JMorven: A Framework for Parallel Non-Constructive Qualitative Reasoning and Fuzzy Interval Simulation.

Allan M. Bruce

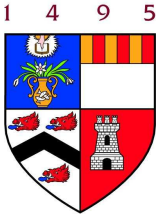
A dissertation submitted in fulfilment

of the requirements for the degree of

Doctor of Philosophy

of the

University of Aberdeen.



Department of Computing Science

2007

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2007

Abstract

The work defended in this thesis introduces a novel approach to constraint-centred qualitative reasoning in a non-constructive manner. Non-constructive approaches have many advantages including the fact that they do not require models to be causally ordered and can therefore reason with systems which contain algebraic loops. This new approach combines reasoning on the spectrum from fully qualitative to fully quantitative. In addition to this, all underlying algorithms have been implemented in parallel to decrease execution times.

Previous work into parallel qualitative reasoning showed that execution time decreased over sequential implementations however the work had two main disadvantages. First, not all stages of execution were implemented in parallel therefore the design was not optimal and secondly, the implementation was presented in the form of a dedicated hardware architecture.

Several methods exist to reason with intervals or fuzzy numbers however no non-constructive approach offers results which are sound and complete. A new qualitative reasoner, named JMorven, was implemented completely from scratch to overcome the limitations described above. JMorven is the successor to Morven but presents a novel set of algorithms working non-constructively and has an abstract parallel architecture which allows it to execute faster when run in distributed computing environments. The novel work presented and tested in this thesis consists of:

1. A novel portable parallel architecture allowing speed-up of every stage of execution.

2. The use of auxiliary variables in a non-constructive environment.

3. More accurate simulation using n-th order Taylor Series.

4. Incorporating several non-constructive fuzzy interval simulation techniques.

5. A method of simulating fuzzy intervals non-constructively which is asymptotically sound and complete.

6. Offering the ability to carry out numerical simulations non-constructively from a qualitative model description.

7. A single non-constructive simulation engine which is capable of reasoning on the spectrum from fully qualitative to fully quantitative.

Acknowledgements

I would especially like to thank my supervisor, Dr. George M. Coghill for giving me the opportunity to conduct my research in this very interesting field, for his guidance and encouragement throughout the duration of my PhD. I would also like to thank my colleagues, Dr. Honghai Liu, Tao You, Hind Al-Ballaa, Mehdi Khoury and the members of the ModKin project at the University of Wales, Aberystwyth for inspiring conversations and many interesting thoughts.

I would like to thank the members of the QR community whom I have interacted with at conferences and workshops, in particular to Dr. Bernhard Rinner for providing his PhD thesis to me and discussing the project.

I am particularly grateful to my parents for their support and encouragement both emotionally and financially. I could not have completed the work without their support. I would also like to thank Daryll, Jim, Kate and my Grandparents for their support and encouragement.

A special thanks to my wife, Catherine, for supporting me throughout all of my student years and putting up with hours of coding and debugging.

A final word of thanks is due to all of my friends for giving me a non-working environment so that I could relax when I needed to. One particular mention to Douglas Currie for his technical expertise from a Software Engineer's point of view.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The work completed for this thesis involves developing methods for non-constructive simulation in a parallel manner for a novel qualitative reasoning inference engine and implementing the ideas in a system completely written from scratch.

Non-constructive simulation approaches can be thought of as generate-and-test methods which involves determining all possible combinations of behaviours and discarding those that are inconsistent. This differs from constructive approaches which require a strict model structure as they construct successor values for system variables and use these to update all remaining variables. Non-constructive techniques offer several advantages over constructive methods. They are more general as they do not impose a strict structure on the input models; models do not need to be causally ordered which allows algebraic loops within models to be reasoned with. Chapter 2.4 discusses the differences between the two approaches in detail.

Parallel computing techniques allow execution times to be decreased dramatically when multiple processing units are available. This is beneficial to qualitative reasoning since

many implementations run slowly due to the large number of possible behaviours produced. Implementing core algorithms in parallel allows the use of multiple processors to carry out the calculations in parallel thus speeding up the execution time of qualitative simulation. Chapter 3 discusses the advantages of parallel computing and methods to determine how efficient parallel algorithms are.

1.1 Motivation

The background context of the project was the development a model-based planner that could be used in harmful environments based on GraphPlan (Blum and Furst, 1997). It became apparent early on that there was not a suitable implementation of a qualitative reasoner that could be used for the purpose therefore a basic qualitative reasoner was to be developed. After a short time, interests shifted toward creating a more useful qualitative reasoning implementation; therefore some research was undertaken into previous efforts to speed up execution times and to obtain more precise simulations than existing techniques.

There was an attempt by another group to implement an existing qualitative reasoner, QSIM (Kuipers, 1986), using parallel algorithms to improve performance. This was successful although there were a few drawbacks with their design and these provided the motivation to create a novel qualitative reasoning system which would overcome these disadvantages. The desire was to implement a new abstract architecture which was portable allowing it to be executed on a wide variety of computer systems and to incorporate parallelisations in every stage possible.

The desire to operate non-constructively relaxes the constraints on the models used in a system in that they may contain algebraic loops and do not require any specific ordering of the equations used to describe the model. No existing qualitative reasoning approach

provides a reasonable and accurate output without including spurious behaviours. The development of a non-constructive semi-quantitative simulation engine commenced that would allow reasoning on the spectrum from fully qualitative to fully quantitative. This presents a very useful tool in the development of many systems allowing simulations to be carried out using pure qualitative parameters, fuzzy intervals or precise quantitative information depending on the known precision of the system being simulated. Bridging the gap between qualitative and numerical simulation like this provides a suitable tool for many different applications, hopefully extending the user-base of qualitative reasoning.

These new approaches have been implemented in a single framework called JMorven. The name follows from its immediate predecessor, Morven, and the fact that the new framework is implemented in the Java programming language. Although JMorven is the successor to an existing qualitative reasoner, the core inference engine is completely novel as it uses non-constructive approaches and is implemented completely in parallel. JMorven adopts several features from its predecessor which are discussed in more detail in chapter 4.

1.2 Novel Contributions

Throughout the work during the PhD several novel features have been implemented in JMorven. These novel contributions are as follows:

1. A novel portable architecture with parallel optimisations: the parallelisations are abstract allowing the implementation to benefit from multiple processors or multiple computers in a distributed computing network. This allows JMorven to take advantage of several different hardware setups.

2. The use of auxiliary variables in a non-constructive environment. Previously, these

have been used constructively which requires ordering of the constraints and limits models to those with no algebraic loops. These auxiliary variables can also be used in parallel which is novel.

3. Using n-th order Taylor Series as an integration method. Most exisitng non-constructive qualitative reasoners use Euler integration since they only reason about one derivative per variable. JMorven uses multiple derivatives and automatically integrates with as much derivative information as possible to give more accurate integration approximations.

4. Incorporating several non-constructive fuzzy interval simulation techniques. Interval simulation makes use of a QR model and a partially specified initial state and uses numerical techniques to simulate the system behaviour over time. Previously, all successful interval simulators have been implemented in a constructive manner limiting the types of model which can be used.

5. The ability to produce asymptotically sound and complete non-constructive simulations. All previous methods produce either sound or complete results but JMorven offers a method which is both sound and complete as the number of iterations is increased. Monte-Carlo methods are also used as a quick approximation to these results.

6. Offering the ability to carry out numerical simulations non-constructively from a qualitative model description. This allows precise numerical simulation trajectories to be calculated if a precise initial state can be defined.

7. A single non-constructive simulation engine which is capable of reasoning on the spectrum from fully qualitative to fully quantitative allowing it to be used throughout the development of any model design from concept through prototype to final product.

1.3 Thesis Organisation

The work contained in this thesis is divided into three main parts. The first part comprising chapters 2, 3, 4 and 5 introduces the field and critically analyses several existing implementations which are used and extended to create JMorven. The second part, consisting of chapters 6 and 7 presents the JMorven implementation and all design choices that were faced during development. The third and last part containing chapters 8 and 9 presents the results and findings of the experiments used to test the JMorven implementation with discussions and conclusions presented reflecting on the work as a whole. The chapters are briefly summarised below:

- **Chapter 2** This chapter introduces the field of qualitative reasoning and the motivations behind it. The different modes of operation of a typical qualitative reasoning engine are briefly summarised. Design choices are presented including ontological choices and algorithm approaches. Finally a brief overview of some existing qualitative reasoners is given with a more detailed discussion of QSIM since it is one of the main predecessors to JMorven.

- **Chapter 3** A brief introduction to parallel computing is given detailing important aspects of performance increases and how to define the efficiency of parallel algorithms. A review of qualitative reasoning and parallel computing techniques combined is presented before a critical analysis of Parallel QSIM; the only known existing qualitative reasoning engine to be implemented in parallel.

- **Chapter 4** This chapter begins with an introduction to fuzzy numbers as a method of representing uncertainty. Two existing fuzzy qualitative reasoners, FuSim and Morven are analysed since they introduced many of the features incorporated into JMorven. A non-constructive synchronous simulator, SyNCSim, is briefly discussed as the ideas used in it are extended and used in JMorven to conduct semi-quantitative and quantitative simulations.

- **Chapter 5** In this chapter, numerical simulation is introduced and techniques for integration are discussed. There is also a brief introduction to interval arithmetic and the problems encountered when using it. There is a critical analysis of the existing semi-quantitative simulation strategies. These fall into two main categories; those that are based on QSIM and those that use fuzzy numbers. A summary of the many approaches is given at the end of the chapter.

- **Chapter 6** JMorven is introduced detailing the algorithms used and how they interact with each other. Each main stage of the qualitative aspect is discussed and how it was implemented in parallel. Finally, some extra features of JMorven are discussed.

- **Chapter 7** This chapter begins by presenting a new representation for fuzzy numbers so that they can make use of interval arithmetic. The integration technique used in JMorven is outlined and there is a discussion of some of the techniques used to aid narrowing intervals when used in a non-constructive manner as in JMorven. All of the methods of simulation are presented which fall into two main categories; those based on real intervals and those that approximate intervals as a group of points.

- **Chapter 8** The results chapter is split into three main sections. The first section presents the qualitative experiments and results from them to verify that JMorven produces the correct output and that auxiliary variables can be used non-constructively. The second section outlines the experiments used to determine how much of a benefit the parallel algorithms offer and how much speed-up is achieved. Finally, the third section shows the performance of each individual technique used in semi-quantitative simulation and summarises the pros and cons of each. The semi-quantitative simulation is also tested to show that it also benefits from the parallel architecture of JMorven.

- **Chapter 9** Finally the results are discussed and a conclusion is drawn reflecting on the initial aims and motivations of the work. Some future work is also proposed.

Chapter 2

Qualitative Reasoning

2.1 Introduction to Qualitative Reasoning

Numerical simulation can provide extremely useful predictions about how systems behave, but it is not always possible to construct quantitative models due to a lack of understanding or a lack of precise numerical information, for example obtaining values for kinetic parameters for the rate of cellular reactions are not reliable with current experimental protocols (de Jong, 2003). If this is the case carrying out a numerical simulation is impossible; however qualitative reasoning can be used instead to suggest behaviours from the information that is known.

Qualitative Reasoning (QR) is an area of Artificial Intelligence which was first studied in the late 1970s and early 1980s (de Kleer, 1977, 1979; Forbus, 1980, 1981; Kuipers, 1986). Qualitative reasoning has much in common with early research conducted in Naïve Physics (Hayes, 1979, 1985) and common-sense reasoning (Kuipers, 1984; Simmons, 1986). The motivation behind QR was to emulate how the human mind performs basic operations without the need for precise numerical information, or to be able to reason when there is some knowledge or information missing.

QR has been used in industrial applications, one of the first being a self-maintaining photocopier (Shimomura et al., 1995) and more recently a diagnosis tool for an engineering plant (Coghill, 2000) and diagnosis in the automotive industry (Price, 2000; Struss and Price, 2004). There are several different fields in which QR has been used including digital circuits (Williams, 1984a; Kaul et al., 1992; Lee, 1999b) including detecting failure modes and its effects (Pugh and Snooke, 1996; Lee, 1999a, 2000; Lee and Ormsby, 1992), tutoring (Lulis et al., 2004; de Koning et al., 2000), diagnosis (de Kleer and Williams, 1987; Ng, 1991; Liu and Coghill, 2005b), system identification (Kay et al., 2000), learning (Coghill et al., 2004) and many others (Bredeweg and Struss, 2004). More recently areas utilising qualitative reasoning are biology (Trelease and Park, 2003; King et al., 2005) and ecology (Salles and Bredeweg, 2003) since these areas lack enough precise data to use numerical techniques. For a review detailing many practical uses of qualitative reasoning see (de Jong, 2003). There is still much interest in the field of QR and a future vision of applications include The Science Bot, virtual vehicles, understanding and managing complex natural systems, interpretation of 4D medical data and robust autonomous problem solvers in the face of uncertain situations (Price et al., 2005).

Qualitative Reasoning can been described as the study of 'reasoning without numbers'. Numerical techniques use real numbers which have infinite cardinality (Coghill, 1996; Shen, 1991). At the other end of the precision spectrum is a single quantity which covers the complete real number line, although this is not very useful as all mathematical operations on this quantity space give the same result. The simplest usable quantity space, termed the signs, therefore is $\{+\ 0\ -\}$ which has a cardinality of three. The signs can also be used to specify the derivative information of a variable in a dynamic system, this information helps to describe how the variable changes over time. This simple representation is very useful in determining behaviours, although something in between purely qualitative and purely numerical is sometimes desired when there is more precise numerical information available, yet not enough to carry out a numerical simulation. As such, some qualitative reasoners use quantity spaces with more precise numerical information than

the signs; for example QSIM (Kuipers, 1986) uses ordinal relations defined by landmark values and intervals between the landmarks. Fuzzy sets (Zadeh, 1965) are an alternative method to analysing complex systems with imprecise or ambiguous information (Zadeh, 1973). The desire to combine both qualitative reasoning and fuzzy numbers was the main motivation for a new type of reasoner (Shen and Leitch, 1993; Coghill, 1996). Fuzzy numbers are used to define quantity spaces allowing a degree of ambiguity to be inherent in models.

To be able to predict the behaviours of a system, some method of defining it is required. A qualitative model is used to define all of the variables of a system to reason about and how the variables relate to one another. Constructing these qualitative models requires a decision to be made be made about what ontology will be used before proceeding. The next section discusses the ontologies used in the field of qualitative reasoning.

2.2 Ontologies

An ontology is a representation of how one perceives the world, therefore the language used to define this world is often called the ontology. There are three common ontologies used in qualitative reasoning, each of which are briefly summarised below.

2.2.1 Process-Oriented Ontology

The process oriented ontology was first developed by Forbus (Forbus, 1981). The process oriented ontology was originally developed as an implementation of Hayes' Naïve physics (Hayes, 1979, 1985). The motivation behind this approach was to create models of steam plants and other similar engineering systems. This type of system can be described by the processes which exist or are created and how they directly or indirectly influence entities within the system. For example, if there is a radiator in a room which is turned on then

there is a heat source causing the process of heat flow from the radiator to all objects in the room causing their temperature to raise. Forbus first proposed Qualitative Process Theory (Forbus, 1984) using the process centred ontology and later went on to develop the Qualitative Process Engine (Forbus, 1990). This ontology has also been utilised in other systems, including a model-based planner based on processes known as Excalibur (Drabble, 1993)

2.2.2 Device-Oriented Ontology

In the device oriented ontology models are described by interconnected devices interacting via lossless ports. It lends itself very well to electrical circuitry where electrical components can be thought of as the devices and the wires connecting them can be approximated by the interconnecting ports. The motivation behind this method was to create engineering models which have a degree of re-usability and hence can be used hierarchically. The first reasoner to use this approach was Envision (de Kleer and Brown, 1984) although several since have also been developed including AutoSteve (Price, 2000) which uses Failure Modes and Effects Analysis (FMEA) as used in Flame (Price et al., 1995) and the Jacquard project (Hunt et al., 1993).

2.2.3 Constraint-Oriented Ontology

In this approach models are created using constraints based on ordinary differential equations which are abstracted for use qualitatively; these are termed Qualitative Differential Equations (or QDEs). The motivation behind this approach is that almost any system can be described using a set of equations and it offers a similar modelling approach to traditional numerical simulation. The Qualitative Physics Compiler (Crawford et al., 1990) was developed to translate QPT models based on processes and compile a set of constraints for use in QSIM thus showing that it is possible to use the constraint-oriented

ontology to model processes. The most famous constraint based qualitative reasoner is QSIM (Kuipers, 1986) which will be described in more detail in section 2.6. Many other systems have followed this approach including FuSim (Shen and Leitch, 1993) and Morven (Coghill, 1996) (the predecessor to JMorven).

2.3 Modes of Operation of a Qualitative Reasoner

To aid analysis of qualitative behaviours, qualitative states are used. A qualitative state can be thought of as a snapshot of the behaviour of the whole model at an instant or interval in time. The current values of all variables make up the qualitative state. For example, a model with three variables A, B, and C may exhibit the following qualitative state:

A: +

B: -

C: 0

For this example, there would be at most 27 possible unique states, however not all are guaranteed to be consistent with the model.

Variables in a model can be separated into two categories, endogenous variables (also known as system variables) and exogenous variables. Endogenous variables are those which are internal to a model or those which the user has no direct control over, for example the heat inside an oven. An exogenous variable is external to the model and can be directly controlled, for example the temperature knob of the oven.

Imprecise information occurs when the exact values of variables or system parameters are not known; instead ranges of values, or qualities, are used to represent all possible values

which the variables or parameters may take. Incomplete information occurs when there is an entity that is incompletely known; often this is the relationship between variables. In this case, monotonic functions can be used which do not require precise detail about the relationship between two variables - these are discussed more in section 2.6.2. Qualitative reasoning predicts the behaviour of systems with this imprecise and incomplete information. As such, there are many behaviours predicted for these systems. This section summarises the different modes of operation of a typical constraint-based qualitative reasoner.

2.3.1 Qualitative Analysis and Transition Analysis

There are two main stages of a typical qualitative reasoner; these are Qualitative Analysis (QA) and Transition Analysis (TA) (Williams, 1984b). During the Qualitative Analysis phase all values of variables are analysed and checked for consistency with the model and the consistent values are then used to generate qualitative states. How this is achieved depends on the implementation; a selected few are discussed later in this chapter. The Transition Analysis phase involves determining the transitions that are possible between the qualitative states. This phase may, or may not be executed depending on the mode of operation required from the qualitative reasoner.

2.3.2 Directed Graphs

There are a few different structures that are used to represent the predicted behaviours of systems, for example trees and graphs. The one used in JMorven is a directed graph so a brief outline of the directed graph is given here.

A directed graph is a structure which contains nodes and directed edges. Nodes in the graph are used to represent qualitative states in a qualitative reasoner and the directed

edges are used to indicate a transition from one state to another in the direction specified. Directed graphs have an advantage in that they can contain cyclic behaviours therefore infinite behaviours from oscillations can be represented and occupy very little memory. Self-transitions are used to dictate that a state may transit to itself; this is typical for most states apart from those that pass through a landmark or real number.

2.3.3 Envisionments

An envisionment is an exhaustive list of all the of the qualitative states a model may exist in. There are two main types of envisionment which are discussed below. Simulation is often termed an attainable or partial envisionment; this is discussed in section 2.3.4.

2.3.3.1 Total Envisionment

A total envisionment is used to determine every possible state a model may exist in for all values of the exogenous variables. The total envisionment may be useful to determine what possible states can occur from a given model. The number of states quickly expands with the complexity of the model or cardinality of the quantity spaces used, thus a complete envisionment is often used.

2.3.3.2 Complete Envisionment

Complete envisionments are very similar to total envisionments in that they display all of the possible states a model may exist in, however they have an additional constraint in that some or all exogenous variables are specified which greatly reduces the complexity of the directed graph. This is a useful tool as all the possible states can be viewed for a system when the inputs to the system are known or can be specified.

2.3.4 Qualitative Simulation

Qualitative simulation is a step-by-step operation which predicts all qualitative behaviours of a model from a given initial state. Qualitative simulation is usually performed using asynchronous simulation or event-drive simulation. An initial state is required for simulation to take place although this does not need to be a fully specified state (if the state is not fully specified similar problems to the total envisionment arise with the complexity of the calculated directed graph). All possible transitions are calculated from the initial state to determine all of the successor states. Simulations can be performed in a similar manner to depth-first search or breadth-first. In a breadth-first manner the successor states are then used to calculate all transitions for the next level of successor states whereas a depth-first search expands one successor state at a time at each level. The simulation carries on until a limit is reached or once all states reach an equilibrium (or steady) state. It is worth noting that a simulation carried out with an empty initial state would provide similar results to a total envisionment in that all possible states would exist in the directed graph.

2.4 Constructive vs Non-constructive Methods for Simulation

There are several different methods for carrying out simulations in QR which can be split into two main categories. These two categories are termed constructive methods and non-constructive methods (Wiegand, 1991) depending how the constraints are used. The two methods are discussed below with a description of the technique and a discussion of some of the advantages and disadvantages associated with each.

2.4.1 Constructive Simulation

Constructive algorithms are defined as ones that generate the the values of the endogenous variables directly from the model definition. This is achieved by using the system equations and integration techniques to predict successor values to the endogenous variables and using the model to construct the remaining variables' values. The values of all variables are constructed during the process hence the term constructive simulation. The most common method of integrating the dynamic equations is to use Euler integration which is the same as first order Taylor Series expansion as shown below:

$$x(t + \delta t) = x(t) + \dot{x}(t).\delta t$$

This formula generates the value of x at time $t + \delta t$ based on the value of x and its derivative at time t with a given time step of δt. This integration is carried out for all system variables resulting in the successor values for all magnitudes of the variables on the left hand side. Once this stage is complete the model equations are then utilised to construct the values of the remaining variables and derivatives. The drawbacks of this approach are that the system equations must be specified in a specific form similar to ordinary differential equations and these equations must be causally ordered. This is not always possible as ordering may be impossible due to algebraic loops. This method will therefore not work when algebraic loops appear in the model. Algebraic loops arise when the value of one variable cannot be computed as it itself must be used to calculate the successor values, for example in the system of equations below (taken from (Cellier, 1991) and simplified by removing all equations extraneous to the algebraic loop) u_3 cannot be calculated due to the algebraic loop.

$$u_3 = U_0 - u_1$$

$$u_1 = R_1 * i_1$$

$$i_1 = i_2 + i_3$$

$$i_3 = \frac{u_3}{R_3}$$

The loop here is found as u_3 appears on the right hand side of the equation for solving i_3. i_3 is required to compute the value of i_1 which is required to find the value of u_1. Finally u_1 is required to compute the value of u_3 therefore an algebraic loop exists and cannot be dealt with by a normal constructive approach. Algebraic loops occur in many different areas including electrical circuits as described above, biological systems (de Jong et al., 2003) and diagnosis (Mosterman et al., 2000).

2.4.2 Non-Constructive Simulation

An alternative approach is to adopt a non-constructive method for simulation. This technique involves generating all possible values that variables may take and using the model equations to discard all inconsistent values. This can be achieved by using integration or predefined transition rules based on integration and the system equations to filter out inconsistent values.

The next stage involves filtering out tuples which are inconsistent with the remaining constraints, i.e. for a tuple to be consistent with the model, it must be consistent with all of the constraints individually. This method does not require any specific type or ordering (Iwasaki and Simon, 1986) of equations making it simpler to produce usable models. The main advantage of using non-constructive methods is that models with algebraic loops can be analysed.

It has been shown that constructive approaches and non-constructive approaches to simulation give the same results in qualitative reasoning (Coghill, 1996; Coghill and Chantler, 1999). In the non-constructive method all behaviours are generated and then tested for consistency with the constraints, however in a constructive algorithm the system variables are propagated and then used to calculate all of the remaining variables in the constraints. This essentially produces the same output as the constraints are responsible in both cases for determining whether to keep or discard certain behaviours. With this in mind, it is clear that a non-constructive algorithm has advantages in that it produces the same output but it can also cope with algebraic loops and also does not require any prior ordering of the constraints.

To demonstrate how a non-constructive system can deal with an algebraic loop, the example from the previous section is used. If it is known that $R1 = 1k\Omega$, $R3 = 20k\Omega$, $U_0 = 60v$ and $i_2 = 7.5mA$ then u_3 can be calculated as follows:
assume a large range for the unknown values

$$i3 = [0 \quad 10^{36}]$$

$$i1 = [0 \quad 10^{36}]$$

$$u1 = [0 \quad 10^{36}]$$

$$u3 = [0 \quad 10^{36}]$$

now loop through the constraints

$$i3 = [0.0000 \quad 5.0000001268882145 * 10^{25}]$$

$$i1 = [0.0075 \quad 5.0000001268882145 * 10^{25}]$$

$$u1 = [7.5000 \quad 5.0000000752373311 * 10^{28}]$$

$$u3 = [0.0000 \quad 52.5000]$$

looping again through the constraints gives:

$$i3 = [0.0000 \quad 0.0026]$$

$$i1 = [0.0075 \quad 0.0101]$$

$$u1 = [7.5000 \quad 10.1250]$$

$$u3 = [49.8750 \quad 52.5000]$$

it is clear at this point that the values are rapidly converging, another loop through the constraints gives:

$$i3 = [0.0025 \quad 0.0026]$$

$$i1 = [0.0100 \quad 0.0101]$$

$$u1 = [9.9937 \quad 10.1250]$$

$$u3 = [49.8750 \quad 50.0062]$$

which gives very narrow ranges. Continuing the loops 3 more times gives us exact numbers as shown:

$$i3 = [0.0025 \quad 0.0025]$$

$$i1 = [0.0100 \quad 0.0100]$$

$$u1 = [9.9937 \quad 10.0003]$$

$$u3 = [49.9997 \quad 50.0062]$$

$$i3 = [0.0025 \quad 0.0025]$$

$$i1 = [0.0100 \quad 0.0100]$$

$$u1 = [10.0000 \quad 10.0003]$$

$$u3 = [49.9997 \quad 50.0000]$$

$$i3 = [0.0025 \quad 0.0025]$$

$$i1 = [0.0100 \quad 0.0100]$$

$$u1 = [10.0000 \quad 10.0000]$$

$$u3 = [50.0000 \quad 50.0000]$$

This shows that within 4 iterations of the constraints the values of unknown variables had been calculated to within approximately 1% of the correct value and that after 7 iterations the exact value was calculated (to within the precision of 32-bit floating point numbers).

2.5 Existing Qualitative Reasoning Engines

Over the past twenty years a lot of research has been undertaken in the area of qualitative reasoning resulting in several qualitative reasoning engines. Forbus' main influence was simulation of engineering systems, in particular steam plants which led to the development of the Qualitative Process Engine (Forbus, 1990) based on his Qualitative Process Theory (Forbus, 1984). As mentioned previously, this reasons about processes and how

they directly or indirectly influence objects in a model. This has been used in several different environments including model-based planning (Drabble, 1993) and ecology (Salles and Bredeweg, 2003).

CA-EN (Bousson and Travé-Massuyès, 1994) is a constraint-based qualitative reasoner which conducts synchronous simulations, i.e. those that are driven by a regular time-step. CA-EN can reason with causes and constraints by using two coupled levels of constraints. The global constraints level defines all of the constraint equations and the local constraint level is used to indicate processes as in QPT. Variables in CA-EN are expressed as either a real interval or a symbol depending on the available information. What results is a simulation algorithm which can reason with multiple degrees of imprecision and produces output envelopes for the simulations. The main disadvantage with this approach however is that it uses constructive methods.

Order of magnitude reasoning (Raiman, 1991) extends traditional qualitative reasoning techniques by determining the relative sizes of expressions, that is whether two expressions are approximately the same, or one is slightly larger, larger, or much larger than the other (and similar for smaller). This was applied to several different models and found to aid the qualitative descriptions to provide a better output. There have been several extensions to basic order of magnitude reasoning, including Raiman's own FOG , O(M) (Mavrovouniotis and Stephanopoulos, 1988) and CHEPACHET (Davis, 1990). Another extension to order of magnitude reasoning has been implemented in the CA-EN simulator described above which uses fuzzy numbers to depict how two variables' magnitudes relate to one another.

Another very popular qualitative reasoning package is QSIM which has influenced the development of many others including FuSim and Morven. QSIM is discussed in more detail in the next section.

2.6 Qualitative SIMulation (QSIM)

QSIM is one of the most developed qualitative reasoning packages. In this section, QSIM and its features are discussed since it is the predecessor to Parallel QSIM - a parallel implementation of QSIM which is critically evaluated in this thesis. Also, JMorven has some features which have evolved from QSIM through the intermediate systems FuSim and Morven.

2.6.1 Introduction to QSIM

QSIM was first developed in the early 1980s by Kuipers (Kuipers, 1986) and was one of the first qualitative reasoners to use the constraint based ontology. These constraints are specified in a special form of mathematical differential equations abstracted for use qualitatively, termed Qualitative Differential Equations or QDEs. The motivation behind using constraints in this form was that engineering and dynamic systems could be easily modelled using these types of equations just as differential equations would be used for numerical simulation. QDEs are discussed in more detail in the next section. Kuipers guarantees that QSIM will find all the possible qualitative behaviours for a system but extraneous behaviours may also be included hence the QSIM algorithm is complete but unsound.

2.6.2 Qualitative Differential Equations

Qualitative Differential Equations (QDEs) are an abstraction of ordinary differential equations (ODEs). Variables in a QDE are the qualitative equivalent to the numerical variables of the corresponding ODE. A typical ordinary differential equation is shown

$$m\ddot{x} = F - kx$$

where x is displacement of a mass m from its rest position, $\dot{x}$ denotes the velocity of the mass, $\ddot{x}$ represents the acceleration of the mass F is the external force applied to the mass and k is the stiffness of the spring. This is an equation for a simple harmonic motion mass-on-a-spring system. To generate a QDE equivalent, the variable x would be qualitative instead of quantitative as in the ODE (see the next section for more information about qualitative values in QSIM). Also QSIM requires constraints to be specified as two or three place predicates. The following set of qualitative differential equations are equivalent to the ODE above (assuming unit mass and unit spring stiffness):

D/DT (A, X)

D/DT (B, X)

MINUS (C, X)

ADD (B, F, C)

Some systems do not have enough known information to model them using strict quantitative functional relations therefore QSIM also defines monotonic function constraints. A monotonically increasing function is whose derivative is positive for all values therefore if one variable increases the other is guaranteed to increase. Similarly for monotonically decreasing functions whose derivatives are always negative. These are defined in QSIM as follows:

M^{+}(P, Q)

M^{-}(Q, R)

To add extra numerical information to the QDEs, QSIM defines corresponding values. These are when a variable is known to be at a certain value, the value of another variable or variables may be explicitly stated. This extra information can help reduce the number of spurious behaviours generated. For example, in monotonic functions it is often found

that $(0, 0)$ is a valid corresponding value stating that when one of the variables in the monotonic constraint is 0 then the other must also be 0. Although this example shows the use of corresponding values in a monotonic constraint, they can be used in any type of constraint. Some work has also been undertaken into using intervals as corresponding values (Say and Kuru, 1993).

2.6.3 Qualitative Variables

Qualitative variables in QSIM are of the form of a $\langle qmag, qdir \rangle$ pair, where $qmag$ denotes the qualitative magnitude of the variable and $qdir$ is the qualitative direction of change or derivative of the variable. The direction of change of the variable can take one of three possible values, inc, dec or std, which represent increasing, decreasing or steady respectively. The possible values of the magnitude depend on how the quantity space is defined. A quantity space in QSIM is an ordered list of possible landmark values $l_1 < l_2 < ... < l_k$ which represent qualitatively important values. Some landmarks may be pre-defined for the model, and new landmarks can be added during simulation. The variable magnitudes may take the value of one of these landmarks or an interval (a range between two landmarks denoted by $]l_j, l_{j+1}[$). The simplest quantity space, the signs, is defined by three landmark values, $-\infty, 0, +\infty$. All negative values lie in the interval $(-\infty, 0[$ and all positive values lie in the interval $]0, +\infty)$. Time also adheres to this landmark representation where temporal landmarks are created at times when variables change to or from a landmark. Landmarks may also be created during the running of a simulation in QSIM when variables reach important behaviours, e.g. a turning point.

QSIM generates Qualitative States at all time points and intervals. States consist of the values for all of the variables in the model at the given time. These states along with the transitions between them form the qualitative behaviours of the simulation in the form of a behaviour tree. How these transitions are defined is discussed in the next section.

P-transitions	$QS(v,t_i)$	$QS(v,t_i,t_{i+1})$	I-transitions	$QS(v,t_i,t_{i+1})$	$QS(v,t_{i+1})$
P1	$< l_j, std >$	$< l_j, std >$	I1	$< l_j, std >$	$< l_j, std >$
P2	$< l_j, std >$	$< (l_j, l_{j+1}), inc >$	I2	$< (l_j, l_{j+1}), inc >$	$< l_{j+1}, std >$
P3	$< l_j, std >$	$< (l_{j-1}, lj), dec >$	I3	$< (l_j, l_{j+1}), inc >$	$< l_{j+1}, inc >$
P4	$< l_j, inc >$	$< (l_j, l_{j+1}), inc >$	I4	$< (l_j, l_{j+1}), inc >$	$< (l_j, l_{j+1}), inc >$
P5	$< (l_j, l_{j+1}), inc >$	$< (l_j, l_{j+1}), inc >$	I5	$< (l_j, l_{j+1}), dec >$	$< l_j, std >$
P6	$< l_j, dec >$	$< (l_j, l_{j-1}), dec >$	I6	$< (l_j, l_{j+1}), dec >$	$< l_j, dec >$
P7	$< (l_j, l_{j+1}), dec >$	$< (l_{j-1}, lj), dec >$	I7	$< (l_j, l_{j+1}), dec >$	$< (l_j, l_{j+1}), dec >$
			I8	$< (l_j, l_{j+1}), inc >$	$< l^*, std >$
			I9	$< (l_j, l_{j+1}), dec >$	$< l^*, std >$

Table 2.1: Transition Rules in QSIM

2.6.4 Transition Rules

To define how Qualitative States transit between one another to create Qualitative Behaviours, QSIM defines a list of transition rules. These transition rules adhere to the Intermediate Value Theorem and the Mean Value Theorem from Calculus (Spivak, 1967). These continuity constraints dictate that for a variable to transit from one value to another, it must pass through all intermediate values. Valid transitions for a variable depend on whether the current time is at an interval or a landmark. Table 2.1 shows transition rules as defined in QSIM for a continuous function v with landmarks $l_{j-1} < l_j < l_{j+1}$. P-Transitions signify transitions from a time point to an interval and I-Transitions denote transitions from a time interval to a point. This is effectively qualitative euler integration, for example, if variable x is increasing then the transition rules state that the successor value of x will be a quantity greater than the current one (or if x is an interval the successor value may be the same interval or the next greater landmark). The intermediate value theorem ensures that the successor quantity is that which is immediately greater to the current quantity before transition to any other quantities.

2.6.5 QSIM Algorithm

The QSIM algorithm aims to generate all consistent qualitative behaviours from the model, consisting of QDEs, and an initial state. QSIM is a type of constraint satisfaction problem solver which solves a constraint network of variables and their domains, or quantity spaces, across a number of constraint relations in the from of QDEs. The

output from the QSIM algorithm is a behaviour tree consisting of all possible qualitative behaviours from the initial state. QSIM achieves this by analysing the initial state and determining all possible successor states by applying the transition rules and checking consistency with the equation constraints. These states are then analysed and their successor states are generated, continuing until no more states are left to analyse. Successor states are not computed for states that satisfy the following conditions:

- The current state is identical to a previous state resulting in the same successors thus avoiding an infinite cycle.
- The current state is a transition state i.e. one that is defined during the transition from one state to another
- The current state is an equilibrium or quiescent state, i.e. one that has no possible unique successor states due to all qualitative directions being steady.
- The current time is $t = \infty$

In the QSIM algorithm described above, the state transitions are constrained by the transition rules and the generated qualitative states are constrained by the qualitative differential equations of the model. This latter constraint filter is split into two sub-components termed the Tuple Filter and Waltz Filter which are discussed below.

2.6.5.1 Tuple Filter

The Tuple Filter in QSIM iterates through a number of tuples provided to check the consistency with the constraints. It achieves this by going through each constraint in turn. Each constraint generates an exhaustive list of all possible tuples and discards any that are inconsistent. Tuples which remain are therefore consistent with that individual constraint.

2.6.5.2 Waltz Filter

The Waltz Filter in QSIM is actually an AC-3 implementation (Mackworth, 1977) rather than a pure Waltz algorithm (Waltz, 1975). This is a pairwise filter which executes on each possible pair of adjacent constraints (two constraints which share a common variable are said to be adjacent). All tuples are checked for consistency and if found to be inconsistent with either constraint they are discarded. In QSIM, the Waltz filter is executed incrementally - after each constraint is checked with the tuple filter, the Waltz filter acts on the set of tuples. This causes some of the tuples of a constraint to be discarded before the tuple filter is again executed thus allowing a slight performance increase.

2.6.6 Form All States

Once all tuples have traversed the constraint filter they are processed to create qualitative states, this is the function of the 'Form All States' stage. These qualitative states are guaranteed to be consistent with the tuples and are used to create qualitative behaviours following the transition rules. QSIM uses a backtracking algorithm which performs a depth-first search. This recursively calls itself with the next constraint if a tuple is found to be consistent with the partial state. Once the final constraint is reached and a tuple is consistent a solution is found in the form of a qualitative state.

2.6.7 Global Filters

As described above QSIM uses constraints in the form of QDEs and transition rules to filter qualitative behaviours which are carried out per-constraint or per-transition. The addition of Global Filters allows QSIM to add constraints which can be applied over a whole qualitative behaviour. One of the most common global filters is one which ensures the law of conservation of energy is not broken (Fouche and Kuipers, 1992). Qualitative

simulation may break this law, e.g. a spring without any external forces may exhibit a behaviour that the amplitude of oscillations increases which is not possible in reality. This global constraint forces these oscillations not to increase thus observing the law of conservation of energy.

2.6.8 Extensions to QSIM

QSIM is one of the most popular qualitative reasoning engines available and as such a lot of further research has gone into expanding it. There are many extensions to QSIM which have been developed to add extra functionality or reduce the number of spurious behaviours. Some of these extensions filter out too many behaviours and impinge on the completeness of the QSIM algorithm resulting in not all real behaviours being included in the simulations. It is beyond the scope of this thesis to detail all of the many extensions to QSIM, the reader is directed to (Kuipers and Chui, 1987; Lee and Kuipers, 1988, 1993; Fouche and Kuipers, 1991; Hossain and Ray, 1997; Hofbaur and Dourdoumas, 2001) for further reading.

One type of extension to QSIM which is of interest to the study of this thesis are the development of semi-quantitative and interval based extensions for example NSIM (Kay and Kuipers, 1992), Q2 (Kuipers and Berleant, 1988), Q3 (Berleant and Kuipers, 1990), QuaSi (Bonarini and Bontempi, 1994a), SQSIM (Kay, 1998) and DecSIM (Clancy and Kuipers, 1998). Some of these are discussed in more detail in chapter 5.

2.7 Summary

In this chapter, the field of Qualitative Reasoning has been introduced with the aims and motivations behind it. These include the ability to reason about models when only imprecise or incomplete knowledge is available about either the model structure and/or the

parameters of the model. Several strategies for accomplishing this have been implemented from the research undertaken. These differing methods have been discussed and it is proposed that the constraint centred approach offers the most flexibility as a representation for defining models. As such, efforts have been concentrated on existing qualitative reasoning engines that adopt the constraint ontology including the popular QSIM. QSIM has been reviewed as it is a direct influence of several of the engines discussed in thesis including Parallel QSIM, the only known existing parallel qualitative reasoning implementation. The review follows a discussion of the differing modes of operation that qualitative reasoning offers including envisionments and simulation. Total and complete envisionments provide a representation of the global behaviours of a system or subset of all behaviours by fixing the values of exogenous variables whereas simulation generates the possible behaviours for specific initial states. Constructive and non-constructive approaches to qualitative reasoning were discussed and it is argued that non-constructive approaches offer an advantage in that they are more general in that they do not impose causal ordering on the model constraints and can reason with models that contain algebraic loops.

Chapter 3

Parallel Qualitative Reasoning

3.1 Introduction to Parallel Computing

Modern computing allows an easy and inexpensive means of combining the power of several processors or computers to be used to run the same process. This allows execution times of these processes to be drastically reduced. A lot of research has been undertaken into developing parallel algorithms for many tasks, and as such there are many standard means of comparing parallel algorithms. Leighton (Leighton, 1992) describes the speed-up, S, as the time taken for the best sequential algorithm to complete over the time taken for the best parallel algorithm to complete. The speedup S_n for an algorithm running on n processors is defined as:

$$S_n = \frac{t_1}{t_n}$$

where S_1 is the sequential time and S_n is the time taken to run on n processors. The efficiency of a parallel algorithm is defined as:

$$E = \frac{S_n}{n}$$

The optimal speed-up of an algorithm is termed *Linear Speed-up*. This is when the efficiency of the algorithm is constant for all n, i.e. when $E = 1$. Although the theoretical optimal speed-up for an algorithm is linear, it is not trivial to achieve. In fact, it is often only possible to achieve speed-up where E decreases as n increases.

To benefit from parallelisations, the tasks to be implemented must not be inherently sequential for example depth-first search (Reif, 1985). There are two main techniques for generating a parallel implementation of a given task, these are known as data parallelisations and algorithmic parallelisations. A good example describing the process between data and algorithm parallelisations is presented by (Greenlaw et al., 1995) which describes generating an algorithm for sorting. The obvious way to do a sort on data D on n processors is to split the data into smaller subsets, d_x, such that

$$\forall_{x \in 1,n} : d_x \subset D$$

$$d_1 \cup ... \cup d_x \cup ... \cup d_n = D$$

and then sort each subset on its own parallel processor. This is an example of data parallelisation. The problem with this approach is that after each processor has executed, each subset is indeed sorted; however the subsets must be combined to create the sorted set D. Using similar approaches to generate larger subsets until the full set is sorted achieves only a marginally better time than the sequential algorithm, therefore an algorithmic parallelisation should be used instead. One such method to do this involves ranking each item to be sorted. The rank can be computed by assigning each parallel unit to a pair of numbers and scoring the entries depending on which is greater. All possible pairs are evaluated resulting in a rank for each entry which can then be used to order the entries in one pass. The interested reader is directed to (Greenlaw et al., 1995) for a detailed discussion of an efficient method of sorting a dataset in parallel.

3.2 Qualitative Reasoning and Parallel Processing

There has been very little research on implementing parallelisations in qualitative reasoning specifically; the only major work carried out to date is that of Platzner and Rinner (Platzner et al., 1997) which is discussed in more detail in section 3.3. However it is worth investigating existing research undertaken in parallelisations used to speed-up similar algorithms that have been used in qualitative reasoning in the past. Since QSIM has been recognised as a type of Constraint Satisfaction Problem (or CSP) solver (Clancy and Kuipers, 1998) it is worth discussing some work undertaken into parallelising CSPs.

Constraint-based qualitative reasoning can be considered a Constraint Satisfaction Problem of $\langle V, D, P \rangle$ where:

- $V = v_1, ..., v_n$ is the set of variables of the CSP
- $D = D_1, ..., D_n$ is the set of domains where D_i holds a set of possible values for variable v_i
- $P = P_1, ..., P_m$ denotes the set of constraint relations. P_j operates on a subset of variables from V such that all variables in V are constrained by at least one element in P.

The qualitative reasoner attempts to find solutions of this CSP where:

- the set V represents the set of qualitative variables in the QDE
- $d_i \in D$ is equivalent to the quantities a variable v_i may take from the quantity space
- P is the set of constraints of the model.

Three main classifications of parallel CSP algorithms were presented by (Luo et al., 1994) which are briefly stated below. A summary of parallel units is given first. Parallel units are disjoint computational nodes which independently solve a problem. Parallel units can be central processing units, separate computers (or nodes) or virtual processors as in Intel's HyperThreading technology.

- **Distributed Agent Based (DAB)** This is where the variables are distributed between parallel units. Due to constraints having multiple variables, there is a lot of communication required between parallel units in this strategy.

- **Parallel Agent Based (PAB)** This is where the domains of the variables are distributed between parallel units. This effectively solves a CSP sub-problem and can use any sequential CSP algorithm. No communication is necessary between parallel units during execution.

- **Function Agent Based (FAB)** This is where functions which are repeatedly executed can be distributed between parallel units. This requires the architecture to have a shared memory type.

Of these strategies, the Parallel Agent Based method is the most common having been used in many CSP implementations including (Lin and Yang, 1995a,b; Burg, 1990). The PAB approach lends itself particularly well to QR due to its ability to find all solutions of a CSP whilst requiring no communication between parallel units and being able to use any sequential CSP algorithm in each parallel branch. The splitting of the whole problem into sub-problems by the domains of the variables makes sense too, the valid tuples of a constraint can be used to create sub-CSPs.

3.3 Parallel QSIM

Parallel QSIM was developed by Marco Platzner (Platzner, 1996) and Bernhard Rinner (Rinner, 1996). The motivation behind their work was to create a more efficient implementation of the popular QR package, QSIM, which was to be used as part of a larger project, *Distributed Real-Time Expert System for Fault Diagnosis in Technical Processes*[1] (Rinner, 1996; Platzner, 1996; Platzner and Rinner, 1995, 1998, 2000; Platzner et al., 1995, 1997). Platzner concentrated on designing and developing co-processors for intensive calculations that were highly repetitive with the idea that a hardware implementation would increase the speed of execution greatly. To take advantage of the hardware and increase efficiency, Parallel QSIM was developed in C. Rinner was responsible for creating the parallel architecture which would be central to Parallel QSIM. Combining their work resulted in an implementation of QSIM which was several orders of magnitude faster than the original LISP implementation of QSIM. This section concentrates on the work of Rinner and the parallel architecture of Parallel QSIM as this is the area of most relevance to the work undertaken for this thesis.

3.3.1 Design Choices

The design of the parallel architecture for Parallel QSIM was aimed at a hardware implementation since concurrent research was being undertaken to develop coprocessors to speed-up several highly repetitive calculations. An architecture was required which would distribute computation between several parallel units which would each contain one of these co-processors. Rinner's aim was to develop a scalable parallel architecture for the most computationally intensive stages of the QSIM algorithm. Scalability was defined by Hwang (Hwang, 1993) as having three aspects:

[1] Project conducted at the Institute for Technical Informatics, Graz University, Austria.

- Problem-size Scalability. This is when a parallel implementation should be able to perform well with an increase in problem size, ideally linear - i.e. if a problem P_1 executes in time t_1 then a problem of c times the complexity P_2 should complete in time $t_2 = c.t_1$

- Machine-size Scalability. For a system to be machine-size scalable, it should be able to exhibit a near-linear speed-up in execution time for the number of parallel units available. i.e. Problems on a machine with n parallel units should take $t = \frac{t_{seq}}{n}$ to complete[2].

- Generation scalability. A system should be able to exhibit good speed-up on current generation computers, but equally future generations of computers should also benefit from similar speed-ups.

Rinner acknowledges that his design does not address the problem of generation scalability; this is due to the fact that since they use a dedicated hardware setup, there might not be a future generation of hardware for their solution. Rinner considers machine-size scalability as the main focus of his scalable design, and comments that problem-size scalability is also considered.

After analysing the runtime of each stage in QSIM it was found that the constraint filter dominates the running time of QSIM. This was then broken down to determine which part of the constraint filter was most computationally expensive. It was found that the execution times of each stage of the constraint filter depends on the mode of operation. For initial state processing, which provides similar results to an envisionment without transitions, the form-all-states stage was highly dominant, however during generation of successor states, the tuple filter took most time to execute. Rinner therefore concentrated his efforts on parallelising the tuple filter and form-all-states stages of the QSIM algorithm, both of which are discussed below. The tuple filter could also benefit from

[2] t_{seq} is sequential time, the time taken for a problem to complete on a sequential (non-parallel) system.

the co-processors developed by Platzner as it required many intensive calculations which could be implemented in hardware.

3.3.2 Tuple Filter

The tuple filter and Waltz filter were combined in the original QSIM algorithm. Rinner devised the figure (shown in figure 3.1) showing the data dependencies of the incremental tuple filter and Waltz filter. In the figure *pvals* denotes possible values for the variables and the tuples are labelled as *tuples*. The original idea behind this incremental filter was

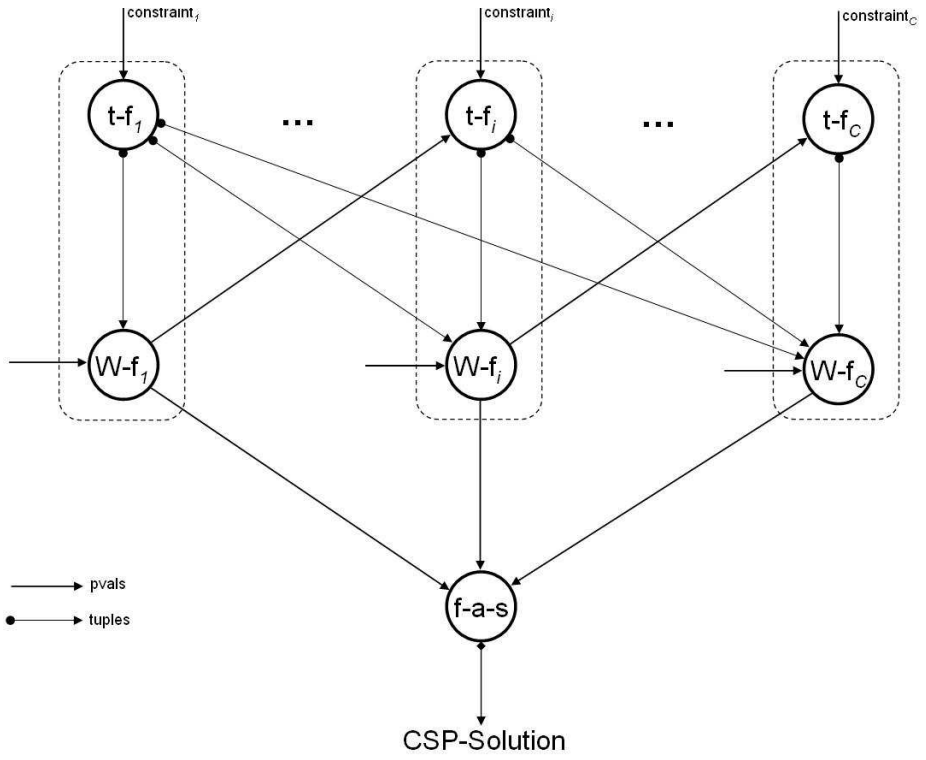


Figure 3.1: Data Dependency of the Incremental Tuple Filter and Waltz Filters. t-f denotes the Tuple Filter stages, W-f the Waltz Filter stages and f-a-s is the form-all-states process. Each Tuple Filter has a constraint as an input and the dotted lines indicate that the Tuple and Waltz Filters are executed sequentially.

that the Waltz filter would discard possible tuples, and future tuple filter stages would have less to filter thus taking less time to execute. In order to parallelise the tuple filter, Rinner decided to separate the incremental tuple and Waltz filters. The problem with this

was that now the tuple filter would have more work to do since it would have more tuples to filter, however this was balanced by the fact that now the Waltz filter needs to be executed only once. Figure 3.2 shows the data dependency of the sequential tuple filter and Waltz filters. With this sequential tuple filter it can be clearly seen that there is no

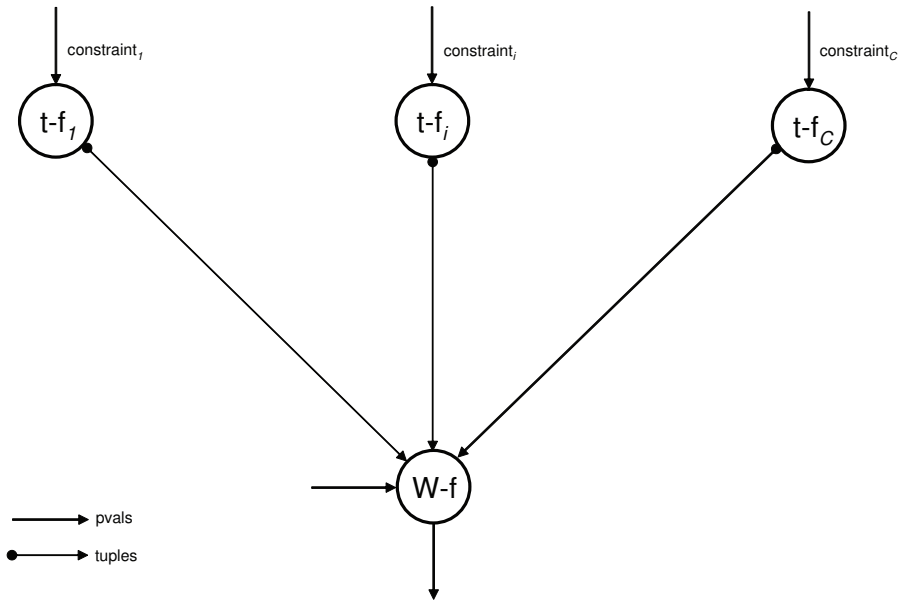


Figure 3.2: Data Dependency of the Sequential Tuple Filter and Waltz Filters. t-f denotes the Tuple Filter stages and W-f the Waltz Filter.

inter-dependency between constraints within the tuple filter. Since there is no data dependency between the constraints, it is trivial to parallelise the tuple filter - each parallel unit simply executes each constraint in turn. The disadvantage with this is that the maximum degree of parallelism is set by the number of constraints i.e. if there are fewer constraints than available parallel units then the utilisation of available resources will be sub-optimal. Rinner carries on to discuss scheduling algorithms for the tuple filter although it is not clear why since the Parallel QSIM was developed for a specific hardware setup of equal processors.

3.3.3 Form-All-States

The Form-All-States stage takes as input, a set of valid tuples for each constraint in the model and creates a solution which is a set of all possible solutions to the CSP. This differs from traditional CSPs in that they often require merely one valid solution or a select few rather than a complete set of all possible solutions.

Rinner summarises three common methods for distributing CSP algorithms, shown in table 3.1. Rinner decided to use a PAB technique and partition the problem into sub-

characteristics	*distributed constraint satisfaction strategies*		
	DAB	*PAB*	*FAB*
preferred problems	naturally distributed	tightly coupled	tightly coupled
algorithm design	specially designed	any sequential	any sequential
memory type	shared / distributed	shared / distributed	shared
communication cost	medium / high	lower	n / a
load balancing	poor / fair	good	good
scalability	poor	fair / good	reasonable
termination detection	difficult	easy	easy
find a solution	poor	fair / good	good
find more solutions	poor	good / excellent	fair

Table 3.1: Characteristics of Basic Distributed CSP Strategies (Luo et al., 1994)

problems by generating sub-sets of tuples which are used to solve sub-CSPs. Each of these CSPs can then be solved and the solution for the whole CSP is merely a union of the solutions of all the sub problems.

3.4 Critical Analysis of Parallel QSIM

Platzner and Rinner admit to their implementation being a specific hardware architecture (Platzner et al., 1995) which limits the usability of their work. To be used widely, a software solution should be developed which is not limited to a specific hardware setup, in other words the solution should be hardware-independent and portable. This would

allow it to be run on a large variety of systems and hence be used for more problems. Due to the hardware limitations, Parallel QSIM was only tested on up to seven parallel units. Whilst the tests were largely successful, it would be interesting to see how QR scales up to more parallel units for complex problems. Any successor to Parallel QSIM should determine how the efficiency of the algorithm behaves over a large range of parallel units and should be compatible with a large number of systems making it a more feasible option.

It is apparent from the design of the Parallel QSIM architecture that machine-size scalability has not been fully met. In (Platzner and Rinner, 2000) the seven processor test-bed demonstrates only a speed-up of $1.15 \leq S_7 \leq 3.5$ with the average speed-up reported as less than $\bar{S}_7 < 2$. (This gives a maximum efficiency of $E \leq 0.50$ and an average of $E < 0.29$ when running on seven processors.)

A runtime analysis shows that of the runtime of QSIM, the constraint filter occupies 80% of the total execution. The tuple filter takes approximately 70% of the constraint filter time to execute which means that the Waltz filter takes approximately 24% of the total execution time to complete. The Waltz Filter execution time would become far more apparent when the constraint filter and form-all-states are run in parallel - for the seven processor system, assuming a near-linear speed-up the Waltz Filter would run for nearly 90% of the total execution time. Even though this is the case, Platzner and Rinner have not implemented a parallel version of the Waltz Filter. One reason this may be was that the original QSIM algorithm uses an incremental Waltz Filter, and although Parallel QSIM uses a type of sequential Waltz Filter, Platzner and Rinner report that they still use some sort of incremental filtering once the result of one tuple filter is received (Platzner and Rinner, 1995). If they were to lose the idea of this incremental Waltz Filter, and instead implement a fully sequential one, further benefits of parallelisation should be apparent. Any new implementation should parallelise all stages of the algorithms including the Waltz Filter.

They concentrated on the qualitative analysis stages of QSIM. Parallelising the transition analysis phase was not attempted, therefore any future parallel system should also attempt to parallelise this stage.

These issues, combined with the limitations of QSIM itself, make it clear that there is need for a new parallel qualitative reasoning engine for complex systems.

Chapter 4

Fuzzy Qualitative Reasoning

4.1 Introduction

Fuzzy computing offers a mathematical method of dealing with vagueness. Fuzzy numbers are ranges of numbers with an associated degree of membership which states how much of a quality a given value has rather than binary 'yes' or 'no'. For example, if we define the region between 5'6" and 6' to be normal height for male adults, and 6' to 6'6" to be tall. If we have two men, A & B, whose height differs by half an inch such that A is 5'11.75" and B is 6'0.25". A would be considered normal height and B considered tall, yet their heights are almost indistinguishable. Instead we can add some extra informative detail to the regions allowing both men to be classified in the same group. Figure 4.1 shows an example of the two regions discussed in normal mathematical terms and in fuzzy terms. If we consider the fuzzy terminology and look at how 'tall' either is, then we can see that A is tall to a degree and B is tall. For this example, this is a much more informative description suggesting that both men are about the same height. It is worth noting that members may belong to more than one group at a time, both men could equally well be considered where they are placed in the 'normal height' group. It is this additional information that becomes useful for systems where uncertainty needs to be dealt with.

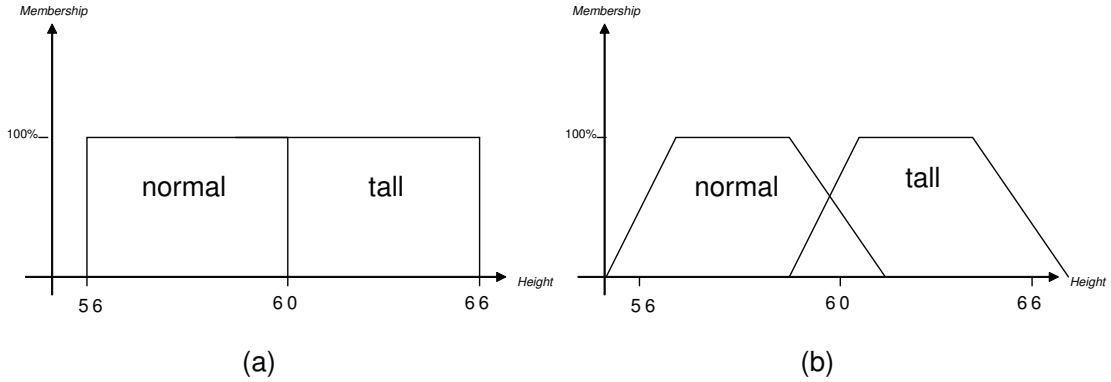


Figure 4.1: Standard Mathematical and Fuzzy Ranges.
(a) shows two ranges for male adults with a crisp range from 5'6" to 6'0" for normal height and 6' to 6'6" for tall men. (b) shows an example fuzzy quantity space for the same ranges.

Fuzzy sets are often mistakenly thought to be similar to probability distributions. The two methods both approach the problem of likelihood. However, fuzzy sets can be thought of as a representation for the degree of truth of a value, whereas probability deals with the degree of belief of an outcome to occur. Probability distributions must combine to cover all possibilities and are defined such that:

$$\sum_{i=1}^{n} P_i = 1$$

where n is the number of all possible outcomes and P_i is the probability of outcome i occurring. There is no such constraint for fuzzy numbers.

Combining qualitative reasoning with fuzzy numbers has been the motivation for creating an inference engine which can cope with ambiguity and uncertainty better than either approach alone. This chapter discusses two such engines: FuSim (Shen, 1991), an immediate successor to QSIM, and Morven (Coghill, 1996)(formerly known as Mycroft) which succeeds and improves on some of the shortcomings of FuSim by implementing many features from a combination of inference engines and offers a different algorithmic approach to solve the model equations. Only features that are used by JMorven are discussed, the interested reader is directed to the original works for a full discussion of all

features. A brief discussion of another successor to Morven, SyNCSim, is also presented.

4.2 FuSim

FuSim (Shen, 1991) was developed as a successor to QSIM adding a fuzzy number representation to the variables of the qualitative models. The motivation of this was to create a system which could deal with ambiguity and imprecision better than a purely qualitative or purely fuzzy approach. Fuzzy numbers allow a finite discretisation of the real number line which ensures that variables have a finite number of possible qualitative values they can take, and that the fuzzy numbers also cover the range of all possible values. These two properties, *Finiteness* and *Coverage*, respectively, are necessary for qualitative reasoning. Another property required is that of *Granularity* which is ensured by selecting an appropriate arbitrary discretisation of the possible ranges of variables. Fuzzy numbers offer a great advantage over crisp regions in that there is no longer an abrupt change transiting from one quantity to the next, instead the change is gradual which is closer to how people think (Zadeh, 1975a,b, 1976). Another advantage to using fuzzy numbers is that due to the semi-quantitative nature, the prerequsites for temporal calculations are readily available in a numerical form therefore fuzzy numbers lend themselves better toward simulation than a purely qualitative approach. This was a motivation of Shen, in particular to create a diagnosis system for continuous dynamic systems.

The following sub-sections describe certain features of FuSim that are of importance to the subject of this thesis.

4.2.1 Fuzzy Four-Tuple Parametric Representation

Fuzzy numbers provide a so-called *soft* boundary for numeric regions, allowing a degree of membership. These regions take shapes similar to probability distribution curves,

although they have subtle differences. Figure 4.2 shows a diagram of an example fuzzy region. It can be seen that the shape of the region is curved and not very efficient to imple-

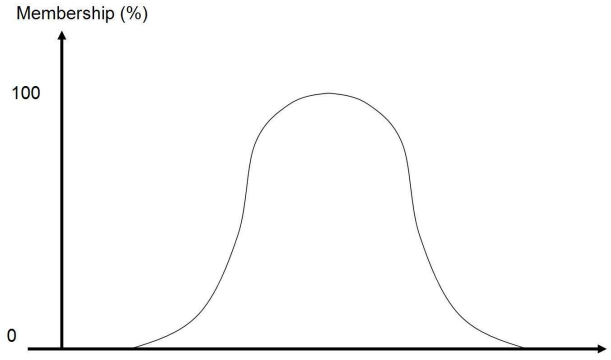


Figure 4.2: Typical Fuzzy Number.

ment nor define arithmetic operations. To overcome this, a simpler representation is used in FuSim which approximates the fuzzy number curve whilst still offering the benefits of fuzzy membership. This representation is known as the Fuzzy Four-Tuple Parametric Representation and offers a much more efficient method to utilise fuzzy numbers. A four-tuple is described using four values a, b, α and β which define the fuzzy number as shown:

$$\mu_A(x) = \begin{cases} 0 & x < a - \alpha \\ \alpha^{-1}(x - a + \alpha) & x \in [a - \alpha \quad a] \\ 1 & x \in [a \quad b] \\ \beta^{-1}(b + \beta - x) & x \in [b \quad b + \beta] \\ 0 & x > b + \beta \end{cases}$$

This is shown graphically in figure 4.3. This representation makes it possible to describe real numbers, real intervals, fuzzy numbers and fuzzy intervals very easily.

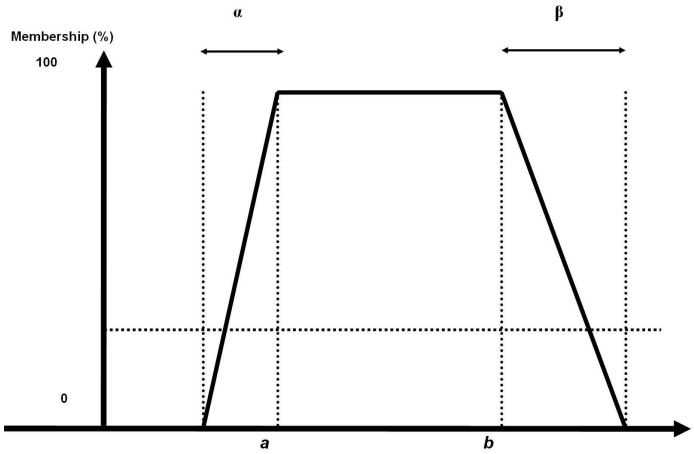


Figure 4.3: Fuzzy Four-Tuple Parametric Representation.

4.2.2 Fuzzy Quantity Spaces

Variables have a domain of quantities from which they can take their values, known as a quantity space. A quantity space in FuSim is a set of convex fuzzy values spanning a region of the real number line, as shown in figure 4.4. Usually the quantities in the

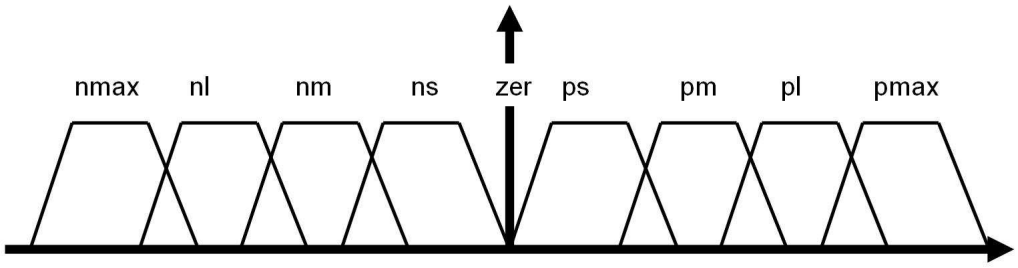


Figure 4.4: Fuzzy Quantity Space.

quantity space overlap giving some degree of ambiguity. Variables can adopt their own associated quantity space and can have different quantity spaces for the magnitude and derivative. The granularity of the quantity space dictates the number of possible tuples that are valid for a constraint; increasing the number of quantities results in a larger number of states being produced which takes longer to calculate. However, this provides more detailed information about the states. It is worth noting that the signs can be defined by the quantity space shown in table 4.1

Quantity	a	b	α	β
-	$-\infty$	0	0	0
0	0	0	0	0
+	0	∞	0	0

Table 4.1: The Signs Quantity Space

4.2.3 Fuzzy Arithmetic in FuSim

To be able to use fuzzy quantities in the constraints, a set of arithmetic operations need to be defined. Table 4.2 shows the definition of arithmetic operations used within FuSim. With the common arithmetic operations defined, it is possible to calculate resulting fuzzy quantities from the constraints. The next section describes how this result is then used to determine which fuzzy quantities are used from the quantity space.

Let:	$m = [a, b, \tau, \beta], n = [c, d, \gamma, \delta]$	
Operation	Result	Conditions
$-n$	$(-d, -c, \delta, \gamma)$	all n
$\frac{1}{n}$	$\left(\frac{1}{d}, \frac{1}{c}, \frac{\delta}{d(d+\delta)}, \frac{\gamma}{c(c-\gamma)}\right)$	$n >_0 0, n <_0 0$
$m + n$	$(a + c, b + d, \tau + \gamma, \beta + \delta)$	all m, n
$m - n$	$(a - d, b - c, \tau + \delta, \beta + \gamma)$	all m, n
$m \times n$	$(ac, bd, a\gamma + c\tau - \tau\gamma, b\delta + d\beta + \beta\delta)$	$m >_0 0, n >_0 0$
	$(ad, bc, d\tau - a\delta + \tau\delta, -b\gamma + c\beta - \beta\gamma)$	$m <_0 0, n >_0 0$
	$(bc, ad, b\gamma - c\beta + \beta\gamma, -d\tau + a\delta - \tau\delta)$	$m >_0 0, n <_0 0$
	$bd, ac, -b\delta - d\beta - \beta\delta, -a\gamma - c\tau + \tau\gamma)$	$m <_0 0, n <_0 0$

Table 4.2: Arithmetic primitives used in FuSim

4.2.4 α-cuts

α-cuts are used in FuSim to generate a crisp quantity from a fuzzy quantity. An α-value is used to dictate the cut-off point for the conversion. Any point in the fuzzy quantity above this α-value is converted to the crisp quantity. An α-value of 0 means that the whole range of the fuzzy number is used, i.e. from $a - \alpha$ to $b + \beta$ whereas an α-value of 1 means that only the region from a to b is used. FuSim uses an α-value that is is greater than all the crossing points of adjacent quantities in the quantity space. This ensures that

no crisp quantities overlap and therefore the temporal calculations remain positive. α-cuts represent the idea of typicality by dictating the minimum membership of a quantity for it to be considered a member which narrows the overall range of a fuzzy number. Figure 4.5 shows a quantity space with and without the α-cut. α-cuts are also used in the approximation principle to reduce the number of possible consistent values a calculated fuzzy interval may be approximated to. See the next section for more information.

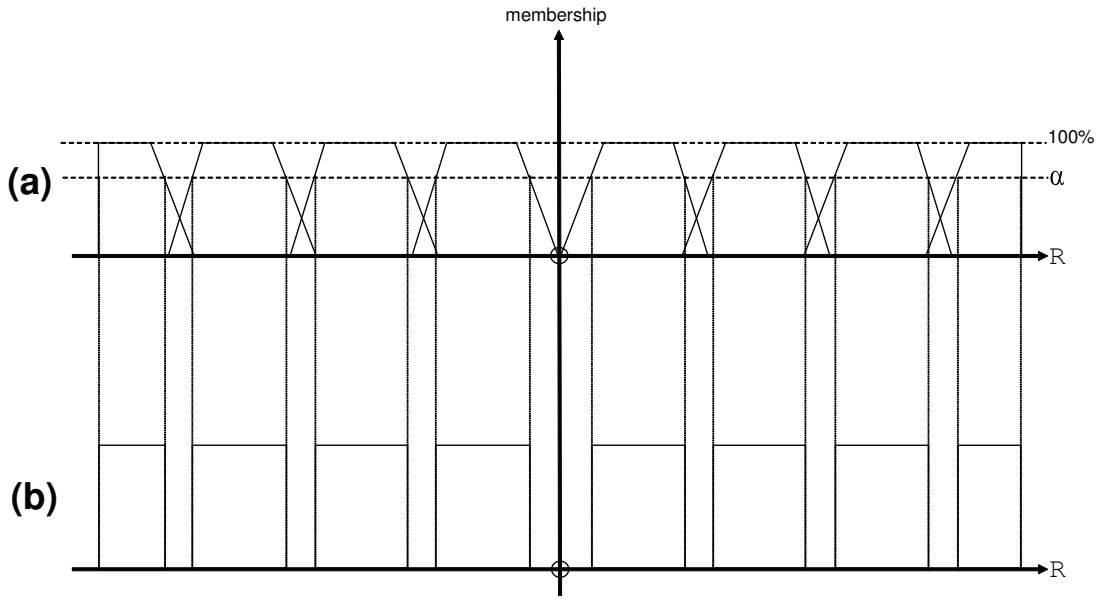


Figure 4.5: Taking the Alpha-cut of a Quantity Space
(a) shows a typical fuzzy quantity space and the alpha-cut value to take. (b) shows the result of taking the alpha-cut of the quantity space.

4.2.5 Approximation Principle

When a constraint is evaluated it is likely that the calculated fuzzy result will not be exactly equal to a quantity from the associated quantity space. In this case, the *Approximation Principle* is used to determine which quantities from the quantity space the calculated value corresponds to. The approximation principle states that the calculated value can approximate by any quantity in the quantity space that overlaps with it. To ensure that no excessive values are included, all quantities and the calculated value are restricted to a

crisp interval using the α-cut. Figure 4.6 shows an example of the approximation principle. It can be seen that the calculated value overlaps with the second, third and fourth quantities in the quantity space, therefore each of these quantities are added to the possible values for the calculation.

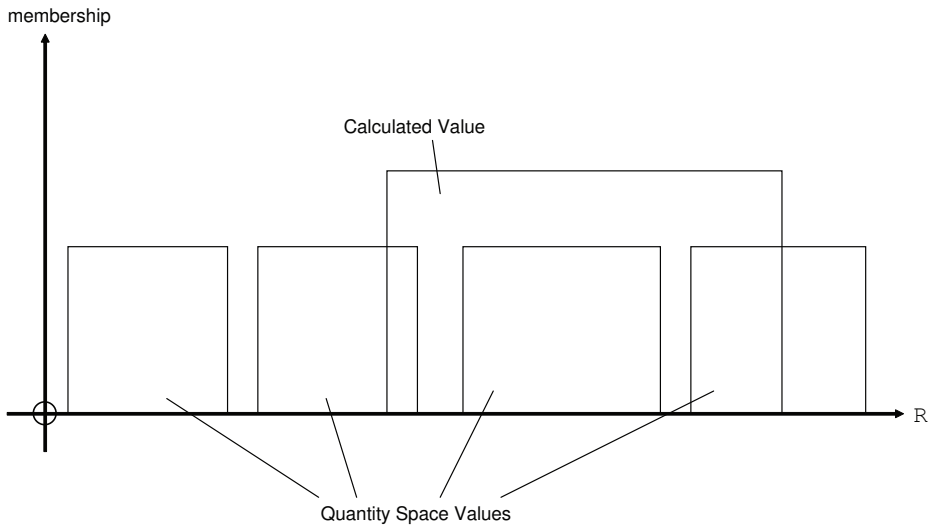


Figure 4.6: The Approximation Principle.

4.2.6 Fuzzy Derivatives

FuSim uses fuzzy quantities for the derivatives of a variable as well as the magnitude. This extra information allows temporal calculations to be made more easily offering a more informative output than using merely the purely qualitative direction of change as in QSIM. Derivatives take on fuzzy values from the quantity space in the same way as the magnitude. With a purely qualitative representation a variable may be known to be increasing, but this could be very slowly or very fast. With the extra information from the fuzzy values it is possible to see how quickly it is changing allowing more precise simulation results.

4.2.7 Function Constraints

QSIM uses monotonic functions to describe the behaviour between two or more variables when the exact mathematical relationship is not known. Whilst this is useful, it is a weak representation especially when it is possible to have the extra information in the form of fuzzy values. FuSim uses Functional Constraints which allows any mapping of possible values for one variable to be consistent with another. This can vary from linear relationships to complex non-linear ones. This addition allows a stronger relationship between variables without knowing the exact mathematical relationship. The variable relations can be thought of as a set of mappings, e.g. for a given function if the left hand side is medium then the right hand side could be either zero or large. There is no constraint on the number of mappings and they can be disjoint. The whole function is described using a lookup table; an example function constraint is shown below in table 4.3.

$A \mapsto B$	zero	small	medium	large	top
zero	1	0	0	0	0
small	0	1	0	0	0
medium	0	0	1	1	0
large	0	0	1	1	1
top	0	0	0	0	1

Table 4.3: Function Constraint in FuSim

4.2.8 State Transitions

As in QSIM, FuSim defines a set of valid state transitions to determine how a system behaves over time. FuSim considers four different types of transitions from variable $\langle A_1, B_1 \rangle$ to $\langle A_2, B_2 \rangle$ summarised as follows:

- **Null-transitions** – The null transition is when no transition takes place, i.e. when $A_1 = A_2$ and $B_1 = B_2$
- **M-transitions** – Magnitude transitions occur when only the magnitude changes, the value of the derivative remains constant, i.e. when $A_1 \neq A_2$ and $B_1 = B_2$
- **R-transitions** – A rate transition is defined as when the derivative changes but the magnitude remains constant, i.e. when $A_1 = A_2$ and $B_1 \neq B_2$
- **MR-transitions** – The final type of transition is the magnitude/rate transition and this occurs when both the magnitude and derivative change instantaneously, i.e. when $A_1 \neq A_2$ and $B_1 \neq B_2$

One restriction is used to the above; if a variable has a real number as the magnitude, then an R-transition is not allowed to occur. This emphasises how the transitions occur around zero which is often implemented as a real number with $a = b = \alpha = \beta = 0$.

4.2.9 Algorithm

Since FuSim is largely based on QSIM, the algorithm has many similarities. Like QSIM, FuSim requires a model in the form of a set of constraints, and an initial state to which it then produces an output of all the possible behaviours of the system. FuSim achieves this by progressing through the following algorithm:

1. Generate a set of valid transitions for each variable in the current State, and calculate the temporal information associated to each transition.
2. Filter the qualitative values for each individual constraint to ensure they are consistent with the model.
3. Filter adjacent constraints to ensure pairwise consistency.

4. Calculate arrival time for all variables and use temporal filtering to remove conflicting values[1].

5. Generate states and use global filtering to remove further states. Mark each remaining state as a successor to the current state.

6. Repeat 1→ 5 until no more changes are observed or a resource limit is met.

As with QSIM, FuSim combines stages 2 & 3 to create an incremental Waltz Filter. The above algorithm has many similarities with the QSIM algorithm, however one main difference is the inclusion of advanced temporal calculations which give FuSim an advantage over other QR techniques. These temporal calculations offer more information as to when states can exist and the amount of time they can exist for, and are a benefit of using fuzzy numbers over purely qualitative values.

4.3 Morven

The motivation of Morven (Coghill, 1996) was to create a constructive qualitative reasoning engine for use in a model-based diagnostic system (MBDS). The motivation behind using constructive techniques was to decrease the number of spurious behaviours. Coghill started by completing an in-depth review of many existing QR techniques and created an implementation which included all the advantages of his research into a single system. The result was a novel framework for fuzzy qualitative reasoning allowing the choice of several algorithms. Additional features to this framework are discussed below along with the central algorithms to Morven.

[1]The original FuSim algorithm used temporal filtering to remove conflicting values however the method was incorrect. Coghill (Coghill, 1996) corrected the temporal filtering calculations and shows that no values can be removed at this stage.

4.3.1 Differential Planes

In QSIM and similar QR engines the variable and model representation is fixed which is quite restrictive in that all derivatives of equations are calculated implicitly. The Predictive Engine (Wiegand and Leitch, 1989) utilises the Predictive Algorithm (Wiegand, 1991) which allows more flexibility in the mode of description. *Differential Planes* were introduced which allow the model to be described at a more detailed level. The first differential plane describes the model in a similar manner to that in which a typical numerical simulator would require. Further differential planes describe the model with further derivatives allowing for more accurate simulation qualitatively. These further differential planes are merely the derivative of the previous one but allow more structural information to be included in the model explicitly. The motivation for this was to reduce the number of spurious behaviours generated by QR systems but in fact it was shown (Coghill, 1996) that they do not. However, differential planes allow the modeller to have control over the number of derivatives for each variable and also allow simpler equations to be used for the derivatives resulting in less states being generated in an envisionment if required.

4.3.2 Fuzzy Vector Envisionment

Vector Envisionment (Morgan, 1988; Coghill, 1992) is another constraint based QR engine, however it only reasons purely qualitatively, i.e. with the $\{+\ 0\ -\}$ quantity space. Morgan introduced multiple derivatives for variables which allows the distinction between linear and non-linear systems which was not possible in QSIM's monotonic function representation. With these extra derivatives it is possible not only to determine the rate of change of a variable, but also the curvature which is very useful. Although this can be achieved in QSIM by explicitly stating that one variable is the derivative of another or by the use of extensions, vector envisionment does this automatically for all variables. Variables are defined using a vector of length equal to the number of derivatives to reason with, e.g. $V = [+ \quad + \quad -]$ states that variable V is positive and increasing, but the

amount it is increasing by is decreasing.

Morven furthers vector envisionment by extending it to the fuzzy domain and terms the feature *Fuzzy Vector Envisionment*. This is using the vector envisionment approach of multiple derivatives per variable but also allowing these to be fuzzy values instead of pure qualitative values. Fuzzy Vector Envisionment and Differential Planes are combined to provide a very powerful simulation technique offering far more information than previously possible with QR approaches. A variable and its derivatives is termed a Variable Vector in Morven.

4.3.3 Constructive Simulation

The distinction between constructive and non-constructive approaches was first hypothesised by (Wiegand, 1991). QSIM and its derivatives use a non-constructive approach to simulation. It was believed that due to this approach extraneous spurious behaviours are generated therefore Wiegand implemented a constructive technique similar to that used in numerical simulators.

Morven incorporates two simulation algorithms, one based on a constructive approach and another semi-constructive approach. The reader is directed to (Coghill, 1996) for details about the constructive algorithms.

4.3.4 Auxiliary Variables

Morven, like its predecessors, uses constraints that are specified using two or three variables. When creating these two or three variable constraints from differential equations it is often required to create a temporary variable. These temporary variables define a range of values which do not necessarily map exactly to any quantities from the quantity space.

QSIM and similar approaches map this temporary variable into the quantity space using the approximation principle thus unnecessarily widening the variable's range an therefore creating unnecessary additional states. To reduce the number of spurious behaviours generated, Morven introduced the use of auxiliary variables. These are variables which are not mapped back to any quantity space, instead the value of an auxiliary variable is kept temporarily for use between constraints. This is made possible by vector envisionment which allows variables to have any number of derivatives including none thus leaving only the magnitude of a variable. An example of the use of an auxiliary variable is given below. If we have the following qualitative differential equation

$$\dot{x} = Px + Q$$

This would be broken down for use in JMorven as shown:

$$auxA = P.x$$

$$\dot{x} = auxA + Q$$

Variable $auxA$ is a temporary variable and as such should not be mapped to a quantity space otherwise spurious states may be generated due to the approximation principle.

4.4 SyNCSim

It is worth mentioning one other fuzzy qualitative reasoner since it also attempts to carry out qualitative simulations in a non-constructive manner. SyNCSim (Bartlett, 2005) is a successor to Morven but approaches the problem of simulation using a non-constructive algorithm allowing it to reason with models which are not necessarily causally ordered, or that contain algebraic loops.

Models in SyNCSim are defined in a similar manner to those in its predecessor, using Qualitative Differential Equations to specify the constraints of the system and using fuzzy quantity spaces to define the domains of the variables. As in Morven, SyNCSim also uses differential planes to explicitly define the derivatives of a model and utilises fuzzy vector envisionment to represent variable values across these differential planes.

The simulations in SyNCSim are carried out synchronously using an internal clock instead of asynchronously as in most existing qualitative reasoners. The motivation behind this approach was to generate an output that not only predicts the order in which events occur but also predicts accurate times between events. At each time step all of the model variables in an initial state are propagated using Euler integration and obeying continuity constraints. If the continuity constraint is breached, the time-step is reduced to the maximum allowable step which ensures the continuity constraint remains consistent; this is termed *Minimum Interval Euler Integration* (Scott and Coghill, 1998). For example, if variable V is currently $p - small$ and after a time-step, δt, the integration phase dictates that V will become $p - large$ then the continuity constraint is breached since V does not pass through $p - medium$. SyNCSim therefore reduces the time-step, δt_{opt}, until V is observed to pass through $p - medium$. This is effectively a form of step-size refinement for use in qualitative simulation. Once the successor values have been calculated, all permutations of the propagated quantities are used to generate states which are then tested for consistency with the constraints. The remaining consistent states are then marked as successor states to the initial state.

The SyNCSim approach offers a method of non-constructive, synchronous qualitative simulation and the results confirm that it is possible to use non-constructive approaches in a synchronous simulator.

4.5 Summary

The purpose of this chapter was to introduce the field of fuzzy numbers for representing numerical imprecision and their use in the field of qualitative reasoning. The combination of fuzzy numbers and qualitative reasoning inspired the development of FuSim and its successors. The fuzzy four-tuple parametric representation is briefly discussed as a method to simplify how fuzzy numbers are defined. This allows fuzzy numbers and the operations on them to be implemented more efficiently than using 'real' fuzzy regions. Fuzzy quantity spaces are defined as a method to specify which fuzzy values are available. This discretization of the real number line allows a qualitative approach to fuzzy numbers and is adopted by FuSim and its successors. Several techniques to aid the use of fuzzy numbers were introduced in FuSim. A fuzzy arithmetic was defined for the core arithmetic functions. α-cuts are also used as a means to convert fuzzy numbers into real crisp intervals aiding temporal calculations and the approximation principle. The approximation principle is used to determine which values from the quantity space a calculated value can be approximated by. Setting a low α-cut means that more quantities approximate the calculated value, whereas a higher α-cut results in less quantities approximating it.

Morven incorporated several useful features over FuSim. The use of multiple derivatives per variable allows the behaviour over time to be simulated more accurately. The addition of differential planes allows extra information to be inherent in the model again increasing the accuracy of simulations. One major improvement in Morven was the addition of auxiliary variables. These are temporary values which are used when equations are broken down into 2 or 3 variable constraints. Since these variables are temporary, no quantity space is assigned which allows the exact values to be used across constraints and reducing the number of spurious solutions.

Chapter 5

Simulation

5.1 Introduction to Simulation

Numerical simulation has been studied for many years. There are many well-known, tried and tested techniques to simulate the behaviour of a system. The most common of these techniques includes Euler Integration, Taylor Series Expansion and Runge-Kutta methods. The approach is very similar in each case, a system of equations are expressed as several ordinary differential equations. These equations are used to estimate the successive values of the variables using some integration techniques. Taylor Series Expansion (and Euler Integration since it is equivalent to a first order Taylor Expansion) uses the current values of derivatives to estimate the values after some time step $\delta t = x - a$. The general formula for Taylor Series is:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + ... + \frac{f^n(a)}{n!}(x-a)^n + ...$$

where $f(x)$ is the estimated value of the function after the time-step and the current value of the system variable is denoted at time $t = a$. Below shows Euler Integration in a form

more familiar to engineers:

$$f(x)_{t=t_0+\delta t} = f(x)_{t=t_0} + \delta t.f'(x)_{t=t_0}$$

Euler Integration is used qualitatively by Morven (Scott and Coghill, 1998) which replaces the transition rules for determining how qualitative states transit between one another. The downside of Euler Integration is that it is not very accurate and can introduce quite large errors into a simulation.

More advanced integration techniques can be used to increase the accuracy of the integration phase. The following equations define the fourth-order Runge-Kutta process (Press et al., 1992):

$$k_1 = h.f(x_n, y_n)$$

$$k_2 = h.f(x_n + \frac{h}{2}, y_n + \frac{k_1}{2})$$

$$k_3 = h.f(x_n + \frac{h}{2}, y_n + \frac{k_2}{2})$$

$$k_4 = h.f(x_n + h, y_n + k_3)$$

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)$$

where h is the time step. The first equation estimates the difference in y before and after the time step. The second equation then uses this to find the difference in half of the time-step more accurately. The third equation then splits the time-step again to get the difference in y more accurately yet again. The last equation then uses these calculations to provide the final estimation for the value of y after the time-step, i.e. y_{n+1}.

This method estimates the gradient of each variable at several intermediate points between the current time and the goal time. These estimates are then combined to estimate the

real gradient to calculate the values at the goal time. Higher order Runge-Kutta methods allow larger step sizes to be used without losing any further accuracy, however the optimal step-size cannot be known for all systems therefore some sort of step-size refinement is common in many algorithms. Froese (Froese, 1961) provides an evaluation of Runge-Kutta type methods for the interested reader.

The methods of simulation described above are termed synchronous simulation, or clock-driven simulation. It differs from asynchronous simulation as discussed in section 2.3.4 in that all variables are propagated at regular time-steps even if no qualitative behavioural changes occur. Asynchronous simulation is driven by events or changes in behaviour whereas synchronous simulation is driven by a clock.

Numerical simulation is a proven technique to estimate how dynamic systems behave over time however they cannot handle imprecision. One method to add imprecision is to use Interval Mathematics. Interval Mathematics is a well researched area and several good text books are available discussing the area, including (Moore, 1966, 1979; Lohner, 1987; Alefeld and Herzberger, 1983). The main problem of interval simulation is that intervals widen unnecessarily over time when using them without any additional knowledge of the rest of the model (termed non-interacting). An example of this is if we take

$$y = 1 - x$$

$$z = 3x + y$$

and set

$$x = [1 \quad 2]$$

then we get

$$y = 1 - [1 \quad 2] = [-1 \quad 0]$$

$$z = 3[1 \quad 2] + [-1 \quad 0] = [2 \quad 6]$$

however solving the two equations to eliminate y shows that the result should in fact be

$$z = [3 \quad 5]$$

This above process generates an approximate hypercube with each side representing the interval of each individual variable after each equation has been solved. The interacting method proposed by Moore (Moore, 1966) uses a connection matrix to transform the co-ordinates using the partial derivatives of the model. This matrix is created using the partial derivatives of the system variables and is created once for each time-step. Using the connection matrix and the associated Jacobian matrix it is possible to calculate the intervals without generating hypercubes hence no unnecessary divergence occurs.

Numerical Simulation is a very useful technique but, as with the main motivation behind Qualitative Reasoning, sometimes only imprecise or incomplete information is known. Creating qualitative envisionments of complex systems can produce a very large number of states which takes a long time to produce therefore simulations are often better suited (Milne, 1991). In this case a combination of the two methods allows behaviours to be generated in reasonable lengths of time. Semi-quantitative techniques can be used to conduct simulation with the knowledge available. This is very useful as it allows analysis of systems with incomplete or imprecise knowledge whereas numerical techniques do not. The rest of this chapter discusses several semi-quantitative simulators that are known in the field including QSIM-types Q2, Q3, NSIM and SQSIM as well as some fuzzy simulators, QuaSi, FRenSi and QFSIM.

5.2 QSIM-based Numerical Simulators

In this section, a handful of semi-quantitative simulation engines based around QSIM are discussed with some comments about their performance.

5.2.1 Q2

The motivation behind Q2 (Kuipers and Berleant, 1988) was to implement a system that would allow numerical information to be embedded into qualitative models allowing extra precision to be included. Not all variables in a system would have associated numerical information therefore traditional simulation techniques could not predict the behaviours of a system, yet a qualitative model may benefit from the extra information and produce a more accurate behaviour, i.e. one with fewer spurious behaviours. Additional numerical information in Q2 can be of one of two types:

- Quantitative landmark ranges $[min \quad max]$
- Bounding envelopes of monotonic function

The range propagator used is similar to that of Davis (Davis, 1987) which sets an initial range for all landmarks which is then narrowed using the constraints until all ranges will not narrow from further propagation. This results in a range or inequality for each variable in the system. These values are used to solve the CSP. Q2 also uses the Mean Value Theorem to propagate values across temporal landmarks.

The additional numerical information in Q2 is used to augment the qualitative descriptions, and as such is only available at landmark time-points similar to the qualitative state descriptions in QSIM. Q2 can therefore be thought of as a special type of global filter for QSIM.

One major disadvantage of Q2 is that the time step is very coarse hence a lot of spurious errors are generated within the intervals calculated.

5.2.2 Q3

Q3 (Berleant and Kuipers, 1990, 1997) extends Q2 by introducing step-size refinement. It achieves this by detecting gaps in the behaviour generated by QSIM and Q2, and then inserts new auxiliary states within this gap. The new auxiliary states are interpolated to help refine the ranges of the states. This is continued until the results are sufficiently precise or until no further narrowing of the ranges can occur. The motivation behind this work was to create a detailed numerical behaviour with more precise results, and thereby also improve the generated qualitative behaviour; thus bridging the gap between qualitative and quantitative simulation. Both Q2 and Q3 maintain the completeness of QSIM however they fail to stop spurious behaviour generation hence are unsound as QSIM was.

Q3 improves on the predicted simulation of a model by providing more information between time-points. It is the belief of the author of this thesis that a system that does not rely on refining qualitative simulation to generate numerical information would be more precise and offer better simulations.

5.2.3 NSIM

The motivation behind NSIM (Kay and Kuipers, 1992, 1993) was to create a simulation engine which could make use of any numerical knowledge in addition to the fully qualitative models. This was thought to be of most importance in monitoring tasks where it is vital to detect an anomaly as early as possible.

Models in NSIM are described using Structural Differential Equations (SDEs), Qualitative Differential Equations (QDEs), and Semi-Quantitative Differential Equations (SQDEs). The SDEs describe the form of the ODE variables qualitatively and the corresponding constraints. These constraints, as in QSIM, are described by arithmetic operators and functional relationships. In addition to QSIM, NSIM describes the functional constraints in more detail, e.g. if they are monotonic, parabolic, sigmoidal etc. Finally, the SQDEs represent the numerical imprecision of the models. Parameters of the model are described by intervals and functional constraints are defined using dynamic envelopes which dictate how two variables change with respect to each other with more information than qualitative functions but not requiring a strict mathematical equation. The QDEs describe the functional relationships and define the domain of each variable used in the SDEs, i.e. the landmark values.

NSIM evaluates the upper and lower bounds of each constraint to predict the behaviours. Interval arithmetic is used to propagate these bounds which results in widening of the intervals. One problem of only using the upper and lower bounds of variables is that this does not always guarantee to enclose all solutions of constraints. For example, if the interval $A = [-1 \quad 2]$ is taken and one constraint is defined as

$$x = A^2$$

then using the extreme points method, the calculated result would be $x = [1 \quad 4]$ however the real result should be $x = [0 \quad 4]$.

The output behaviours of NSIM are claimed to result in tighter bounds than are produced by Q2, although Kay (Kay, 1998) admits that this is only before a certain amount of time has passed as NSIM behaviours diverge whereas Q2 bounds remain more stable. NSIM aimed to provide a simulation strategy that improved its predictions as the amount of information known increased - NSIM is a step in the right direction for such a system.

NSIM is now used mainly as part of the SQSIM package as described in the next section.

5.2.4 SQSIM

SQSIM (Kay, 1998) is a semi-quantitative simulation engine based on QSIM and uses the same model representation as NSIM. SQSIM makes use of the semi-quantitative simulation to refine the behaviour tree of the qualitative simulation thus reducing the number of spurious behaviours generated. It also uses the qualitative predictions to aid some of the semi-quantitative inferences, e.g. if the derivative of variable A is calculated semi-quantitatively to be in the range $[-1 \quad 1]$ and the qualitative prediction is that $qmag(A) = inc$ then the semi-quantitative range can be reduced to $(0 \quad 1]$.

SQSIM combines the inferences made by QSIM, Q2 and NSIM and thus reduce the imprecision in the predictions made by each. QSIM is used as the central simulation process. Q2 and NSIM are then used to augment numerical information and envelopes to the predictions and finally SQSIM combines the predictions to generate the semi-quantitative states. There are several ways in which QSIM, Q2, and NSIM are combined which are briefly outlined below:

- **Dynamic Envelope Intersection** Envelopes are generated with NSIM and Q2 separately. NSIM produces an output envelope which is initially very well bound but the interval arithmetic causes it to widen after a time, whereas Q2 defines a constantly wide envelope. SQSIM combines these two envelopes using NSIM initially until the envelope produced by Q2 is tighter.

- **Event Intersection** The envelope prediction from NSIM is combined with the event descriptions from Q2 which can reduce the event ranges and time taken for each event. SQSIM uses this which can also lead to further reductions in other events in the system.

- **Extremum Detection** This is when the envelope produced by NSIM contains an extremum: a local minimum or maximum. When the lower and upper bounds of the NSIM envelope exhibit this extremum, the QSIM behaviour is modified to ensure that it also contains this local behaviour.

- **Order Reduction** Order Reduction is when the NSIM envelope diverges beyond the Q2 envelope extremities, i.e. if the maximum and minimum values of the Q2 envelope are surpassed by the NSIM envelope, then the Q2 minimum and maximum values are used for the magnitude of the envelope and the derivative is set to zero.

- **Re-Simulation** SQSIM uses the previous states to predict values for successor states, however sometimes the final states can be used to narrow predecessor states particularly when equilibria are reached. In this case, re-simulation can help reduce the predicted behaviour between the initial and final states.

Since SQSIM combines several other inference engines, it succeeds in improving on them individually however it still has a few drawbacks. Generated envelopes can still suffer from rapid divergence resulting in a predicted output which differs greatly from the real output. Since models are based on the QSIM specification, variables are limited to one derivative hence only Euler integration can be used which introduces errors in integration. These simulators also infer single intervals, inferring about fuzzy ranges is thought to be more useful for ambiguous models.

5.3 Fuzzy Simulators

In the following sub-sections semi-quantitative simulation engines based on fuzzy numbers are presented.

5.3.1 QFSIM

QFSIM (Vescovi and Travé-Massuyès, 1992) was motivated by integrating numerical simulation methods into a qualitative environment. This is achieved using fuzzy numbers to instantiate qualities with some numerical information. QFSIM uses the same fuzzy four-tuple representation and fuzzy quantity space definition as used in FuSim (Shen, 1991). QFSIM presents the following two methods:

- **The Extremity Method**. This method extends Euler's method to incorporate fuzzy operators based on the Extension Principle (Zadeh, 1965). The method is reported to be complete but not sound in that it predicts all possible outcomes of a model but also contains spurious behaviours. The extremity method has one main disadvantage in that it is not easily generalized for complex models.
- **The Discretisation Method**. This method involves generating a set of discrete points from the fuzzy parameters and simulating this group of points to produce an output that is sound but not complete. This method is also used in their Qualitative Behaviour Generation stage to produce a global output from the group of simulations of points.

Problems with QFSIM are that the method is only applicable to first and second order systems; this limitation restricts the number and complexity of models that it can reason with. QFSIM also uses a constructive approach which has limitations as mentioned in chapter 2.4.1.

5.3.2 QuaSi

QuaSi (Bonarini and Bontempi, 1994a) is a framework of simulators for simulating systems with imprecision using fuzzy numbers. QuaSi is based on the extension principle

which is used to map an input fuzzy relation to some output when applied to a crisp algebraic system. However, this requires an infinite number of calculations for the fuzzy representation used in QuaSi therefore an approximation proposed by Nguyen (Nguyen, 1978) is used which approximates the fuzzy region using multiple α-cuts to generate intervals which can then be used by normal interval arithmetic (Moore, 1966). There are three main algorithms within the framework, each of which are briefly described below:

5.3.2.1 QuaSi I

The original QuaSi I (Bonarini and Bontempi, 1994c) algorithm uses fuzzy numbers and either a system of ODEs or a set of fuzzy rules to define the model to be simulated. It works along the lines of a typical numerical simulator in that at each time step the current values are used for numerical integration and then the model is used to construct the rest of the values. This is extended to intervals using the non-interacting approach of Moore's interval arithmetic (Moore, 1966). Since QuaSi uses fuzzy numbers, several α-cuts are taken to discretize the fuzzy values which are then simulated. The resulting simulator is guaranteed to bound all of the results of the real simulation however it suffers from excessive widening of the intervals due to approximating the system variables as an n-hypercube at each step.

5.3.2.2 QuaSi II

QuaSi II (Bonarini and Bontempi, 1994b) incorporates the interacting approach presented by Moore based on the connection and Jacobian matrices. Instead of approximating the variables as an n-hypercube, this method maintains the interactions between the variables and hence produces results which contain far fewer spurious errors. They achieve this using the property of sufficiency of vertices (or PSV) which determines if it is possible to compute any intervals from the extreme points only. This test is not sufficient for all

systems therefore the correct results cannot be guaranteed however QuaSi II does not suffer from unnecessary divergence of the simulated intervals. Since QuaSi II is based on the interacting approach, it therefore must use a constructive approach to simulation as the system equations are used to construct the connection and Jacobian matrices. Hence this method cannot be used to develop a non-constructive algorithm for simulation.

5.3.2.3 QuaSi III

QuaSi III (Bontempi, 1996) extends QuaSi II by introducing an optimisation technique to replace the original sampling problem. The optimisation works by first taking an initial starting point, integrating and then using the gradient to sample a new point. If a maximum or minimum is found, then the optimisation routine halts. This improves upon the PSV problem in QuaSi II although it still does not guarantee to bound all real results.

QuaSi presents an interesting simulation engine which produces positive results; however there are a few drawbacks. QuaSi exhibits exponential complexity with the order of the system being simulated. Computationally it is also very intensive when simulations need to be achieved over a long time since at each time-step the optimisations need to be carried out. Finally, the QuaSi approach is constructive therefore it cannot cope with algebraic loops.

5.3.3 FRenSi

FRenSi (Keller et al., 1999) is a fuzzy simulator based on the QuaSi approach. FRenSi generates an N-hypercube[1] fuzzy region and the external surface is approximated using corner points and cubic splines. FRenSi suffers from the same problem as QuaSi in that the computational complexity is too great for it to be used in practice. In his thesis Keller

[1] $N = n + k$ where n is the number of system variables in the model and k is the number of parameters

(Keller, 1999) outlines a simplified FRenSi algorithm which is less processor-intensive. This simpler algorithm takes the fuzzy regions and splits them into α-cuts and the corners are numerically integrated as before, however instead of using splines to describe the borders of the hypercube, several equally spaced samples are used instead. At each step, the minimum and maximum values of each α-cut of each variable are recorded. What results is a simulator which maintains a similar level of error yet executes in much less time. Unfortunately, FRenSi still uses constructive methods therefore does not offer a technique that can reason with more general systems as non-constructive techniques can.

5.4 Summary

Of the semi-quantitative simulators described above, those based on the non-interacting approaches of Moore's interval arithmetic use Euler's method of integration which introduces unnecessary errors. Even going one step further and using a second order integration technique would be of great advantage to reducing these errors. Unfortunately since the simulators are based on QSIM and its derivatives, it is only possible to use first order integration methods as only one derivative of each variable is available within the model. These simulators also tend to have problems with unnecessary interval divergence which introduces spurious results.

Methods based on the interacting approach offer a good simulation with few errors however these methods require a lot of computations and therefore are not suited for practical use. Moreover the interacting method requires a constructive approach to solving the system equations and therefore cannot cope with algebraic loops (and also require the equations to be ordered). Table 5.1 summarises the simulators discussed in this chapter.

It is clear that there is room for a novel simulator which would operate non-constructively yet be able to simulate with few integration errors and no spurious results due to diverging

intervals. It would also be beneficial for it to be efficient for practical use.

Simulator	Interacting?	Constructive?	Fuzzy	Disadvantages
Q2	N	N	N	Coarse time-step Asynchronous
Q3	N	N	N	Maintains spurious behaviours of Q2 Asynchronous
NSIM	N	N	N	Interval divergence Extreme points not complete
SQSIM	N	N	N	Interval divergence Asynchronous
QFSIM	N	Y	Y	Doesn't handle complex models Constructive
QuaSi	Y	Y	Y	Constructive Very slow
FRenSi	Y	Y	Y	Constructive

Table 5.1: Summary of existing semi-quantitative simulators

Chapter 6

JMorven

6.1 Introduction to JMorven

JMorven is a novel abstract parallel architecture framework for qualitative reasoning and simulation capable of reasoning in a fully qualitative manner, a semi-quantitative manner and a fully numerical manner. This is all achieved using non-constructive algorithms which are more general, being able to cope with systems regardless of whether they are causally ordered or contain algebraic loops. JMorven has been written completely from scratch in the Java language for maximum portability. JMorven succeeds and improves upon the its predecessor, Morven (formerly known as Mycroft (Coghill, 1996)), by:

- incorporating parallelisations throughout the design which allows the framework to be distributed, substantially decreasing execution time and thus making it more applicable to industrial application.
- using totally non-constructive algorithms and the ability to use auxiliary variables in non-constructive algorithms.
- using n-th order Taylor series integration which increases the accuracy over Euler

integration.

- incorporating several novel simulation algorithms which allow non-constructive simulations to be carried out semi-quantitatively and numerically, and also providing output charts of the simulation behaviours. Algorithms include regular-spaced methods and Monte-Carlo techniques. See chapter 7 for more information.

- Simulating on the spectrum from qualitatively to quantitatively and doing so from the same model representation.

The original motivation to create a qualitative reasoning engine was for it to be used in developing a model-based planner. Since no portable implementation was available it was decided to create a basic qualitative reasoner. During the early stages of development, it was evident that there was place for an improved strategy toward qualitative simulation. Reading into the area showed that one research group had looked into the benefits of parallel computing to make QSIM more efficient however there were a few drawbacks with their work. This motivated the development of a better system which would make use of an abstract parallel architecture allowing it make use of a wide variety of computing environments.

As discussed in chapter 3, Platzner and Rinner (Platzner et al., 1997; Platzner and Rinner, 1998, 2000) proposed and demonstrated that it was possible to speed up the popular QSIM package (Kuipers, 1986). They achieved positive results by porting QSIM to the C language, by parallelising several stages and using a dedicated hardware setup to carry out intensive calculations. There were a few drawbacks with the work undertaken by Platzner and Rinner. One limitation of the work on parallel-QSIM was that the architecture was designed to run on a restricted number of parallel units resulting in a non-scalable implementation. Overcoming this limitation by developing an abstract and portable architecture should result in a more usable qualitative reasoner that would allow it to be used to solve a wider variety of problems. Another drawback with their work was that they did not

parallelise all stages of execution.

Another motivation for JMorven was to develop a simulation engine capable of reasoning on the spectrum from fully qualitative to fully quantitative which would mean that the same tool could be used throughout the development of any model design from concept through prototype to final product. This is useful in the beginning stages of design when not all factors are known, nor is specific numerical knowledge therefore full qualitative models are most useful. As the design continues, more numerical information is known hence a semi-quantitative model can be used to predict behaviours. Finally, a full quantitative model can be derived leading to the design of the final product.

JMorven is the first known fuzzy qualitative reasoning engine to make use of parallelisations to benefit execution time. JMorven benefits from parallelisations in all stages of the reasoner which is a first for QR. These parallelisations are incorporated into an abstract, portable architecture which allows JMorven to be scalable. It is also the first to provide a range of simulations from fully qualitative to fully quantitative. Merging these together provides a parallel architecture for simulation which is also thought to be novel. Having this framework implementing these features is believed to be useful and contribute significantly to the field of qualitative reasoning.

The rest of this chapter discusses the qualitative aspects of the JMorven framework, in particular the stages of qualitative analysis and transition analysis that have been parallelised. There is also a discussion about the use of auxiliary variables in a non-constructive algorithm.

6.2 Design

Although JMorven is based largely on its predecessor, Morven, several design choices have had to be made due to JMorven reasoning in a non-constructive manner. The result is that JMorven combines many features from several existing QR packages including QSIM, FuSim, Parallel QSIM and Morven. As with these packages, JMorven implements the Qualitative Analysis (QA) and Transition Analysis (TA) phases. These are discussed in more detail below. Figure 6.1 shows how the QA and TA phases are combined to create the qualitative component of JMorven.

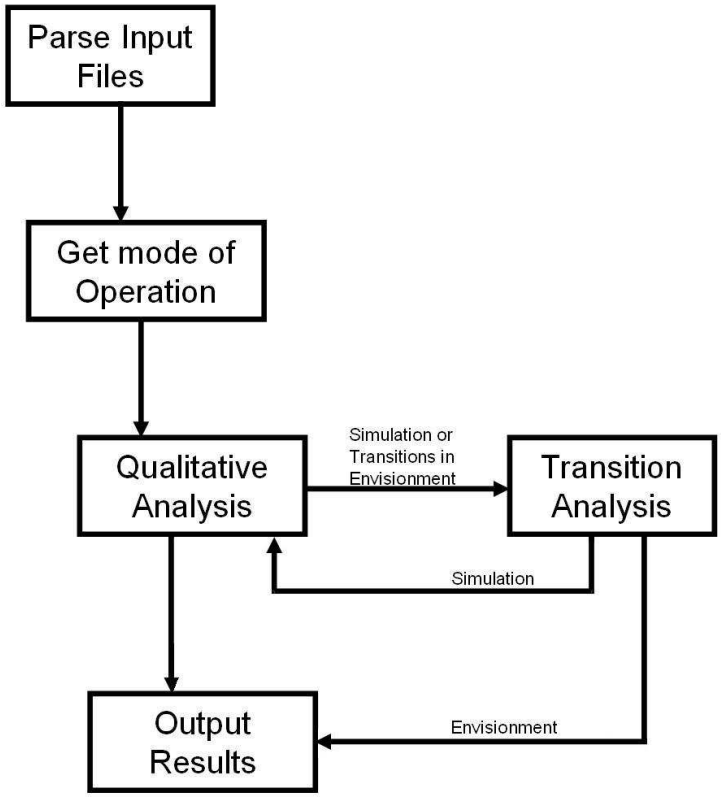


Figure 6.1: Flowchart of QA and TA phases in JMorven Operation.

6.2.1 Qualitative Analysis

The Qualitative Analysis (QA) phase in JMorven is responsible for analysing the constraints and ensuring that qualitative states are consistent with them. This is typically split into two stages, the constraint filter and the state generator. The aim of the constraint filter is to generate and test sets of tuples which are consistent with every constraint. There are several methods for achieving this but the most common involves a two step procedure consisting of a Tuple Filter and a Pairwise Filter. The Tuple Filter generates sets of valid tuples for each constraint in the model which are guaranteed to be unique and consistent for the current constraint only. To ensure all tuples are consistent with the complete model, a pairwise filter is used to test each pair of constraints for inconsistencies. Once the pairwise filter has completed, a set of consistent tuples remain which are then used to generate qualitative states. Each qualitative state can then be analysed by the Transition Analysis phase for simulation.

6.2.2 Transition Analysis

The Transition Analysis (TA) phase takes each qualitative state and generates a list of possible successor states to be tested by the QA stage. Generating these successor states is achieved either by integration or by following a set of predefined transition rules depending on whether simulations or envisionments are required respectively. JMorven allows two modes of simulation; one which reasons qualitatively in that all variables in the states adhere to the values in the quantity space. A directed graph is generated for all states reachable from the initial state therefore all qualitative behaviours can be extracted from this graph. The alternative simulation mode in JMorven is carried out numerically using exact numbers or fuzzy numbers to represent imprecise values. This simulation mode is discussed in more detail in chapter 7.

6.3 Abstract Parallel Architecture

JMorven implements a novel parallel architecture, as shown in in figure 6.2, which allows it to make best use of the available resources, whether it be multiple processors or multiple computers in a distributed computing environment. This architecture was developed in Java for maximum portability. The following sections describe the main stages in detail, including the primary function of the stage and how it has been implemented in parallel.

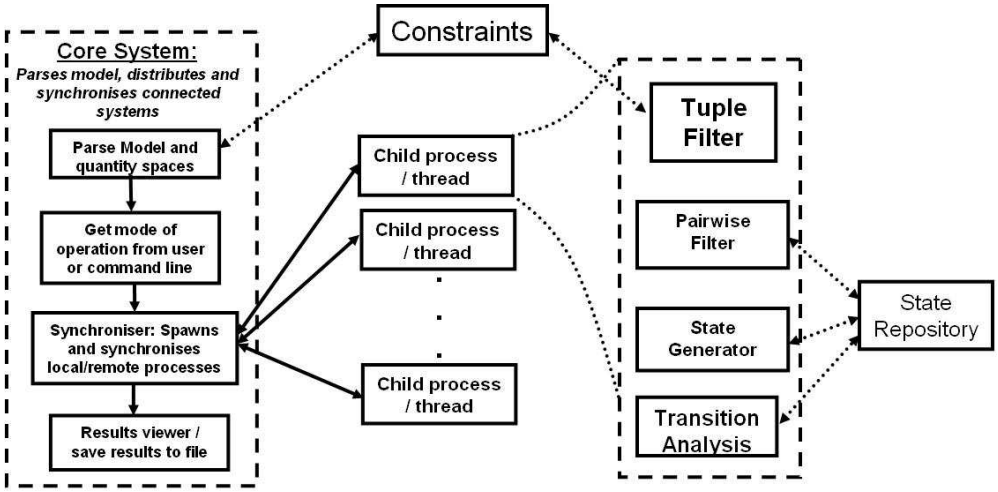


Figure 6.2: The JMorven Parallel Architecture Overview.

6.3.1 Parallel Tuple Filter

The tuple filter is responsible for iterating through each constraint in turn and providing a list of valid tuples. A valid tuple is a set of consistent variable values for the constraint, e.g. for the constraint shown:

$$A = B + C$$

If B is *small* and C is *medium* then possible values for A may include *medium* or *large* depending on the quantity space used. The tuple filter is either given a list of tuples to check for consistency or it can generate all possible consistent tuples from an exhaustive list of all possible quantities.

The tuple filter in QSIM was analysed by Platzner & Rinner (Platzner et al., 1997). They constructed the data-dependency graph of the tuple filter and Waltz filter shown in figure 3.1. The incremental Waltz Filter used in QSIM could be replaced by a sequential pairwise filter. Platzner & Rinner found that while the incremental one was designed so that tuples could be discarded earlier thus having fewer tuples to filter, the speed benefit was not great. This speed increase was certainly not significant compared to the speed increase by paralellising the tuple filter. Splitting the tuple filter and Waltz filter into sequential filters results in the data dependency graph shown in figure 3.2. This allows the parallel tuple filter to execute each constraint in its own execution unit since there are no inter-dependencies. The Waltz filter merely waits for all constraints to be fully filtered before proceeding with its own filtering method.

JMorven uses containers to hold certain entities during execution. These containers are simple data structures similar to linked lists and do not have any mechanism to protect data when accessed from multiple execution units. This allows optimal performance to be achieved when accessing the contents of the containers, but care must be taken when accessing to ensure that if multiple execution units access the data concurrently then no corruption will take place. Mutexes are used to protect against this.

Each constraint is distributed equally between a number of containers; the number of containers is equal to the maximum number of execution units available. One JMorven-Thread[1] (see Appendix D.1 for the Java code) is created for each container therefore in each JMorvenThread containing a number of constraints to analyse and generate a valid set of tuples.

Each JMorvenThread iterates through all of the constraints in the container. For each constraint an exhaustive list of all tuples for all quantities in the associated quantity spaces

[1] A JMorvenThread is a wrapper for a standard Java thread which offers some housekeeping. This includes keeping track of the number of threads running concurrently, utility methods for waiting on threads to complete and timing mechanisms to determine how much time each thread takes to execute.

is generated. Each tuple is then checked in turn to determine if it is consistent with the constraint. If the tuple is inconsistent it is discarded. If it is consistent it is then stored in the output container to be tested by the pairwise filter.

A diagram of how the Tuple Filter is implemented is shown in figure 6.3.

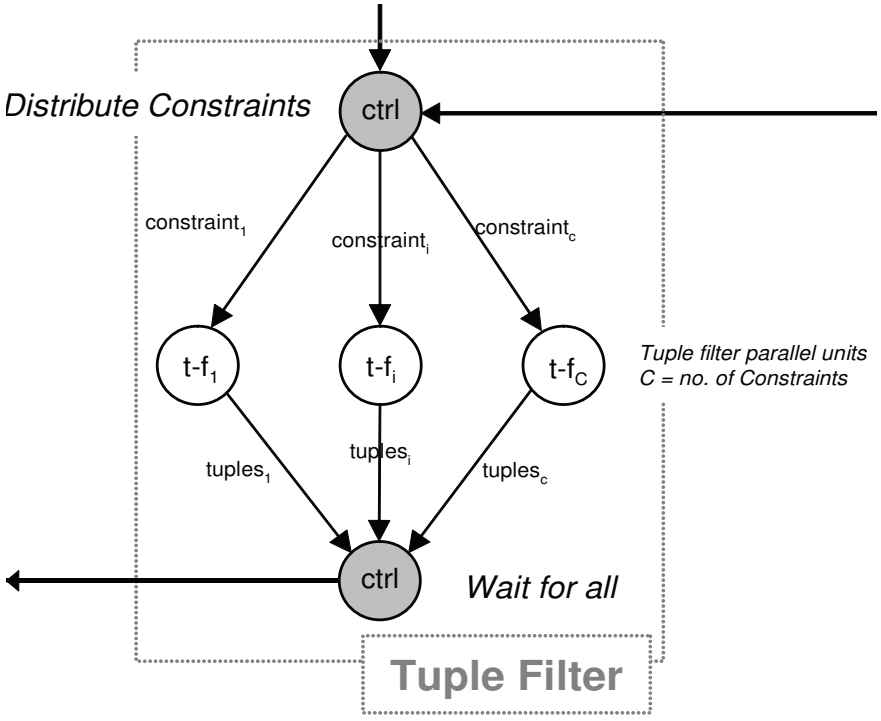


Figure 6.3: The Tuple Filter in parallel.

6.3.2 Parallel Pairwise Filter

The Pairwise Filter is used to ensure that all tuples resulting from the Tuple Filter are consistent across all constraints, since the tuple filter only reasons per-constraint. This is achieved by testing each pair of adjacent constraints (two constraints are said to be adjacent if they each share a common derivative of a variable, i.e. vector element). Each valid pairing then iterates through all possible tuples and discards those that are not common to both constraints resulting in a list of tuples numerically equal to or less than the input. An example of the reasoning behind the pairwise filter follows. If we have constraints C1

and C2 from the single tank example as shown in Appendix A.1 with tuples as shown:

C1:[V' q_i q_o] = [*p-small p-large p-medium*]
C2:[V q_o] = [*p-medium p-medium*]

The only vector element common to both of these constraints is q_o which is consistent for the given values (*p-medium* in both constraints) therefore the pairwise filter would not discard this pair, however if we had:

C1:[V' q_i q_o] = [*p-small p-large p-medium*]
C2:[V q_o] = [*p-large p-large*]

then the pair would be discarded since the value of q_o would be inconsistent across the pair of constraints.

Each individual pair of constraints can be filtered independently of all others since there are no data dependencies between them, therefore allowing the pairwise filter to be easily parallelised. JMorven implements this pairwise filter by first creating an exhaustive list of all possible pairs of constraints for the model and removing those that are not adjacent. JMorven then creates a number of containers equal to the number of execution units available. Each adjacent pair of constraints is then distributed equally amongst these containers and a new JMorvenThread is created for each container. Each JMorvenThread then iterates through all pairs of constraints. For each adjacent pair, all of the consistent tuples are tested to ensure they are consistent with both constraints. If a tuple is found that is inconsistent then this tuple is removed from the container. After all tuples have been checked, the container contains only tuples that are valid with that pair of constraints which is then passed to the state generator.

A diagram of the data dependency of the pairwise filter is shown in figure 6.4.

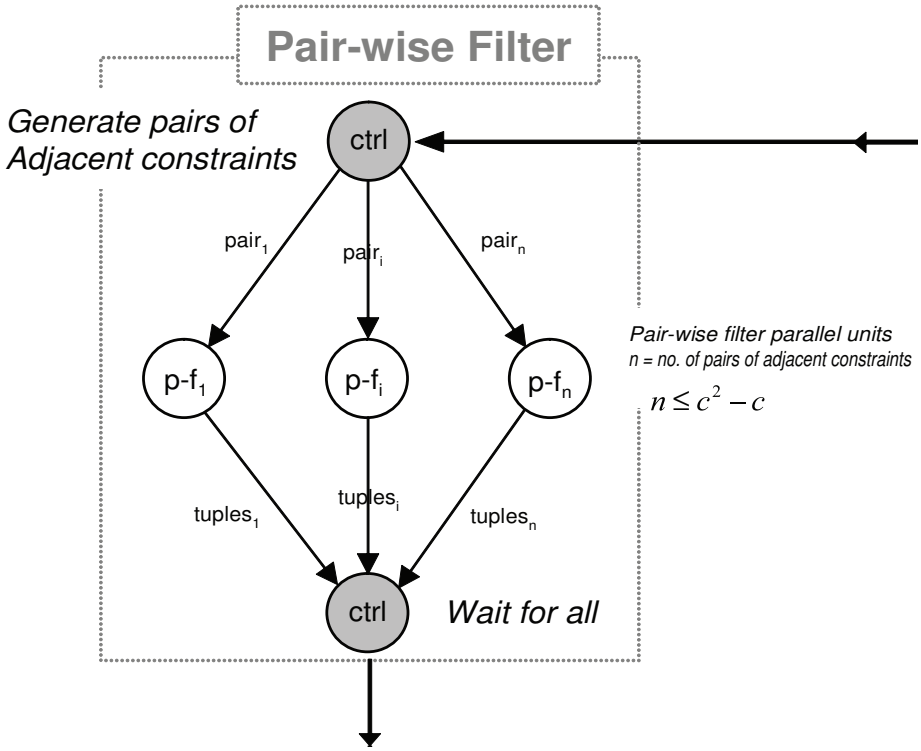


Figure 6.4: The Pairwise Filter in parallel.

6.3.3 Parallel State Generator

The State-generator is the most computationally expensive stage of Qualitative Analysis, especially when creating an envisionment of all possible states. This is the process of iterating through each set of tuples and creating unique states for every combination of variables' derivatives possible.

The State Generator is implemented in JMorven by iterating through each constraint in turn, in a breadth-first manner, starting with an empty initial state. Each constraint has an associated number of tuples which are consistent with it. These tuples are distributed equally among a number of containers equal to the number of execution units available. Each tuple container also has an associated states container. Each states container has a copy of the same contents; the partially defined consistent states up to this point. Each

pair of containers is then used to generate a new set of partial states for the set of tuples in the next constraint. Each pair of containers are used as input to the JMorvenThread and the output is a set of valid partial states for the next iteration. The following pseudo-code outlines the process:

```
initialise input state-containers with empty initial states
iterate c through all constraints
    split tuples in constraint into tuple-containers
    iterate t through tuple-containers
        create JMorvenThread with input containers to produce output state-containers
    copy output state-containers into new input state-containers for next iteration
```

Each JMorvenThread in the State Generator is implemented by iterating through each set of tuples in the tuple-container and each partial state in the state-container. If the state is consistent with the tuple then the tuple is set and added to the output state-container. The following pseudo-code describes the process:

```
iterate through t tuples in tuple-container
    iterate through s states in state-container
        if tuple t is consistent with state s
            create copy of state s and set tuple t - store in output state-container
```

This allows the state-generation to run in parallel as shown in figure 6.5.

6.3.4 Parallel Transition Analysis

The Transition Analysis (TA) phase involves determining how qualitative states transit between one another from one point in time to another. This is achieved by following a set of transition rules as shown in figure 6.6 or by numerically integrating variables to

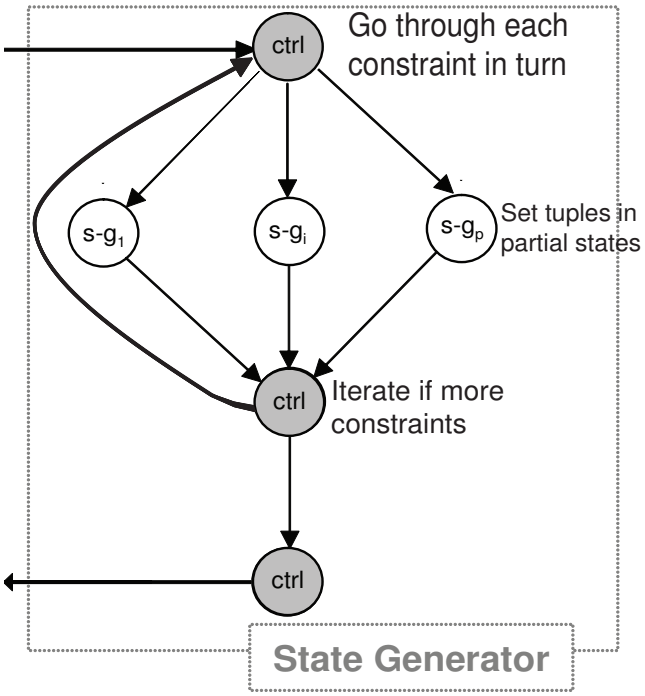


Figure 6.5: The State Generator in parallel.

find successor values. The numerical integration transitions are discussed in more detail in chapter 7.

The transition rules are shown for the purely qualitative quantity space for simplicity however the idea applies to fuzzy quantities too. For the quantities adjacent to zero in a fuzzy quantity space, the diagram above still applies, e.g. if we have variable A = [*p-small n-small*] then, from the diagram, it can transit to either [*zero n-small*], [*p-small zero*], or

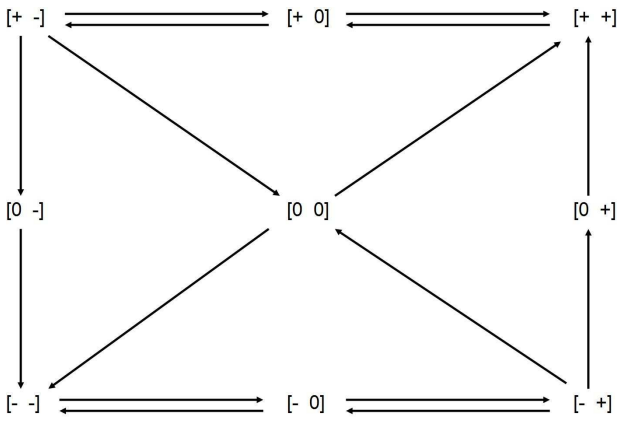


Figure 6.6: The Transition Rules in JMorven.

[*zero zero*]. In addition to this the derivative may change to any adjacent quantity i.e. *zero* or *n-medium* which gives additional variable values of [*p-small n-medium*] or [*zero n-medium*]. A more general set of rules is given below:

- Quantities can only transit to adjacent quantities in the quantity space due to the continuity constraint (quantities can also stay as they are unless the quantity has zero width and has a non-zero derivative).

- If there is derivative information available this dictates the direction of the transition, i.e. if the derivative is greater than zero then the quantity must transit to the next larger quantity in the quantity space (similarly for negative derivatives and smaller quantities).

- If the quantity has zero width then any derivative change must also trigger a change in the quantity, e.g. if we have the quantity *one* which represents exactly the number one as a fuzzy quantity and we have variable A = [*one zero*] then transitions to [*one n-small*] and [*one n-positive*] would be invalid since the quantity *one* did not change.

The transition analysis phase takes each qualitative state and applies the transition rules to each variable within the state and generates a list of possible successor states. These successor states are only consistent with the transition rules and must be filtered through the qualitative analysis stage to check that they are consistent with the constraints too.

Since the transition rules are applied to each state independently, the TA phase can be easily parallelised. Each execution unit is given a number of states to analyse and generate successor states. The states are distributed naïvely, assuming that each execution unit has the same processor power available.

A diagram of the TA implementation is shown below in figure 6.7.

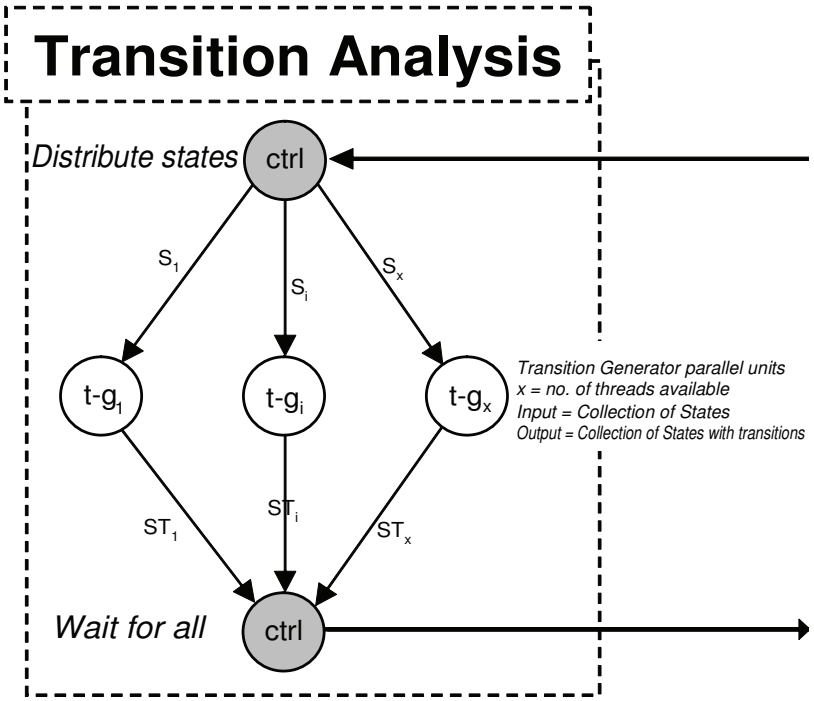


Figure 6.7: The JMorven Transition Analysis Phase in Parallel.
S_i denotes a State without transitions and ST_i is a State with transitions calculated.

6.3.5 Complete Parallel Implementation

In this chapter, each stage of the JMorven qualitative reasoning engine has been discussed, and a description given of how it has been implemented in parallel. Figure 6.8 shows a complete diagram of JMorven implemented in parallel.

6.4 Auxiliary Variables

Models in JMorven are defined using a set of qualitative differential equations. These equations are split into two or three place constraints for use internally. This restriction is to enable the tuple filter to execute efficiently and also allows ease of parsing. If a QDE requires more than one of these two or three place constraints, temporary variables need to be used. JMorven incorporates auxiliary variables like its predecessor, Morven. However there are several problems implementing auxiliary variables in JMorven since it

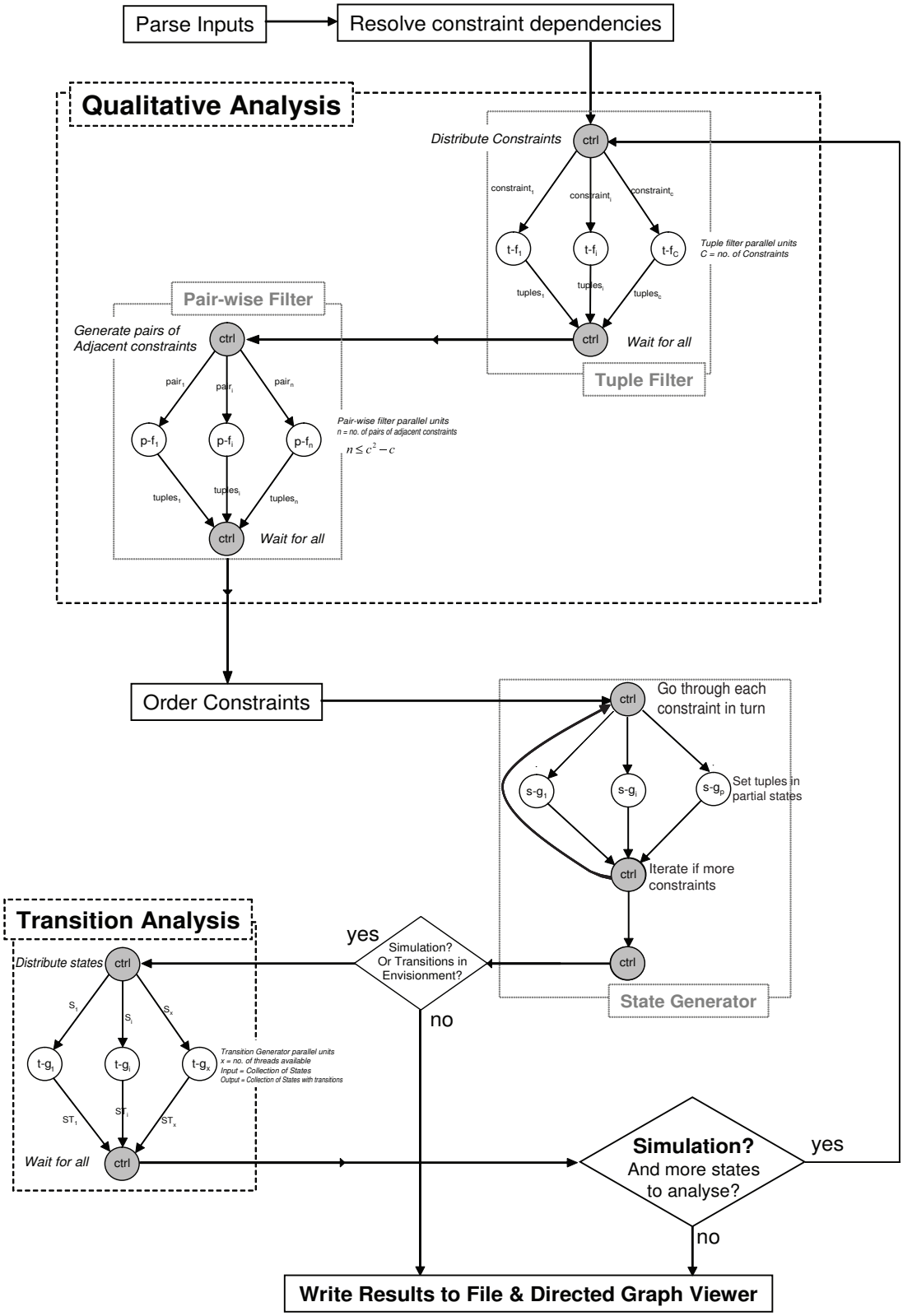


Figure 6.8: The JMorven Qualitative Parallel Implementation.

reasons non-constructively. This is discussed in more detail below.

6.4.1 Non-constructive Auxiliary Variables

Auxiliary variables are used temporarily to store the values of a JMorven variable across constraints. Auxiliary variables are implemented in JMorven as a fuzzy four tuple which is initially unset, then when constrained this interval is set. Within a constraint the union of all possible temporary values is used for the auxiliary variable to ensure that all tuples remain valid using this temporary value. Across different constraints the intersection of temporary values is taken which is effectively the same as the pairwise filter on normal variables. The intersection of values is taken to ensure that all temporary values are consistent with the whole model. For example (using crisp quantities to keep the example simple), if these two constraints are part of a model

$$T = A * B$$

$$D = C + T$$

where A is $p - small = [0.5 \quad 0.75]$ and B can be either $p - medium = [1.0 \quad 1.5]$ or $p - large = [1.5 \quad 2]$ then the auxiliary variable T can take the values of $[0.5 \quad 1.125]$ or $[0.75 \quad 1.5]$. Since this is within a single constraint then the union of these values is taken therefore $T = [0.5 \quad 1.5]$. If we then evaluated the second constraint and found that $T = [0.75 \quad 1.667]$ then the intersection of both values of T would be taken resulting in $T = [0.75 \quad 1.5]$. Notice that this does not necessarily match any quantity in the quantity space.

Since JMorven reasons non-constructively there is no requirement for ordering equations thus JMorven must be able to handle unordered auxiliary variables or loops. Before execution, JMorven loads the model file and parses the constraints, then it finds all of the dependencies of each auxiliary variable in an attempt to order them automatically to avoid

iteration. If a cycle is detected, i.e. there is an algebraic loop in the model constraints, then iteration will have to take place. Iteration of the constraints occurs because narrowing the interval of one auxiliary variable may have an effect on another if there is a loop or the constraints are unordered. This iteration continues until there is no change in any of the auxiliary variables concerned (typically only three iterations are required, one which narrows the maximum side of the fuzzy tuple, one which narrows the minimum and a final one to ensure the changes don't have any further effect).

6.5 Summary

In this chapter, the motivations for JMorven were discussed and how JMorven was designed and implemented. The chapter details the qualitative aspects of JMorven including the parallel architecture created for increasing efficiency when executed in environments with multiple processors available (Chapter 7 presents the semi-quantitative and quantitative contributions of JMorven). These parallelisations are discussed with the reasoning behind the design of each major stage and how it is implemented in JMorven. The filters (tuple and pair-wise) and transition generator are classic examples of data parallelisation in that the data to be filtered can be split easily for use in each execution unit. The state generator is a more complex stage and required an algorithm parallelisation to make best use of available resources. The result is that all of the stages in JMorven have been parallelised resulting in JMorven being the first fully parallel qualitative reasoning system available. In addition to this, JMorven uses non-constructive algorithms which make it more general since they have the ability to cope with systems regardless of whether they are causally ordered or contain algebraic loops.

Chapter 7

Non-constructive Fuzzy Interval Simulation

As discussed in chapter 5, there are a number of simulation strategies already available yet none have achieved good results when using non-constructive algorithms. In this chapter, a non-constructive synchronous simulator, as part of the JMorven framework, is presented and discussed. This simulator has several different algorithms for tackling the problem. What results is a simulator that can reason with fuzzy numbers in a non-constructive manner and which does not suffer from unnecessary interval divergence. SyNCSim (discussed in Chapter 4.4) showed that it is possible to conduct synchronous simulation using non-constructive algorithms. JMorven uses this knowledge to apply similar techniques to semi-quantitative and quantitative simulation.

7.1 Twin Interval Fuzzy Numbers

Fuzzy numbers in JMorven are represented by fuzzy four-tuples. However to aid the calculations in the synchronous simulation environment this has been modified to a new

representation termed *Twin Interval Fuzzy Numbers*. Fuzzy numbers are represented by two numerical intervals: The inner interval is defined from $[a \quad b]$ and the outer interval from $[a-\alpha \quad b+\beta]$. Since α and β are zero or positive then the inner range is guaranteed to be a subset of the outer range, from Moore (Moore, 1966):

$$A \subset B \longrightarrow f(A) \subset f(B)$$

for intervals A and B. This representation can be used to specify a real interval if α and β are zero, i.e. both ranges are equal. Also if the intervals have zero width, this representation can also be used to represent a real number. It is worth noting that this could be extended to represent more complex fuzzy numbers where each interval could be the result of taking the α-cut with different values for α.

7.2 Updated Interval Arithmetic

There are several different definitions for interval arithmetic in the literature. Table 7.1 is an amalgam of these with the addition of a flag to determine if the operands refer to the same interval. One point to note about this table over that used in existing qualitative reasoning engines is that this table defines the arithmetic properties for operations whose operands may span zero which is more important for interval simulation than qualitative simulation since the time-steps are generally far smaller. The result of this is a more complex table however implementation-wise it does not increase computation time.

Qualitative trigonometric operations (Liu and Coghill, 2005a; Coghill et al., 2005) have been included in JMorven. However, JMorven extends the qualitative trigonometry by adding the ability to calculate the result of trigonometric operations on intervals. This is made difficult in that trigonometric functions are non-monotonic and therefore the extreme points are not sufficient to calculate the real result. The process below outlines how the trigonometric interval $[c \quad d]$ is calculated in JMorven:

Let:	$m = [a \quad b], n = [c \quad d]$	
Operation	Result	Conditions
$-n$	$[d \quad c]$	all n
$\frac{1}{n}$	$[\frac{1}{d} \quad \frac{1}{c}]$	$c, d > 0$ or $c, d < 0$
	$[-\infty \quad \infty]$	$c \leq 0$ and $d \geq 0$
$m + n$	$[a + c \quad b + d]$	all m, n
$m - n$	$[a - d \quad b - c]$	$m \neq n$
	$[0 \quad 0]$	$m = n$
$m \times n$	$[ac \quad bd]$	$m = n$ and $(c, d > 0$ or $c, d < 0)$
	$[0 \quad bd]$	$m = n$ and $c \leq 0$ and $d \geq 0$
	$[ac \quad bd]$	$m \neq n$ and $a, c > 0$
	$[bc \quad ad]$	$m \neq n$ and $a > 0$ and $d < 0$
	$[bc \quad bd]$	$m \neq n$ and $a > 0$ and $c \leq 0$ and $d \geq 0$
	$[ad \quad bc]$	$m \neq n$ and $b < 0$ and $c > 0$
	$[bd \quad ac]$	$m \neq n$ and $b < 0$ and $d < 0$
	$[ad \quad ac]$	$m \neq n$ and $b < 0$ and $c \leq 0$ and $d \geq 0$
	$[ad \quad bd]$	$m \neq n$ and $a \leq 0$ and $b \geq 0$ and $c > 0$
	$[bc \quad ac]$	$m \neq n$ and $a \leq 0$ and $b \geq 0$ and $d < 0$
	$[min(ad, bc) \quad max(ac, bd)]$	$m \neq n$ and $a \leq 0$ and $b \geq 0$ and $c \leq 0$ and $d \geq 0$
$\frac{m}{n}$	$[1 \quad 1]$	$m = n$
	$m \times \frac{1}{n}$	$m \neq n$

Table 7.1: Interval Arithmetic Operations as Defined in JMorven

$m \neq n$ denotes that the intervals do not correspond to the same variable whereas $m = n$ indicates that the intervals do correspond to the same interval. $a, c > 0$ indicates that both intervals are positive whereas $b, d < 0$ dictates that both intervals are negative. $c \leq 0$ and $d \geq 0$ (as well as $a \leq 0$ and $b \geq 0$) governs that the interval spans zero. It is possible to define $m \times n$ for when both intervals span zero however it has been left out in this table for simplicity.

```
if (d − c) ≥ 2π then return [min(f_trig)  max(f_trig)]
determine result of f_trig([c  d]) ⟶ [lo  hi]
if lo > hi then swap lo and hi
find nearest quadrant < lo using lastQuad = c − (c  mod  π/2)
set nextQuad = lastQuad
while nextQuad < d
    update [lo  hi] with f_trig(nextQuad)
    increment nextQuad by π/2
return [lo  hi]
```

Having a fully correct interval arithmetic definition is important to reduce the number of spurious behaviours generated. Some simulators use a naïve interval arithmetic based on merely the extreme points or assume that intervals do not span zero. In many of these cases not all values may be included in the calculated results and in some cases extraneous width of intervals may be calculated particularly in the case when operations are carried out on the same variable, e.g. $A \times A$.

7.3 Integration Techniques

As discussed in section 5.1 there are many different techniques used to approximate the integration process for simulation. They are summarised below with respect to non-constructive simulation.

- Euler Integration is the simplest form of approximate integration. It has the benefit of being simple and very efficient however it suffers from large errors even when very small time-steps are used. It is used in existing systems since often only one derivative is known for each variable.

- Runge-Kutta methods find an estimate of the derivatives at successive time-steps and refine them using the system of equations to achieve more accurate results. The second order and fourth order variants are the most common and provide an increased level of accuracy when used in simulation. However, these cannot be used non-constructively since they require interaction between the variables of the model in a similar way to Moore's interacting approach (Moore, 1966).

- Taylor Series uses as much derivative information as possible unless a threshold is defined and reached. This allows Taylor Series to be more accurate for variables of an order greater than two (Euler Integration is the equivalent of first order Taylor

Series). Since JMorven reasons with multiple derivatives, this technique is particularly suitable. The more derivatives there are specified in the model results in a more accurate simulation. It is also suited toward non-constructive simulation since each variable can be integrated individually.

It is clear that Taylor Series offers the best method for integration for use in JMorven. To integrate using Taylor Series, each variable is considered in turn and each derivative can be calculated with the exception of the last derivative since there is no further derivative information available. This last derivative of a variable is left undefined although the model constraints should constrain it. A Taylor Series is defined as:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + ... + \frac{f^n(a)}{n!}(x-a)^n + ...$$

where $f(x)$ is the estimated value of the function after a certain time-step. The current value of the system variable is denoted at $t = a$. JMorven integrates a variable vector as shown in pseudo-code below:

```
set n = number of derivatives in variable V_t
iterate x from 0 to n-1
    set P_{t+δt} = P_t from deriv x of Variable V_t
    iterate y from x+1 to n
        calculate f = δt^(y-x) / (y-x)!
        increment P_{t+δt} by f × P_t
    set P_{t+δt} as deriv x of Variable V_{t+δt}
return V_{t+δt}
```

7.4 Non-Constructive Interval Simulation

Since JMorven uses a Twin Interval representation for Fuzzy Numbers, it can be summarised as interval simulation since simulating in the fuzzy domain is merely an extension of interval simulation in this way. To simulate non-constructively a model and a partially-defined initial state are required. The more precision available in the initial state results in a more precise simulation. The initial state is first analysed to ensure that it is consistent before proceeding with the simulation. The simulation process is an iterative one which starts from time t_0 until t_{end} in steps of δt. The first part of the process is to integrate each variable in turn as described in section 7.3. Once the integration is complete, the constraints are used to determine the values of all undefined derivatives from the integration process. This results in a state which is fully specified although it may contain extraneous width in the intervals. To narrow the intervals, the constraints are used to narrow the intervals of all possible values repeatedly until no more changes occur or a threshold is reached. During this analysis the new updated values of all variables are used which help to narrow other variables within constraints. One technique that is also used is Inverse Constraint Operations. This is described below.

7.4.1 Inverse Constraint Operations

Inverse Constraint Operations is the process of applying the inverse, or inverses, of a constraint on a tuple. The aim is to discard more inconsistent values and minimise the width of all intervals within the constraint resulting in fewer spurious behaviours being generated. For most constraints there is at least one associated inverse constraint which is arithmetically inverse. For example, the addition constraint has two inverse constraints defined as shown below:

$$A = B + C$$

has the inverse constraints:

$$B = A - C$$

$$C = A - B$$

It may not be sufficient to merely calculate the interval for A based on the values of B and C since the calculated result may help in the reduction of the interval width of the other variables. Applying the inverse constraints helps reduce the interval width of all variables within the constraint, not just the left hand side variable. It also reduces the number of times the model constraints need to be looped before no changes are apparent (as described in chapter 6.4.1). If a constraint does not have an inverse then this process is skipped for that constraint.

The following example shows how using the inverse constraints help reduce more than just the interval of the left hand variables within a constraint. If the constraint below is considered

$$A = B + C$$

and have initial values of $A = [5 \quad 6]$, $B = [3 \quad 4]$, and $C = [2.5 \quad 3.5]$ which are calculated from the integration step. Applying the basic constraint arithmetic narrows A as shown below. B and C are used to determine what the consistent range of values for A could be.

$$[5.5 \quad 7.5] = [3 \quad 4] + [2.5 \quad 3.5]$$

however $A = [5 \quad 6]$ from integration therefore taking the intersection of the two ranges narrows A to become $A = [5.5 \quad 6]$

To narrow the rest of the variables the inverse constraints should be used. The first one is taken:

$$B = A - C$$

then the interval for B can be narrowed as shown:

$$[2 \quad 3.5] = [5.5 \quad 6] - [2.5 \quad 3.5]$$

but from integration, $B = [3 \quad 4]$. Taking the intersection of the intervals gives $B = [3 \quad 3.5]$.

Finally, using the last inverse constraint:

$$C = A - B$$

the interval for C may also be narrowed as shown:

$$[2 \quad 3] = [5.5 \quad 6] - [3 \quad 3.5]$$

but $C = [2.5 \quad 3.5]$ from integration therefore taking the intersection $C = [2.5 \quad 3]$

This now results in all variables being narrowed as much as possible as shown:

$$[2.5 \quad 3] = [5.5 \quad 6] - [3 \quad 3.5]$$

After this process the intervals for A, B and C are narrowed as much as possible when reasoning over this constraint alone. However, these updated values may result in further narrowing in other constraints; hence the process is repeated until no more changes are made in the whole model or a threshold is reached. Due to the narrowing of intervals using this Inverse Constraint Operations, not much looping of the whole model is required.

This process could be parallelised since each inverse constraint can be carried out independantly of the others however the overhead of creating and synchronising threads for

such a small process would mean that the benefits would be small at best.

7.5 Fuzzy Interval Simulation

This section describes the different simulation strategies used in JMorven for simulating fuzzy numbers as real intervals.

7.5.1 Basic Interval Simulation

The most basic mode of simulation in JMorven is a straightforward interval simulation engine. This mode uses the interval arithmetic defined in table 7.1, the Inverse Constraint Operations discussed in section 7.4.1 and n-th order Taylor Series is used to integrate current values to predict the successor values (where n is equal to the number of derivatives defined for the variable).

The initial state is defined for time $t = 0$. The user is asked for a time-step of δt and a total time to simulate over t_{tot} therefore there will be

$$num = \frac{t_{tot}}{\delta t}$$

distinct time steps therefore num distinct states created during the process of simulation. At each time-step all derivatives of all variables are set to an initial range equal to $[\infty \quad -\infty]$ and stored in the repository. Once the integration phase has completed, all derivatives are set in the repository with the following trivial process setting $minimise$ to false (this is the process of taking the union of all values in the qualitative auxiliary variables):

```
deffun:  updateInterval
params:
    float:  newMin newMax t
    string:  var
    int:  deriv
    boolean:  minimise
        old = getInterval for deriv of var at t
        if minimise XOR (newMin > old.min)
          old.min = newMin
        if minimise XOR (newMax < old.max)
          old.max = newMax
```

When constraining and narrowing intervals using the Inverse Constraint Operations, the above process is repeated but with $minimise$ set to true to ensure the intervals narrow (this is the process of taking the intersection of the values of qualitative auxiliary variables).

Even with these extra measures to reduce spurious trajectories, the intervals widen over time. This interval divergence is observed more as the initial interval increases. The output from this mode is complete but not sound, i.e. all of the correct solutions are bounded by the output intervals however spurious solutions are also included in the output.

7.5.2 Sub-Interval Simulation

Several methods were researched to find ways to decrease the amount of interval divergence for simulated trajectories. One method which was found to be successful was Sub-Interval Simulation. In this process, each of the intervals are divided into x regularly

spaced distinct sub-intervals that cover the whole original interval. These intervals do not overlap and if the union of all sub-intervals is taken the original interval is restored. This is guaranteed to bound all real solutions of the problem as proved by Moore (Moore, 1966):

$$A \subset B \longrightarrow f(A) \subset f(B)$$

States are generated each having a unique combination of all sub-intervals. Each state forms the initial state for a simulation to proceed using the basic interval simulation strategy. Since JMorven reasons about fuzzy numbers, a strategy has to be devised to output a fuzzy trajectory. For each variable, if the sub-interval to be simulated is completely bounded by the interval $[a \quad b]$ (where a and b are the two parameters of the fuzzy four-tuple [$a\ b\ alpha\ beta$]) then the interval is considered as an inner range, otherwise the interval is considered as an outer range. These inner and outer ranges are combined to form the Twin-Interval fuzzy numbers as described in section 7.1.

The motivation for this technique was that the width of the initial interval has a direct influence on the amount of divergence on simulated trajectories. Initial intervals which are narrower suffer less divergence over time, hence having a method which is guaranteed to bound all solutions but diverge less is desired. As the sub-interval widths tend to zero, the output trajectories tend toward the real solution with no unnecessary interval divergence. Hence, this method is complete but not sound however soundness is achieved as the width of the sub-intervals tend to zero. Figure 7.1 shows the simple oscillator model defined in section 5.1 simulated with an original interval and several sub-interval simulations with the number of sub-intervals being increased.

The advantage of this process is clear, however there is one disadvantage to the approach. Due to each interval being divided into sub-intervals there is an exponential number of initial states to simulate from. This seems to be a major problem at first, however during experimental testing one behaviour was observed that reduces the number of states

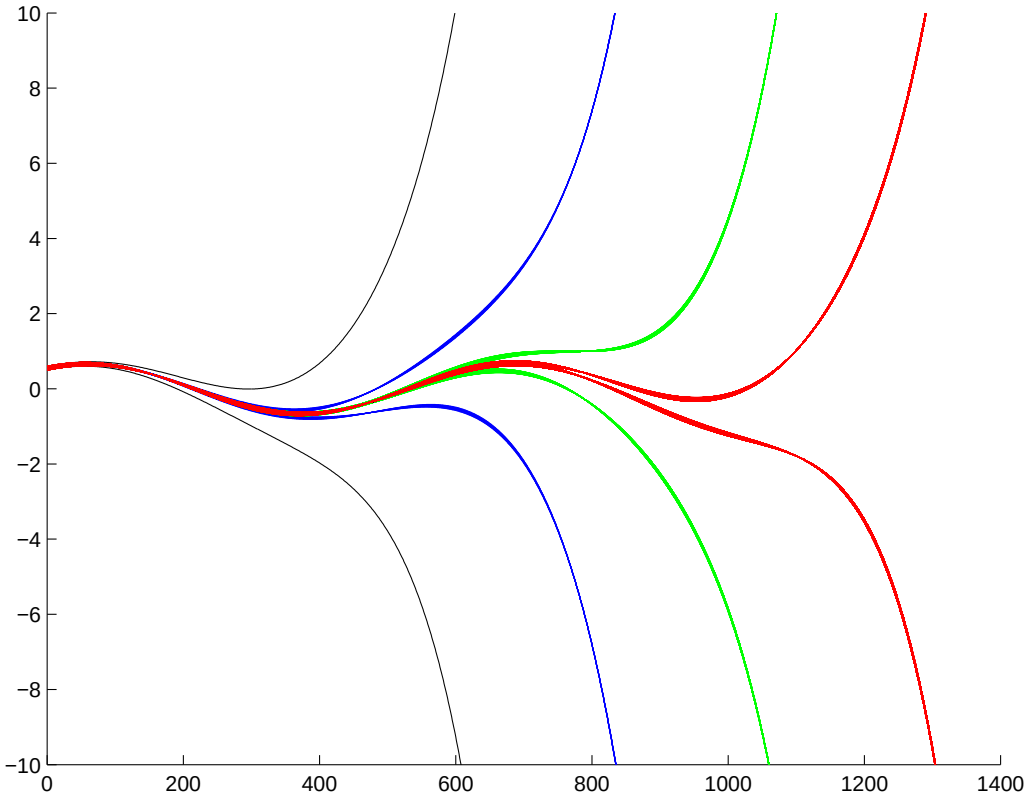


Figure 7.1: Undamped spring model simulated using the sub-interval method. This shows the simulated trajectories of the spring model amplitude against time. The interval shown in black is the spring model simulated using the basic interval simulation strategy. The interval shown in blue shows the same problem using the sub-interval method splitting the original interval into ten distinct sub-intervals. Similarly, the green interval uses 100 sub-intervals and finally the red interval simulates with 1000 sub-intervals.

greatly. The observation is that not all initial states are consistent once the original intervals are split. This reduces the number of simulations to be executed making this process more feasible than originally thought. One other method to decrease execution time is to take advantage of the parallel framework of JMorven. Since each initial state can be simulated individually, it is trivial to make this method benefit from parallelisations. The only interaction between execution units is the updating of intervals after integrating and constraining occurs which occupies a very small fraction of the whole simulation process therefore close to linear speed-up is possible. The results of the parallelisations are discussed in more detail in chapter 8.5.4.

7.5.3 Monte-Carlo Interval Simulation

Another method to simulate intervals is proposed using Monte-Carlo methods (Rubinstein, 1981). For each interval a number of sub-intervals are chosen at random with a very small initial width. Each sub-interval is then simulated individually and the results of each simulation are used to update a central simulation repository containing the union of all fuzzy intervals for all variables. The motivation behind this is that due to the very small initial intervals the simulated trajectories will not suffer from much divergence. Also, the random intervals should show more of the real solutions quicker than the sub-interval method. The number of iterations can be set to a threshold, however the process can be stopped at any time if the desired output is met.

Monte-Carlo techniques have been criticised for being slow and missing combinations of inputs (Kahaner et al., 1989). JMorven addresses these issues; firstly, since each initial state can be simulated individually as before, parallelisations can be used to speed up the process dramatically. The problem with missing combinations is less of a problem in JMorven since each simulation reasons about an interval. This has two benefits:

- Since intervals are used, it is more likely to cover the whole original interval as a finite number of intervals can be used to recreate the interval

- Due to interval calculations diverging, the output trajectory bounds more than just the real solution of the initial intervals, hence the missing combinations are likely to be bounded by these extraneous inclusions.

Theoretically the Monte-Carlo Interval Simulation method is complete as the number of iterations tends toward infinity. However it is not sound as there is still a degree of interval divergence apparent due to each iteration having an initial interval width greater than zero.

7.6 Point simulation

Point simulation is defined in JMorven as simulating with all intervals having zero width, i.e. real numbers. These points are simulated precisely and the trajectory remains with zero width over all time. Point simulation itself can therefore be thought of as a traditional numerical simulation technique and offers the same advantages and disadvantages. The main advantage is that a dynamic system can be simulated precisely for the precisely known initial state. The disadvantage is that this method alone cannot reason with any imprecision. This method can be used to make use of its advantage and overcome the disadvantage by approximating an interval as a group of points. The following subsections discuss several methods in which this is implemented.

7.6.1 Extreme Point Simulation

The Extreme Point Simulation method uses the Point Simulation method to simulate from a number of states. The intervals in the initial state are used to create two points, one at each extreme of the interval.

The motivation behind this method is that there is no interval divergence in each of the simulations hence the final output should have no divergence. Taking the extreme points of each interval gives an approximate range of possible values whilst maintaining an efficient method.

This method is sound but incomplete in that the solution found contains no spurious solutions but it does not cover every possible solution. A method which improves upon the completeness is proposed in the next section.

7.6.2 Regular-Spaced Point Simulation

As with the sub-interval method, the Regular-Spaced Point Simulation method takes each interval in the initial state and splits it into a number of states. These states contain a number of points regularly spaced which approximates the interval.

The motivation behind this is that the point method guarantees no unnecessary divergence and using a set of points for each interval should cover most of the possibilities for the final solution.

As the number of points increases the final solution tends toward the real solution, hence this method is theoretically sound and complete as the number of points tends to infinity. This is due to each point being infinitesimally close to the neighbouring point so that no values are missed out. Since points are being simulated, there is no initial interval width therefore no unnecessary divergence occurs.

As with the sub-interval method, this method is exponential in the number of intervals however it also has the benefit that not every state will be consistent and it can also benefit from parallelisations as shown in 8.5.4.

7.6.3 Monte-Carlo Point Simulation

The final method of simulation in JMorven uses Monte-Carlo methods to simulate a number of points from the initial state intervals. Each interval is taken and a number of points are chosen at random within the bounds of the interval. Each state is simulated using the Point Simulation approach.

The motivation behind this method was to combine the advantages of all other methods

of simulation in JMorven. Using points guarantees that the solution is sound and using Monte-Carlo methods makes the solution tend to completeness more efficiently than Regular-Spaced Point Simulation. This method can also be parallelised to benefit from parallel machines as shown in 8.5.4.

7.7 Parallel Simulation

The simulation process of a single point or interval can be carried out in a small amount of time. The extended methods described above use multiple instances of this simulation method hence run slower; therefore some method is required to speed up the process. Since each instance can be executed independently it can be executed in its own parallel unit. All of the above methods (apart from basic interval simulation) can benefit from parallelisations. A number of containers are created equal to the number of execution units available and each initial state is distributed equally among these containers. Each container is then executed in its own execution unit and a single repository of intervals is updated from the output of each simulation.

7.8 Summary

Simulation of intervals is not a trivial process especially when required to deal with algebraic loops. JMorven defines a number of non-constructive simulation approaches which are summarised in table 7.2. The performance of each of these is discussed in chapter 8.5

Approach	Sound?	Complete?	Comments
Basic Interval	N	Y	Rapid interval divergence occurs
Sub-Interval	Y*	Y	Less divergence but computationally expensive
Monte-Carlo Interval	N	Y**	Still suffers from divergence
Extreme Point	Y	N	No divergence but doesn't cover all real solutions
Regular-Spaced Point	Y	Y**	No divergence, close to covering all real solutions but slow
Monte-Carlo Point	Y	Y**	No divergence, close to covering all real solutions most efficiently

Table 7.2: Summary of Simulation Approaches used in JMorven
* Soundness is achieved as the interval width tends to zero, i.e. as the number of intervals tend toward infinity.
** Monte-Carlo methods achieve completeness as the number of iterations tend toward infinity. The Regular-Spaced method achieves this as the number of points tend toward infinity.
In practice, tending toward infinity is not required; merely tending toward the resolution of the floating point representation is required although this would still result in too many iterations to complete in a reasonable amount of time.

Chapter 8

Results

The JMorven framework has been described in chapters 6 and 7. In this chapter several experiments have been undertaken using the JMorven framework to test the hypotheses made including:

- The use of parallel algorithms to speed-up all stages of execution.
- The use of auxiliary variables in a non-constructive environment.
- The use of n-th Taylor Series for a better approximation to integration.
- Using several techniques to simulate fuzzy intervals.
- Displaying asymptotically sound and complete simulation of fuzzy intervals.
- Carrying out semi-quantitative and quantitative simulations using a qualitative model.
- The ability to simulate on the spectrum from fully qualitative to fully quantitative.

The experiments conducted to test these hypotheses are shown below and the reasoning behind them:

- a verification test for the qualitative aspects of JMorven.
- a model with an algebraic loop simulated to show JMorven can reason with algebraic loops.
- speed-up benefits of the parallelisations in all qualitative stages.
- more accurate non-constructive integration using n-th order Taylor Series and demonstrating the ability to simulate quantitatively from a qualitative model.
- non-constructive numerical simulation methods for semi-quantitative and quantitative models and showing that one method is asymptotically sound and complete. This experiment also shows JMorven's ability to use auxiliary variables within a non-constructive environment.
- a model simulated on the spectrum from fully qualitative to fully quantitative.
- speed-up benefits of numerical simulations.

8.1 Qualitative Experiments

Since JMorven is capable of reasoning in a fully qualitative manner, it is relevant to carry out a basic experiment to ensure that the output is as expected with respect to its predecessors FuSim (Shen, 1991) and Morven (Coghill, 1996). To recap, JMorven succeeds Morven in that the underlying algorithms are implemented in a non-constructive manner which allows JMorven to reason with general models regardless of whether they are causally ordered or contain algebraic loops. Constructive techniques were used in Morven as they were thought to generate fewer spurious behaviours however this was shown not to be the case (Coghill, 1996) as the output from a constructive and non-constructive algorithm are identical (Coghill and Chantler, 1999).

JMorven adds two main additional features which did not exist at all in its predecessors.

The first being a parallel algorithm which allows JMorven to be executed in environments where multiple execution units are available. This novel feature speeds up execution greatly therefore allowing more complex model behaviours to be generated in less time. Another addition to JMorven is a non-constructive numerical simulation algorithm which can provide semi-quantitative and fully quantitative output behaviours (this stage can also be carried out in parallel). The inclusion of such a simulation algorithm allows JMorven to simulate output behaviours as accurately as possible with all of the available information. This ranges from imprecise variable and parameter values using fuzzy numbers to exact numerical simulations using points.

8.1.1 Envisionments

To verify the validity of the pure qualitative aspect of JMorven, the coupled tanks system is used. This is shown diagrammatically in figure 8.1

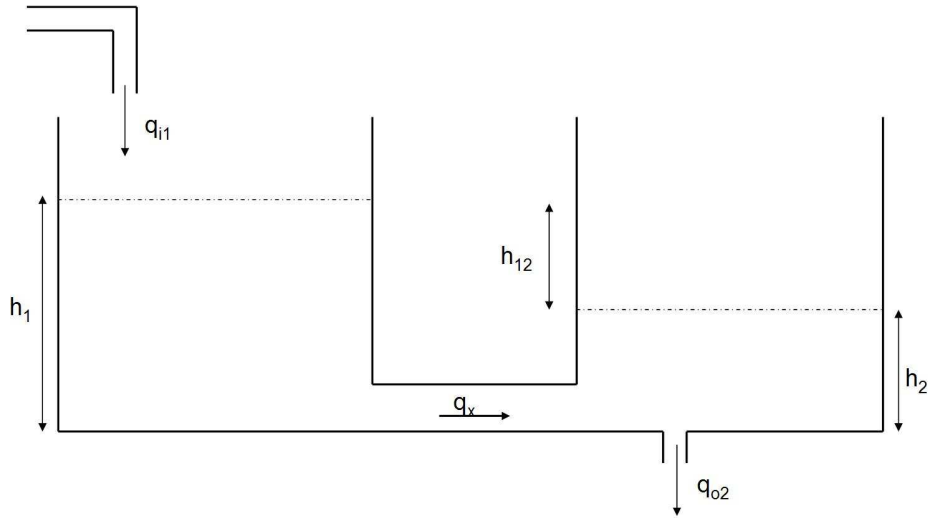


Figure 8.1: Coupled tanks model
showing two tanks of water with heights h_1, h_2 and their difference h_{12}. One inflow tap q_{i1} and one outflow plug q_{o2} determine the flow in and out of the tanks and the cross-flow q_x describes the flow between them.

and is described by equations below (the model is also shown in JMorven format in Appendix A.2):

$$q_{o2} = M^{+}(h_2)$$

$$q_x = M^{+}(h_{12})$$

$$h_{12} = h_1 - h_2$$

$$h_1' = q_{i1} - q_x$$

$$h_2' = q_x - q_{o2}$$

The coupled tanks model was used to generate a total envisionment and complete envisionment holding the exogenous variable $q_{i1} = [+ \quad 0]$. The total envisionment produces 188 states and 19941 transitions. The complete envisionment results in a graph with 28 states with 71 transitions between them, and one equilibrium state as shown:

$$h_1 = [+ \quad 0 \quad 0]$$

$$h_2 = [+ \quad 0 \quad 0]$$

$$h_{12} = [0 \quad 0]$$

which is the expected result. The graph and state repository are shown in Appendix B.1 and Appendix C.2 respectively. This output has been compared to the output of Morven and verified as correct. Although this was a trivial experiment to carry out it was necessary to ensure the results of the JMorven qualitative reasoner matches that of its predecessor.

8.1.2 Simulation

The verification of the qualitative simulation in JMorven was tested using a simple mass on a spring system. This model is described by the following equation:

$$x'' = F - mx$$

where x'' is the acceleration of the mass m, x is the displacement of the mass from its equilibrium position and F is the Force on the mass. The JMorven model can be found in Appendix A.3. Figure 8.2 shows the directed graph within JMorven of the output of this model when using the signs quantity space. From the graph, it is clear to see that some sort of cyclic behaviour is apparent. Upon analysis of the state repository (see Appendix C.1) it can be seen that this is indeed the oscillatory behaviour expected of the mass on a spring.

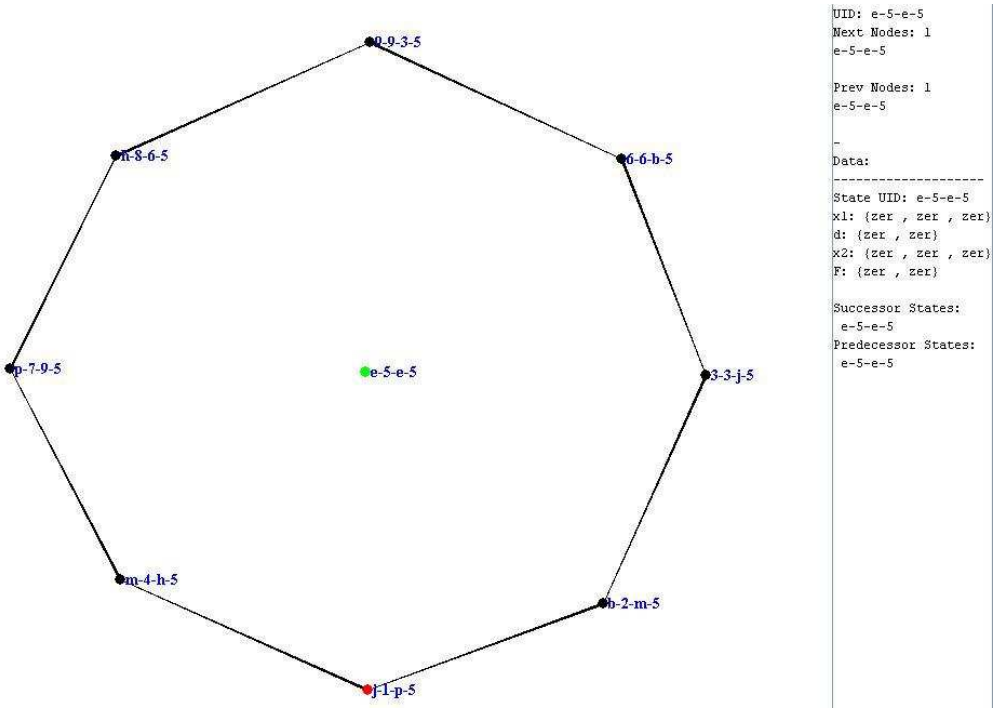


Figure 8.2: Mass on a Spring Simulation Graph
Nodes represent states within the state repository and edges denote transitions between states. Edges are directed from thick to thin to represent the direction of the transition. Self transitions are not shown on the graph. The node shown in red was the one chosen for the initial state.

8.2 Algebraic Loops

JMorven uses non-constructive algorithms which have the benefit of being able to reason with algebraic loops. This experiment uses a model of an electrical circuit defined by the equations in section 2.4.1 and the JMorven model can be found in appendix A.4. JMorven loops round the constraints until all intervals are narrowed as much as possible as shown in the worked example in section 2.4.2. This example was used to test JMorven and the following output was observed:

```
Starting simulation...
Before narrowing and constraining the initial state:
Variable:  u3
        Derivative:  0 Initial:u3:0:Set (0.0 , 100000.0 , 0.0 , 0.0)

After narrowing and constraining the initial state:
Variable:  u3
        Derivative:  0 u3:0:0.0:Set (50.0 , 50.0 , 0.0 , 0.0)
```

which is exactly as expected. To test handling algebraic loops with intervals the following initial state was given to JMorven:

$$R1 = [1990 \quad 2010]$$

$$R3 = [19500 \quad 20500]$$

$$U_0 = [99 \quad 101]$$

$$i_2 = [0.014 \quad 0.016]$$

The output from JMorven is show below:

```
Starting simulation...
Before narrowing and constraining the initial state:
Variable:  u3
        Derivative:  0 Initial:u3:0:Set (0.0 , 100000.0 , 0.0 , 0.0)

After narrowing and constraining the initial state:
Variable:  u3
        Derivative:  0 u3:0:0.0:Set (59.900314 , 67.32529 , 0.0 , 0.0)
```

JMorven calculated the value of u_3 to be $[59.900314 \quad 67.32529]$ which can be verified by working through the equations by hand. These simple tests show that JMorven works non-constructively and can reason with algebraic loops.

8.3 Analysis of Parallelisation Benefits

One of the major contributions of the JMorven framework is that of the parallel algorithms for all major stages of qualitative analysis. As discussed in chapter 3, the optimal performance of a parallel algorithm is linear and when the efficiency

$$E = \frac{S_n}{n} = 1$$

where S_n is the speed-up observed when running on n execution units. Platzner and Rinner (Platzner et al., 1997) report an average speed-up $\bar{S_7} < 2$ which is far from optimal. This section discusses the results of the parallelisations within JMorven and how they compare to the findings of Platzner and Rinner.

8.3.1 Tuple Filter

The times were recorded using the average of five executions of the full coupled tanks system with 21 quantities in the quantity spaces and 2 input flows and 2 output flows as shown in figure 8.3.

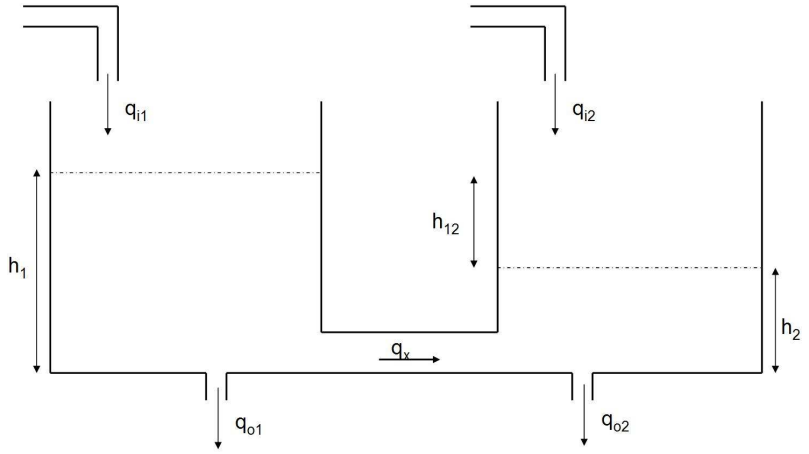


Figure 8.3: Full Coupled tanks model
showing two tanks of water with heights h_1, h_2 and their difference h_{12}. Two inflow taps q_{i1}, q_{i2} and two outflow plugs q_{o1}, q_{o2} determine the flow in and out of the tanks and the cross-flow q_x describes the flow between them.

The Tuple Filter occupies approximately 5% of the running time of the qualitative analysis phase. Parallelisations aim to make it execute more quickly when using multiple execution units. The results achieved are summarised below in table 8.1. The benefits of the parallelisations are not as much as initially thought however due to the small amount of time of execution of the tuple filter, the implementation was not optimised (the tuple filter takes less than 5% of the running time of the QA phase and approximaitely 1% of the total running timwe). It can be seen from the table that there is a benefit from the parallelisations; the implementation could be optimised to achieve closer to linear speed-up.

The method used to parallelise the tuple filter has a few drawbacks in that the number of execution units to be used is limited by the number of constraints in the model. This means that the maximum number of execution units to be utilised depends on the complexity

	number of execution units, n							
	1	2	3	4	5	6	7	8
Time Taken (s)	4.744	4.085	3.600	3.197	2.693	2.521	2.336	2.183
Speed-up, S_n	1.000	1.161	1.318	1.484	1.762	1.882	2.031	2.173
Efficiency, E	1.000	0.581	0.493	0.371	0.352	0.314	0.290	0.272

Table 8.1: Parallelisation Benefits of the JMorven Tuple Filter

of the model. Also, in a worst-case scenario, there would be one more constraint than number of execution units available, and all tuples are filtered in exactly the same time resulting in the tuple filter taking twice as long to execute than it would with just one less constraint.

This limitation is not too great a problem since the tuple filter takes a very small amount of time to complete especially for simple models with a small number of constraints. As the model grows in complexity, so does the number of constraints; therefore the maximum number of usable execution units is also increased.

8.3.2 Pairwise Filter

The pairwise filter is the least computationally expensive stage of the qualitative analysis phase. The times were recorded using the average of five executions of the full coupled tanks system with 21 quantities in the quantity spaces and 2 input flows and 2 output flows. Due to the small execution time there is very little benefit from the parallelisations as can be seen in table 8.2.

	number of execution units, n							
	1	2	3	4	5	6	7	8
Time Taken (s)	0.646	0.455	0.382	0.339	0.356	0.360	0.374	0.344
Speed-up, S_n	1.000	1.420	1.691	1.906	1.815	1.794	1.727	1.878
Efficiency, E	1.000	0.710	0.564	0.477	0.363	0.299	0.247	0.235

Table 8.2: Parallelisation Benefits of the JMorven Pairwise Filter

If there are C constraints in the model, then there will be at most $C^2 - C$ possible pairings of constraints to be filtered in the pairwise filter. This suffers from the same drawbacks as

the tuple filter however there are more pairs to be executed than constraints in the tuple filter so the limitations are not so apparent.

The results look positive for the first half of the table, using 4 processors over 1 takes around half the time to execute, however the benefits then plateau. This is due to the small amount of time taken for this stage since there is not much work to be done. The algorithm could be optimised to allow for a greater benefit from the parallelisations but since the state generator is by far the most expensive stage, most of the effort was concentrated on it.

8.3.3 State Generator

The state generator in JMorven is the most computationally intensive stage, occupying approximately 95% of the running time of the QA phase therefore this is the most important stage for which to optimise parallelisations. The times were recorded using the average of five executions of the coupled tanks system with 9 quantities in the quantity spaces and 1 input flow and 1 output flow. The results are summarised in table 8.3.

	number of execution units, n							
	1	2	3	4	5	6	7	8
Time Taken (s)	41.452	20.631	14.689	11.645	9.524	8.484	8.214	6.830
Speed-up, S_n	1.000	1.999	2.822	3.560	4.352	4.886	5.047	6.069
Efficiency, E	1.000	1.000	0.941	0.890	0.870	0.814	0.721	0.759

Table 8.3: Parallelisation Benefits of the JMorven State Generator

It can be seen that there is a large performance increase from the parallelisations in the state generator. Using eight execution units results in an observed speed-up greater than six times over the sequential version which is a huge benefit. The speed-up is close to linear. However, there is a drop in efficiency which could be due to communication costs from the master node to the child parallel units, although it is not known how this could be tested. There is another area where the parallel algorithm is not fully utilised - for the first few iterations of a simple model, there are not enough tuples to make use of the

maximum number of execution units available, therefore the maximum speed-up will not be observed during these iterations. This limit is not apparent after the first few iterations as all execution units can be used and hence optimal speed-up can be obtained from then on. The machine on which tests were carried out is an eight processor Sun server running Solaris 5.8. The threading model in this version of Solaris is quite old and it is thought that a newer version may show even greater benefits from the parallelisations.

Since the state generator is the most computationally expensive stage in qualitative analysis; this means that even if only the state generator were implemented in parallel the whole process would be speeded up greatly. There is no shared memory used in this algorithm for generating states therefore the algorithm would also work well in a distributed computing environment.

8.3.4 Transition Analysis

The transition analysis phase was also parallelised. This is responsible for calculating the transitions between states. Table 8.4 shows the results of the parallelisations. The model used was the coupled tanks model with nine quantities in the quantity space for each variable and one input and output, however the α-cut was set to zero to obtain the maximum number of transitions possible. There were 755 states with 13966 transitions in this test.

	number of execution units, n							
	1	2	3	4	5	6	7	8
Time Taken (s)	8.466	4.598	3.489	2.750	2.394	2.170	1.905	1.677
Speed-up, S_n	1.000	1.841	2.426	3.079	3.536	3.901	4.444	5.048
Efficiency, E	1.000	0.921	0.809	0.770	0.707	0.650	0.635	0.631

Table 8.4: Parallelisation Benefits of the JMorven Transition Analysis

The transition generation shows very good benefits from the parallelisations. This stage occupies a lot of the overall execution time of an envisionment, typically over 50% if transitions are required.

8.3.5 Qualitative Parallelisations Summary

The qualitative stages of JMorven have been parallelised and the results of each individual stage have been presented above. For an envisionment with transitions calculated the QA phase occupies just under 50% of the total runtime leaving just over 50% for the TA phase. The QA phase is dominated by the State Generation which typically occupies about 95% of the runtime. The transition generation (the only stage in TA) and state generation stages together dominate the total runtime therefore these are the stages which are required to show the greatest benefits from parallelisations. From the results, it is clear that these two stages do benefit greatly from the parallelisations. The state generator shows the biggest improvements; this is due to most optimisations being undertaken on this stage. The transition generation suffers a little in that it does not display as much of a speed-up as the state generator. The tuple and pairwise filters have a good theoretical algorithm for parallel computation however are let down by the implementation in JMorven. There are still benefits from the parallelisations showing that it is possible to speed-up these stages too, but since these stages do not occupy much of the total running time they have not been optimised like the transition and state generators.

8.4 n-th Order Taylor Series Integration

JMorven uses n-th order Taylor Series to integrate in a non-constructive manner. The following example shows the difference between using first order (Euler Integration) and second order Taylor Series Expansions to simulate a simple mass on an undamped spring model. The model is described by the following equation:

$$x'' = F - mx$$

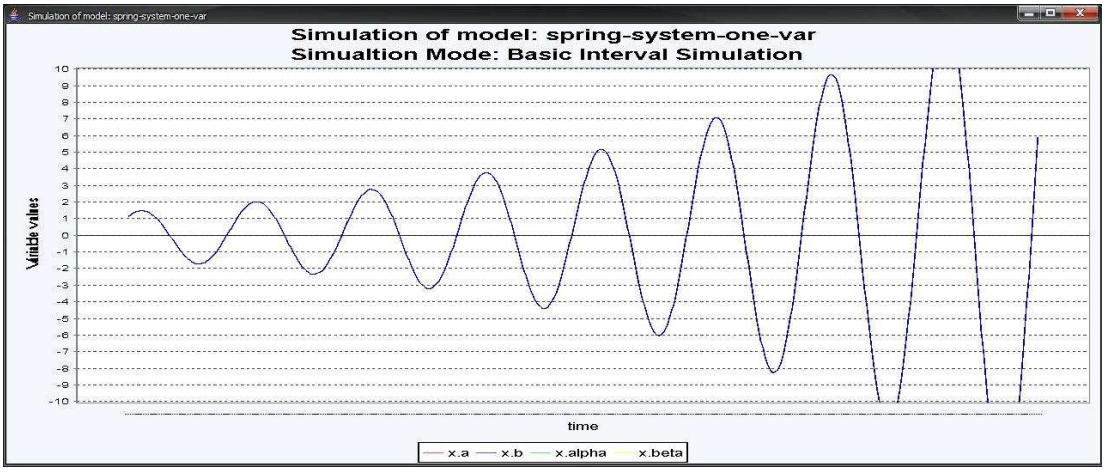


Figure 8.4: Undamped spring model simulated using Euler Integration. Quantitative simulation of the mass on a spring showing the trajectory of x

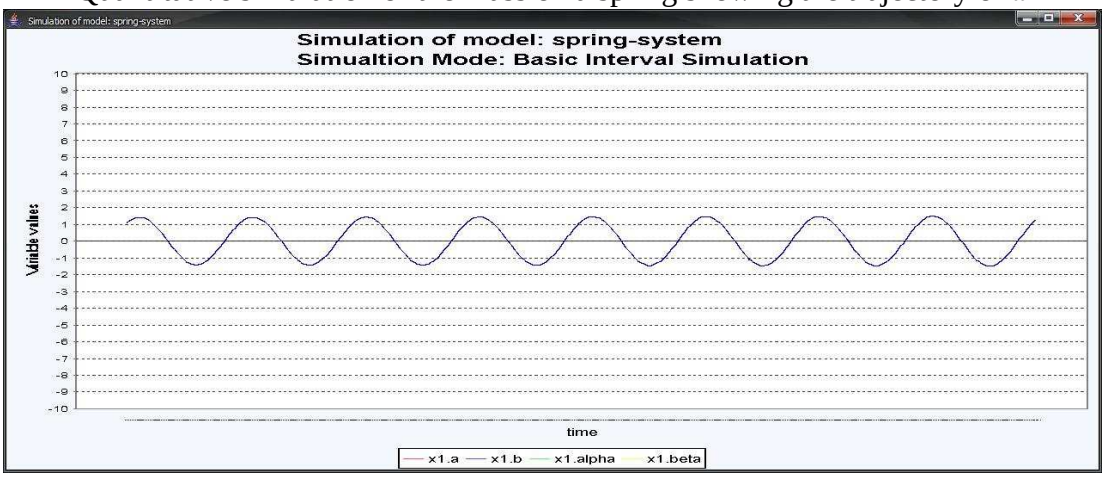


Figure 8.5: Undamped spring model simulated using second order Taylor Series. Quantitative simulation of the mass on a spring showing the trajectory of x

This qualitative model is given the quantitative initial state specifying $x = 1, x' = 1, x'' = -1$. JMorven can simulate this quantitatively from the quantitative initial state with no modification to the qualitative model. Figure 8.4 shows the simulated behaviour using non-constructive Euler Integration. The model is undamped and has no external forces therefore the amplitude of oscillations should remain constant however due to errors in the integration approximations the amplitude is unstable and is increasing exponentially. Figure 8.5 shows the same model but using a second order non-constructive Taylor Series to approximate the integration step. It can be seen that the amplitude of oscillations remain far more stable. From this simple experiment, it is clear that when using a higher order integration estimate, it is possible to provide more accurate simulations.

8.5 Fuzzy Interval Simulation

In addition to qualitative simulation, JMorven also presents a framework for simulating at a semi-quantitative level and fully quantitative level. The fuzzy calculations are based on using intervals to depict fuzzy numbers as detailed in chapter 7.1. Setting these intervals to have zero width results in a fully quantitative simulation. The results of each method and a brief discussion is presented below with a more in-depth discussion and conclusion presented in chapter 9.

8.5.1 Simulation Methods Using Real Intervals

This section discusses the results of the simulation modes in JMorven that use real intervals for simulating trajectories of models. The model used to test out the semi-quantitative simulations is simple mass on a spring. The model description can be found in Appendix A.3. The model was simulated using the initial values of $x = 1$, $x' = 1$; these are precise values with intervals of zero width hence JMorven carried out a fully numerical simulation to obtain the output graph as shown in figure 8.5.

8.5.1.1 Basic Interval Simulation

The mass on a spring was simulated with an initial state specifying $x = [0.9, 1.1, 0.05, 0.05]$, $x' = [1]$ and $F = [0]$ and the rest of the values remain unspecified. The basic interval simulation output can be seen in figure 8.6. The simulation is of the first ten seconds from the initial state. It can be seen that even with a fairly precise input fuzzy interval quite a lot of excessive widening occurs. This output is not very useful as it is not apparent that any of the expected oscillations occur. This demonstrates the problem with basic interval simulation.

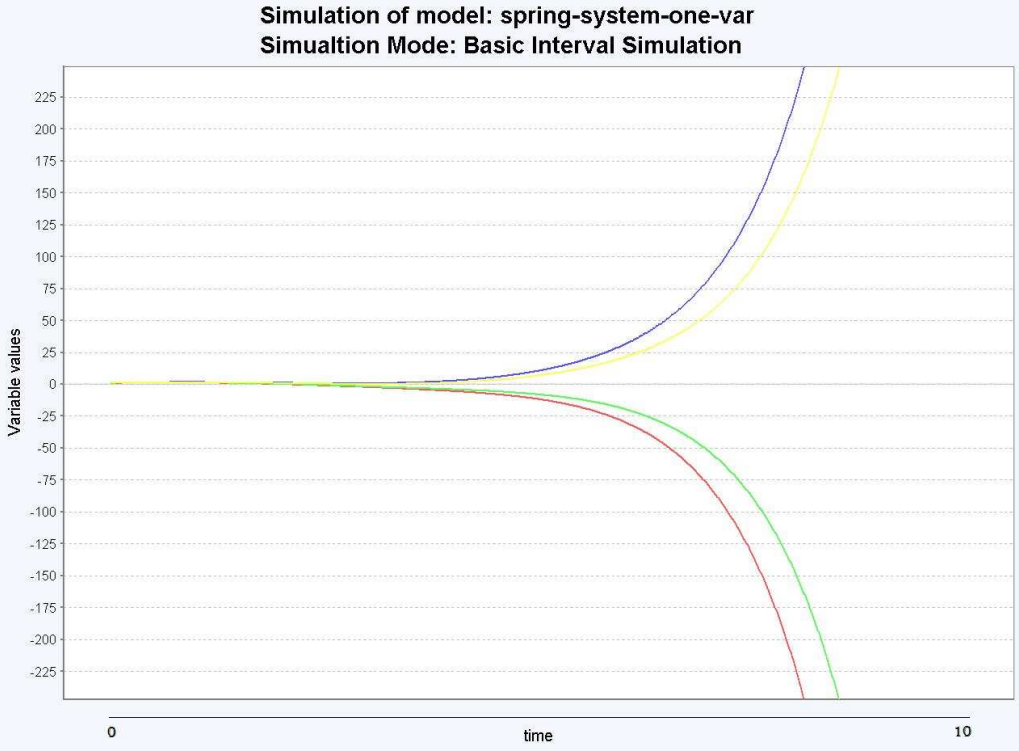


Figure 8.6: Basic Interval Simulation of Mass on a Spring
Simulation of the mass on a spring showing the trajectory of x. The blue line indicates the simulated trajectory of the $a - \alpha$ fuzzy parameter. The yellow line is the a parameter, green is b and red is $b + \beta$.

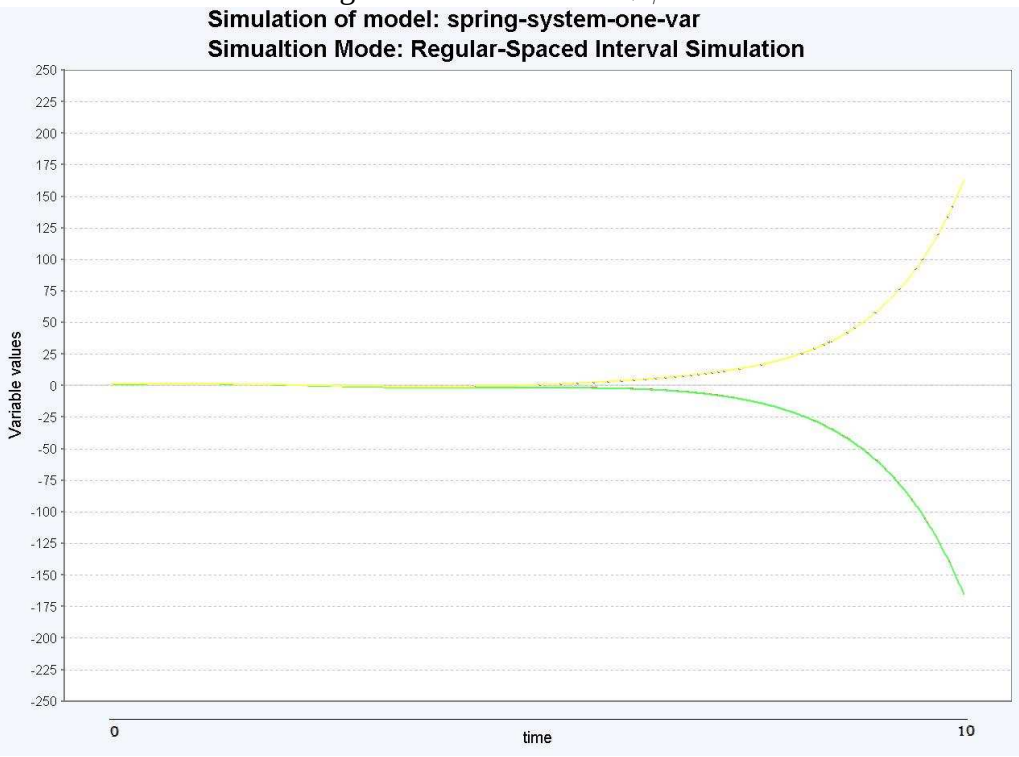


Figure 8.7: Sub-Interval Simulation of Mass on a Spring
Simulation of the mass on a spring showing the trajectory of x using 10 sub-intervals

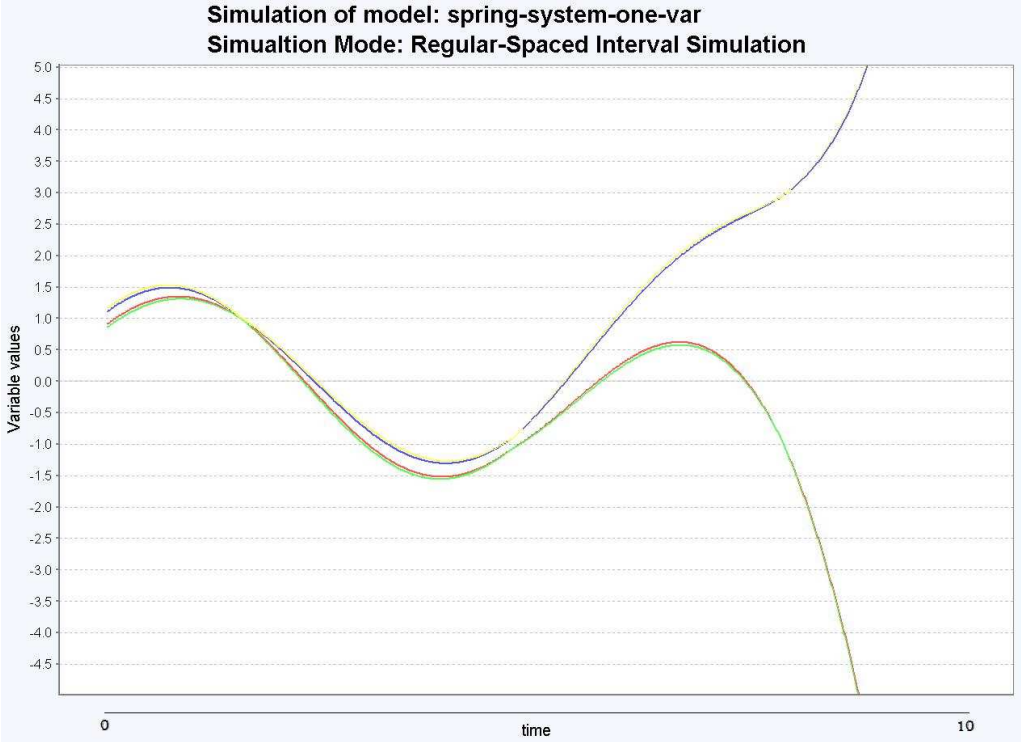


Figure 8.8: Sub-Interval Simulation of Mass on a Spring
Simulation of the mass on a spring showing the trajectory of x using 100 sub-intervals

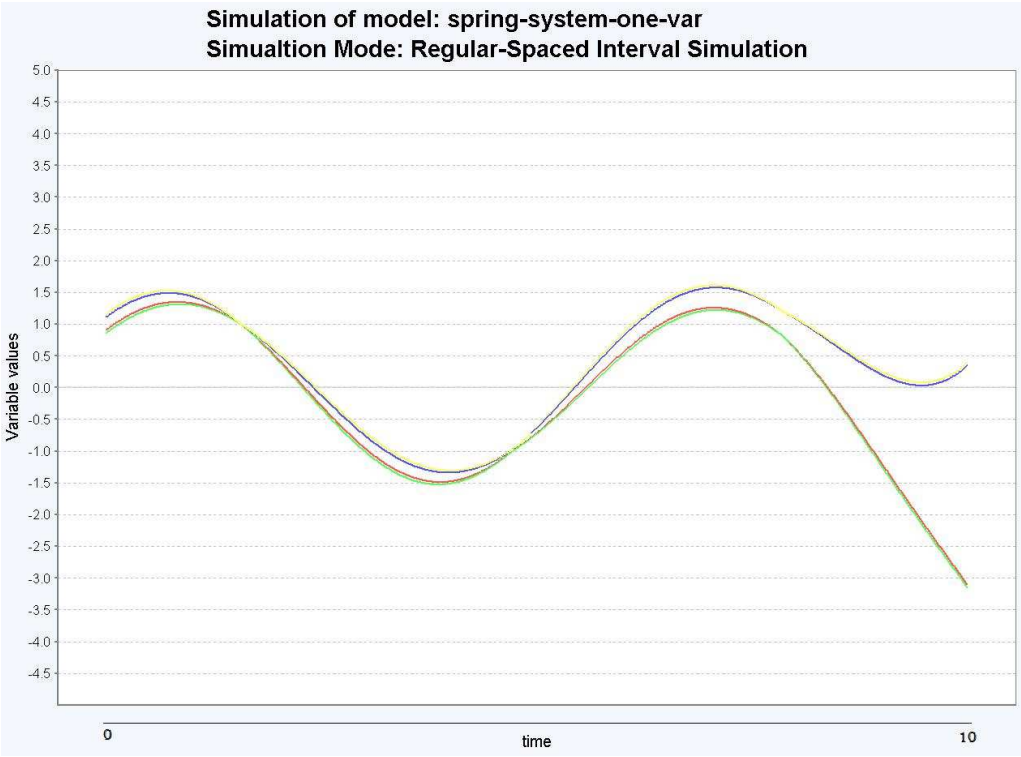


Figure 8.9: Sub-Interval Simulation of Mass on a Spring
Simulation of the mass on a spring showing the trajectory of x using 1000 sub-intervals

8.5.1.2 Regular-spaced Interval Simulation

One method that JMorven uses to simulate intervals more precisely is regular-spaced interval simulations. The same inputs were used for these tests as in section 8.5.1.1. Figure 8.7 shows the output using 10 sub-intervals. It is clear to see that the interval still diverges; however it does not widen as rapidly as with basic interval simulation. Figure 8.8 shows the output using 100 sub-intervals. It can be seen that now the interval widens far less and some useful trajectory results can be seen. Figure 8.9 shows the output with 1000 sub-intervals. From this set of outputs it can be seen that as the number of sub-intervals is increased the resulting interval suffers less from excessive widening. Theoretically, as the number of sub-intervals approaches infinity the output trajectory will approach the real solution since each interval will have infinitesimal width. The major drawback with this method is that it takes a long time to execute since each sub-interval needs to be simulated individually.

8.5.1.3 Monte-Carlo Interval Simulation

The best simulation method in JMorven that reasons with real intervals is the Monte-Carlo Interval method. The results of the simulation with this method are shown in figure 8.10. The advantages of this method are that the interval does not widen as rapidly as with the sub-interval method even when using 1000 sub-intervals since each iteration of the Monte-Carlo method is defined as having a very small interval width. Also the execution time was far quicker as fewer iterations were required to obtain the output trajectory. The disadvantages are that this method relies on random numbers therefore there will be subtle differences between simulations of the same model with the same input parameters. Also, this method is not guaranteed to bound all possible solutions although it will approximate the output solution very closely.

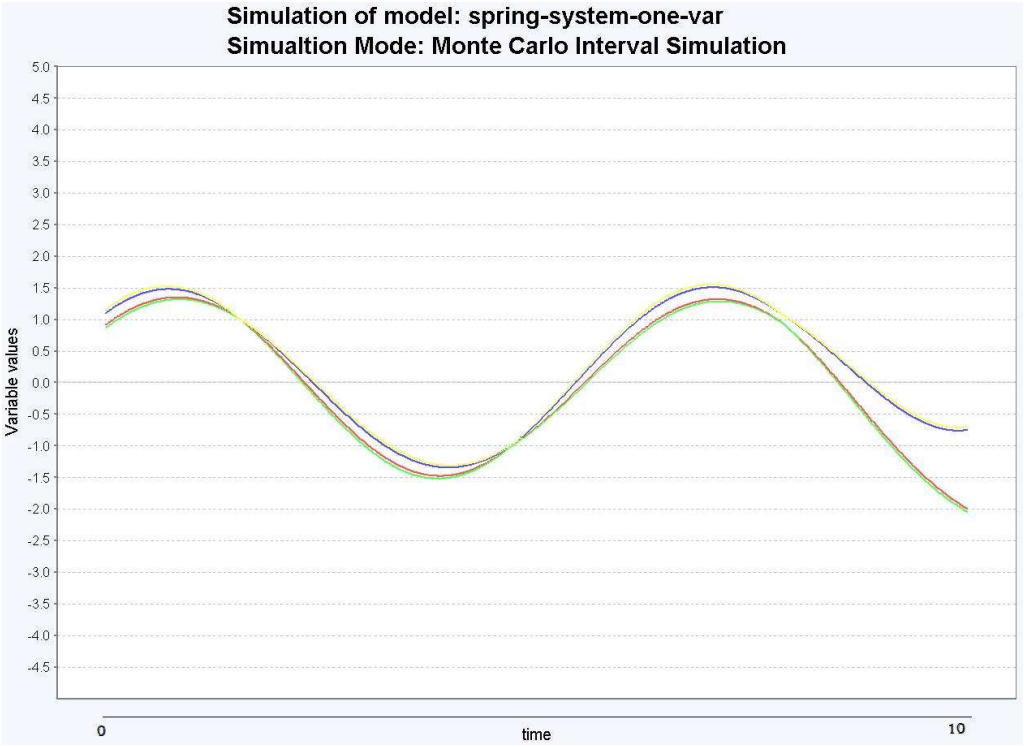


Figure 8.10: Monte-Carlo Interval Simulation of Mass on a Spring
Simulation of the mass on a spring showing the trajectory of x using Monte-Carlo methods

8.5.2 Simulation Methods by Approximating Intervals

This section discusses the results of the simulation modes in JMorven that approximate intervals for simulating trajectories of models using groups of points. The model used to test out the semi-quantitative simulations is the Van der Pol oscillator. This is described by the following equation:

$$\ddot{x} = -P(x^2 - 1)\dot{x} - Qx$$

The model description can be found in Appendix A.5. This model utilises auxiliary variables to ensure no unnecessary divergence of the intervals takes place, hence these tests also verify the ability of JMorven to use auxiliary variables in a non-constructive environment.

The experiment was simulated using the initial values of $x = 1$, $x' = 1$, $P = 1$ and

$Q = 1$; these are precise values with intervals of zero width hence JMorven was used to carry out a fully numerical simulation to obtain the output graphs as shown in figures 8.11 and 8.12. The simulations are carried out for 40 seconds after the initial state.

8.5.2.1 Extreme Points Simulation

There are some methods included in the JMorven framework to simulate by approximating intervals. The first one simulates the endpoints of each interval specified in the initial state. The Van der Pol oscillator was simulated specifying $x, x' = [0.5, \quad 1.5, \quad 0.25, \quad 0.25]$ and $P, Q = [1]$ in the initial state with all remaining values left unspecified. Figures 8.13 and 8.14 show the output trajectories of the model for x and x' respectively. Using the extreme points method provides an output with no widening of the intervals but not all of the solutions are bound by the output. However this method does provide a very efficient approximation to the desired output. The implementation of this method takes the extreme points of the fuzzy interval as $[(a - \alpha) \quad (b + \beta)]$ therefore the output is a single interval rather than fuzzy.

8.5.2.2 Regular-Spaced Point Simulation

The regular spaced point method is similar to the regular-spaced interval method except that points at regular spaces are used instead of regular spaced-intervals. This aims to approximate the interval as a group of points so that the output will bound close to all solutions but with no excessive widening of the output trajectory since points are being simulated instead of intervals. Figures 8.15 and 8.16 show the output of the regular-spaced point method using five points to approximate each interval. Since there are two intervals in the initial state of the Van der Pol oscillator for this problem there are 25 unique combinations of points to simulate. The output provides a very good trajectory with no widening and the intervals appear relatively close to the expected output. On a close inspection it

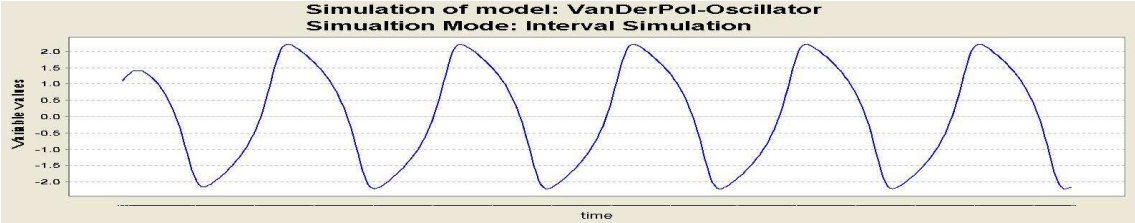


Figure 8.11: Numerical Simulation of Van der Pol Oscillator
Quantitative simulation of the Van der Pol oscillator showing the trajectory of x

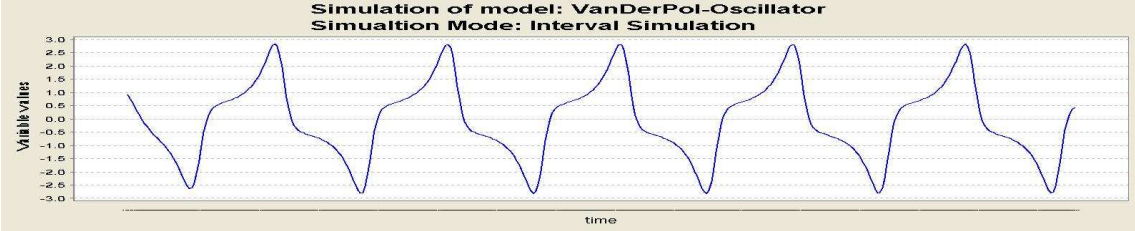


Figure 8.12: Numerical Simulation of Van der Pol Oscillator
Quantitative simulation of the Van der Pol oscillator showing the trajectory of x'

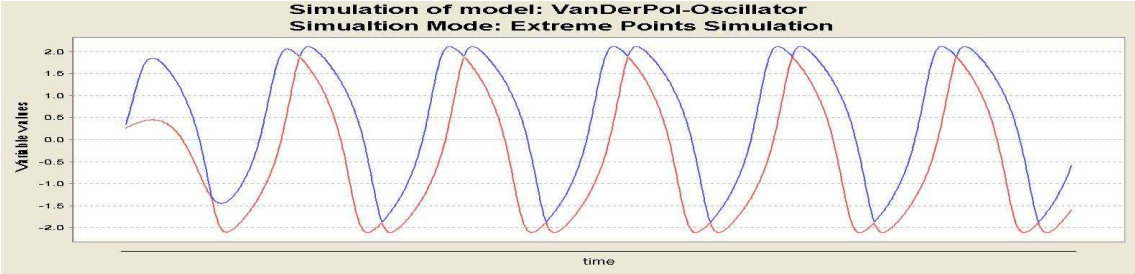


Figure 8.13: Extreme Points Simulation of Van der Pol Oscillator
Simulation of the Van der Pol oscillator showing the trajectory of x using the extreme points method

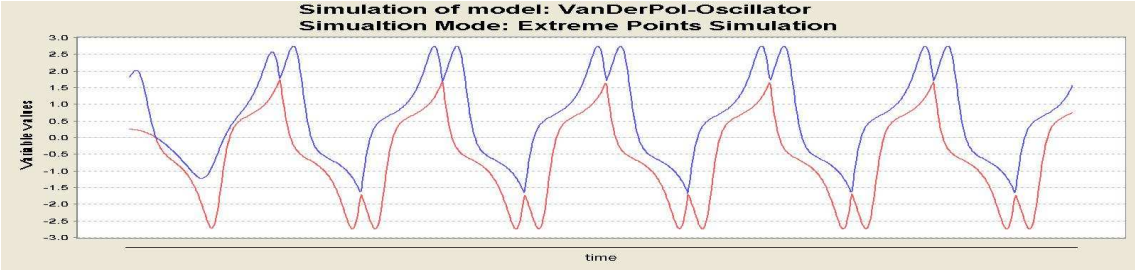


Figure 8.14: Extreme Points Simulation of Van der Pol Oscillator
Simulation of the Van der Pol oscillator showing the trajectory of x' using the extreme points method

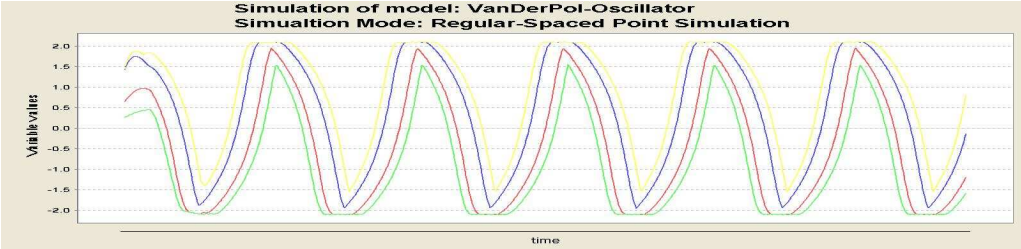


Figure 8.15: Regular-Spaced Points Simulation of Van der Pol Oscillator Simulation of the Van der Pol oscillator showing the trajectory of x using the regular-spaced points method with 5 points per interval. The yellow line indicates the simulated trajectory of the $a - \alpha$ fuzzy parameter. The blue line is the a parameter, red is b and green is $b + \beta$.

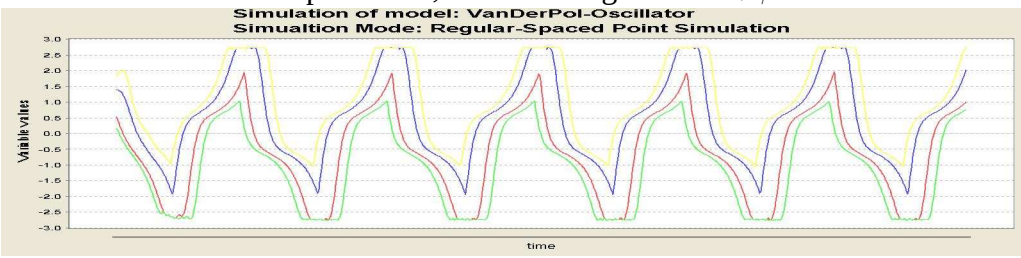


Figure 8.16: Regular-Spaced Points Simulation of Van der Pol Oscillator Simulation of the Van der Pol oscillator showing the trajectory of x' using the regular-spaced points method with 5 points per interval.

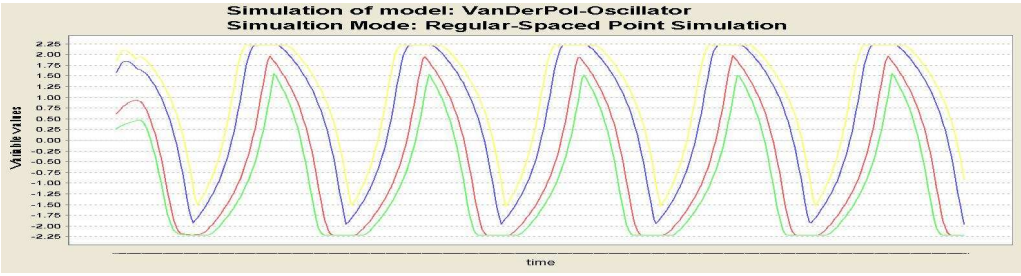


Figure 8.17: Regular-Spaced Points Simulation of Van der Pol Oscillator Simulation of the Van der Pol oscillator showing the trajectory of x using the regular-spaced points method with 20 points per interval.

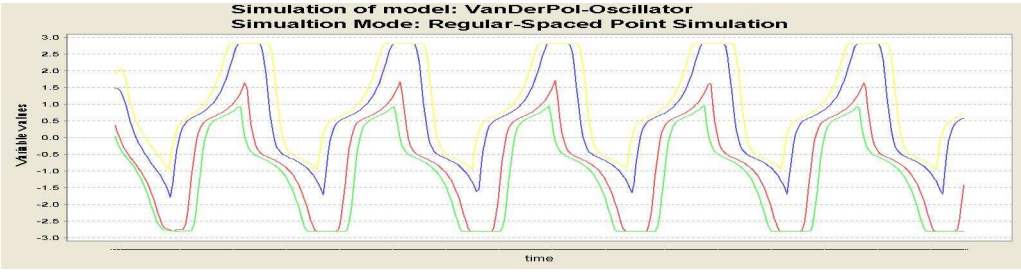


Figure 8.18: Regular-Spaced Points Simulation of Van der Pol Oscillator Simulation of the Van der Pol oscillator showing the trajectory of x' using the regular-spaced points method with 20 points per interval.

can be seen that there are a few errors around the peak of each oscillation. Figures 8.17 and 8.18 show the same method except using 20 points to approximate the interval. There is a noticeable difference between the two around the peaks of oscillations in the trajectory for x' but the main difference is observed in the first oscillation. There is very little difference between the outputs for x. Approximating the intervals using five to ten points offers a reasonable output trajectory executed in a reasonable length of time, however as the number of points is increased the results of the simulation tend to become sound and complete.

8.5.2.3 Monte-Carlo Point Simulation

The final mode for semi-quantitative simulation in JMorven is Monte-Carlo Point Simulation. This method offers the benefits of being able to produce simulations with no excess widening of the intervals. It also produces an output close to being sound and complete within the quickest time of any of the other methods. As such it is the mode most recommended for use if very accurate simulations are required.

Figures 8.19 and 8.20 show the output of the Monte-Carlo method with 100 initial states. It can be seen that these graphs are very similar to the regular-spaced point method with 20 points however the Monte-Carlo version takes far less time to execute making it a better choice. This makes the Monte-Carlo Point Simulation technique a good method to approximate the outcome of the sound and complete simulation of the regular-spaced point technique.. The disadvantage with this method however is that since it uses random points within the intervals, no two outputs will be identical.

To test the Monte-Carlo method further, the Van der Pol oscillator was simulated again except using the following initial values: $x, x', Q = [1]$ and $P = [0.9, \quad 1.1, \quad 0.1, \quad 0.1]$. Having a parameter as an interval causes a different form of output as can be seen in figures 8.21 and 8.22. The initial values for x and x' are real numbers hence have no width

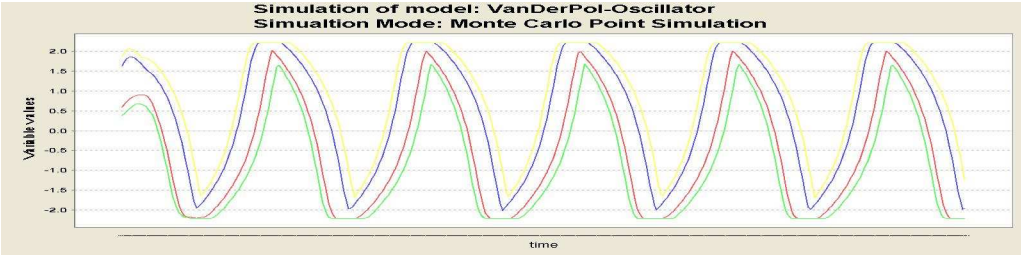


Figure 8.19: Monte-Carlo Points Simulation of Van der Pol Oscillator
Simulation of the Van der Pol oscillator showing the trajectory of x using the Monte-Carlo points method with 100 initial states.

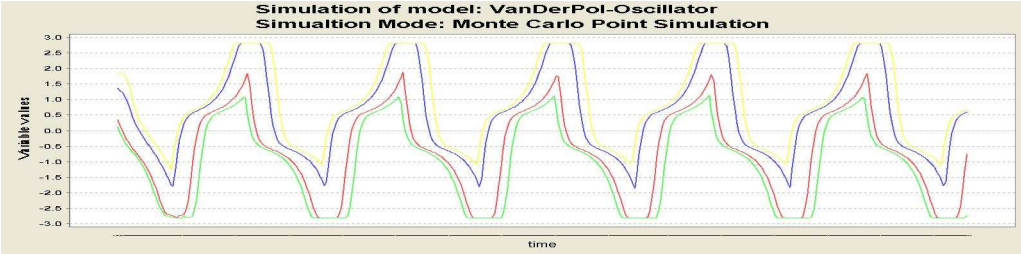


Figure 8.20: Monte-Carlo Points Simulation of Van der Pol Oscillator
Simulation of the Van der Pol oscillator showing the trajectory of x' using the Monte-Carlo points method with 100 initial states.

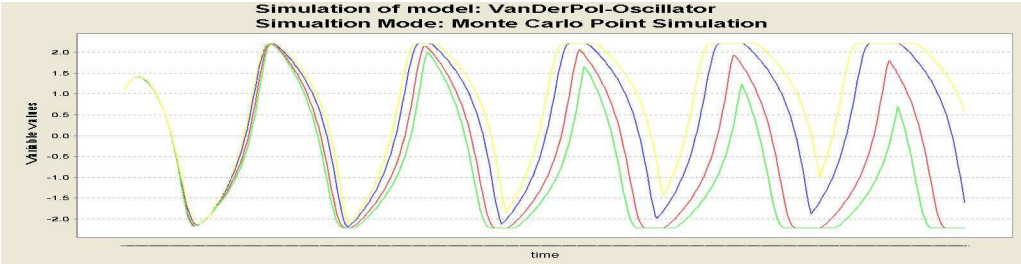


Figure 8.21: Monte-Carlo Points Simulation of Van der Pol Oscillator
Simulation of the Van der Pol oscillator with fuzzy parameter P showing the trajectory of x using the Monte-Carlo points method with 100 initial states.

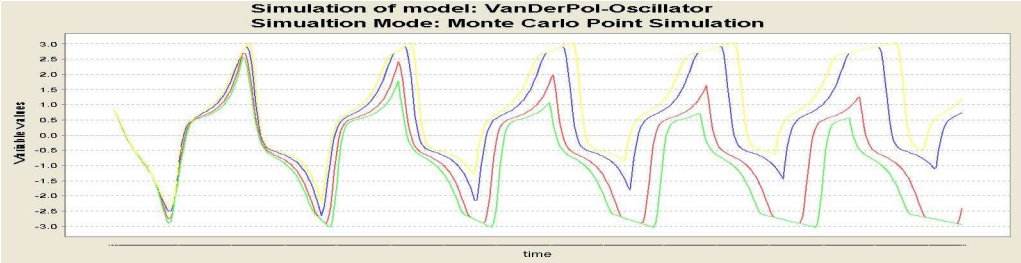


Figure 8.22: Monte-Carlo Points Simulation of Van der Pol Oscillator
Simulation of the Van der Pol oscillator with fuzzy parameter P showing the trajectory of x' using the Monte-Carlo points method with 100 initial states.

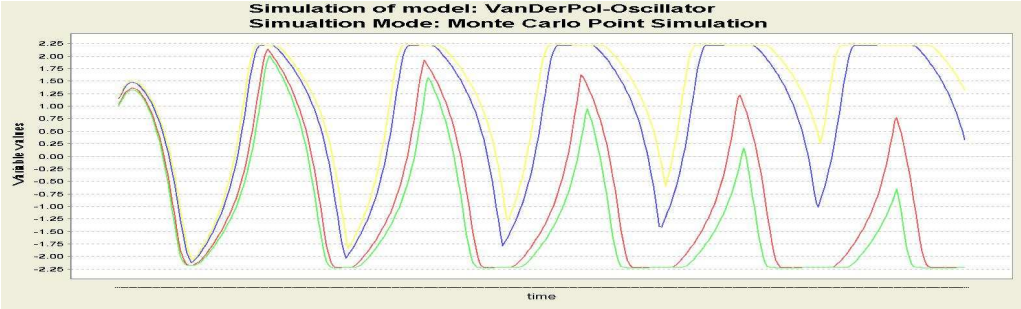


Figure 8.23: Monte-Carlo Points Simulation of Van der Pol Oscillator
Simulation of the Van der Pol oscillator with all initial values and parameters having a fuzzy interval showing the trajectory of x using the Monte-Carlo.

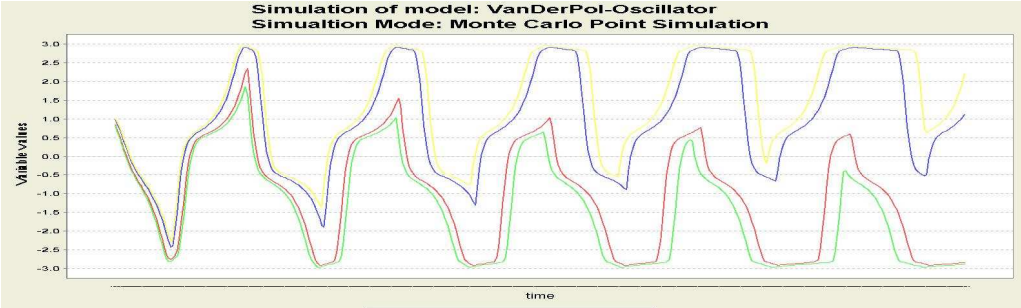


Figure 8.24: Monte-Carlo Points Simulation of Van der Pol Oscillator
Simulation of the Van der Pol oscillator with all initial values and parameters having a fuzzy interval showing the trajectory of x' using the Monte-Carlo.

however since one of the parameters has interval width it causes the values of x and x' to widen. These outputs have be verified with the results reported in (Keller, 1999) showing that JMorven produces the correct results and therefore also showing that auxiliary variables can be used in a non-constructive environment.

The final test for the fuzzy interval simulation was using the Monte-Carlo Points method on the Van der Pol oscillator with $x, x', P, Q = [0.9, \quad 1.1, \quad 0.05, \quad 0.05]$ which is similar to what a real problem might be like; i.e. all parameters and initial values having a fuzzy interval. The simulated trajectories can be found in figures 8.23 and 8.24.

The outputs show that the initial interval is very narrow and how it widens with time. This is not due to errors in simulation but due to the parameters being incompletely specified. The simulation still results in a very useful output to determine how the system could

behave in reality. Once a graph like this has been created it is possible to use it for fault detection; the initial values may represent the acceptable tolerances in the system.

8.5.3 Simulation Spectrum

This section briefly demonstrates JMorven simulating in the spectrum from fully qualitative to fully quantitative using the simple mass on a spring model as used in section 8.1.2 and defined by the following equation:

$$x'' = F - mx$$

where F is the external force applied on the mass m, x is the displacement of the mass and (x'' is the acceleration of the mass. For all experiments, the force was set to zero and the mass is set as unity.

8.5.3.1 Qualitative Simulation

The qualitative simulation results in the following state transitions in order:
$x = [+,+,-] \rightarrow [+,0,-] \rightarrow [+,-,-] \rightarrow [0,-,0] \rightarrow [-,-,+] \rightarrow [-,0,+] \rightarrow [-,+,+] \rightarrow [0,+,0]$
and then loops round again displaying oscillatory behaviour (the qualitative simulation figure is shown in figure 8.2). Even with the course quantity space of the signs it is still possible to observe a trend in the behaviour of the simulation.

8.5.3.2 Semi-Quantitative Simulation

The initial state was specified with fuzzy values $x = [0.9 \quad 1.1 \quad 0.05 \quad 0.05]$ and $x' = [0.95 \quad 1.05 \quad 0.025 \quad 0.025]$. The simulated trajectory is shown in figure 8.25.

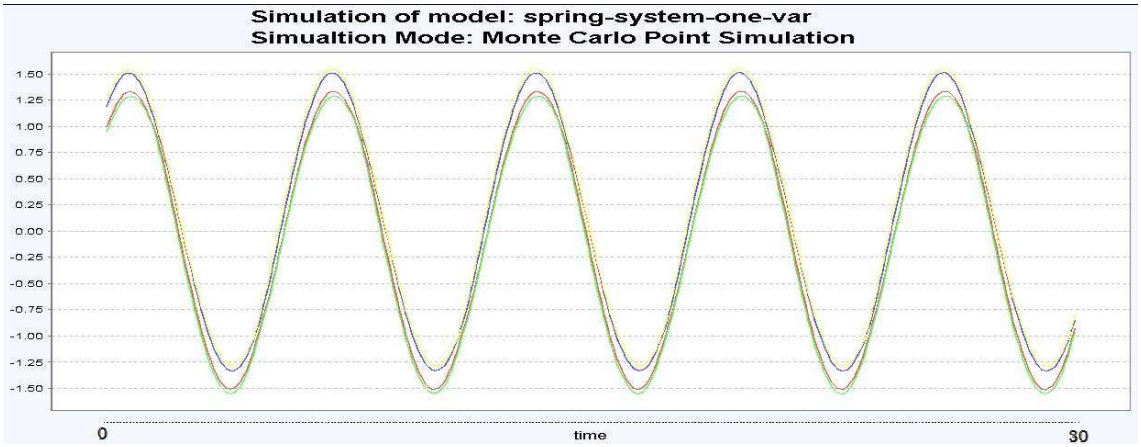


Figure 8.25: Semi Quantitative Simulation of the Mass on a Spring model

It can be seen that there are oscillations in the simulation and that the mass stays within a small threshold at all times. This is due to the intervals specified in the initial state; if the interval was wider so would the threshold be and similarly if the initial interval is narrower the threshold would be narrower too.

8.5.3.3 Quantitative Simulation

The initial state was specified with exact values $x = 1$ and $x' = 1$. The simulated trajectory is shown in figure 8.26.

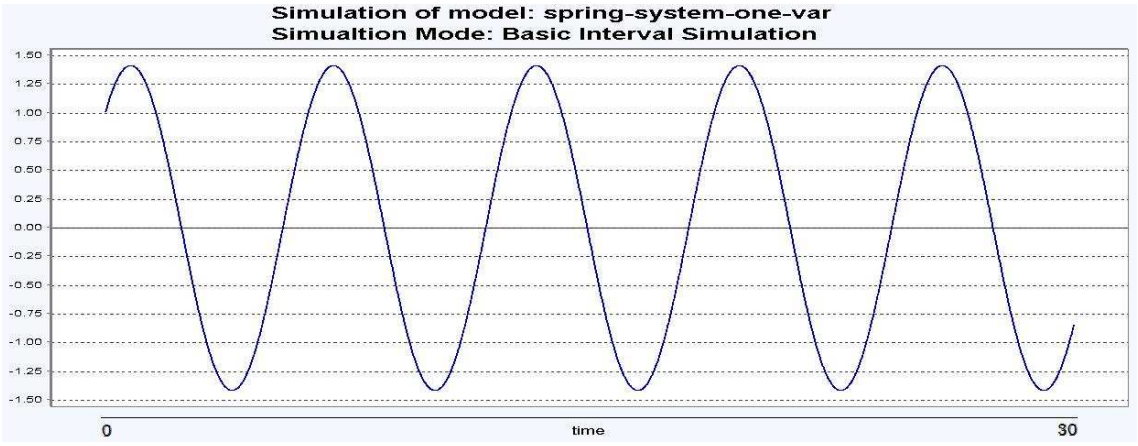


Figure 8.26: Quantitative Simulation of the Mass on a Spring model

This simulation shows that since the initial state was specified with exact numerical information then the simulation results in a definite trajectory too. These experiments show that JMorven is capable of simulating on the spectrum from fully qualitative using the signs quantity space, through semi-quantitative adopting fuzzy numbers, and fully quantitative using exact numbers.

8.5.4 Parallel Semi-Quantitative Simulation

The JMorven parallel framework can be used to speed-up the semi-quantitative simulation too. To test the parallelisations, the Van der Pol oscillator was used with the Regular Spaced Point simulation method with ten points per interval. The results of the times taken are summarized in table 8.5

	number of execution units, n							
	1	2	3	4	5	6	7	8
Time Taken (s)	88.155	50.177	39.625	30.012	26.388	22.716	18.288	17.210
Speed-up, S_n	1.000	1.757	2.225	2.937	3.341	3.881	4.820	5.122
Efficiency, E	1.000	0.879	0.742	0.734	0.668	0.647	0.689	0.640

Table 8.5: Parallelisation Benefits of JMorven Semi-Quantitative Simulation

It is clear to see that the parallelisations benefit the time taken for semi-quantitative simulation to execute. Running the same simulation on eight processors results in a speed-up of over five times which is a great advantage.

Chapter 9

Discussion, Conclusions and Future Directions of the JMorven Framework

9.1 Discussion

JMorven offers two major improvements to existing qualitative reasoning implementations; speeding up the execution by implementing parallelisations into the core algorithms, and offering an advanced mechanism for simulating from fully qualitatively to fully quantitatively in a non-constructive manner. JMorven is the only current qualitative reasoner that benefits from parallelisations in every stage. It is also the only system that can simulate on the spectrum from fully qualitative to fully quantitative. It is also worth mentioning that JMorven is also the only semi-quantitative system that uses automatic n-th order methods for integration depending on the amount of derivative information provided, and it is the only system that uses auxiliary variables in a non-constructive environment. Each of these novelties are discussed below.

JMorven was completely written from scratch and with parallelisations in mind. These parallelisations were implemented for every major stage of execution and offer substantial

gains over the sequential running time. In qualitative mode, the runtime is dominated by two main stages; the state generator and the transition generator. As such, most effort was concentrated on these areas to optimise the algorithms to achieve the greatest speed-up from parallelisations. The state generator offers the greatest benefits displaying a speed-up in excess of six times over the sequential running time when running on an eight processor machine. The algorithm suffers slightly during the early iterations as there may not be enough tuples to fully utilise all of the available execution units. This is only apparent for the first few iterations of the state generator which take a small amount of the total time of state generation. This is one reason why the speed-up is not linear. Another reason may be due to the operating system used on the test machine. It is believed that a more up-to-date version of the Solaris operating system would allow greater benefits to be observed due to a better threading model. The transition generation displays a positive speed-up too. However it is not quite as great as the benefits in the state generator. A speed-up of over five times is observed when running on eight processors which is still an excellent performance gain. The transition generation should theoretically allow very close to linear speed-up however it is thought that the implementation of some of the core methods in JMorven do not perform well in parallel which affects the performance. These methods are used heavily in the tuple filter and pairwise filter stages which is why they do not offer a major performance gain. Since these filters take a very small percentage of time to execute efforts were not concentrated on optimising them. If time allowed, optimising these methods might result in better benefits for the filters and should also improve the transition generation too. Overall, the qualitative stages have been shown to benefit greatly from parallelisations therefore I suggest this strategy is valid for speeding up the implementation of a qualitative reasoner.

The semi-quantitative simulation mode in JMorven allows models to be simulated without precise information yet still produces very useful trajectories for the variables over time. There are two main methods used to simulate semi-quantitatively; one which uses real intervals and interval arithmetic and another which approximates intervals as a set

of points. These two methods have similar sub-strategies which include extreme method, regular-spaced methods and Monte-Carlo methods. Generally the extreme methods (Basic Interval and Extreme Points) offer a very quick output to be generated but they are not accurate. The Basic Interval method suffers from excess widening of any intervals due to the nature of interval arithmetic and the Extreme Points method does not bound all solutions however it does not suffer from the intervals widening. The Basic Interval is therefore complete in that it bounds all of the real solutions to the problem however it is unsound as it also contains spurious behaviours in the form of interval divergence. The Extreme Points method on the other hand is incomplete as it does not bound all solutions however it is sound as it does not contain any spurious behaviours. This is true in general of the two main techniques. All of the methods using real intervals are complete but not sound and all of the methods that approximate intervals as groups of points are sound but incomplete. The regular-spaced and Monte-Carlo point methods improve the solutions and offer sound and complete results in the limiting cases when an infinite number of points are used or when the interval is split into an infinite number of sub-intervals. In reality, it is possible to generate a finite set of points or intervals with zero width since numbers in computers have a finite representation. In this case, the results would be sound and complete to the precision of the floating point number representation. The Regular-Spaced Interval method allows the model to be simulated for longer before the intervals diverge, and the Regular-Spaced Point approach bounds more of the real solution as the number of points increases. These methods achieve greater performance when more regular spaces are used, however this can take a very long time to execute. To overcome this, Monte-Carlo methods are also implemented which give a good trade-off between achieving close to the real solution whilst executing in a reasonable length of time. The best approach appears to be the Monte-Carlo Point method which produces a solution very close to the real one and in good time. The disadvantage of it is that it uses random numbers however with a sufficient number of iterations it is impossible to tell the difference between two different Monte-Carlo simulations.

To make the integration phase more accurate, several techniques were researched and it was decided that, for non-constructive methods, n-th order Taylor Series was the best choice. This uses all of the derivative information for each variable to estimate the successor values after a time-step δt. It was shown that when using second order methods over first order methods, the accuracy of the integration increased dramatically. This allows the modeller the design choice of adding extra derivatives to the model thus allowing more accurate simulations to be carried out.

To allow accurate simulation in a semi-quantitative manner, JMorven adopts the use of auxiliary variables. These variables are not mapped to any quantity space and thus do not unnecessarily diverge any intervals when using them. These have previously only been implemented in a constructive environment where there are strict rules about the ordering of constraints. JMorven implements these and gets around the ordering constraints by using Inverse Constraint Operations and looping round the constraints until no further narrowing of the intervals can take place. It was found that if looping through the constraints was necessary then only a small amount was required therefore the runtime is not severely affected by this technique.

9.2 Conclusions

One aim of developing a new qualitative reasoner was to improve the runtime performance by incorporating parallelisations throughout all of the main stages. Existing methods attempted this, however they had a few drawbacks in that not all stages were parallelised and that they were designed for very specific hardware setups. This motivated the development of an abstract parallel architecture which would be portable and allow it to run on a wide variety of systems including single processor workstations, multi-processor servers or in distributed network environments. After an analysis of the runtime of each

phase in the qualitative reasoner, efforts were concentrated on the two stages which occupied over 95% of the running time, although all stages have been parallelised. The results show that the parallelisations offer a vast benefit over the sequential running time.

Auxiliary variables were implemented in the non-constructive algorithms of JMorven. The difficulty with this is due to the non-causally ordered property of non-constructive methods; an auxiliary variable's value may be required to be used before being set. To get around this JMorven loops around the constraints containing the auxiliary variable updating the values until no more changes occur. JMorven also uses *Inverse Constraint Operations* which narrows the ranges of values more quickly thus there are fewer iterations of the constraints required to calculate the precise values of auxiliary variables. The output from JMorven was verified against the output of Morven showing that the use of auxiliary variables in a non-constructive algorithm is possible.

An aim of JMorven was to allow it to reason on the spectrum from fully qualitative to fully quantitative while maintaining a non-constructive algorithm. The non-constructive method is more general in that it does not require causally ordered models and would allow models with algebraic loops to be simulated. This was not a trivial problem to solve since interval arithmetic produces interval divergence. JMorven uses n-th order Taylor Series for integration which allows the integration phase to be more accurate when there is more derivative information available. There are several approaches to the simulation problem presented in JMorven, each offering some advantages and disadvantages. From the results it can be seen that it is possible to simulate fuzzy intervals in a non-constructive manner. JMorven does this by approximating intervals as a group of points. This can take a long time to simulate for naïve approaches therefore a method based on Monte-Carlo techniques was implemented. This produces excellent results in a reasonable amount of time and is the method recommended for use.

Non-constructive methods allow reasoning with more general models that do not need to

be causally ordered. This also allows models with algebraic loops to be dealt with. Since JMorven uses non-constructive methods, the input models do not need to be causally ordered. One of the optimisations for speed in JMorven re-orders the constraints depending on the number of valid tuples for each constraint. This means that internally the models are unlikely to be causally ordered; however, JMorven still continues to reason and provide the expected results. This shows that qualitative reasoning and quantitative simulation can be successfully carried out using non-constructive algorithms.

Overall JMorven has successfully met the original aims and provides a framework for use in a large variety of systems requiring simulation with ambiguity or imprecision.

9.3 Future Work

Throughout the design and implementation of JMorven several features have been thought of and hypothesised. These include:

- **Qualitative Parallel Optimisations:** The tuple and pairwise filters do not exhibit many benefits from the parallelisations which is thought to be due to some of the core methods in JMorven. Optimising these might allow them to show a similar speed-up to the state and transition generators. Since these stages take a small percentage of time to execute compared with the State Generator and Transition Analysis, and time for the project did not allow, this has not been attempted.

- **Speed-up of complete execution:** It would be benificial to find out the speedup of total execution time of running JMorven. Tests were only conducted for each stage individually but seeing how the total execution time is affected by parallelisations would be a benefit in showing their advantage.

- **Semi-Quantitative Parallel Optimisations:** Semi-quantitative simulations could be optimised by utilising a local repository of the intervals in each execution unit instead of a global one. This would allow less mutexes and less communication between execution units, therefore should display more benefits from the parallelisations. This would be of greatest benefit when implemented in a distributed network environment.

- **Iterative Simulation:** A strategy that could be used would be to create an iterative method for semi-quantitative simulation in JMorven which would carry out the simulations using regular-spaced methods but would iteratively simulate with more points. This could be used to show a graph generated in real-time and as more iterations complete, the graph would be updated. For example, the Regular-Spaced Interval method could start with two sub-intervals which would give a very approximate output but then it could carry on to use ten sub-intervals and refine the output once this had completed. This would offer a method that did not require estimating how many sub-intervals to use as too few might not give a precise enough output, however too many would take too long to execute. This method could be stopped at any time when the output was sufficiently precise as decided by the user.

- **Web Service:** Deploying JMorven as a web-service would allow it to be used over the internet and would be executed on large multiple-node networks. Since JMorven has an abstract parallel architecture, it would be able to make best use of the available resources and would provide an efficient solution to simulation to a wide user base.

- **Scheduling Algorithm** To run efficiently as a web service, or in an environment where the processor power is not the same in every execution unit, some sort of scheduling algorithm should be implemented.

- **Genetic Algorithms:** It is thought that genetic algorithms could be used along with the Monte-Carlo methods to produce faster, more accurate output trajectories for simulating fuzzy intervals.

- **Step-size refinement** At the moment JMorven uses a fixed step size for numerical simulations. Using step-size refinement is a proven method to improve the accuracy and/or the execution time of simulations. Adopting this technique would be a useful addition to JMorven.

- **Qualitative/Quantitative Refinement** It is thought that the results of the quantitative simulations could benefit the qualitative simulations by removing some or all of the spurious behaviours. It may also be possible that qualitative simulation could help some of the interval methods of quantitative simulation by avoiding unnecessary interval divergence.

- n**-tuple Fuzzy Numbers** JMorven uses *Twin Interval Fuzzy Numbers* as a representation for fuzzy numbers. This allows fuzzy numbers to be reasoned with using standard interval arithmetic. This could be extended to have fuzzy numbers with many intervals which would allow more precise fuzzy numbers to be simulated.

There have already been a number of additions to JMorven implemented by third parties including:

- A graphical model builder
- A graphical qualitative behaviour viewer
- A natural language generator for output behaviours
- The inclusion of constraints using complex numbers

This shows that JMorven is an extensible framework and these additions should make JMorven become a more powerful tool.

References

Alefeld, G. and Herzberger, J. (1983). *Introduction to Interval Computation*. Academic Press.

Bartlett, T. H. (2005). *Non-Constructive Synchronous Fuzzy Simulation and Envisionment*. PhD thesis, University of Wales, Aberystwyth, Ceredigion, Cymru, UK.

Berleant, D. and Kuipers, B. J. (1990). Combined qualitative and numeric simulation with q3. In *Proceedings of the Fourth International Workshop on Qualitative Reasoning about Physical Systems*, pages 140–152, Lugano, Switzerland.

Berleant, D. and Kuipers, B. J. (1997). Qualitative and quantitative simulation: Bridging the gap. *Artificial Intelligence*, 95(2):215–256.

Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300.

Bonarini, A. and Bontempi, G. (1994a). A qualitative simulation approach for fuzzy dynamical models. *ACM Transactions on Modelling and Computer Simulation*, 4(4):285–313.

Bonarini, A. and Bontempi, G. (1994b). A qualitative simulation approach for fuzzy models. In Guasch, A. and Huber, R. M., editors, *Modeling and Simulation ESM 94 (Proceedings of the European Simulation Multiconference 1994)*, pages 420–424, Ghent, Belgium. SCS International.

Bonarini, A. and Bontempi, G. (1994c). Qualitative simulation of approximate models: An approach based on fuzzy sets. In Trappl, R., editor, *Cybernetics and Systems Research '94 (Proceedings of the 12th European Meeting of Cybernetics and Systems Research)*, pages 359–366, Singapore. World Scientific Publishing.

Bontempi, G. (1996). Quasi iii: A software tool for the simulation of fuzzy dynamical system. In Javor, A., Lehmann, A., and Molnar, I., editors, *Modeling and Simulation ESM 96 (Proceedings European Simulation Multiconference 1996)*, pages 615–619, Ghent, Belgium. SCS International.

Bousson, K. and Travé-Massuyès, L. (1994). Putting more numbers in the qualitative simulator ca-en. In *Proceedings of the 2nd International Conference on Intelligent Systems Engineering*, pages 62–69, Hamburg, Germany.

Bredeweg, B. and Struss, P. (2004). Current topics in qualitative reasoning. *AI Magazine*, 24(4):13–16.

Burg, B. (1990). Parallel forward checking: First part. Technical Report TR-594, Institute for New Generation Computer Technology.

Cellier, F. (1991). *Continuous System Modelling*. Springer-Verlag.

Clancy, D. J. and Kuipers, B. J. (1998). Qualitative simulation as a temporally-extended constraint satisfaction problem. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pages 240–247, Madison, Wisconsin. American Association for Artificial Intelligence.

Coghill, G. M. (1992). Vector envisionment of compartmental systems. Master's thesis, University of Glasgow.

Coghill, G. M. (1996). *Mycroft: A Framework for Constraint-based Fuzzy Qualitative Reasoning*. PhD thesis, Heriot-Watt University, Edinburgh.

Coghill, G. M. (2000). A feasability study on model-based diagnosis of an ammonia washer system. Technical report, CORUS(UK).

Coghill, G. M., Bruce, A. M., Wisley, C., and Liu, H. (2005). Integrating fuzzy qualitative trigonometry with fuzzy qualitative envisionment. In *Proceedings of the 5th annual UK Workshop on Computational Intelligence, UKCI-05*, pages 97–104, London, UK.

Coghill, G. M. and Chantler, M. J. (1999). Constructive and non-constructive asynchronous qualitative simulation. In *Proceedings of the 13th Inernational Workshop on Qualitative Reasoning, QR-99*, pages 51–61, Loch Awe, Scotland.

Coghill, G. M., Garrett, S. M., and King, R. D. (2004). Learning qualitative metabolic

models. In *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI-2004*, pages 445–449, Valncia, Spain.

Crawford, J., Farquhar, A., and Kuipers, B. (1990). Qpc: A compiler from physics models into qualitative differential equations. In *Proceedings of the 8th National Conference on Artificial Intelligence, AAAI-90*, pages 365–372, San Mateo, CA.

Davis, E. (1987). Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331.

Davis, E. (1990). Order of magnitude reasoning in qualitative differential equations. In Weld, D. S. and de Kleer, J., editors, *Reasings in Qualitative Reasoning about Physical Systems*, pages 422–434.

de Jong, H. (2003). Qualitative simulation and related approaches for the analysis of dynamical systems. Technical Report 5128, INRIA (Institut National de Recherche en Informatique et en Automatique).

de Jong, H., Geiselmann, J., Hernandez, C., and Page, M. (2003). Genetic network analyzer: Qualitative simulation of genetic regulatory networks. *Bioinformatics*, 19(3):336–344.

de Kleer, J. (1977). Multiple representations of knowledge in a mechanics problem solver. In *IJCAI-77*, pages 299–304.

de Kleer, J. (1979). Causal and teleological reasoning in circuit recognition. Technical Report 529, MIT AI Lab., Cambridge MA.

de Kleer, J. and Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83.

de Kleer, J. and Williams, B. (1987). Diagnosing multiple faults. *Artificial Intelligence*, 32(1):91–130.

de Koning, K., Bredeweg, B., Breuker, J., and Wielinga, B. (2000). Model-based reasoning about learner behaviour. *Artificial Intelligence*, 117:173–229.

Drabble, B. (1993). Excalibur: A program for planning and reasoning with processes. *Artificial Intelligence*, 62(1):1–40.

Forbus, K. D. (1980). Spatial and qualitative aspects of reasoning about motion. In *AAAI-80*, Stanford, CA.

Forbus, K. D. (1981). Qualitative reasoning about physical processes. In *IJCAI-81*, Vancouver, BC.

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

Forbus, K. D. (1990). The qualitative process engine. In Weld, D. S. and de Kleer, J., editors, *Readings in qualitative reasoning about physical systems*, pages 220–235. Morgan Kaufmann Publishers Inc., San Francisco, CA.

Fouche, P. and Kuipers, B. J. (1991). Abstracting irrelevant distinctions in qualitative simulation. In *Proceedings of the Fifth International Workshop on Qualitative Reasoning about Physical Systems*, pages 232–244, Austin, Texas.

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

Froese, C. (1961). An evaluation of runge-kutta type methods for higher order differential equations. *Journal of the ACM*, 8(4):637–644.

Greenlaw, R., Hoover, H. J., and Ruzzo, W. L. (1995). *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, Oxford, UK.

Hayes, P. J. (1979). The naive physics manifesto. In Michie, D., editor, *Expert Systems in the Microelectronics Age*, pages 242–270. Edinburgh University Press.

Hayes, P. J. (1985). The second naive physics manifesto. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, pages 1–36. Morgan Kaufmann Publishers Inc.

Hofbaur, M. and Dourdoumas, N. (2001). Lyapunov based reasoning methods. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, 31(6):546–558.

Hossain, A. and Ray, K. S. (1997). An extension of qsim with qualitative curvature. *Artificial Intelligence*, 96(2):303–350.

Hunt, J. E., Price, C. J., and Lee, M. H. (1993). Automating the fmea process. *Intelligent Systems Engineering Journal*, 2(2):119–132.

Hwang, K. (1993). *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill.

Iwasaki, Y. and Simon, H. (1986). Theories of causal ordering: A reply to de kleer and brown. *Artificial Intelligence*, 29:63–68.

Kahaner, D., Moler, C., and Nash, S. (1989). *Numerical Methods and Software*. Prentice-Hall, Engelwood Cliffs, NJ.

Kaul, N., Biswas, G., and Bhuva, B. (1992). Multi-level qualitative reasoning applied to cmos digital circuits. *International Journal of Artificial Intelligence in Engineering*, 7:125–137.

Kay, H. (1998). Sqsim: A simulator for imprecise ode models. *Computers and Chemical Engineering*, 23:27–46.

Kay, H. and Kuipers, B. J. (1992). Numerical simulation envelopes for qualitative models. In *Proceedings of the Sixth International Workshop on Qualitative Reasoning about Physical Systems*, pages 24–27, Edinburgh, Scotland.

Kay, H. and Kuipers, B. J. (1993). Numerical behaviour envelopes fo qualitative models. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 606–613, Cambridge, MA.

Kay, H., Rinner, B., and Kuipers, B. J. (2000). Semi-quantitative system identification. *Artificial Intelligence*, 119:103–140.

Keller, U., Wyatt, T. K., and Leitch, R. R. (1999). Frensi - a fuzzy qualitative simulator. In *Proceedings of Workshop on Applications of Interval Analysis to Systems and Control (with special emphasis on recent advances in Modal Interval Analysis)*, pages 305–313, Girona, Spain.

Keller, U. E. (1999). *Qualitative Model Reference Adaptive Control*. PhD thesis, Heriott-Watt University.

King, R. D., Garrett, S. M., and Coghill, G. M. (2005). On the use of qualitative reasoning to simulate and identify metabolic pathways. *Bioinformatics*, 21(9):2017–2026.

Kuipers, B. J. (1984). Commonsense reasoning about causality: Deriving behaviour from structure. *Artificial Intelligence*, 24:169–203.

Kuipers, B. J. (1986). Qualitative simulation. *Artificial Intelligence*, 29(3):289–338.

Kuipers, B. J. and Berleant, D. (1988). Using incomplete quantitative knowledge in qualitative reasoning. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-88*, pages 324–329, St. Paul, Minn.

Kuipers, B. J. and Chui, C. (1987). Taming intractable branching in qualitative reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-87*, pages 23–28, Milan, Italy.

Lee, M. H. (1999a). Qualitative circuit models in failure analysis reasoning. *Artificial Intelligence*, 111:239–276.

Lee, M. H. (1999b). Qualitative modelling of linear networks in ecad applications. In *Proceedings of the 13th Inernational Workshop on Qualitative Reasoning, QR-99*, pages 146–152, Loch Awe, Scotland.

Lee, M. H. (2000). Many-valued logic and qualitative modelling of electrical circuits. In *Proceedings of the 14th International Workshop on Qualitative Reasoning*, pages 89–96, Michoacan, Mexico.

Lee, M. H. and Ormsby, A. R. T. (1992). Qualitative modelling of electrical circuits. In *Proceedings of the 6th International Workshop on Qualitative Reasoning, QR-2000*, pages 155–169, Edinburgh, Scotland.

Lee, W. W. and Kuipers, B. J. (1988). Non-intersection trajectories in qualitative phase space: A global constraint for qualitative simulation. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-88*, pages 286–291, St. Paul, Minn.

Lee, W. W. and Kuipers, B. J. (1993). A qualitative method to construct phase portraits. In *Proceedings of the Seventh International Workshop on Qualitative Reasoning about Physical Systems*, pages 132–137, Orcas Island, Washington.

Leighton, F. T. (1992). *Introduction to Parallel Algorithms and Architectures*, volume 2. Morgan Kaufmann, San Mateo, CA.

Lin, W. and Yang, B. (1995a). Fast parallel tree search with static load-balancing forward checking technique. In *Proceedings of the ISCA International Conference on Parallel and Distributed Computing Systems*, pages 33–38, Orlando. FL.

Lin, W. and Yang, B. (1995b). Probabilistic performance analysis for parallel search techniques. *International Journal of Parallel Programming*, 23(2):161–189.

Liu, H. and Coghill, G. M. (2005a). Can we do trigonometry qualitatively? In *Proceedings of the 19th International Workshop on Qualitative Reasoning QR05*, pages 199–205, Graz, Austria.

Liu, H. and Coghill, G. M. (2005b). A model-based approach to robot fault diagnosis. *International Journal of Knowledge Based Systems*, 18:225–233.

Lohner, R. J. (1987). Enclosing the solutions of ordinary initial and boundary value problems. In Kauchner, E. W., Kulisch, U., and Ullrich, C., editors, *Computer Arithmetic: Scientific Computation and Programming Languages*, pages 255–286. Wiley, Teubner, Stuttgart.

Lulis, E., Michael, A. J., and Evens, M. W. (2004). Using qualitative reasoning in the classroom and in electronic teaching systems. In *Proceedings of the 18th International Workshop on Qualitative Reasoning, QR-2004*, pages 121–127, Evanston, IL.

Luo, Q. P., Hendry, P. G., and Buchanan, J. T. (1994). Strategies for distributed constraint satisfaction problems. In *Proceedings of the 13th International DAI Workshop*, pages 207–221, Seattle, WA.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8:99–118.

Mavrovouniotis, M. and Stephanopoulos, G. (1988). Formal order-of-magnitude reasoning in process engineering. *Computing and Chemical Engineering*, 12(9/10):867–881.

Milne, R. (1991). Second generation expert systems: The application gap. In *Proceedings of the 11th International Conference on Expert Systems and Their Applications(General Conference on 2nd Generation Expert Systems)*, pages 259–264, Nanterre, France.

Moore, R. E. (1966). *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NY.

Moore, R. E. (1979). *Methods and Applications of Interval Analysis*. Studies in Applied Mathematics, Philadelphia.

Morgan, A. (1988). *Qualitative Behaviour of Dynamic Physical Systems.* PhD thesis,

University of Cambridge.

Mosterman, P. J., Manders, E. J., and Biswas, G. (2000). Qualitative dynamic behaviour of physical system models with algebraic loops. In *Proceedings of the Eleventh International Workshop on Principles of Diagnosis (DX'00)*, pages 155–162, Mexico.

Ng, H. T. (1991). Model-based, multiple fault diagnosis of time-varying, continuous physical devices. *IEEE Expert*, 6(6):38–43.

Nguyen, H. T. (1978). A note on the extension principle for fuzzy sets. *Journal of Mathematical Analysis and Applications*, 64(2):369–380.

Platzner, M. (1996). *Design, Implementation, and Experimental Evaluation of Coprocessor Architectures for Fast Qualitative Simulation*. PhD thesis, Technical Univesrity of Graz, Graz, Austria.

Platzner, M. and Rinner, B. (1995). Improving performance of the qualitative simulator qsim – design and implementation of a specialized computer architecture. In *Proceedings of the ISCA International Conference on Parallel and Distributed Computing Systems*, pages 494–501, Orlando, FL.

Platzner, M. and Rinner, B. (1998). Design and implementation of a parallel constraint satisfaction algorithm. *International Journal in Computers and Their Applications*, 5(2):106–116.

Platzner, M. and Rinner, B. (2000). Toward embedded qualitative simulation. *IEEE Intelligent Systems*, 15(2):62–68.

Platzner, M., Rinner, B., and Weiss, R. (1995). Exploiting parallelism in constraint satisfaction for qualitative simulation. *The Journal of Universal Computer Science*, 1(12):811–820.

Platzner, M., Rinner, B., and Weiss, R. (1997). Parallel qualitative simulation. *Simulation Practice and Theory - International Journal of the Federation of European Simulation Societies*, 5(7-8):623–638.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1992). *Numerical Recipes in C: The Art of Scientific Computing*, chapter 16, pages 710–722. Cambridge University Press, 2nd edition edition.

Price, C. J. (2000). Autosteve: Automated electrical design analysis. In *ECAI-2000: Prestigious Applications of Artificial Intelligence*, pages 721–725, Berlin.

Price, C. J., Pugh, D. R., Wilson, M. S., and Snooke, N. (1995). The flame system: Automating electrical failure mode and effects analysis (fmea). *Reliability and Maintainability Symposium*, pages 90–95.

Price, C. J., Travé-Massuyès, L., Milne, R., Ironi, L., Bredeweg, B., Lee, M. H., Struss, P., Snooke, N., Lucas, P., and Cavazza, M. (2005). Qualitative futures. In *Proceedings of the 19th International Workshop on Qualitative Reasoning QR05*, pages 29–37, Graz, Austria.

Pugh, D. and Snooke, N. (1996). Dynamic analysis of qualitative circuits for failure mode and effects analysis. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 37–42.

Raiman, O. (1991). Order of magnitude reasoning. *Artificial Intelligence*, 51:11–38.

Reif, J. (1985). Depth-first search is inherently sequential. *Information Processing Letters*, 20:229–234.

Rinner, B. (1996). *Design, Prototype Implementation and Experimental Evaluation of a Scalable Multiprocessor Architecture for Qualitative Simulation*. PhD thesis, Technical University of Graz, Graz, Austria.

Rubinstein, R. Y. (1981). *Simulaton and the Monte-Carlo Method*. John Wiley ans Sons, Inc., New York.

Salles, P. and Bredeweg, B. (2003). Qualitative reasoning about population and community ecology. *AI Magazine*, 24(4):77–90.

Say, A. C. C. and Kuru, S. (1993). Improved filtering for the qsim algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):967–971.

Scott, J. A. and Coghill, G. M. (1998). Qualitative euler integration with continuity. In *Proceedings of the 12th International Workshop on Qualitative Reasoning*, pages 114–122, Cape Cod, MA.

Shen, Q. (1991). *Fuzzy Qualitative Simulation and Diagnosis of Continuous Dynamic Systems*. PhD thesis, Heriot-Watt University, Edinburgh, UK.

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

Shimomura, Y., Ogawa, K., Tanigawa, S., Umeda, Y., and Tomiyama, T. (1995). Development of self-maintenance photocopiers. In *The Seventh Conference on Innovative Applications of Artificial Intelligence IAAI-95*, pages 171–180, Montral, Canada.

Simmons, R. (1986). Commonsense arithmetic reasoning. In *Proceedings of the Fifth International Conference on Artificial Intelligence (AAAI-86)*, pages 118–124.

Spivak, M. (1967). *Calculus*. Benjamin, New York.

Struss, P. and Price, C. J. (2004). Model-based systems in the automotive industry. *AI Magazine*, 24(4):17–34.

Trelease, R. B. and Park, J. (2003). Qualitative reasoning in integrative biology: Evolving a network experiment modeling system for collaborative multilevel systems research. In *Proceedings of the AIME'03 Workshop: Model-based and Qualitative Reasoning in Biomedicine*, Cyprus.

Vescovi, M. R. and Travé-Massuyès, L. (1992). A constructive approach to qualitative fuzzy simulation. In *6th International Workshop on Qualitative Reasoning about Physical Systems (QR'92)*, pages 268–280, Edinburgh.

Waltz, D. (1975). *Understanding Line Drawings of Scenes with Shadows*. McGraw-Hill, New York.

Wiegand, M. E. (1991). *Constructive Qualitative Simulation of Continuous Dynamic Systems*. PhD thesis, Heriot-Watt University, Edinburgh.

Wiegand, M. E. and Leitch, R. (1989). A 'predictive engine' for the qualitative simulation of dynamic systems. In *Proceedings of the fourth International Conference on the Application of Artificial Intelligence in Engineering*, Cambridge, UK.

Williams, B. C. (1984a). Qualitative analysis of mos circuits. *Artificial Intelligence*, 24:281–346.

Williams, B. C. (1984b). The use of continuity in a qualitative physics. In *Proceedings of the Fourth National Conferenceon Artificial Intelligence (AAAI-84)*, pages 350–354, Austin, Texas.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, 8:338–353.

Zadeh, L. A. (1973). Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics*, 3:28–44.

Zadeh, L. A. (1975a). The concept of a linguistic variable and its application to approximate reasoning, part 1. *Information Sciences*, 8:199–249.

Zadeh, L. A. (1975b). The concept of a linguistic variable and its application to approximate reasoning, part 2. *Information Sciences*, 8:301–357.

Zadeh, L. A. (1976). The concept of a linguistic variable and its application to approximate reasoning, part 3. *Information Sciences*, 9:43–80.

Appendix A

JMorven Models

A.1 Single Tank Model

```
model-name:  single-tank
short-name:  fst

NumSystemVariables:  2
variable:  qo range:  zero p-max NumDerivatives:  1 qspaces:  tanks-qs tanks-qs2
variable:  V range:  zero p-max NumDerivatives:  2 qspaces:  tanks-qs tanks-qs2 tanks-qs2

NumExogenousVariables:  1
variable:  qi range:  zero p-max NumDerivatives:  1 qspaces:  tanks-qs tanks-qs2


Constraints:
NumDiffPlanes:  2

Plane:  0 NumConstraints:  2
Constraint:  func (dt 0 qo) (dt 0 V) NumMappings:  9
        Mappings:
                n-max n-max
                n-large n-large
                n-medium n-medium
                n-small n-small
                zero zero
                p-small p-small
                p-medium p-medium
                p-large p-large
                p-max p-max
Constraint:  sub (dt 1 V) (dt 0 qi) (dt 0 qo)

Plane:  1 NumConstraints:  2
Constraint:  func (dt 1 qo) (dt 1 V) NumMappings:  5
        Mappings:
                nl-dash nl-dash
                ns-dash ns-dash
                zero zero
                ps-dash ps-dash
                pl-dash pl-dash
Constraint:  sub (dt 2 V) (dt 1 qi) (dt 1 qo)

NumVarsToPrint:  3 VarsToPrint:  V qi qo
```

A.2 Coupled Tanks Model

```
model-name:  coupled-tanks
short-name:  cpdt

NumSystemVariables:  5
variable:  h1 range:  zer pos NumDerivatives:  2 qspaces:  tanks-qs tanks-qs2 tanks-qs2
variable:  h2 range:  zer pos NumDerivatives:  2 qspaces:  tanks-qs tanks-qs2 tanks-qs2
variable:  h12 range:  neg pos NumDerivatives:  1 qspaces:  tanks-qs tanks-qs2
variable:  qx range:  neg pos NumDerivatives:  1 qspaces:  tanks-qs tanks-qs2
variable:  qo range:  zer pos NumDerivatives:  1 qspaces:  tanks-qs tanks-qs2

NumExogenousVariables:  1
variable:  qi range:  zer pos NumDerivatives:  1 qspaces:  tanks-qs tanks-qs2


Constraints:
NumDiffPlanes:  2

Plane:  0 NumConstraints:  5
Constraint:  func (dt 0 qo) (dt 0 h2) NumMappings:  3
        Mappings:
                neg neg
                zer zer
                pos pos
Constraint:  func (dt 0 qx) (dt 0 h12) NumMappings:  3
        Mappings:
                neg neg
                zer zer
                pos pos
Constraint:  sub (dt 0 h12) (dt 0 h1) (dt 0 h2)
Constraint:  sub (dt 1 h1) (dt 0 qi) (dt 0 qx)
Constraint:  sub (dt 1 h2) (dt 0 qx) (dt 0 qo)

Plane:  1 NumConstraints:  5
Constraint:  func (dt 1 qo) (dt 1 h2) NumMappings:  3
        Mappings:
                neg neg
                zer zer
                pos pos
Constraint:  func (dt 1 qx) (dt 1 h12) NumMappings:  3
        Mappings:
                neg neg
                zer zer
                pos pos
Constraint:  sub (dt 1 h12) (dt 1 h1) (dt 1 h2)
Constraint:  sub (dt 2 h1) (dt 1 qi) (dt 1 qx)
Constraint:  sub (dt 2 h2) (dt 1 qx) (dt 1 qo)

NumVarsToPrint:  3 VarsToPrint:  h1 h2 h12
```

A.3 Spring System Model

```
model-name:  spring-system
short-name:  spr

NumSystemVariables:  3
variable:  d range:  n-max p-max NumDerivatives:  1 qspaces:  fuzzy-qs fuzzy-qs2
variable:  x1 range:  n-max p-max NumDerivatives:  2 qspaces:  fuzzy-qs fuzzy-qs2
        fuzzy-qs2
variable:  x2 range:  nl-dash pl-dash NumDerivatives:  2 qspaces:  fuzzy-qs2 fuzzy-qs2
        fuzzy-qs2

NumExogenousVariables:  1
variable:  F range:  n-max p-max NumDerivatives:  1 qspaces:  fuzzy-qs fuzzy-qs2


Constraints:
NumDiffPlanes:  2

Plane:  0 NumConstraints:  3
Constraint:  func (dt 0 d) (dt 0 x1) NumMappings:  9
        Mappings:
                n-max n-max
                n-large n-large
                n-medium n-medium
                n-small n-small
                zero zero
                p-small p-small
                p-medium p-medium
                p-large p-large
                p-max p-max
Constraint:  func (dt 1 x1) (dt 0 x2) NumMappings:  5
        Mappings:
                nl-dash nl-dash
                ns-dash ns-dash
                zero zero
                ps-dash ps-dash
                pl-dash pl-dash
Constraint:  sub (dt 1 x2) (dt 0 F) (dt 0 d)

Plane:  1 NumConstraints:  3
Constraint:  func (dt 1 d) (dt 1 x1) NumMappings:  5
        Mappings:
                nl-dash nl-dash
                ns-dash ns-dash
                zero zero
                ps-dash ps-dash
                pl-dash pl-dash
Constraint:  func (dt 2 x1) (dt 1 x2) NumMappings:  5
        Mappings:
                nl-dash nl-dash
                ns-dash ns-dash
                zero zero
                ps-dash ps-dash
                pl-dash pl-dash
Constraint:  sub (dt 2 x2) (dt 1 F) (dt 1 d)

NumVarsToPrint:  1 VarsToPrint:  d
```

A.4 Algebraic Loop Model

```
model-name:  LRcircuit
short-name:  LR

NumSystemVariables:  1
variable:  u3 range:  zero p-max NumDerivatives:  0 qspaces:  fuzzy9

NumExogenousVariables:  4
variable:  R1 range:  R-min R-max NumDerivatives:  0 qspaces:  resistors
variable:  R3 range:  R-min R-max NumDerivatives:  0 qspaces:  resistors
variable:  U0 range:  U-min U-max NumDerivatives:  0 qspaces:  voltages
variable:  i2 range:  i-min i-max NumDerivatives:  0 qspaces:  currents

NumAuxiliaryVariables:  3
variable:  u1
variable:  i3
variable:  i1

Constraints:  NumDiffPlanes:  1
Plane:  0 NumConstraints:  4
Constraint:  mul (dt 0 u1) (dt 0 R1) (dt 0 i1)
Constraint:  div (dt 0 i3) (dt 0 u3) (dt 0 R3)
Constraint:  sub (dt 0 u3) (dt 0 U0) (dt 0 u1)
Constraint:  add (dt 0 i1) (dt 0 i2) (dt 0 i3)

NumVarsToPrint:  1 VarsToPrint:  u3
```

A.5 Van der Pol Oscillator Model

```
model-name:  VanDerPol-Oscillator
short-name:  vdpo

NumSystemVariables:  2
variable:  x1 range:  n-max p-max NumDerivatives:  1 qspaces:  fuzzy-qs fuzzy-qs
variable:  x2 range:  n-max p-max NumDerivatives:  1 qspaces:  fuzzy-qs fuzzy-qs

NumExogenousVariables:  3
variable:  one range:  one one NumDerivatives:  0 qspaces:  constants
Variable:  P range:  n-max p-max NumDerivatives:  0 qspaces:  fuzzy-qs
Variable:  Q range:  n-max p-max NumDerivatives:  0 qspaces:  fuzzy-qs

NumAuxiliaryVariables:  5
Variable:  A
Variable:  B
Variable:  C
Variable:  D
Variable:  E

Constraints:
NumDiffPlanes:  1

Plane:  0 NumConstraints:  7
Constraint:  func (dt 1 x1) (dt 0 x2) NumMappings:  9
        Mappings:
                n-max n-max
                n-large n-large
                n-medium n-medium
                n-small n-small
                zero zero
                p-small p-small
                p-medium p-medium
                p-large p-large
                p-max p-max
Constraint:  mul (dt 0 A) (dt 0 Q) (dt 0 x1)
Constraint:  mul (dt 0 B) (dt 0 x1) (dt 0 x1)
Constraint:  sub (dt 0 C) (dt 0 one) (dt 0 B)
Constraint:  mul (dt 0 D) (dt 0 P) (dt 0 x2)
Constraint:  mul (dt 0 E) (dt 0 D) (dt 0 C)
Constraint:  sub (dt 1 x2) (dt 0 E) (dt 0 A)

NumVarsToPrint:  2 VarsToPrint:  x1 x2
```

Appendix B

JMorven Diagrams

B.1 Coupled Tanks Graph

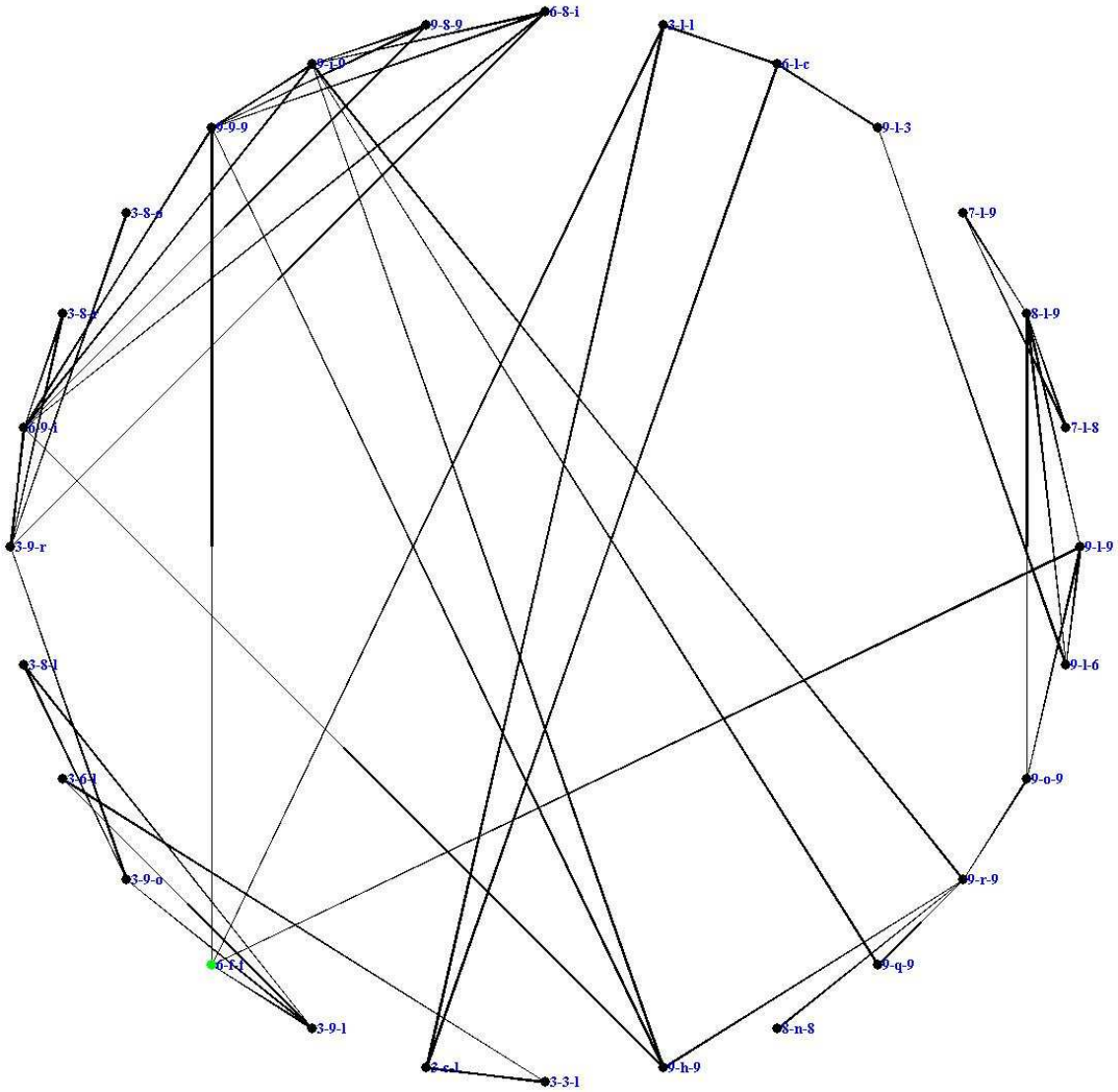


Figure B.1: Coupled Tanks Graph of Complete Envisionment

Appendix C

JMorven State Repositories

C.1 Spring System State Repository

```
======================================
= S T A T E _ R E P O S I T O R Y
======================================
--------------------
State UID: e-5-e-5
x2:{zer , zer , zer}
F:{zer , zer}
x1:{zer , zer , zer}
d:{zer , zer}

Successor States:
e-5-e5
Predecessor States:
e-5-e5

--------------------
State UID: h-8-6-5
x2:  {pos , zer , neg}
F: {zer , zer}
x1:  {zer , pos , zer}
d:  {zer , pos}

Successor States:
9-9-3-5
Predecessor States:
p-7-9-5

--------------------
State UID: b-2-m-5
x2:  {neg , zer , pos}
F: {zer , zer}
x1:  {zer , neg , zer}
d:  {zer , neg}

Successor States:
```

```
j-1-p-5
Predecessor States:
3-3-j-5

--------------------
State UID: 6-6-b-5
x2:  {zer , neg , zer}
F: {zer , zer}
x1:  {pos , zer , neg}
d:  {pos , zer}

Successor States:
3-3-j-5
Predecessor States:
9-9-3-5

--------------------
State UID: 9-9-3-5
x2:  {pos , neg , neg}
F: {zer , zer}
x1:  {pos , pos , neg}
d:  {pos , pos}

Successor States:
9-9-3-5, 6-6-b-5
Predecessor States:
9-9-3-5, h-8-6-5

--------------------
State UID: 3-3-j-5
x2:  {neg , neg , pos}
F: {zer , zer}
x1:  {pos , neg , neg}
d:  {pos , neg}

Successor States:
3-3-j-5, b-2-m-5
Predecessor States:
3-3-j-5, 6-6-b-5

--------------------
State UID: m-4-h-5
x2:  {zer , pos , zer}
F: {zer , zer}
x1:  {neg , zer , pos}
d:  {neg , zer}

Successor States:
```

```
p-7-9-5
Predecessor States:
j-1-p-5

--------------------
State UID: p-7-9-5
x2:  {pos , pos , neg}
F: {zer , zer}
x1:  {neg , pos , pos}
d:  {neg , pos}

Successor States:
h-8-6-5, p-7-9-5
Predecessor States:
p-7-9-5, m-4-h-5

--------------------
State UID: j-1-p-5
x2:  {neg , pos , pos}
F: {zer , zer}
x1:  {neg , neg , pos}
d:  {neg , neg}

Successor States:
m-4-h-5, j-1-p-5
Predecessor States:
b-2-m-5, j-1-p-5

--------------------------- End of State Repository
Number of states in repository = 9 (13 transitions)
```

C.2 Coupled Tanks State Repository

```
=====================================
= S T A T E _ R E P O S I T O R Y
=====================================
--------------------
State UID: 3-3-l
h1:  {pos , neg , pos}
h2:  {pos , neg , neg}
h12:  {pos , neg}

Successor States:
3-3-l, 3-c-l
Predecessor States:
3-c-l, 3-3-l, 3-6-l

--------------------
State UID: 3-9-l
h1:  {pos , neg , pos}
h2:  {pos , pos , neg}
h12:  {pos , neg}

Successor States:
3-9-l, 6-f-f, 3-9-o, 3-6-l
Predecessor States:
3-9-l, 3-8-l

--------------------
State UID: 3-8-l
h1:  {pos , neg , pos}
h2:  {zer , pos , neg}
h12:  {pos , neg}

Successor States:
3-9-l, 3-9-o
Predecessor States:



--------------------
State UID: 3-9-r
h1:  {pos , pos , pos}
h2:  {pos , pos , neg}
h12:  {pos , neg}

Successor States:
6-9-i, 3-9-r
Predecessor States:
6-9-i, 3-9-r, 3-9-o, 6-8-i, 3-8-o, 3-8-r
```

```
--------------------
State UID: 3-8-r
h1:  {pos , pos , pos}
h2:  {zer , pos , neg}
h12:  {pos , neg}

Successor States:
6-9-i, 3-9-r
Predecessor States:


--------------------
State UID: 3-9-o
h1:  {pos , zer , pos}
h2:  {pos , pos , neg}
h12:  {pos , neg}

Successor States:
3-9-r
Predecessor States:
3-9-l, 3-8-l

--------------------
State UID: 3-8-o
h1:  {pos , zer , pos}
h2:  {zer , pos , neg}
h12:  {pos , neg}

Successor States:
3-9-r
Predecessor States:


--------------------
State UID: 3-6-l
h1:  {pos , neg , pos}
h2:  {pos , zer , neg}
h12:  {pos , neg}

Successor States:
3-3-l
Predecessor States:
3-9-l

--------------------
State UID: 9-9-9
h1:  {pos , pos , neg}
h2:  {pos , pos , neg}
```

```
h12:  {pos , pos}

Successor States:
6-9-i, 6-f-f, 9-i-9, 9-9-9
Predecessor States:
6-9-i, 9-i-9, 9-8-9, 9-9-9, 6-8-i, 9-h-9

--------------------
State UID: 9-8-9
h1:  {pos , pos , neg}
h2:  {zer , pos , neg}
h12:  {pos , pos}

Successor States:
9-9-9, 9-i-9, 6-9-i
Predecessor States:


--------------------
State UID: 6-9-i
h1:  {pos , pos , zer}
h2:  {pos , pos , neg}
h12:  {pos , zer}

Successor States:
6-9-i, 3-9-r, 9-9-9, 9-i-9
Predecessor States:
3-9-r, 9-8-9, 6-8-i, 6-9-i, 9-9-9, 9-i-9, 9-h-9, 3-8-r

--------------------
State UID: 6-8-i
h1:  {pos , pos , zer}
h2:  {zer , pos , neg}
h12:  {pos , zer}

Successor States:
6-9-i, 3-9-r, 9-9-9, 9-i-9
Predecessor States:


--------------------
State UID: 3-l-l
h1:  {pos , neg , pos}
h2:  {pos , neg , pos}
h12:  {pos , neg}

Successor States:
3-c-l, 3-l-l, 6-f-f, 6-l-c
```

```
Predecessor States:
3-c-l, 3-l-l, 6-l-c

--------------------
State UID: 9-l-3
h1:  {pos , neg , neg}
h2:  {pos , neg , pos}
h12:  {pos , pos}

Successor States:
9-l-3, 6-l-c
Predecessor States:
6-l-c, 9-l-3, 9-l-6

--------------------
State UID: 7-l-9
h1:  {pos , pos , neg}
h2:  {pos , neg , pos}
h12:  {neg , pos}

Successor States:
7-l-9, 8-l-9
Predecessor States:
7-l-9, 7-l-8

--------------------
State UID: 7-l-8
h1:  {zer , pos , neg}
h2:  {pos , neg , pos}
h12:  {neg , pos}

Successor States:
7-l-9, 8-l-9
Predecessor States:



--------------------
State UID: 9-l-9
h1:  {pos , pos , neg}
h2:  {pos , neg , pos}
h12:  {pos , pos}

Successor States:
9-l-6, 9-o-9, 9-l-9, 6-f-f
Predecessor States:
9-l-9, 8-l-9

--------------------
```

```
State UID: 8-l-9
h1:  {pos , pos , neg}
h2:  {pos , neg , pos}
h12:  {zer , pos}

Successor States:
9-o-9, 9-l-6, 9-l-9
Predecessor States:
7-l-9, 7-l-8

--------------------
State UID: 9-l-6
h1:  {pos , zer , neg}
h2:  {pos , neg , pos}
h12:  {pos , pos}

Successor States:
9-l-3
Predecessor States:
9-l-9, 8-l-9

--------------------
State UID: 9-r-9
h1:  {pos , pos , neg}
h2:  {pos , pos , pos}
h12:  {pos , pos}

Successor States:
9-r-9, 9-i-9
Predecessor States:
9-r-9, 9-o-9, 8-n-8, 9-i-9, 9-h-9, 9-q-9

--------------------
State UID: 9-q-9
h1:  {pos , pos , neg}
h2:  {zer , pos , pos}
h12:  {pos , pos}

Successor States:
9-r-9, 9-i-9
Predecessor States:


--------------------
State UID: 9-o-9
h1:  {pos , pos , neg}
h2:  {pos , zer , pos}
h12:  {pos , pos}
```

```
Successor States:
9-r-9
Predecessor States:
9-l-9, 8-l-9

--------------------
State UID: 8-n-8
h1:  {zer , pos , neg}
h2:  {zer , zer , pos}
h12:  {zer , pos}

Successor States:
9-r-9
Predecessor States:


--------------------
State UID: 6-l-c
h1:  {pos , neg , zer}
h2:  {pos , neg , pos}
h12:  {pos , zer}

Successor States:
9-l-3, 3-l-l, 3-c-l, 6-l-c
Predecessor States:
9-l-3, 3-l-l, 3-c-l, 6-l-c

--------------------
State UID: 3-c-l
h1:  {pos , neg , pos}
h2:  {pos , neg , zer}
h12:  {pos , neg}

Successor States:
3-l-l, 3-c-l, 3-3-l, 6-l-c
Predecessor States:
3-l-l, 3-3-l, 3-c-l, 6-l-c

--------------------
State UID: 9-i-9
h1:  {pos , pos , neg}
h2:  {pos , pos , zer}
h12:  {pos , pos}

Successor States:
9-r-9, 6-9-i, 9-i-9, 9-9-9
Predecessor States:
```

```
9-r-9, 6-8-i, 9-8-9, 6-9-i, 9-9-9, 9-i-9, 9-h-9, 9-q-9

--------------------
State UID: 9-h-9
h1:  {pos , pos , neg}
h2:  {zer , pos , zer}
h12:  {pos , pos}

Successor States:
9-r-9, 6-9-i, 9-i-9, 9-9-9
Predecessor States:



--------------------
State UID: 6-f-f
h1:  {pos , zer , zer}
h2:  {pos , zer , zer}
h12:  {pos , zer}

Successor States:
6-f-f
Predecessor States:
3-9-l, 3-l-l, 9-l-9, 6-f-f, 9-9-9

--------------------------- End of State Repository
Number of states in repository = 28 (71 transitions)
```

Appendix D

JMorven Code

D.1 JMorvenThread

The code below shows the implementation of a thread in JMorven, termed a JMorven-Thread. This provides a mechanism for keeping track of the ID of each thread, the number of threads queued and running as well as utility methods which allow the current thread to wait until either a thread is complete, all threads are complete or one queued thread can be executed.

```
/*
* JMorvenThread.java
*
* Created on 09 May 2005, 16:01
*/

package JMorven.Utilities;

import java.util.*;

/**
* This is a Thread wrapper which displays debug info for timings/IDs if requested.
* This class should also carry out some housekeeping to have control over the
* number of Threads running at once and also provide a mechanism to wait for
* Threads to become available or finish .
* @author Allan M. Bruce
*/
public class JMorvenThread extends Thread
{
    /**
    * The number of threads currently running and queued
    */
    private static int mNumQueued = 0, mNumRunning = 0;

    /**
    * The maximum number of threads to spawn at one time
    */
    private static int mMaxThreads;

    /**
    * Flag to determine whether to show timing/IDs of usage
    */
    private static boolean mVerbose;

    /**
    * A counter for the ID of the next spawned thread
    */
    private static long mIDs = 0;

    /**
    * Lock for synchronizing access to shared variables
    */
    private static Object mStaticLock = new Object();
```

```
/**
* The ID of the instantiated thread
*/
private long mID;

/**
* Initialise the parameters for the threads - call at start of day before
* using the Class
* @param xiMaxThreads The maximum number of threads to spawn at one time
* @param xiVerbose Flag to determine whether to show timing/IDs of usage
*/
public static void init(int xiMaxThreads, boolean xiVerbose)
{
    mVerbose = xiVerbose;
    mMaxThreads = xiMaxThreads;
}

/**
* The run method - displays extra info if in debug mode
*/
public void run()
{
    synchronized(mStaticLock)
    {
        mNumRunning++;
    }

    // if verbose, take note of the time and print a start message
    long lStartTime = 0;
    if (mVerbose)
    {
        System.out.println("Thread " + mID + " started");
        System.out.flush();
        lStartTime = System.currentTimeMillis();
    }

    // do the work!
    super.run();

    // if verbose, find out the time taken and print a message
    if (mVerbose)
    {
        long lEndTime = System.currentTimeMillis();
        long lTimeTaken = lEndTime - lStartTime;
        System.out.println("Thread " + mID + " finished, time taken = "
                + lTimeTaken/1000.0 + "s");
        System.out.flush();
    }

        // decrement the counter and notify all threads
    synchronized(mStaticLock)
    {
        mNumRunning--;
        mNumQueued--;
        mStaticLock.notifyAll();
    }
}

/**
* This waits for threads to start and finish their work.
*/
public static void waitForThreadsToFinish()
{
    synchronized(mStaticLock)
    {
        // wait until we have started if we haven't already
        while (mNumQueued != 0)
        {
            try
            {
                mStaticLock.wait();
            }
            catch (Exception exc)
```

```
                {
                }
            }
        }
    }

    /**
    * This waits for a thread to finish
    * @return the number of threads still running
    */
    public static int waitForAThreadToFinish()
    {
        synchronized(mStaticLock)
        {
            if (mNumQueued == 0)
                return 0;

            try
            {
                mStaticLock.wait();
            }
            catch (Exception exc)
            {
            }
            return mNumRunning;
        }
    }

    /**
    * This waits until a Thread becomes available - i.e.  until the counter drops
    * below the number of threads specified in init()
    */
    public static void waitUntilThreadBecomesAvailable()
    {
        synchronized(mStaticLock)
        {
            while (mNumRunning > mMaxThreads)
            {
                try
                {
                    mStaticLock.wait();
                }
                catch (Exception e)
                {
                }
            }
        }
    }

    /**
    * Creates a new instance of JMorvenThread
    */
    public JMorvenThread(Runnable xiTarget)
    {
        super(xiTarget);

        synchronized(mStaticLock)
        {
            mID = mIDs++;
            mNumQueued++;
        }

        if (mVerbose)
        {
            System.out.println("Thread " + mID + " created");
            System.out.flush();
        }
    }

}
```

Appendix E

Publications

Below are a list of papers which have been peer reviewed and accepted for publication:

- A. M. Bruce and G. M. Coghill,"Implementing Parallelisations in a Qualitative Reasoning Engine", *Proceedings of the 5th International Conference on Recent Advances in Soft Computing, RASC2004*, pp 390-396, Nottingham, UK, 2004

  (Received 'Best Student Presentation Award')

- A. M. Bruce and G. M. Coghill, "Parallel Fuzzy Qualitative Reasoning", *Proceedings of the 19th International Workshop on Qualitative Reasoning, QR2005*, pp 110-116, Graz, Austria, 2005

- A. M. Bruce and G. M. Coghill, "Implementing Parallelisations in a Fuzzy Qualitative Reasoning Engine", *Proceedings of the 5th annual UK Workshop on Computational Intelligence, UKCI2005*, pp 36-43, London, UK, 2005

- G. M. Coghill, A. M. Bruce, C. Wisley and H. Liu, "Integrating Fuzzy Qualitative Trigonometry with Fuzzy Qualitative Envisionment", *Proceedings of the 5th annual UK Workshop on Computational Intelligence, UKCI2005*, pp 97-104, London, UK, 2005