracted qdir. In addition, an algorithm has been developed to test each time–point state to determine when a variable stops chattering. This occurs when a related variable changes and begins to constrain the chattering variable. When this occurs, a single direction of change is identified from the lists provided above.

### Landmark chatter

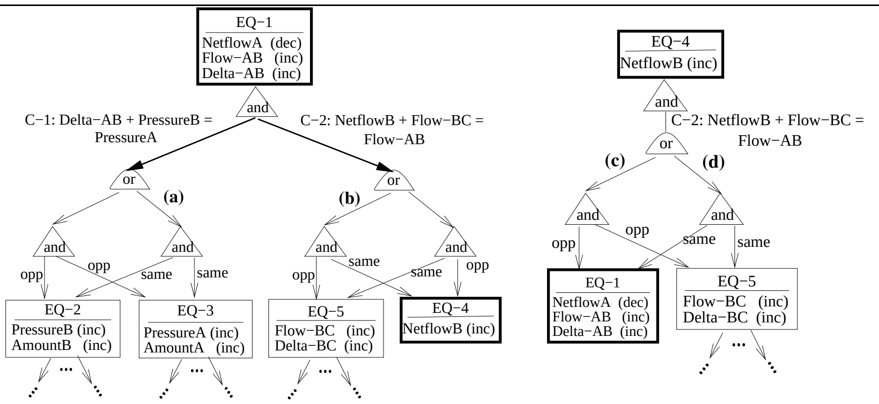
Dynamic chatter abstraction is able to handle chatter around zero without any additional processing. The algorithm described above does not assume that the derivative constraint prevents the integral variable from chattering. Instead, information about the constraining power of the derivative constraint is represented via labels within the chatter dependency graph. For example, if a variable's derivative is explicitly represented within the model, then both the `qmag` and the `qdir` of the derivative variable must be unconstrained and the `qdir` of the derivative variable must begin to head toward zero. These restrictions are specified within the chatter dependency graph. If both a variable and its derivative are identified as chattering, then the derivative variable will exhibit chatter around zero and its qualitative magnitude is also abstracted.

### Complexity

The size of the chatter dependency graph is polynomial in the number of chatter equivalency classes. Since the evaluation algorithm is performing constraint satisfaction, we expect that the worst case complexity of the algorithm may be exponential, however, the average case complexity is much better. In fact, we have not encountered an exponential time factor in any of the models tested (see table 2). This is because the algorithm performs a directed search of the potentially chattering region simply to identify variables as chattering or non–chattering as opposed to enumerating all possible solutions within this search space. Conflicts are often encountered well before an entire path is traversed within the search space. A detailed complexity analysis of the dynamic chatter abstraction algorithm still must be performed.

| Edge Label | Description |
|---|---|
| **(same)** | The qdirs of the variables within *node* must remain the same. |
| **(opposite)** | The qdirs of the variables within *node* must change to the opposite sign. For the qdirs to change, *node* must be classified as a chattering node. |
| **(chatter)** | The variables within *node* must be free to chatter. This link does not place specific constraints on the values of the node variable's qdirs. This label is used when processing the `D/DT` constraint. |
| **(qmag <var> <lmark>)** | The qualitative magnitudes of the variables within *node* must be unconstrained around *lmark*. |
| **(qdir <var> <qdir>)** | The qdir of *var* must have the value *qdir*. *Qdir* must be different from the current value and thus *var* must be free to chatter. This link is required when a variable is steady at the time-interval state being evaluated or has an abstracted qdir. |

Table 1: Dependency graph edge labels




Equivalency node $EQ_1$ is evaluated as follows for the time–interval state $S_1$ following the initial state. The dependency and–or subgraph for $EQ_1$ displayed in figure 5 is redisplayed here. To simplify the structure of the display, nodes $EQ_4$ and $EQ_1$ are displayed in two places in the above graph. (The order of the traversal of the and–or subgraphs has been slightly modified for the sake of this presentation.)

**Evaluating** $EQ_1$

*Step-1* Instantiate a partial state description $S_p$ with the qualitative values of the variables in $EQ_1$ and `InflowA` since it is constant. Push $EQ_1$ on a stack maintaining a list of visited nodes called $VN$.

*Step-2* To satisfy constraint $C_1$ traverse link (a). This link requires the variables in $EQ_2$ and $EQ_3$ to remain the same. Add this information to $S_p$.

*Step-3* To satisfy constraint $C_2$, traverse link (b). This link requires the variables in $EQ_4$ to move in the `opposite` direction. Therefore, the variables in $EQ_4$ must be free to chatter. Since this node is classified as `chatter-unknown`, call the algorithm recursively to determine the status of $EQ_4$.

**Evaluating** $EQ_4$

*Step-4* Traverse link (c) in the above graph. To satisfy this link, both nodes must be free to chatter. A cycle, however, occurs in the calling sequence since $EQ_1 \in VN$. Therefore, the algorithm backtracks.

*Step-5* Traverse link (d). Since this link is satisfied by the current state, classify $EQ_4$ as chattering and return (i.e. `NetflowB` will chatter.)

*Step-6* $EQ_4$ is therefore free to chatter. Augment $S_p$ with the qualitative value information to satisfy link (b).

*Step-7* The QSIM state completion algorithm is called to ensure that there exists a consistent completion of $S_c$. Variables that have not been identified as chattering must retain their current qualitative values within this completion. A state is identified and $EQ_1$ is classified as chattering.

Figure 6: Dependency graph evaluation for the W-tube.

## Evaluation and Discussion

Dynamic chatter abstraction has been empirically evaluated using a corpus of over 20 models obtained from various researchers within the field of qualitative reasoning. The results were validated against a standard QSIM simulation to show that the algorithm identifies a variable as chattering if and only if it exhibits chatter within the same region of the state space in the standard simulation. Table 2 compares dynamic chatter abstraction with chatter box abstraction on a few of these models with respect to the time required to perform the abstraction.[4] In all of the models tested, dynamic chatter abstraction performed better than chatter box abstraction, and in fact, as the number of chattering variables increased, the speed up provided by dynamic chatter abstraction increased significantly. In addition, a number of improvements are are being implemented to increase the efficiency of the dynamic chatter abstraction. In particular, a propagation phase will be added prior to the dependency graph evaluation to reduce the complexity of this step.

Dynamic chatter abstraction provides a scalable solution that is able to efficiently solve the problem of chatter. However, since the algorithm incorporates information about how the simulation algorithm processes the constraints within the model, extensions to the simulation algorithm may require modifications to the dynamic chatter abstraction algorithm. On the other hand, chatter box abstraction uses the basic simulation algorithm as its main inference engine. As a result, extensions to the simulation algorithm can be seamlessly integrated. Thus, both algorithms offer advantages.

## Conclusions

Recent discussions within the field of qualitative reasoning have questioned the usefulness of qualitative *simulation* when reasoning about the behavior of complex dynamical systems. In general, these discussions have focused on the complexity of the simulation and the difficulty of scaling up to larger, more complex systems due to irrelevant distinctions. Chatter is a major source of these distinctions. The technique presented in this paper solves this problem. This technique in combination with the DecSIM component simulation algorithm (Clancy and Kuipers 1997b) significantly extends the range of models that can be tractably simulated using qualitative simulation and thus supports the application of qualitative simulation to tasks such as monitoring, diagnosis and design for complex, real–world systems.

[4]Some of the examples used result in an infinite simulation and thus were terminated at an arbitrary state limit.

## Acknowledgments

This work has taken place in the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the QR Group is supported in part by NSF grants IRI-9504138 and CDA 9617327, by NASA grants NAG 2-994 and NAG 9-898, and by the Texas Advanced Research Program under grant no. 003658-242. Source code is included in the latest QSIM release available at **http://www.cs.utexas.edu/users/qr/**.

## References

D. J. Clancy and B. J. Kuipers. Behavior Abstraction for Tractable Simulation in *Proceedings of the Seventh International Workshop on Qualitative Reasoning about Physical Systems*, pp. 57-64, 1993.

D. J. Clancy and B. Kuipers. Static and dynamic abstraction solves the problem of chatter in qualitative simulation In *Proceedings from the Fourteenth National Conference on Artificial Intelligence, AAAI-97*, Providence, RI, July 1997.

D. J. Clancy and B. Kuipers Model Decomposition and Simulation: A component based qualitative simulation algorithm. Submitted for publication.

D. DeCoste. Goal–directed qualitative reasoning with partial states. Technical Report 57, The Institute for the Learning Sciences, University of Illinois at Urbana–Champaign, August 1994.

J. De Kleer and J. S. Brown. A Qualitative Physics Based on Confluences. In *Artificial Intelligence* 24:7-83, 1984.

K. D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

P. Fouché and B. J. Kuipers. Towards a Unified Framework for Qualitative Simulation. In *Proceedings of the Fifth International Workshop on Qualitative Reasoning about Physical Systems*, 295-301, 1991.

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

B. J. Kuipers, C. Chiu, D. Dalle Molle and D. R. Throop. Higher-order derivative constraints in qualitative simulation. *Artificial Intelligence*, 51:343-379, 1991.

B. Porter, J. Lester, K. Murray, K. Pittman, A. Souther, L. Acker, and T. Jones. AI research in the context of a multi-functional knowledge base: The botany knowledge base project. Technical Report AI88-88, University of Texas at Austin, 1988.

J. Rickel and B. Porter. Automated Modeling of Complex Systems to Answering Prediction Questions To appear in *Artificial Intelligence*, 1997.

D. Weld and J. de Kleer. *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufman, Los Altos, CA, 1990.

| **Model** | **Vars** | **Chat Vars** | **Number of Behaviors** No chatter abstraction: **Envisionment** | **Behavior tree** (no lms) | Chatter abstraction: **Behavior tree** (w/ lms) | **Simulation time (sec)** **Dynamic Chatter** | **Chatter Box** |
|---|---|---|---|---|---|---|---|
| W Tube | 16 | 1 | 3 | $> 1845$ | 1 | 0.9 | 3.4 |
| Glucose-insulin Interaction | 11 | 2 | 155 | $> 2807$ | 41 | 14 | 52 |
| Van der Pol Equation$^A$ | 10 | 4 | 43 | $> 1788$ | $12^B$ | 2.3 | 15 |
| Controlled Hot/Cold tank$^A$ | 14 | 5 | 24 | $> 601^C$ | 14 | 5.0 | 48.2 |
| Turgor Stomates | 19 | 7 | 509 | $> 2598$ | 1 | 2.4 | 20.5 |
| Cooling Plant$^A$ | 15 | 10 | 659 | $> 4095$ | 1 | 7.7 | 418 |
| Heart Model | 42 | 28 | 689 | $> 2784$ | 200 | 100 | $C$ |

$A$ – Exhibits chatter around zero.

$B$ – Oscillatory behavior results in infinite behavior tree. Simulation terminated once the structure within the tree could be determined.

$C$ – Could not be simulated to completion due to resource limitations.

- Each model was tested both with and without chatter abstraction. When chatter abstraction was not used, both an envisionment and a behavior tree simulation without landmark introduction were used. The number of behaviors generated are listed above using a state limit of 5000. Note that while an envisionment often results in a smaller number of behaviors, the total number of qualitatively distinct behaviors represented is the same as there are an infinite number of paths within the envisionment graph.
- Both types of abstraction generated the same number of behaviors; however, dynamic chatter abstraction performed significantly better than chatter box abstraction with respect to simulation time.
- HOD constraints were used whenever applicable. Ignore qdirs does not work on models exhibiting chatter around zero. On other models ignore qdirs can be used, however, it requires a significant amount of work by the modeler to identify which variables are chattering.

Table 2: Evaluation of Dynamic Chatter and Chatter Box Abstraction