Advancing the Use of Sparse Knowledge for Qualitative Models and Simulations

MSc Thesis *(Afstudeerscriptie)*

written by

Stefania Ionescu
(born December 27th, 1993 in Bucharest, Romania)

under the supervision of **dr. Bert Bredeweg** and **dr. Jaap Kamps**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
July 20th, 2018	prof. dr. Ronald de Wolf (chair)
	dr. Kaspar Beelen
	dr. Anders Bouwer
	dr. Bert Bredeweg
	dr. Jaap Kamps
	dr. Marijn Koolen

Contents

Abstract

The fundamental problem of process-oriented qualitative models and simulations is to extract as much useful information as possible from sparse or incomplete descriptions. These qualitative reasoning approaches address the gap between the infinite complexity of the world and our partial knowledge of it, which defies traditional numerical modelling and generalises the reasoning to the qualitative behaviour capturing the key aspects. We take a theoretical approach to this problem, and make the following contributions. First, we develop a new partial axiomatisation that is better suited for simulation analysis. Second, using this axiomatisation, we identify inconsistencies of state graphs resulted from the use of sparse knowledge, and suggest solutions for these. Third, we introduce the problem of early inconsistency detection, requiring non-trivial inequality reasoning in the context of partial knowledge. Fourth, we formulate a generalisation of this problem, called the "combining changes problem", and analyse it from a complexity theoretic perspective, proving it to be polynomial under reasonable assumptions. Fifth, we derive a practical procedure for the early identification of contradictory relations, making qualitative reasoning more efficient by reducing the number of eligible compound terminations. As a general result, our work demonstrates the value of a theoretical approach to this problem when grounded by practical examples, realised using Garp3, clarifying the concepts and problems, and motivating the methods we developed.

Acknowledgments

First and foremost, I would like to thank my supervisors, Dr. Bert Bredeweg and Dr. Jaap Kamps, for their guidance during my first steps into the vast domain of Qualitative Reasoning, and for the patience of answering my questions, even when emails were sent at unreasonable hours. I am very grateful for their time and effort spent in guiding my writing of this thesis, as well as for having faith in good final results even when progress was really slow.

Lots of thanks also to the other members of the committee, Prof. Ronald de Wolf, Dr. Kaspar Beelen, Dr. Anders Bouwer, and Dr. Marijn Koolen who had the patience of going through pages of sometimes impenetrable mathematical notation. A huge thanks to Dr. Wielemaker for helping with all the SWIProlog related questions, and especially for his clarifications on the working of the profiler which was extremely valuable in the early phase of our work. Next, I would also like to thank Prof. Benedikt Löwe for the mentoring done throughout my two years in the Master of Logic, and to Dr. Thomas Forster for guiding my steps into taking this program.

Many thanks to my parents for all their support during the many years of learning that lead to this thesis. Last but not least, special thanks to Vlad for all the moral support, understanding for my panic as the deadline approached, and having to patiently listen to a lot of qualitative reasoning in the past few months. Also for proofreading, especially the acknowledgements (please, stop looking at me now... seriously).

Chapter 1

Introduction

Since there is an enormous gap between the infinite complexity of the world and our finite knowledge of it, people react to the environment using their internal representations, which are based on incomplete information (Kuipers, 1994). This incomplete information is usually expressed qualitatively, by the relative difference between quantities, rather then quantitatively, by giving in exact numerical values. For example, two people discussing future weather conditions would say "the temperature in Amsterdam will be slightly higher than the one in Utrecht", rather than "the temperature in Amsterdam will be equal to the one in Utrecht plus one degree".

Qualitative Reasoning (QR), which is a field of Artificial Intelligence, investigates how this type of reasoning can be automated (Van Harmelen et al., 2008). The aim of this field is to develop the means for creating a model that captures the available qualitative information (system structure, assumptions, conditions etc.), and that can infer possible outcomes through simulation. The need for reasoning with incomplete knowledge and for human-machine interaction motivates QR applications in a variety of subjects such as physics (De Kleer, 1990), ecology (Bredeweg and Salles, 2009), engineering (Abbott et al., 1988), and robotics (Pin et al., 1992). Lastly, it is very useful in education, as it gives a natural learning environment for students in science, enabling them to formalise their knowledge into models and use it through scenario simulations (Bredeweg and Forbus, 2003).

The focus of this thesis is on addressing the *resolution problem*, which is one of the three main QR problems according to Weld and De Kleer (2013). In short, since usually not all the necessary information is possessed, there is a need for reasoning techniques that work even under partial knowledge. This means our focus is on addressing the following research question:

How can we maximise the information inferable from sparse knowledge in process-oriented qualitative reasoning?

Our main approach for tackling this question is theoretical. We formulate a new axiomatisation based on the earlier attempts (see next paragraph for citations), but tailored to the needs of addressing our research question focussing on simulation rather than modelling. This axiomatisation is then used both to point out incoherences and to find procedures for early identification of incompatible system behaviour developments. The time analysis of these procedures is mostly done theoretically. Occasionally, we look at some particular examples to investigate practical benefits of our algorithms. This case analysis is done by running simulations in Garp3 (Bredeweg et al., 2009), which is a workbench based on an executable language that allows users to construct, simulate, and analyse models based on qualitative knowledge.

This thesis is structured around four main chapters, each addressing a specific question and contributing towards answering our main research question. Chapter 2 investigates how the earlier formalisations of process-oriented qualitative reasoning can be adapted to allow for theoretical analysis of simulations. Consequently, this chapter brings three main contributions. First, it provides a brief literature review of process-oriented QR. Second, it builds the clarity and the vocabulary needed to theoretically discuss qualitative simulations by developing a partial axiomatisation. This is mostly based on the earlier formalisations of Bredeweg et al. (2009), Liem (2013), and Linnebank (2004), but focusses on defining the concepts needed in simulations and on providing clear notations and definitions, similar to the work of Weld (1988). Third, to illustrate those definitions, a novel model from the field of Auction Theory is introduced. Fourth, a large simulation example (Kansou et al., 2017) is used for further clarifying the definitions and the simulation steps, and for a practical illustration of the problems associated with sparse knowledge.

In Chapter 3, the question addressed is of whether, because of sparse knowledge usage, principles of consistency are violated by the current qualitative simulation approach. First, this chapter underpins three types of incoherent behaviours that could be predicted by the previously axiomatised simulation approach. These violate either (a) *model consistency*, i.e. each state must be in accordance with the model, both in point of dependencies and relations, (b) *path consistency*, i.e. each possible sequence of events predicted should be coherent, and in particular should not contain contradictory assumptions or misrepresent extrema, or (c) *inexplicit inequality consistency*, i.e. the simulation result should also be in accordance with the inequalities that are not explicitly stated. Second, it finds solutions for solving these particular types of incoherences.

The focus of Chapter 4 is on whether there is a general formulation of a problem that is related to the one of early identification of system behaviour developments and that can be efficiently solved. This is an important issue since, because of the sparse nature of the available knowledge, evolution incompatibilities are hard to detect. This chapter brings three main contributions. First, it provides a formulation of a related problem, namely the combining changes problem (*COMB*). Second, under some explicit assumptions, classical complexity theoretical results (Boesch and Gimpel, 1977; Arora and Barak, 2009) are used to solve this problem in polynomial time. Third, a faster algorithm for *COMB*, which is better for practical use, is provided.

Chapter 5 investigates how the solutions for the general problem presented in the previous chapter can be adapted for efficiently using the sparse knowledge in qualitative simulations. First, it finds that, in the case of non-changing models, there is a reduction from *COMB* to the problem of finding plausible simultaneous next-state developments of systems (*ELIG*). Second, it adapts the previous solution to work under changing models. Third, the benefits of these solutions are illustrated by returning to the two examples from the second chapter.

This thesis finishes with a discussion and conclusions in Chapter 6, which presents the general conclusions and directions for future work.

Chapter 2

Modelling and Simulation in the Context of Qualitative Reasoning

This chapter aims to provide a clear understanding of process-oriented Qualitative Reasoning, which is necessary for answering the research question posed in the introduction. To achieve this, first, a partial axiomatisation for process modelling and simulation is developed. Second, two examples of simulations are then used with the double purpose of illustrating the concepts introduced in the earlier parts of the chapter, and of hinting potential problems resulting from sparse knowledge usage.

2.1 Qualitative Modelling and Simulation

As mentioned in the introduction, this thesis considers an executable language with logical constraints. Therefore, after modelling a system, the user can describe a situation (by means of an initial scenario) and run the model to analyse the possible qualitative developments. Hence, the process can be broken into three phases, namely: understanding the model language, introduced in Subsection 2.1.1, describing a state in the world, discussed in Subsection 2.1.2 and simulating the model, presented in Subsection 2.1.3. This section is based on the previous formalisations of Bredeweg et al. (2009), Liem (2013), and Weld (1988).

2.1.1 Modelling Language

In this phase of modelling, the user acquires the means of describing their knowledge about a system in a standardised form. There are various reasons for which modelling is important. At a conceptual level, the act of formally expressing all the aspects of one's knowledge can help both organising it and revealing the missing parts that can be later developed (Alessi, 2000). At a practical level, since it is expressed in a standardised form, other people can later understand and use the information contained within the model. Moreover, an algorithm can now interpret the information and use it to produce predictions (i.e. run simulations).

Therefore, one needs first to understand a standardised language in order to construct models. In this thesis, we build upon the language proposed by Bredeweg et al. (2009). This is also used in Garp3, which will be the workbench used for running examples. There are a multitude of ingredients used to construct this language, and this subsection presents the structural ones.

Entities are the basic structural compounds of the system which do not change during the simulation phase. Formally, we define an entity as a finite set of quantities, $E = \{Q_1, ..., Q_n\}$. *Quantities* are the changeable features of entities. For instance, the quantity "size" of the entity "population" might change over time. Similarly, the quantity "bid" of an entity "bidder" has different values throughout a simulation. At any given time, a quantity has also the tendency to either increase, remain steady or decrease. This tendency is given by the value of the derivative of a quantity at that given moment.

Hence, a quantity Q is defined as a variable, which has another associated variable named *(first order) derivative*, denoted by dQ. Moreover, if $\mathcal{Q}$ is a set of quantities, then the set of associated derivatives is denote by $d\mathcal{Q}$, i.e. $d\mathcal{Q} = \{dQ | Q \in \mathcal{Q}\}$. A *parameter* is defined inductively; any quantity is a parameter, and so are the derivatives of any parameter. For practical reasons, not all order derivatives are considered. In general, for a given set of quantities, the associated parameters are only these quantities together with their first order derivatives, that is $\mathcal{P}_{\mathcal{Q}} := \mathcal{Q} \cup d\mathcal{Q}$. Notice that this is different from the usual definition of a parameter (Weld and De Kleer, 2013).

The qualitative values that parameters can take are described by *landmarks* and *intervals*. For instance, a bid could be "zero", "plus", or "maximum", where "zero" and "maximum" are points (i.e. landmarks) while "plus" is an interval. The meanings associated with these are that the bid stands at zero, a positive value, or the maximum amount the bidder is willing to offer, respectively. Formally, landmarks are considered constants that can be partially ordered by the strict relation $<$. Two landmarks l_u, l_v within the set $\mathcal{L}$ of all landmarks are considered consecutive if $l_u < l_v$ and there is no $l_t \in \mathcal{L}$ such that $l_u < l_t < l_v$. As always, given a set of landmarks L, a minimal landmark is an l_1 such that there is no $l_0 \in L$ with $l_0 < l_1$ (and similarly for the maximal).

An interval is also a constant which is associated either with two consecutive landmarks or with a minimal or maximal landmark. So, for the consecutive landmarks $l_u < l_v$, the associated interval will be $i_{u,v}$. If l_1 is minimal, then $i_{0,1}$ is its associated interval, while if l_u is maximal it will have $i_{u,u+1}$ as an associated interval. The partial order $<$ also extends over intervals, so $l_u < i_{u,v} < l_v$. However, the relation is not strict over intervals, as $i_{u,v} < i_{u,v}$ is valid. The intuition for this is that there are more values within a qualitative interval. For example, if there are two bids that are at "plus", there is still the possibility that one is larger than the other, as being "plus" varies from being almost zero to being almost at maximum.

A given quantity can take different values depending on its nature. Therefore, while modelling the system, the user needs to specify the qualitative values that each quantity can take. So, for each quantity, there is an associated *magnitude space* formed of landmarks and intervals, in which its value can vary. The *landmark space* associated with a quantity Q is defined as a set of landmarks totally ordered by $<$, $L_Q = \{l_1 < l_2 < ... < l_m\}$ that quantity Q can take, while the *interval space* of Q is the set of intervals associated with these landmarks, that is $I_Q = \{i_{1,2}, i_{2,3}, ..., i_{m-1,m}\}$ possibly together with one or both of $i_{0,1}$ and $i_{m,m+1}$. The *magnitude space* of Q is hence the set of landmark and interval spaces of Q, i.e. $M_Q = L_Q \cup I_Q$.

Intuitively, the value of a derivative of a parameter P shows the tendency of change for that parameter. So, a derivative should be either negative, indicating that P is decreasing, zero, indicating that it is stable, or positive, indicating that it is increasing. Hence, for any parameter P, $L_{dP} = \{0\}$, $I_{dP} = \{-, +\}$, and $M_{dP} = \{- < 0 < +\}$.

The model for a bidder, which is depicted in Figure 2.1, illustrates the concepts defined above (see Section 2.2 for details and motivation on the auction model). The entity *Bidder* has two associated quantities, namely *Bid* and *To_absolute*. Using the notation, $Bidder = \{Bid, To\_absolute\}$. The landmarks for the quantity *Bid* are 0 and *Max*, and there is only one interval, namely *Plus*. Hence, the landmark space is $L_{Bid} = \{0, Max\}$, the interval space is $I_{Bid} = \{Plus\}$, and the magnitude space is $M_{Bid} = \{0, Plus, Max\}$. As usual, the magnitude space for the derivative is $M_{dBid} = \{-, 0, +\}$.

Note however that the magnitudes are not necessarily equal between quantities, even though they have the same name. If two quantities $Bidder_1$, $Bidder_2$ have *Max* in their magnitudes space, formally they have the two associated landmarks Max_{Bidder_1} and Max_{Bidder_2}, which in general might not be equal. However, to simplify notation, the indices are sometimes dropped.

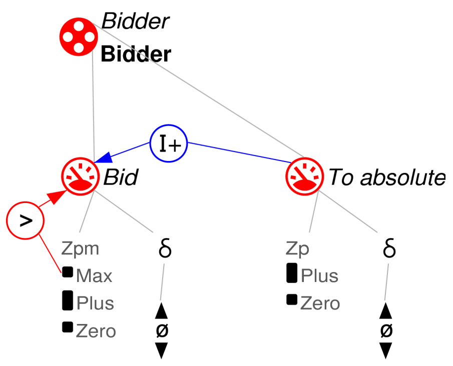


Figure 2.1: A model fragment for the English Auction model.

The complexity of a system usually comes not from the number of elements composing it, but rather from the interrelated nature of those elements. Usually, a change in one quantity determines a change in another, which in turn might have various effects on other quantities. For example, if a bidder is not the absolute winner, that is it the value of its *To_absolute* quantity is positive, this determines an increase of its bid. This is modelled using *causal ingredients*, which are of three types:

- *Proportionality* (which can be either positive or negative) models a correlation between the tendencies of two quantities. An example is the proportionality between the size and the number of births of a population: if the first increases, this will determine an increase in the second. More precisely, if Q_1 is positively proportionally influenced by Q_2, Q_1 has no other dependencies, and Q_2 is increasing, then Q_1 should also be increasing.

  More formally, the two types of proportionality are denoted by $P+$ or $P-$, and each of them is an antisymmetric binary relation on $\mathcal{Q}$.

- *Influence* (which can be positive or negative) models the dependency between the magnitude of a quantity and the tendency of another. To take an example, the number of deaths negatively influences the size of a population as, if the number of deaths is positive, and there are no births or immigration, then the size of the population should decrease. More precisely, if Q_1 is positively (directly) influenced by Q_2, Q_1 has no other dependencies, and Q_2 is positive, then Q_1 should also be increasing.

  Formally, the two types of influences are denoted by $I+$ or $I-$, and each of them is an antisymmetric binary relation on $\mathcal{Q}$.

  We also say that a *dependency* is either an influence or a proportionality relation.

- *Correspondence*, which can be directed or undirected, shows co-appearing magnitudes. For example, in an auction with two bidders, a zero difference between the two bids means that the two bidders are not the absolute winners, so their *To_absolute* quantities should be *Plus*. Hence, a bidding difference of zero unidirectionally corresponds to a *To_absolute* value of *Plus*.

  Formally, the directed (value) correspondence set C is a binary relation on $\mathcal{P}_{\mathcal{Q}} \times \cup_{P \in \mathcal{P}_{\mathcal{Q}}} M_P$. More precisely,

$$C \subseteq \{((P_1, v_1), (P_2, v_2)) | P_i \in \mathcal{P}_{\mathcal{Q}}, v_i \in M_{P_i}, P_1 \neq P_2\}.$$

  In the example above we have the correspondence $((\textit{Bid\_difference}, 0), (\textit{To\_absolute}, +))$.

Additionally, the undirected correspondence is symmetric. However, since the undirected correspondence can be expressed in terms of directed correspondences we will only use the latter, to which we will refer to simply as correspondences.

With the elements presented above, one has the needed language to construct a model. The inequalities extended to quantities, as well as the option of modelling by fragments, are also useful, but will be discussed in the next subsection. However, on its own, this language is not sufficient as, in order to use it, one should be able to provide a description of the current state of the world, as well as to run it in order to predict possible future developments.

2.1.2 World Description

Up to this point, the language for encoding the general information that remains true throughout any simulation was presented. That is only the information that does not depend on any particular value. However, in order to make the model useful, one should be able to supply information about the state of the world at a particular time. This means providing details about the magnitudes of the quantities used and their derivatives, the relations between different quantities, etc. In this subsection, we build the vocabulary used to provide the model with a (partial) description of its state at a particular time.

Firstly, *(in)equalities* are used to compare certain ingredients. These are binary relations $R \in \{<, \leq, =, \geq, >\}$, which can appear between two quantities, landmarks or derivatives, a quantity and a landmark, or a derivative and zero. More precisely, for a given set of quantities $\mathcal{Q}$, the elements of the relations R are of the form:

- (Q_1, Q_2), for $Q_1, Q_2 \in \mathcal{Q}$, i.e. comparing two quantities;
- (Q, l), for $l \in L_Q$, i.e. comparing a quantity and a landmark;
- (l, l'), for $l, l' \in L_Q$, $Q \in \mathcal{Q}$, i.e. comparing two landmarks;
- (dQ_1, dQ_2), for $Q_1, Q_2 \in \mathcal{Q}$, i.e. comparing two derivatives;
- $(dQ, 0)$, for $Q \in \mathcal{Q}$, i.e. comparing a derivative and zero.

This is also extended to inequalities on terms, that is on expressions obtained from parameters together with arithmetic operators.

The example in Figure 2.2 shows that the difference between the maximum bids of the two bidders needs to be negative and large, i.e. *Neg_large*. This is expressed by an inequality between a term and a landmark, and is given by $(\mathit{Max}_{\mathit{Bid}_1} - \mathit{Max}_{\mathit{Bid}_2}, \mathit{Neg\_epsilon}) \in <$.

In addition, one should also be able to specify qualitative values for different ingredients in order to compare them. That is, if at a certain moment the bid of the first bidder is 0, and a prediction of possible future developments from that state is desired, one should be able to specify this value. A specification of one or more such values will be referred to as an *assertion.* Formally, this is defined as follows:

- A *magnitude assertion* for the set of quantities $\mathcal{Q}$ is a (partial) function $mag : \mathcal{Q} \to \cup_{Q \in \mathcal{Q}} M_Q$, such that $mag(Q) \in M_Q$ for all $Q \in \mathcal{Q}$ and $mag(Q)$ defined. Hence, a magnitude assertion for $\mathcal{Q}$ is a function that gives to (some) quantities in $\mathcal{Q}$ some magnitude within their associated magnitude space.
- A *derivative assertion* for the set of quantities $\mathcal{Q}$ is a (partial) function $der : d\mathcal{Q} \to \{-, 0, +\}$.
- A *qualitative value assertion* for the set of quantities $\mathcal{Q}$ is a (partial) function $val : \mathcal{Q} \cup d\mathcal{Q} \to \cup_{Q \in \mathcal{Q}} M_Q \cup \{-, 0, +\}$ such that $val_m := val|_{\mathcal{Q}}$ is a magnitude assertion, and $val_d := val|_{d\mathcal{Q}}$ is a derivative assertion.

As an example, in Figure 2.2 there is a magnitude assertion for the set of quantities $\mathcal{Q} = \{Bid_1, Bid_2, Bid\_difference\}$ is a total function, such that:

- $mag(Bid_1) = mag(Bid_2) = mag(Bid\_difference) = 0$,
- $mag(dBid\_difference) = 0$, and
- $mag(dBid_1) = mag(dBid_2) = +$.

Moreover, since there are no derivative specifications, the qualitative value assertion is the same as the magnitude assertion.

This gives all the elements to define a *scenario*, that is a partial qualitative description of a situation. A scenario captures information about the qualitative values and the relations between different ingredients of the model. Moreover, the two elements of the scenario must be consistent with each other. For example, if two quantities are equal, and they also have asserted values, then their values should not be specified as being unequal.

An example of a scenario, realised using Garp3, is pictured in Figure 2.2. This scenario presents an auction with two bidders, each of them currently bidding zero, but with a tendency to increase the bid. In addition, the difference between the two bidders is currently zero and stable. Finally, the second bidder has a higher valuation than the first one, meaning that they are willing to bid (significantly) more.

Formally, a scenario s is a tuple $\langle \mathcal{E}, val, <, \leq, = \rangle$, where $\mathcal{E}$ is a set of entities, val is a qualitative value assertion for $\mathcal{Q}$, and $<, \leq, =$ are inequalities on $\mathcal{E}$. In addition to this, if $val(P_1)$ and $val(P_2)$ are defined, and $P_1 \ R \ P_2$ for some

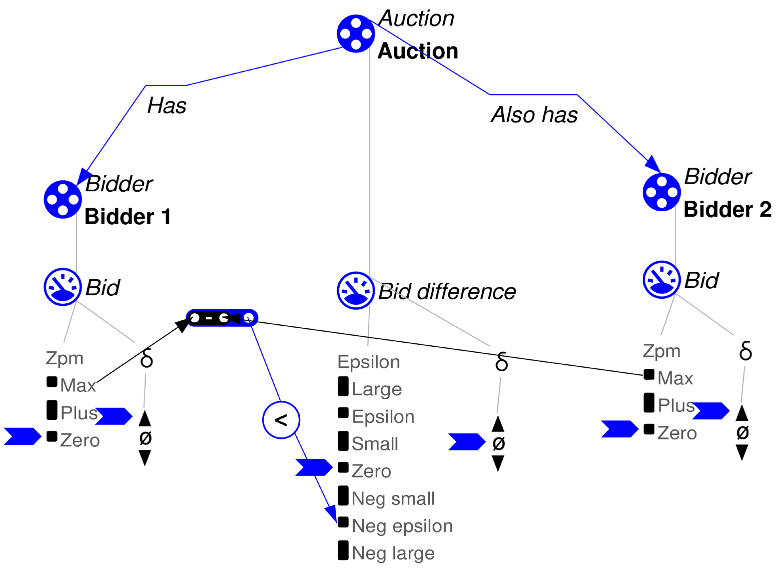


Figure 2.2: English Auction with two bidders: initial scenario

$R \in \{<, \leq, =\}$, then the relation between P_1 and P_2, if specified, should be $val(P_1)\ R\ val(P_2)$, where P_1, P_2 are parameters in $\mathcal{E}$, formally in $\mathcal{P}_{\cup_{E \in \mathcal{E}} E}$.

Another useful concept is that of *model fragments* (MF), used to describe how parts of the system behave under certain conditions. One such example was already mentioned in Figure 2.1. The conditions are shown in red, and the consequences in blue. As a result, the model fragment reads as follows: if there is an entity *Bidder* with quantities *Bid* and *To_absolute*, such that *Bid* is at a value smaller than *Max*, then the *To_absolute* quantity positively influences the *Bid*. Notice that the same model fragment can be used multiple times. For example, if there are two bidders respecting the conditions of this model fragment, as it is the case in Figure 2.2, then the model fragment would apply for both.

An *active* model fragment for a given scenario, is a model fragment that has the conditions satisfied by that scenario. In order to provide a formal definition for model fragments, a classification of the different types of ingredients is needed.

The *behaviour ingredients* of a scenario are the relations and the value assertion function. That is, the behaviour ingredients of $s = \langle \mathcal{E}, val, <, \leq, = \rangle$ are $val, <, \leq, =$. The *structural ingredients* of a scenario are the set of its entities (which also contains information about quantities), together with agents, configurations, attributes, and assumptions. The last four types are beyond the scope of this thesis, so we will not insist on these. In general, the *conditionals* of a scenario are its structural and behavioural ingredients.

Using this, MFs are then rules that link a set of conditionals to a set of causal ingredients and possibly (in)equality restrictions. Hence, they can be determined by a function, say the *model fragment function* $mf : \mathcal{P}(Cond) \rightarrow \mathcal{P}(Caus)$, where $Cond$ is the set of all possible conditionals, and $Caus$ is the set of causal ingredients together with (in)equality relations.

Therefore, for every state there are associated proportionality, influence and correspondence relations, plus possible (in)equality restrictions, that are given by the MF function. Those will be referred to as the set of *active causal ingredients*, which, for a scenario s, is given by $\bigcup_{Co \subseteq Cond_s} mf(Co)$.

So, in this section, an axiomatisation for a state description was provided. In addition, model fragments were used to account for case distinctive behaviours of the system. The focus of the next section is on running a scenario in the context of a given model fragment function.

2.1.3 Model Simulation

Having constructed a model, and presented it with an associated scenario, it is now time to run the model with the scenario in order to predict possible system behaviours. This is referred to as a *simulation*. Throughout this subsection, the focus will be on providing the vocabulary for describing the simulation of a model fragment function and a scenario.

To begin with, a *state* represents a complete version of a scenario. Differently from a scenario, a state has known qualitative values for all the magnitudes of quantities. More precisely, a state s is a scenario for which the magnitude assertion is total, i.e. the qualitative value assertion restricted on quantities (val_m) is a total function. Furthermore, a state s extends a scenario s' if all the elements of s are supersets of the respective ones in s'. If the scenario is under-specified, it can have more than one state extension. In that case the scenario is called *ambiguous*.

In order to capture the dynamic nature of a system, states must be able evolve into new ones. The way a state changes is by *terminating* the current one. Hence, a *simple termination* is defined as a pair capturing a cause and a set of resulting changes. So, a simple termination for a state s is a pair (c, Re), where c is a cause, and Re is a set of results. Moreover, the tuple needs to satisfy the termination validity criteria which will be defined later. The set of results is a collection of relations on the parameters of s, while the different types of causes are defined below.

A *cause* c of a termination for a state $s = \langle \mathcal{E}, val, <, \leq, = \rangle$ is a pair that gives the type of change (i.e. a constant from Table 2.1) and a quantity (or a pair of quantities) from $\cup_{E \in \mathcal{E}} E$, i.e. $c = (cause\_name, Q)$, for some Q $in \cup_{E \in \mathcal{E}} E$. Table 2.1 presents the causes grouped into three possible categories. Formally, there are four categories, as there are also exogenous terminations on derivatives (Bredeweg et al., 2009), but these necessitate additional constraints, and are outside the scope of this thesis. In additon, for the derivative and inequality causes, there are also assumed versions of the constants, e.g. $(assumed\_derivative\_stable\_to\_down, Q)$.

As stated before, in order for a pair $t = (c, Re)$ to be a termination of state

Class of causes	Type of cause
Value causes	$(\mathit{to\_point\_above}, Q)$
	$(\mathit{to\_point\_below}, Q)$
	$(\mathit{to\_interval\_above}, Q)$*
	$(\mathit{to\_interval\_below}, Q)$*
Derivative causes	$(\mathit{derivative\_stable\_to\_down}, Q)$*
	$(\mathit{derivative\_stable\_to\_up}, Q)$*
	$(\mathit{derivative\_down\_to\_stable}, Q)$
	$(\mathit{derivative\_up\_to\_stable}, Q)$
Inequality causes	$(\mathit{from\_equal\_to\_greater}, (Q_1, Q_2))$*
	$(\mathit{from\_equal\_to\_smaller}, (Q_1, Q_2))$*
	$(\mathit{from\_smaller\_to\_equal}, (Q_1, Q_2))$
	$(\mathit{from\_greater\_to\_equal}, (Q_1, Q_2))$
	$(\mathit{from\_smaller\_or\_equal\_to\_smaller}, (Q_1, Q_2))*$
	$(\mathit{from\_smaller\_or\_equal\_to\_equal}, (Q_1, Q_2))$
	$(\mathit{from\_smaller\_or\_equal\_to\_greater}, (Q_1, Q_2))*$
	$(\mathit{from\_greater\_or\_equal\_to\_greater}, (Q_1, Q_2))*$
	$(\mathit{from\_greater\_or\_equal\_to\_equal}, (Q_1, Q_2))$
	$(\mathit{from\_greater\_or\_equal\_to\_smaller}, (Q_1, Q_2))*$

Table 2.1: Types of causes

$s = \langle \mathcal{E}, val, <, \leq, = \rangle$, t needs to satisfy the *termination validity criteria*. This means that only certain pairs of causes and results are valid. Moreover, these are valid for the state s only when a certain set of conditions (constraints) of s is valid. The *condition function* of s, denoted by *cond*, is a function from the set of simple terminations to the power set of constraints of s. Hence, $\mathit{cond}(t)$ is the set of constraints for the termination t. Table 2.2 outlines the termination validity criteria. This is based on the work of Linnebank (2004).

Not all terminations have the same level of priority, as some happen before others. The set of terminations is therefore divided into two parts, namely *immediate terminations*, which are from a point or from equality, i.e. the ones marked with * in Table 2.1, and *non-immediate terminations*, which are to a point or to equality. Hence, the set of simple terminations $\mathcal{T}_s$ is the disjoint union of the set of immediate terminations $\mathcal{I}_s$ and the set of non-immediate terminations $\mathcal{N}_s$. That is, $\mathcal{T}_s = \mathcal{I}_s \dot{\cup} \mathcal{N}_s$.

Because of the inter-related nature of the elements of the system, one termination is usually not sufficient to reach another state. Instead, more terminations would need to co-occur. To capture this behaviour, a *compound termination* for

Cause type	Elements of $cond(t)$	Elements of results set
(to_point_above, Q)	$val(Q) = i_{u,u+1}$, for $i_{u,u+1} \in I_Q$ $val(dQ) = +$	$val(Q) = l_{u+1}$ $val(dQ) \geq 0$
(to_point_below, Q)	$val(Q) = i_{u,u+1}$, for $i_{u,u+1} \in I_Q$ $val(dQ) = -$	$val(Q) = l_u$ $val(dQ) \leq 0$
$(to\_interval\_above, Q)$	$val(Q) = l_u$, for $l_u \in L_Q$ $val(dQ) = +$	$val(Q) = i_{u,u+1}$ $val(dQ) \geq 0$
$(to\_interval\_below, Q)$	$val(Q) = l_{u+1}$, for $l_{u+1} \in L_Q$ $val(dQ) = -$	$val(Q) = i_{u,u+1}$ $val(dQ) \leq 0$
$(derivative\_stable\_to\_down, Q)$	$val(dQ) = 0$ $val(ddQ) = -*$	$val(dQ) = -$ $val(ddQ) \leq 0*$
$(derivative\_stable\_to\_up, Q)$	$val(dQ) = 0$ $val(ddQ) = +*$	$val(dQ) = +$ $val(ddQ) \geq 0*$
$(derivative\_up\_to\_stable, Q)$	$val(dQ) = +$ $val(ddQ) = -*$	$val(dQ) = 0$ $val(ddQ) \leq 0*$
$(derivative\_down\_to\_stable, Q)$	$val(dQ) = -$ $val(ddQ) = +*$	$val(dQ) = 0$ $val(ddQ) \geq 0*$
$(from\_equal\_to\_greater, (Q_1, Q_2))$	$Q_1 = Q_2$ $dQ_1 > dQ_2*$	$Q_1 > Q_2$ $dQ_1 \geq dQ_2*$
$(from\_equal\_to\_smaller, (Q_1, Q_2))$	$Q_1 = Q_2$ $dQ_1 < dQ_2*$	$Q_1 < Q_2$ $dQ_1 \leq dQ_2*$
$(from\_smaller\_to\_equal, (Q_1, Q_2))$	$Q_1 < Q_2$ $dQ_1 > dQ_2*$	$Q_1 = Q_2$ $dQ_1 \geq dQ_2*$
$(from\_greater\_to\_equal, (Q_1, Q_2))$	$Q_1 > Q_2$ $dQ_1 < dQ_2$	$Q_1 = Q_2$ $dQ_1 \leq dQ_2*$

Table 2.2: Termination validity criteria. Note that the inequality transitions from a weak constraint to a strong one were omitted, but these are similar to the mentioned ones. In addition, if the constraints marked with star do not appear explicitly, but could be added while maintaining consistency of the state, then the cause is assumed.

a state s is defined to be a set of simple terminations of s. Since immediate terminations take priority over non-immediate one, a compound terminations only has elements from the immediate terminations, if such elements exist, otherwise it is composed by non-immediate terminations. Formally,

$$T \subseteq \begin{cases} \mathcal{I}_s\texttt{, if } \mathcal{I}_s \neq \emptyset \\ \mathcal{N}_s\texttt{, otherwise} \end{cases},$$

where $\mathcal{I}_s$ is the set of immediate terminations of s, $\mathcal{N}_s$ is the set of non-immediate ones, and T is a compound termination. This rule of prioritising immediate terminations will be referred to as the *epsilon ordering rule.*

The set of compound terminations of s is denoted by $\overline{\mathcal{T}_s}$. The set of causes of the compound termination T is the set of causes that belong to the simple terminations within T, that is $\{c|(c, Re) \in T\}$. Similarly, the results of T is the union of the result sets of the simple terminations, i.e. $\cup_{(c,Re)\in T} Re$. Moreover, the condition function naturally extends over compound terminations by letting $cond(T) = \cup_{t\in T} cond(t)$.

The physical world is continuous, in the sense that there are no jumps in tendencies between consecutive states of the same system. So, the *continuity criterion* states that the tendency of a quantity cannot jump from the positive to the negative interval or vice-versa without first passing through zero. Another way of saying this is that a pair of states (s_1, s_2) satisfies the continuity criterion iff for every parameter P appearing resulting from the entity spaces of the two states the following holds:

- if $val_1(dP) = -$, then $val_2(dP) \leq 0$
- if $val_1(dP) = +$, then $val_2(dP) \geq 0$

At this point a state and a compound termination can produce a new state by implementing the resulting changes, captured by the termination, into the old state. A *transition scenario* from state s with the compound termination T is then defined as the scenario s' obtained from s by changing with the results in T, that is with $Re_T = \cup_{(c,Re)\in T} Re$.

However, simply making the changes within a state does not guarantee the consistency of the state, as it is not necessarily in accordance with it active causal ingredients. Therefore, we say that a state $s = \langle \mathcal{E}, val, <, \leq, = \rangle$ with its active causal ingredients $P+, P-, I+, I-, C$ is *stable* if:

- There is an influence balance. This happens when the qualitative value assertion *val* is plausible with the influence relation of s. Formally, that is when for every $Q \in \cup_{E\in\mathcal{E}} E$, the relation

  $$dQ = \sum_{(Q_1,Q)\in I+} f_1(Q_1) - \sum_{(Q_2,Q)\in I-} f_2(Q_2),$$

  where the functions f_i are strictly increasing and with zero as a fixed point, holds.

- There is a proportionality balance. Similarly, this happens when the qualitative value assertion *val* is plausible with the proportionality relation of s. Formally, that is when for every $Q \in \cup_{E \in \mathcal{E}} E$, the relation

$$dQ = \sum_{(Q_1,Q) \in P+} g(dQ_1) - \sum_{(Q_2,Q) \in P-} g(dQ_2).$$

  where the functions g_i are strictly increasing and with zero as a fixed point, holds. Notice that in this case, because of the possible values of the derivatives, the relation is equivalent with $val(dQ) = \sum_{(Q_1,Q) \in P+} val(dQ_1) - \sum_{(Q_2,Q) \in P-} val(dQ_2)$.

- There is a correspondence fit. This happens when the qualitative value assertion *val* is plausible with the correspondence relation of s. Formally, that is when for every $P_1, P_2 \in \cup_{E \in \mathcal{E}} E$ and $v_i \in M_i$, such that $((P_1, v_1), (P_2, v_2)) \in C$, if $val(P_1) = v_1$, then $val(P_2) = v_2$.

Notice that the influence and proportionality balances are separate, and not mixed. The reason for this is that in qualitative reasoning, the dependencies are interpreted as causalities, so a quantity cannot have both incoming influences and proportionalities.

There are now all the components needed to define the possible developments of a current state. A *successor* of a state s is a continuous and stable state s', extending the transition scenario of s and some compound termination T. The change between the successive states s and s' is captured by a transition. Hence, a *transition* is formally defined as a pair of successor states (s, s'). Now there is a visual representation of the system's development by the states and the transitions between them. This is in the form of a *state graph* (or behaviour graph), which is a directed graph $G = (V, E)$ where V is a set of states and E is the set of transitions between them. Moreover, any behaviour graph must be maximal, in the sense that there is no state in V with a successor outside V. A path in this state graph, which shows a possible development of the system, is referred to as a *track*. Examples for those are discussed in the next section.

This finalises our axiomatisation of the system. As mentioned in the introduction of this section, this is only a partial one meant to bring the conceptual clarity needed for investigating the research question. Throughout this section, the formal definitions were illustrated using examples from an English Auction model. These example will be discussed in the next section.

2.2 Continuous English Auction with Two Bidders from a Qualitative Perspective

This section has the purpose of explaining the English Auction model into more detail in order to further exemplify the notions introduced before. In doing so, the first subsection discusses the English Auction, the second one the modelling, and the last one, the simulation.

2.2.1 Continuous English Auction

During a classical English Auction, an item is presented to people in the room, and the ones interested bid for this item. Each person has a *valuation*, that is a maximum value that they are prepared to bid. During the bidding, one person starts with an offer. If anybody is willing to offer more, they can challenge the highest current bid by offering a higher amount. The auction stops when nobody wants to bid over the highest offer. The item is usually indivisible, so there can only be one winner (McAfee and McMillan, 1987).

This system is obviously not continuous. However, for theoretical analysis in Auction Theory, the continuous version is used. According to that, the bidders increase their offer in a continuous manner, and the winner is the person that offers more than any other one by at least some epsilon (an arbitrarily small constant). This version is similar to the ascending bid or Japanese Auction (Milgrom and Weber, 1982), where the interested people stay in a room where an initial price is displayed. The price increases continuously, and the bidders that are not interested in purchasing the item at that price exit the room. The winner is the last bidder remaining in the room.

For clarity, this section aims at modelling the continuous version of the English Auction for two bidders only, but the same principles can be used for larger numbers of bidders. To achieve this aim, the model fragments need to be constructed based on entities. This is done in the Subsection 2.2.2.

2.2.2 Qualitative Model

Having described what a Continuous English Auction is, this subsection presents the model fragments used for its qualitative modelling. Figure 2.1, already showed one such model fragment. This encodes an obvious strategy for a bidder, namely that if their bid is not maximum, then the difference they have until becoming the absolute winner positively influences their bid. Another model fragment, shown in Figure 2.3, says that once at the maximum value, the bid stagnates, i.e. one cannot bid over their valuation. The fragment in Figure 2.4 ensures that the bidding difference is indeed the first bid minus the second one, that is *Bid_difference* = *Bid_a* − *Bid_b*. There are also proportionality relations: a positive one from *Bid_a* to *Bid_difference*, as an increase in the first bid produces an increase of the difference, and a negative one from *Bid_b* to *Bid_difference*, as if the former increases the latter would decrease. In addition, the *Epsilon* and *Neg_epsilon* landmarks have the same absolute values, but opposite signs, i.e. their sum is zero.

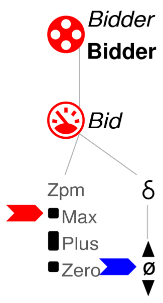


Figure 2.3: English Auction with two bidders: model fragment for the strategy according to which bidders do not bid over their valuation.

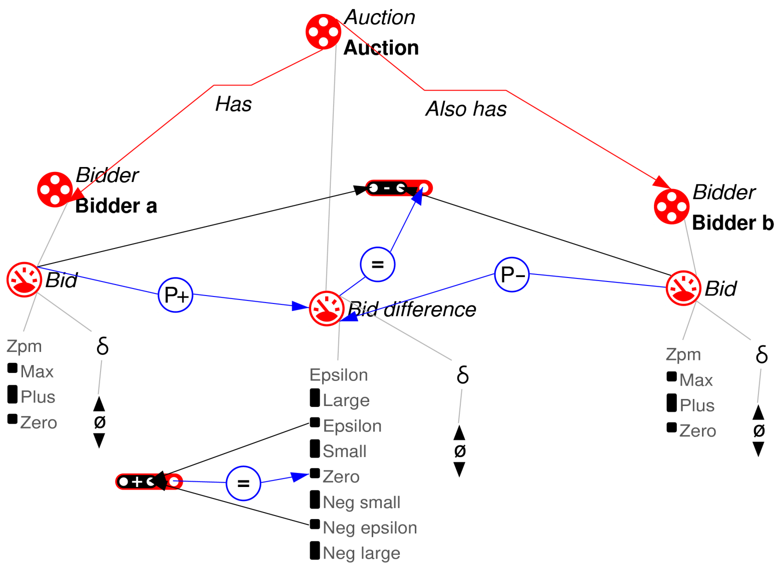


Figure 2.4: English Auction with two bidders: model fragment for the behaviour of *Bid_difference*.

The only quantities not covered by the previously introduced model fragments are the *To_absolute* quantities of the two bidders. To do this, the model fragment from Figure 2.4, is extended to the one in Figure 2.5, which shows the correspondences for the *To_absolute* quantities. A bidder is the absolute winner if the difference between them and any other bidder is at least epsilon. For the case of two bidders, this means that $To\_absolute_{Bidder_a} = 0$ iff $Bid\_difference \geq Epsilon$, and $To\_absolute_{Bidder_b} = 0$ iff $Bid\_difference \leq Neg\_epsilon$. This is precisely what is encoded by the value correspondences in the model fragment. In addition, whenever *To_absolute* has a non-zero value, it is proportionally influenced by the $Bid\_difference \geq Epsilon$. In other words, whenever $Bid\_difference \leq Epsilon$ this negatively proportionally influences $To\_absolute_{Bid\_a}$, see Figure 2.6, and whenever $Bid\_difference \geq Neg\_epsilon$ this positively proportionally influences $To\_absolute_{Bid\_b}$, see Figure 2.7. So, $To\_absolute_{Bid\_a}$ is a quantity that is zero whenever *Bid_difference* is *Large* or *Epsilon* and otherwise it increases if and only if *Bid_difference* is decreasing.

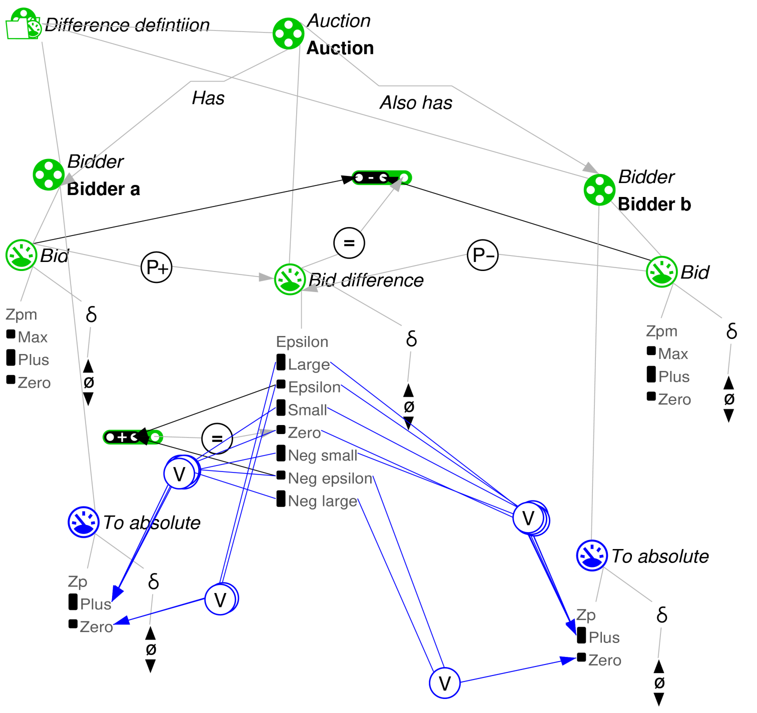


Figure 2.5: English Auction with two bidders: model fragment showing the correspondences from the *Bid_difference* quantity to the *To_absolute* quantities.

This finalises the building of the model for the English Auction. It was done via six model fragments that encode the obvious strategies of the bidders, i.e. "do not bid over valuation", "if the bid did not reach the valuation, increase bid iff not the absolute winner", as well as the behaviour for the other quantities, i.e. the *Bid_difference* which is the difference between the two bids, and the two *To_absolute* quantities. In the next subsection the simulation is discussed.

2.2.3 Example of a Simulation

This section presents the simulation for the scenario in Figure 2.2. Remember that according to that scenario, the maximum of Bid_1 is lower than the maximum of Bid_2 by a non-negligible amount, that is an amount greater than *Epsilon*. In addition, the two bids are currently at zero, as is their difference.

This scenario is firstly extended to a state, i.e. state 1 in Figure 2.8a. According to Table 2.2, this state has four possible simple terminations, which are pictured in Figure 2.8b. Since the two quantities *Bid* are at 0 with a positive derivative, (*to_interval_above*, *Bid_i*) are hence simple terminations for both bidders, i.e. for both $i = 1$ and $i = 2$. In addition, since the derivative of

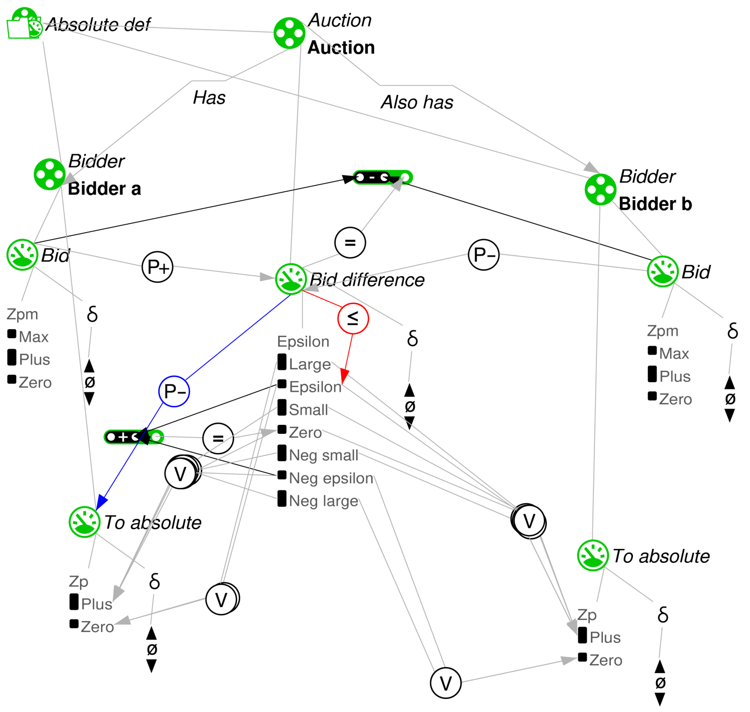


Figure 2.6: English Auction with two bidders: model fragment showing how *Bid_difference* influences the *To_absolute* of *Bid_a*.

Bid_difference, and second order derivatives are unknown, there are also assumed derivative terminations for this quantity. So the other two simple terminations are (*assumed_derivative_stable_to_up*, *Bid_difference*), and (*assumed_derivative_stable_to_down*, *Bid_difference*). The compound terminations in Figure 2.8c are obtained by combining the simple ones. Since the two value terminations are immediate and non-assumed, by epsilon ordering, they should always co-appear. Moreover, the two assumed terminations are contradictory, so they cannot co-occur. Therefore, the three compound terminations that may produce successor states are obtained from the two value terminations together with one or no assumed derivative change. Each of these gives a successor for 1.

To continue this example, consider state 2. In this state both bids are, according to the valuation, *Plus* and have a positive derivative. Moreover, the *Bid_difference* is 0 and decreasing. According to the model fragments in figures 2.5, 2.6, and 2.7, both *To_absolute* quantities are *Plus*, the one corresponding to *a* is increasing, while the one corresponding to *b* is decreasing. This state has five simple terminations, as shown in Figure 2.9a, but only one of those, namely (*to_interval_below*, *Bid_difference*) is an immediate termination, so this forms alone the only compound termination, see Figure 2.9b. Consequently, only one successor for state 2 exists, as shown in Figure 2.9c.

With similar reasoning, the entire state graph can be constructed; this graph

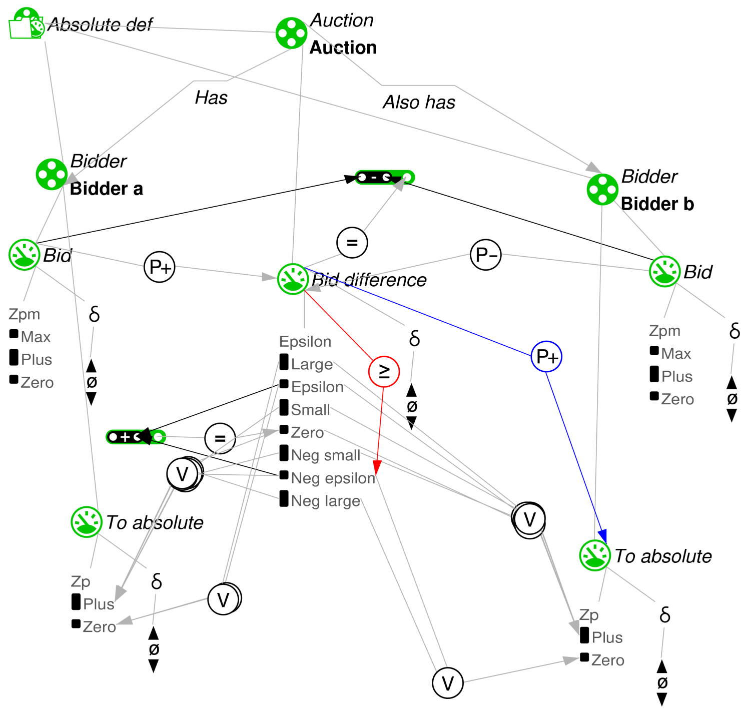
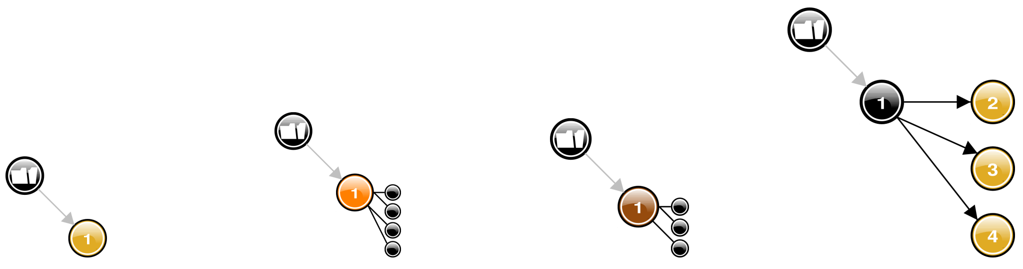


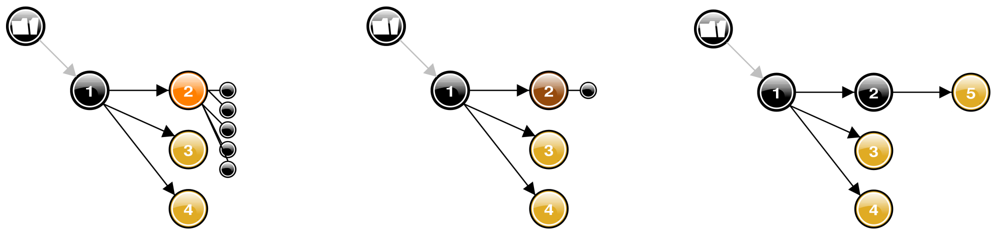
Figure 2.7: English Auction with two bidders: model fragment showing how *Bid_difference* influences the *To_absolute* of *Bid_b*.



(a) The extension of the initial scenario. (b) The simple terminations. (c) The compound terminations. (d) The successors.

Figure 2.8: The stepwise identifications for the successors of the state extending the initial scenario in Figure 2.2. To see the diagram with the specific values and their derivatives, please refer to Figure 2.11.

is pictured in Figure 2.10. One of its tracks is shown in Figure 2.11. The values of the quantities are represented by circles at the height of the corresponding interval or landmark, and the derivatives are shown by arrows within the circles. The arrows next to the cycles are for the second order derivatives. To read it out, according to this track both bidders are increasing their bids until the last two steps. The difference between their bids is always non-positive, meaning that the second bidder always has a higher offer. This difference does not distance itself



(a) The simple terminations. (b) The compound terminations. (c) The successors.

Figure 2.9: The stepwise identifications for the successors of the second state for the simulation with the initial scenario from Figure 2.2. To see the diagram with the specific values and their derivatives, please refer to Figure 2.11.

from zero by at least epsilon until the last state, namely state 8. Also until this last state, none of the bidders is the absolute winner. In the penultimate state, namely state 10, the first bidder reaches his maximum valuation, and, therefore, they stop increasing their bid. The second bidder then increases their offer until they become the absolute winner.

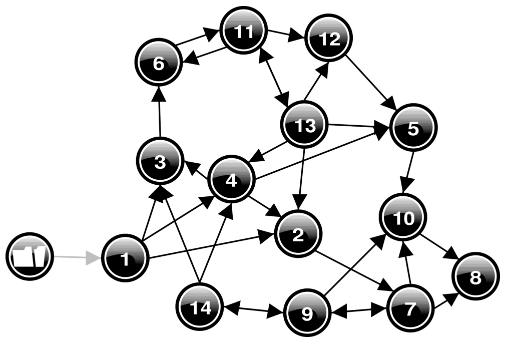


Figure 2.10: The state graph of the model for the two bidders continuous English Auction, together with the initial scenario from Figure 2.2.

From the state graph, notice that state 8 is the only state without any successors. This is, therefore, the only equilibrium point of the auction, that is the only point when bidders do not have an incentive to change their offer. Since, in this state, the revenue, i.e. highest offer, equals the second highest valuation plus *Epsilon*, and *Epsilon* is arbitrarily small, the result of the simulation is in accordance with Vickrey's theorem (Vickrey, 1961). This theorem also proves that the expected revenue for the English Auction equals the second highest valuation.

All the diagrams in this section were produced using Garp3, and had the purpose of clarifying the concepts introduced in the previous section. Next, we investigate one way of computing the state graph in a given setting, by referring to the example of the reasoning engine in Garp3 (Bredeweg et al., 2009).

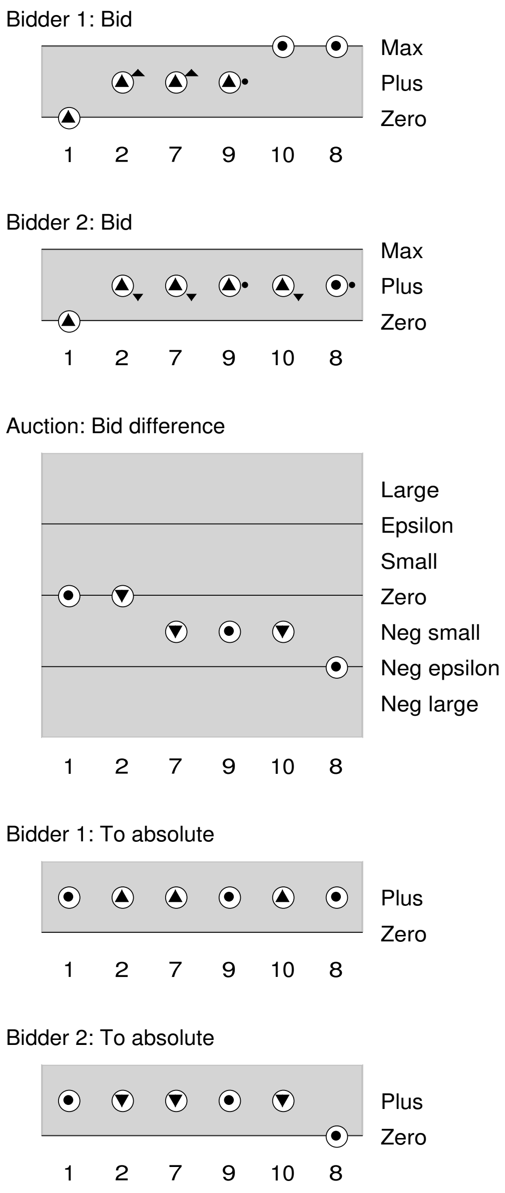


Figure 2.11: A path in the state graph from Figure 2.10.

2.3 Reasoning Engine

The previous section exemplifies the general axioms for qualitative modelling and simulations which were introduced in Section 2.1. In this section, the question of how qualitative simulations can be carried out in practice is addressed. To answer this, Garp3 is considered as an example of an engine that automatically performs simulations for provided model fragments and scenarios. Based on the work of Bredeweg et al. (2009) and our own analysis of the code, this chapter gives an overview of the engine's reasoning aspects that are relevant for this thesis.

2.3.1 Outline

As explained in Section 2.1, a simulation is based on a given library of model fragments, which encode the general behaviour of the system, and on a scenario, which provides some information about a state the system could be in. Given these two elements, the engine firstly extends the provided scenario to one or more initial states, depending on whether the scenario was ambiguous. After this, for each state, its successors are found and added to the state-graph, together with the respective transitions.

The process of finding states and transitions is done in phases. Firstly, when a state is created, either as an extension of a scenario or as a successor of an existing state, its state is marked as *interpreted*. Next, the simple terminations of the state are identified and the state is marked as *terminated*. Then, the eligible compound terminations are found, via a procedure that we will later describe, and the state is marked as *ordered*. Lastly, the state becomes *closed* when its valid successors obtained from the short-listed compound terminations are identified, and the respective transitions are found. At this point, it is known that the state at hand cannot be further continued to a successor. Figure 2.12 gives an visualisation of these phases of a state.

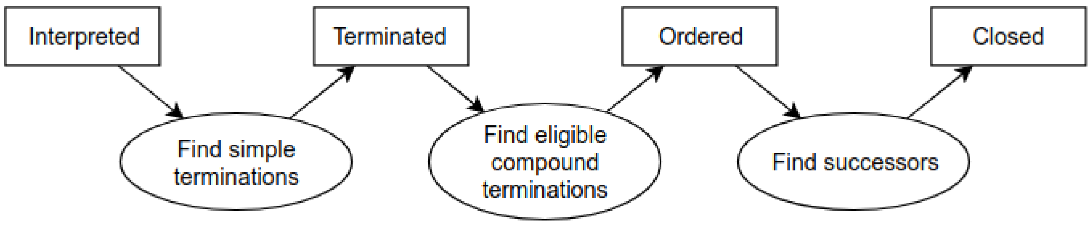


Figure 2.12: The phases of a state in the state-graph generation process

Each of the three phases has its own procedure. Firstly, in order to identify the simple terminations, and hence change the phase of the current state from interpreted to terminated, the rules shown in Table 2.2 are used. Secondly, the state becomes ordered by finding combinations of simple terminations that might lead to valid successors. More precisely, a series of rules that will be later introduced is used to early identify incompatible combinations of simple terminations, and hence reduce the number of eligible compound terminations. Lastly, the state becomes closed. This is the most complex phase in terms of the number of steps involved, as it contains continuity checks, the identification of the new applicable model fragments, and, in case the current compound termination results in a successor, a check to see if that state already exists in the graph. From the perspective of our axiomatisation, for each compound termination T, in this phase, the engine first finds the transition scenario from s with T. Then it applies the continuity criterion, and stabilises the state.

Two particular procedures used within these phases are worth mentioning in the context of this thesis. One is model fragment selection, which does repeated checks for finding active model fragments (Bredeweg et al., 2009). The other one is inequality reasoning, which verifies the consistency of a scenario with a given model fragment function. The latter one will be further discussed at the end of this section.

From the three phases, a closer examination is only needed for the ordering one. The termination phase is already clarified due to the axiomatisation from Section 2.1. Regarding the closing phase, for the purpose of this thesis there are only two important aspects to mention. First, it requires at least two calls to the inequality reasoner for each eligible compound termination. Second, given a transition scenario and a compound termination, it gives a continuous and stable extension of the scenario with the termination, i.e. it finds valid successors.

Having provided a general overview of the reasoning engine, the remaining two subsections focus on further discussing some of the procedures involved, namely inequality reasoning and state ordering.

2.3.2 Inequality Reasoner

A key component of the engine is the inequality reasoner. This is a procedure that given a set of relations, S, together with an extra relation r, tries to add the relation r to the set S. There are three possible answer based on the compatibility between r and S. First, if S and r form an inconsistent system, then r cannot be added to the system. Second, if r can be inferred from S, then it is deducible, so there is no need to add it to the system. Third, if r is consistent with S but not inferable, then it is added to S. For the purpose of this thesis, the inequality reasoner will be only used to check for consistencies of relation sets. Obviously, this can be done. To check if a set S is consistent, S together with a tautology, such as $0 = 0$ is passed to the inequality reasoner. Then S is consistent if and only if $0 = 0$ is found to be deducible.

Since, at an internal level, the information in models and scenarios can be represented by qualitative systems of (in)equalities, the inequality reasoning procedure is essential. It is always used at least twice in the closing of a state for each eligible compound termination. In addition, it is occasionally also used in the ordering phase, as will be discussed in the next subsection.

Even though extremely useful, inequality reasoning is computationally expensive. This is because the detection of all implicit contradictions requires the full transitive closure of the internal system of (in)equalities, together with addition and subtraction inferences (Bredeweg et al., 2009). In fact, according to Veber et al. (2004), the problem of finding a solution for system of (in)equalities is **NP**-complete, as **SAT**, the satisfiability problem for sets of clauses, reduces to this inequality reasoning problem.

As a result, there is a link between better using sparse knowledge and improving the time complexity of the reasoning procedure. Since the closing of a state uses inequality reasoning, this is in turn also time expensive. Therefore, reducing the number of transition scenarios requiring closing is useful. Hence, lowering the number of eligible compound terminations found in the ordering phase corresponds to lowering the time required by the simulation.

In the next subsection, some methods for reducing eligible combinations currently in use are presented.

2.3.3 Ordering a State

Within this subsection, it is assumed that there is a *terminated state*, that is a state together with a set of its simple terminations, which will be denoted by $(s = \langle \mathcal{E}, val, <, \leq, = \rangle, \mathcal{T})$. Since the total number of compound termination is exponential in the number of simple terminations, $\mathcal{O}\left(2^{|\mathcal{T}|}\right)$, the engine cannot analyse all the possible combinations. Hence, some rules must be used in order to lower this amount. The aim of this phase is to find a short-list of eligible compound terminations. In the current subsection, the approach of Garp3 (Bredeweg et al., 2009) is presented.

To form this short-list, the epsilon ordering rule is firstly used. As explained in the previous section, the set of terminations $\mathcal{T}$ is partitioned in the class of immediate and non-immediate terminations. If the first one is non-empty, then all the compound terminations must be a subset of it, while if it is empty, the compound terminations are formed from non-immediate simple ones.

Secondly, three more rules are used in order to infer constants on the termination combinations. In general, these rules will be referred to as *combination concepts*. A *combination concept function*, for a given state and model fragment function, is a function, *comb*, that, given a set of simple terminations, say S, returns a set of *combination constraints*, that is of logical relations that constrain the possible ways of combining the terminations in S. One example of such a rule is $t_1 \in T \leftrightarrow t_2 \in T$, which says that t_1 and t_2 must co-occur. This will be abbreviated as $t_1 \leftrightarrow t_2$ Another example is when t_1 and t_2 are incompatible, which is given by the relation $t_1 \notin T \vee t_2 \notin T$, and is abbreviated as $\neg t_1 \vee \neg t_2$.

One combination concept is the *mutually exclusive terminations constraint.* This rule identifies terminations that might not appear together, which may happen in the case of terminations with assumed causes. For example, if a bid difference is at *Epsilon* with an unknown derivative, then there could be assumed terminations that either change it to *Small* or *Large*, and these two cannot be combined. That is, combinations of terminations with causes such as $(\mathit{assumed\_to\_interval\_above}, Q)$ and $(\mathit{assumed\_to\_interval\_below}, Q)$, are not allowed.

Another combination concept is the *correspondence ordering.* When there are correspondences between values within the magnitude spaces of two parameters, these two parameters do not change values independently. To explain this, let us suppose that within the correspondence set from the causal ingredients of s there is the pair $((P_1, v_1), (P_2, v_2))$ for some parameters P_1, P_2, and values v_1, v_2 within their magnitude spaces, $v_i \in M_{P_i}$. As a reminder, this means that whenever P_1 has value v_1, P_2 must have the corresponding value v_2. Suppose moreover that the value assertion of s, that is *val*, is such that $val(P_1) = v_1$, and $val(P_2) = v_2$. Then, if there is no simple termination changing the value of P_1, either above or below, then any simple termination changing the value of P_2, if such exists, must not occur in eligible compound terminations. In other words, if $\mathcal{T}$ contains a termination changing the value of P_2 but none changing the value of P_1, then the first one cannot be used in any combination. This is because P_1 cannot change value, so it will remain at v_1, and by the value correspondence, P_2 should not change either from v_2.

Table 2.3 presents an overview of the rules used in the correspondence ordering. As in the example before, those are under the assumption of an existing correspondence $((P_1, v_1), (P_2, v_2))$. This table analyses the cases when the two parameters are both at the corresponding values, both not at the corresponding values, and when only P_2 is at the corresponding value v_2. The fourth case, when only P_1 is at the corresponding value, that is when $val(P_1) = v_1$ and $val(P_2) \neq v_2$, cannot appear, as it does not satisfy the correspondence constraint, so s is not stable, and hence it cannot be a state.

The last combination concept in use is the *mathematical ordering.* This is a rule that checks if combinations of changes can occur from the perspective of relation-correctness. More precisely, if we have two parameters, P_1, P_2, and three terminations, a value t_i termination for each P_i and an inequality termination t_3 for the pair (P_1, P_2), then there are certain bounds on how these terminations can appear together. For instance, if currently $P_1 = P_2$, then a change of inequality can only appear together with at least a change in the values of the two parameters, i.e. $t_3 \rightarrow t_1 \vee t_2$.

To apply this, the engine takes every pair of quantities that have a specified relation between them and that is still valid under the previous two combination concepts. Since this step is computationally demanding, it is actually important to be the last one that is checked, in order to have the least amount of valid combinations as input. In addition, every considered pair needs to have inequality constraints on the landmarks that could be reached through combinations. Finally, the inequality reasoner is used to check which combinations of terminations are valid.

Lastly, after using all these three combination concepts, the *constraint cross product* of the simple terminations is taken. This means that the engine finds the subset of the cross product such that all the compound terminations in this subset satisfy the conditions returned by the combination concept function, that is the conditions within $comb(\mathcal{T})$.

As a final mention, an additional step needs to be taken in the case of immediate terminations. Due to their nature, immediate terminations must always occur together. So, if the set of immediate terminations is non-empty, then the

Value assertion	Combination constraint	Description
$val(P_1) = v_1$, $val(P_2) = v_2$	$t_2 \to t_1$	If P_1 and P_2 are both at the corresponding values, then P_2 cannot change value alone. In the particular case when there are no terminations changing the value of P_1, then P_2 cannot change at all, so any value termination on P_2 can be eliminated.
$val(P_1) = v_1'$, $val(P_2) = v_2'$	$t_1 \to t_2$	If P_1 and P_2 are both at adjacent values to the corresponding ones, then P_1 cannot move alone to the corresponding value. Similarly, when there are no terminations changing the value of P_2, then P_1 cannot change at all, so any value termination on P_1 can be eliminated.
$val(P_1) = v_1'$, $val(P_2) = v_2$	$\neg t_1 \lor \neg t_2$	If P_2 is at the corresponding value, while P_1 is at an adjacent one, P_1 cannot change to the corresponding value, while P_2 moves away from it.

Table 2.3: Correspondence ordering constrains for parameters P_1, P_2, where $v_1 \in M_{P_1}$ corresponds to $v_2 \in M_{P_2}$. Each v_i' is a value within M_{P_i} that is adjacent to v_i (either smaller or larger), and each t_i is a termination on P_i with value or derivative causes that change the parameters either from v_i to v_i', or the other way around.

constraint cross product is further reduced by eliminating all the compound terminations that are a subset of another compound termination.

This finalises the explanation of how the eligible compound terminations are selected in Garp3. The next section analyses the impact of using these combination concepts by considering a large simulation example.

2.4 A Large Simulation

In this section, a large scientific model is discussed as an example of how the engine functions and how important each phase of the state is in the simulation. For this purpose, the model of cellulose hydrolysis, which was introduced by Kansou et al. (2017), is used. Explaining the motivation behind the model is outside the scope of this thesis. Instead, this section analyses the first few steps in the simulation of this model. The most attention is paid to state 3, as this is also used as an example at the end of Chapter 5.

Quantity	Magnitude	Derivative
$Concentration_{Activenzyme}$	*Plus*	+
$Concentration_{Freeenzyme}$	*Plus*	+
$Concentration_{Product}$	*Plus*	+
$Concentration_{Substrat}$	*Max*	0
$Concentration_{Surfaceenzyme}$	*Plus*	+
$Quant\ enzyme_{Enzyme}$	*Total*	0
$Quantity\ of\ submol_{Sugar}$	*Total*	0
$Rate\ in_{Additionenzyme}$	*Plus*	−
$Rateads_{Surfaceenzyme}$	*Plus*	+
$Ratecat_{Activenzyme}$	*Plus*	+
$Ratecomp_{Activenzyme}$	*Plus*	+
$Ratedes_{Freeenzyme}$	*Plus*	+
$Rateoff_{Activenzyme}$	*Plus*	+
$Surface\ available_{Substrat}$	*Plus*	−
$Surface\ covered_{Substrat}$	*Plus*	+
$Surface\ max_{Substrat}$	*Max*	0

Table 2.4: Cellulose Hydrolysis: values of quantities in state 3.

As an initial scenario, the *basic enzymatic reaction restart* is considered. The resulting state graph has 83 states, and takes almost two days to be fully computed. [1] For the first 14 states of this simulation, we used the profiler procedure in SWIProlog (Graham et al., 2004) to record the CPU times for each phase in simulating in the state. As expected from the theoretical complexity results, for all these states the most time was spent in the closing of a state; while the terminating and ordering phases were always carried out in less than 1 minute, some closing of states took almost 1 hour. For instance, one state with only two valid successors but 255 compound terminations took 34 minutes to close. In addition, also in accordance with the theoretical results, most time was spent in the inequality reasoner (more than 80%). This suggests that, for practical reasons as

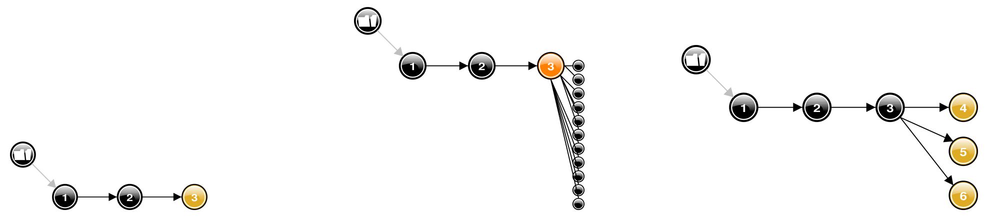
[1]These results were obtained on an Intel Core i5 1.7GHz processor, with 4 GB of RAM machine running Ubuntu 16.04.

Label	Simple termination cause
1	$(to\_point\_below, Surface\ available_{Substrat})$
2	$(to\_point\_above, Concentration_{Surfaceenzyme})$
3	$(to\_point\_above, Surface\ covered_{Substrat})$
4	$(to\_point\_above, Concentration_{Product})$
5	$(to\_point\_above, Concentration_{Freeenzyme})$
6	$(to\_point\_below, Rate\ in_{Additionenzyme})$
7	$(to\_point\_above, Concentration_{Activenzyme})$
8	$(assumed\_derivative\_up\_to\_stable, Concentration_{Surfaceenzyme})$
9	$(assumed\_derivative\_up\_to\_stable, Concentration_{Freeenzyme})$
10	$(assumed\_derivative\_up\_to\_stable, Concentration_{Activenzyme})$
11	$(assumed\_derivative\_up\_to\_stable, Rateads_{Surfaceenzyme})$

Table 2.5: Cellulose Hydrolysis: simple terminations for state 3.

well, it would be useful to lower the number of calls to the inequality reasoner.

Let us now consider a particular state, which has valuation in accordance with Table 2.4. From this, 11 simple terminations are found (see Table 2.5). One can check that these are in accordance with the termination validity criteria from Table 2.2. The number of eligible compound terminations identified is 511. Notice that this is reduced as, by combining all the simple terminations, $2^{11} - 1 = 2047$ compound terminations would have been found. Figure 2.13 shows the development of the state graph throughout these phases.



(a) The extension of the initial scenario. (b) The simple terminations. (c) The successors.

Figure 2.13: Cellulose Hydrolysis: the stepwise identifications for the successors of state 3.

2.5 Conclusion

To summarise, this chapter made four contributions. First, and most importantly, it has introduced a new axiomatisation that focuses more on the simulation aspects than the previous formalisation attempts (Bredeweg et al., 2009; Liem, 2013; Weld, 1988). It used clear notations and definitions for simple terminations, which are possible next-state developments of a system, and compound terminations, which correspond to simultaneous developments. This leads to clear definitions for the simulation output, i.e. state graph, which is the overview of the possible future behaviours of the system. All those definitions are indispensable in order to perform a theoretical analysis of the process-oriented qualitative simulations, in their current form. Second, since this is done in the context of the earlier formalisations, this chapter also serves as a literature review.

Third, for clarifying those definitions, the continuous English Auction with two bidders model was used. This is a novel example extending the applications of qualitative simulations to the field of Auction Theory. The results of the simulation were in accordance with the theoretical ones in the field, hence implying the correctness of our model.

Fourth, Garp3 (Bredeweg et al., 2009) was used as example of a qualitative reasoning engine. Its inner workings were illustrated by means of a large example of cellulose hydrolysis (Kansou et al., 2017). Three important phases were identified for obtaining successors, namely termination, ordering, and closing. Also, a very important procedure within simulation is the inequality reasoner, which checks the consistency of systems of qualitative (in)equalities. This was shown by Veber et al. (2004) to be **NP**-complete. The brief practical case analysis carried out in the last section revealed that this theoretical result correlates with slow running times. As a result, since the closing phase requires multiple calls to the inequality reasoner, the number of compound terminations requiring analysis during closing should be lowered. This can be done by better using sparse knowledge for early identification of incompatible compound terminations.

In the next chapter, the problem of whether sparse knowledge leads to incoherent behaviours is investigated. The axiomatisation build in this chapter proves useful in identifying and analysing different situations that could result in not meeting consistency principles.

Chapter 3
Criteria for Simulation Consistency

As already mentioned in the previous chapter, data is sparse in qualitative models, in the sense that not all valid relations in the model are made explicit. Consequently, the simulation is based on incomplete information, leading in some cases to incoherent results. The goal of this chapter is to use the axiomatisation created before in order to identify incoherences, as well as to find solutions for eliminating them.

Three principles of coherence are discussed. The first section investigates model consistency, that is whether transitions in simulation results are in accordance with the constraints of the system. The second section considers paths in the resulting state graph. As a principle, each state graph should be path consistent, meaning that the information recorded in successions of states should describe a coherent behaviour. The last section discusses consistency from the perspective of inexplicit (in)equalities, as, even when relations are not explicit but only inferable from either a scenario or a model fragment, the state graph should still be consistent with them.

3.1 Model Consistency

Every system is described qualitatively by its model fragments. Therefore, in order to have coherent results, the simulation result should be in accordance with the model ingredients. This principle is reffered to by the term *model consistency.* Depending on which type of ingredients are considered, this principle branches out into two parts. Firstly, the simulation result should respect the information provided by the dependency relations, that is it should be *dependency consistent.* Secondly, it should be in accordance with the relations in the model, that is it should be *relation consistent.* In this subsection, we discuss each of these two principles and investigate the extent to which the current procedures in qualitative reasoning respect *model consistency.*

3.1.1 Dependency Consistency

As already mentioned, dependency consistency is the principle which states that each transition should be in accordance with the dependency relations given by the model fragment function, that is with the influences and proportionalities. Because each state is stable, it is clear that the influence and proportionality balances are respected. However, dependencies could provide other information as well.

To see this, first consider an example. Suppose there is a meeting room in which people enter via an entrance corridor, and from which they exit via an exit corridor. In terms of qualitative modelling, this means that we have the entity room with three quantities: the number of people in the room, to which we will refer to as *No_room*, the number of people in the entrance corridor, *No_entrance*, and the number of people in the exit one, *No_exit*. In addition, there is a positive influence from *No_entrance* to *No_room*, as if there are people at the entrance, then they will go in the room, so the number of people in the room will increase (if none of them are exiting). Similarly, there is a negative influence from *No_exit* to *No_room*.

The model fragment for the system described above is shown in Figure 3.1. The conditionals, which, in this case, are the entity *Room* together with its three quantities, are drawn in red. The causal ingredients returned by the model fragment, namely the two inferences, are pictured in blue.

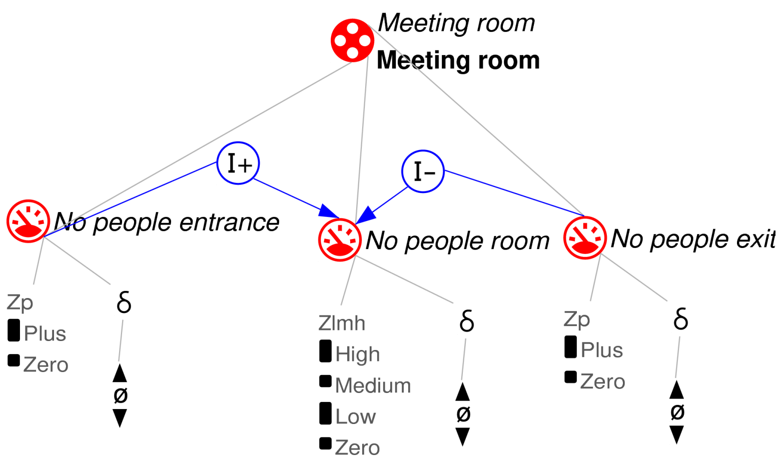


Figure 3.1: The model fragment for a room with entrance and exit corridors.

Given this system, the following initial scenario is considered. Suppose that at a certain meeting there is a medium number of people in the room, and the number of people at the entrance is constant (people are coming in at a constant pace). Suppose moreover that more and more people in the room decide that the meeting is not worth their time, so the number of people in the exit hall increases. Hence, we have the scenario from Figure 3.2.

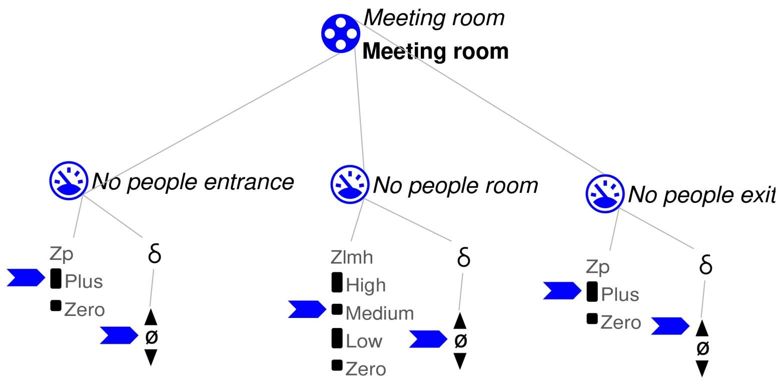


Figure 3.2: An initial scenario for the meeting room model.

Next, the evolution of this situation is analysed. To begin with, according to the scenario, the tendency of the number of people in the room is completely balanced at 0 by the ones entering and the ones exiting. So, $dNo\_room = f(No\_entrance) - g(No\_exit)$. Since $dNo\_room = 0$, then we should have $f(No\_entrance) = g(No\_exit)$. Moreover, as the derivative of No_exit is positive and g is strictly monotonous, $g(No\_exit)$ is increasing, so we will immediately have in the next state $f(No\_entrance) < g(No\_exit)$, and, hence, $dNo\_room < 0$. Therefore, we would expect an unique successor of the state extending the given scenario, which would be the same as the current state, but with $dNo\_room = -$. However, without considering the second order derivatives, this is not the case, because for each of the three values of dNo_room, the resulting state is stable and continuous, but neither is justified by an simple termination according to Table 2.2. This can also be exemplified by running this simulation in Garp3 with the default training preferences. According to this, no successors of state 1, the state extending the initial scenario from Figure 3.2, can be found.

This is a general issue extending beyond our example. Such a problem might appear whenever there is a quantity with at least two incoming dependencies. One way of fixing it is to consider second order derivatives. To understand why doing so solves the issue, we return to the example before. Now, when extending the given scenario, the second order derivatives of the three quantities will also be identified. Hence, $ddNo\_room = df(No\_entrance) - dg(No\_exit)$ is deduced since $dNo\_room = f(No\_entrance) - g(No\_exit)$. Because f and g are continuous and strictly monotonous, $ddNo\_room = 0 - minus = minus$. As now the derivative of dNo_room is negative, this will trigger only one derivative termination on No_room, changing its value from 0 to $-$. Even though using second order derivatives produces the expected output, this will mean that each quantity would have an additional parameter. This increases the number of parameters by 50%, hence requiring more analysis when, in fact, these might only be needed for one case. Moreover, doing this just moves the issue onto second order derivatives, so, it is not entirely solved. In Garp3, this solution is implemented by customising the simulation preferences.

Another way of producing the expected behaviour would be to add extra causes for terminations based on the model fragment function. More precisely, for a state $s = \langle \mathcal{E}, val, <, \leq, = \rangle$, the Table 2.1 is extended with the four causes in Table 3.1. As before, the starred causes are for immediate terminations. The associated sets of constraints and results set are presented in Table 3.2.

Class of causes	Type of cause
Model consistency causes	$(mc\_derivative\_stable\_to\_down, P)^*$ $(mc\_derivative\_stable\_to\_up, P)*$ $(mc\_derivative\_down\_to\_stable, P)$ $(mc\_derivative\_up\_to\_stable, P)$

Table 3.1: Model consistency causes

For clarity purposes, let us now explain the second validity criteria in Table 3.2. Because of the inference and proportionality balance, we have that

$$dP = \sum_{P_i \in Pr_+} f_i(dP_i) - \sum_{P_j \in Pr_-} f_j(dP_j) + \sum_{P_k \in In_+} f_k(P_k) - \sum_{P_u \in In_-} f_u(P_u),$$

where $Pr_+ = \{P_i | (P_i, P) \in P_+\}$, $Pr_- = \{P_j | (P_j, P) \in P_-\}$, $In_+ = \{P_k | (P_k, P) \in I_+\}$, and $In_- = \{P_u | (P_u, P) \in I_-\}$. So if all dP_i, P_k are steady or increasing, all dP_j, P_u are steady or decreasing, and at least one of them all have the tendency to change, then P should immediately have the tendency to increase. That is, there is a termination with $dP = +$.

In the meeting room example, there are two influences to No_room. The positive one comes from a steady quantity, while the negative one comes from an increasing quantity. This state description matches the constraints of $(mc\_derivative\_stable\_to\_down, No\_room)$, which is an immediate termination changing the value of dNo_room to $-$.

To summarise, originally the principle of dependency consistency is not met without considering higher order derivatives, even though there is sufficient information in the model. This situation was firstly exemplified using the meeting room model. Afterwards, it was generalised to any quantity with at least two incoming dependencies. A solution for this general formulation of the problem was then provided. More precisely, a new class of causes was introduced that accounts for the changes determined by information deducible from dependencies. The next section considers the coherence of the system from the point of view of relations.

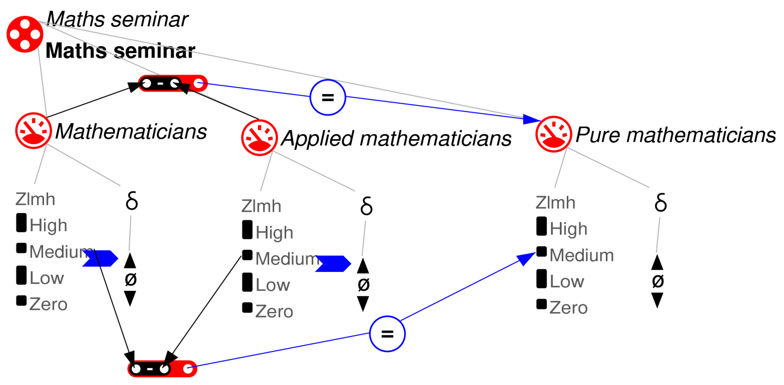
Cause type	Elements of $cond(t)$	Elements of results set
$(mc\_derivative\_stable\_to\_down, P)$	$val(dP) = 0$ $\forall (P', P) \in P_+ \ val(ddP') \leq 0$ defined $\forall (P', P) \in P_- \ val(ddP') \geq 0$ defined $\forall (P', P) \in I_+ \ val(dP') \leq 0$ defined $\forall (P', P) \in I_- \ val(dP') \geq 0$ defined at least one of the inequalities is strict	$val(dP) = -$
$(mc\_derivative\_stable\_to\_up, P)$	$val(dP) = 0$ $\forall (P', P) \in P_+ \ val(ddP') \geq 0$ defined $\forall (P', P) \in P_- \ val(ddP') \leq 0$ defined $\forall (P', P) \in I_+ \ val(dP') \geq 0$ defined $\forall (P', P) \in I_- \ val(dP') \leq 0$ defined at least one of the inequalities is strict	$val(dP) = +$
$(mc\_derivative\_down\_to\_stable, P)$	$val(dP) = -$ $\forall (P', P) \in P_+ \ val(ddP') \geq 0$ defined $\forall (P', P) \in P_- \ val(ddP') \leq 0$ defined $\forall (P', P) \in I_+ \ val(dP') \geq 0$ defined $\forall (P', P) \in I_- \ val(dP') \leq 0$ defined at least one of the inequalities is strict	$val(dP) = 0$
$(mc\_derivative\_up\_to\_stable, P)$	$val(dP) = +$ $\forall (P', P) \in P_+ \ val(ddP') \leq 0$ defined $\forall (P', P) \in P_- \ val(ddP') \geq 0$ defined $\forall (P', P) \in I_+ \ val(dP') \leq 0$ defined $\forall (P', P) \in I_- \ val(dP') \geq 0$ defined at least one of the inequalities is strict	$val(dP) = 0$

Table 3.2: Termination validity criteria for model consistency causes for state s with active causal ingredients P_+, P_-, I_+, I_-.

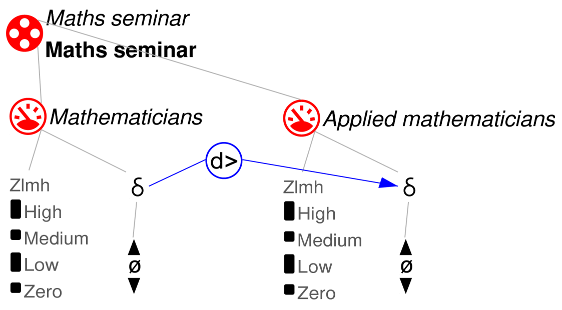
3.1.2 Relation Consistency

Besides dependencies, other model fragment ingredients that should be respected are the relations between parameters. The ones that are made explicit in the system are trivially considered, as it is a requirement for the stability of states. However, the relations that are inferable from the explicit ones by taking derivatives are not necessarily considered. In general, if there is an equality between terms, then the relation obtained by taking derivatives should hold as well.

For an example, consider the case of a mathematics seminar where both pure and applied mathematicians are coming. So, the number of pure mathematicians equals the total number of mathematicians minus the number of applied ones. This, together with the analogous condition on the medium values of each quantity, are pictured in Figure 3.3a, i.e. $No\_pure = No\_math - No\_applied$. In addition, it is assumed that this seminar is more popular among pure mathematicians, in the sense that the derivative of the number of mathematicians is always greater than the derivative of the number of applied ones, as shown in Figure 3.3b. That is, $dNo\_math > dNo\_applied$.



(a) The model fragment giving equality constraints.



(b) The model fragment showing derivative constraints.

Figure 3.3: The two model fragments of the mathematical seminar example.

This model fragment information completely determines the value derivative for the number of pure mathematicians. Because of the equality between quantities, by taking derivatives, it can be inferred that $dNo\_pure = dNo\_math - dNo\_applied$. Since, moreover $dNo\_math > dNo\_applied$, $dNo\_pure > 0$ should be inferred. However, if, for instance, initially all the values are low and all the three quantities are increasing, as in Figure 3.5, $dNo\_pure > 0$ is not inferred. One path in the resulting state-graph is shown in Figure 3.5. According to that path, the number of pure mathematicians remains at medium, with an unknown derivative.

The solution for this issue is to extend the set of equality relations with the corresponding derivative ones. For Garp3, doing so will surely solve the issue, as by explicitly adding the constraint on derivatives in Figure 3.3b, this issue is

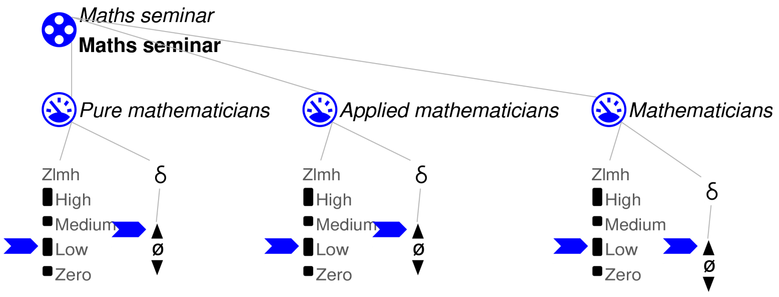


Figure 3.4: Initial scenario for the mathematical seminar.

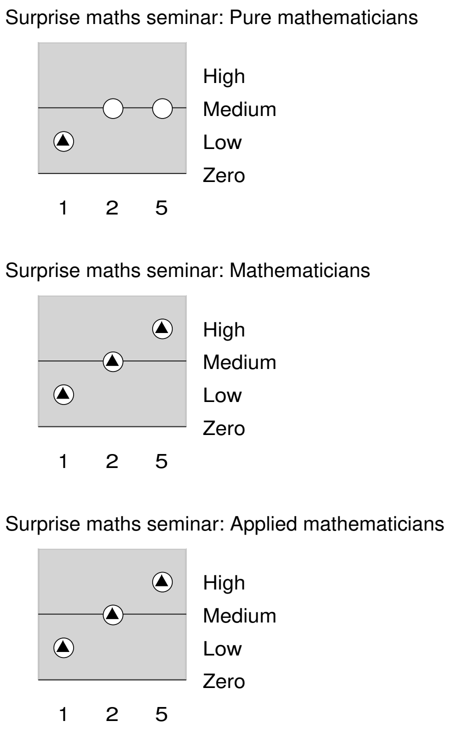


Figure 3.5: Result of the simulation of the mathematical seminar.

solved. At a higher level, this issue could be solved by updating the inequality reasoner with the derivative taking rule.

This finalises our discussion on relation consistency. In the next section, the focus is moved from individual transitions, to sequences of transitions.

3.2 Path Consistency

The coherence of all transitions is not enough to ensure the coherence of the entire state graph. This is because each state only provides partial information of the system at a particular time. Therefore, there could still be situations with tracks that do not describe a valid development of the system. We call the principle which states that each track contains a correct evolution of the system *path consistency.* The purpose of this section is to check whether path consistency holds according to our axiomatisation. In order to do this, the next two sections analyse two aspects of this principle.

3.2.1 Assumption Consistency

During a simulation, some states are consistent only due to some assumptions about the system. However, these assumptions are not recorded in state's representation. Therefore, there might be cases when contradictory assumptions are made within the same path. This obviously leads to incoherent tracks. Hence, we introduce the *assumption consistency* principle which states that once an assumption or inference is made within a path, all the remaining states in the path must be in accordance with it.

One example of assumption inconsistency comes from inferred inequalities between landmarks. Considering the example of population dynamics, assume that in some state the number of births equals the number of deaths, and that both of them are at their medium value, as in Figure 3.6. It should then be the case that in any future state developed from this one, if the two quantities have the medium value again, they should be considered equal. That is, for any track containing this state, two medium points should be equal in any later state.

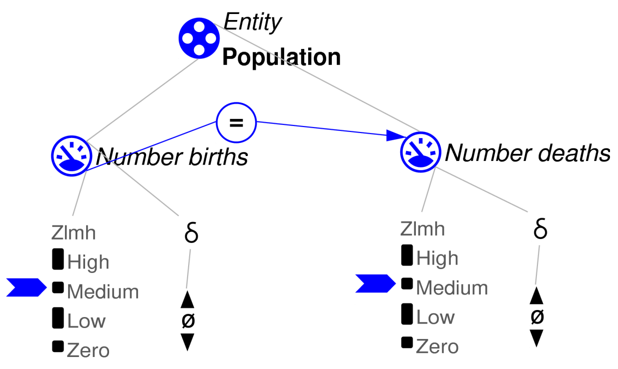


Figure 3.6: A state where the number of deaths and the one of births are equal. Both of them have the medium magnitude.

The approach of Garp3 already gives a solution to this issue. Using one of the optional simulation preferences, namely the "Constrain interaction between possible worlds (derive landmark relations)", the engine considers the previously inferred landmark relations in order to ensure consistency within the state-graph from a parameter inequality perspective. However, this is not sufficient for the simulation to make sense. There might still be incoherent behaviour between states within the same path, as we will see in the next example.

As in Section 3.1, we consider the example of the meeting room, this time with more detailed quantity spaces for the number of people in the entrance and in the exit. More precisely, this time we have the model fragment in Figure 3.7. Each of the quantities can take values from the magnitude space $\{0, Low, Medium, High\}$. The initial scenario is the one from Figure 3.8: all quantities are at their medium value, with the number of people in the room and in the entrance being constant, and the number of people in the exit increasing.

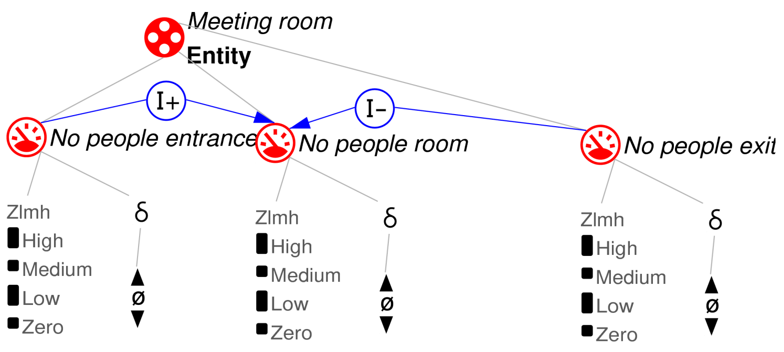


Figure 3.7: The model fragment for a meeting room with entrance and exit corridors.

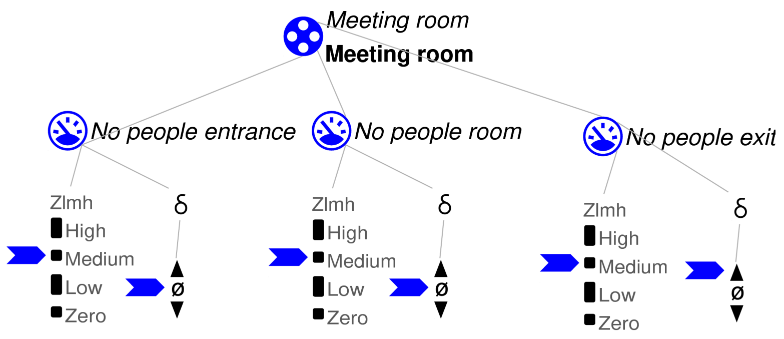


Figure 3.8: An initial scenario for a meeting room with entrance and exit corridors.

As before, it should be inferred that as long as both quantities are at the medium value, the entrance and the exit balance each other out, resulting in the number of people in the room remaining steady. Moreover, if the number of people in the exit is greater than the medium, then they should outnumber the ones in the entrance, so the number of people in the room should be decreasing. However, this is not the case. As shown in Figure 3.9, even though the number of people in the entrance remains medium, and the number of people in the exit remains high, the number of people in the room does not necessarily decrease. Instead, this number may have any possible derivative. Remember that, in the figure, the position of a circle shows the value of the quantity, while the drawing within it represents the derivative. For example, the number of people in the room is decreasing in state 5, is steady in state 7, and is increasing in state 8. Its magnitude in the first state is *Medium*, while in the last two states is 0.

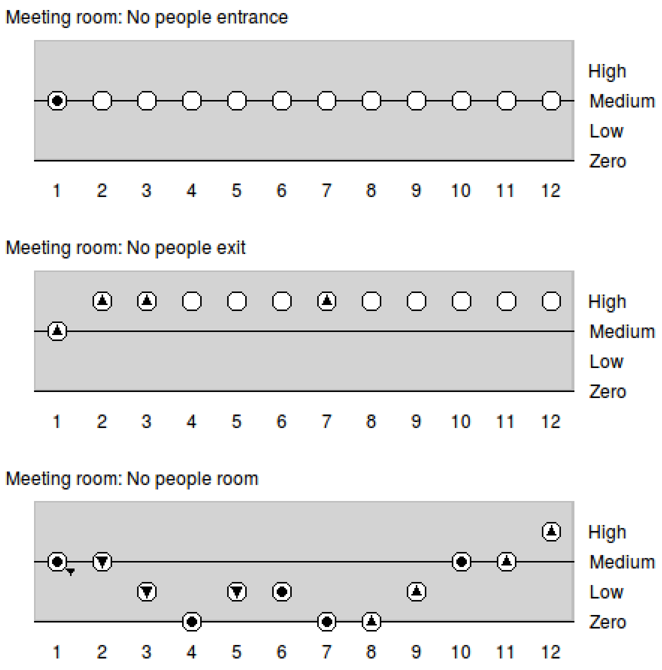


Figure 3.9: Value assertion for the states within the state graphs of the meeting room example.

The reason for this behaviour is that the engine does not consider how strong the dependency is. This could be added by associating a quantity with the same magnitude space as the influencing quantity or its derivative to each influence and proportionality pair. Moreover, since the influences and proportionalities are associated with strict monotonous functions with a fixed point at zero, the two quantity spaces must be in full correspondence. To give an example, assume $(Q, Q') \in I$, so there is an associated quantity, $I^+_{(Q,Q')}$, with magnitude space M_Q. Since it is an influence relation, it means that there exists a strictly monotonous function f such that Q contributes to the derivative of Q' with $f(Q)$. Therefore, it has a contribution of zero if Q is zero, it has a medium one if Q is at medium and so on. This is why Q and $I^+_{(Q,Q')}$ should have the same landmarks, which should also correspond to each other.

Formally, we modify the model fragment function. That is, if $mf(Cond)$ currently gives the causal ingredients P_+, P_-, I_+, I_-, and C, for each dependency we will add a new quantity, and new correspondences to C. Table 3.3 outlines the changes to the model fragment function.

Since within the model the influence and proportionality balances must hold,

Dependency	Quantity name	Magnitude space	Addition to C
$(Q, Q') \in I_+$	$I^+_{(Q,Q')}$	M_Q	$((Q, I^+_{(Q,Q')}), (l, l)), \forall l \in M_Q$
$(Q, Q') \in I_-$	$I^-_{(Q,Q')}$	M_Q	$((Q, I^-_{(Q,Q')}), (l, l)), \forall l \in M_Q$
$(Q, Q') \in P_+$	$P^+_{(Q,Q')}$	M_{dQ}	$((Q, P^+_{(Q,Q')}), (l, l)), \forall l \in M_Q$
$(Q, Q') \in P_-$	$P^-_{(Q,Q')}$	M_{dQ}	$((Q, P^-_{(Q,Q')}), (l, l)), \forall l \in M_Q$

Table 3.3: Introducing dependency quantities: the changes in the result of $mf(Cond)$, where P_+, P_-, I_+, I_-, and C are its current active causal ingredients, and $\mathcal{E}$ is the set of entities from its returned conditionals. The new quantity is added under a new entity named *Dependency*.

we should also add relations such that

$$dQ = \sum I^+_{(Q,Q')} - \sum I^-_{(Q,Q')} + \sum P^+_{(Q,Q')} - \sum P^-_{(Q,Q')}.$$

These equations are also added to the conditionals of $mf(Cond)$.

The result of this change is that landmark relations for the contribution of dependencies are also tracked. Coming back to the meeting room example, the model fragment in 3.7 is completed with two dependency quantities, namely $I^+_{No\_entrance,No\_room}$ and $I^-_{No\_exit,No\_room}$, each having the magnitude space $\{0, Low, Medium, High\}$. Besides the respective correspondences, the relation $dQ = I^+_{No\_entrance,No\_room} - I^-_{No\_exit,No\_room}$ is added. In the initial scenario from Figure 3.8, the landmark relation $0 = Medium_{I^+_{No\_entrance,No\_room}} - Medium_{I^-_{No\_exit,No\_room}}$ is recorded. Therefore, under the simulation preference "Constrain interaction between possible worlds (derive landmark relations)", the engine will ensure track consistency. As a result, when the number of people in the exit is at high, and the one of the people in the entrance is at medium, the number of people in the room will be decreasing.

To summarise, in this section we introduced quantities for each dependency in a model fragment. By doing so, it was ensured that the relations between the amount of influence and proportionality are also tracked. As a result, the coherence within a path was improved upon by ensuring more assumptions are made explicit. In the next section, the paths describing behaviours with extremum values are considered.

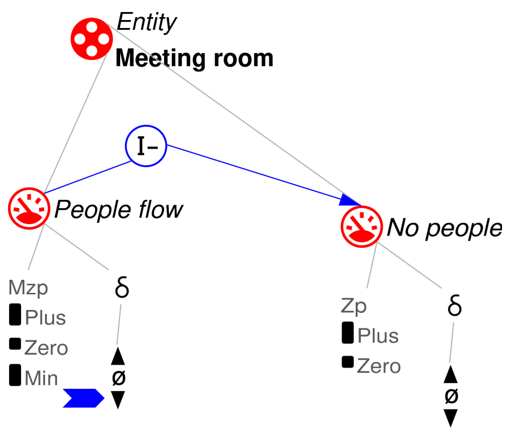
3.2.2 Extrema Consistency

Other information that can be deduced from a path, but is not encoded in a state, is the difference between an extremum and a plateau point. For example, if, when considering a track, a quantity has a negative derivative at some state s_1, then, in the successor of that state, s_2, the derivative becomes zero, and, immediately after, in s_3, it is positive, then state s_2 corresponds to an extremum point for the quantity in question. Differently, if in the successor of s_2 the derivative of the quantity remains steady, then s_2 is the start of a plateau interval. This distinction is extremely important as transitions from extrema should be immediate. That is, if there is some parameter at a landmark value with a tendency to change, and another parameter at an extrema, then those two quantities should change their value in the same state; contrarily, if the second parameter is at a plateau, then only the first one should change. We will refer to this principle as *extrema consistency.* In this subsection we investigate whether state-graphs are in accordance with this principle.

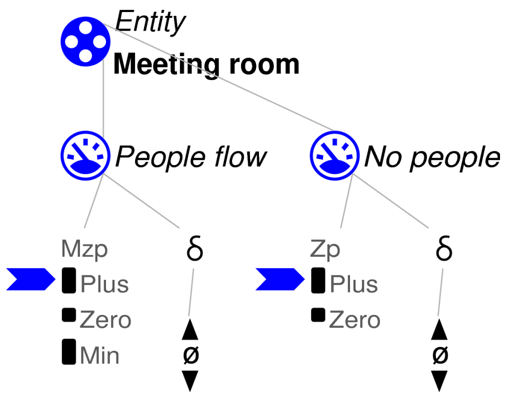
To start, let us investigate what is the path development in the case of an extremum point according to the axiomatisation in Chapter 2. Looking back at Table 2.2, it can be seen that a value and a derivative to-point simple termination of a quantity can co-occur within the same compound termination. That is, we can have a quantity transiting, for example, from value $i_{0,1}$ to l_1, and its derivative going from $+$ to 0. If the resulting state is an extremum, then its successor should have an opposite value for the derivative of the quantity and return to the previous value. However, according to the same table, the change in value is not possible, as the derivative is 0. Hence, a derivative change takes place instead, modifying the value of the derivative from 0 to $-$. This is problematic as, in the same transition, another termination from point to value can take place, resulting in an extrema inconsistent path.

To give an example, we once again model a meeting room, this time in the transition period between two consecutive meetings. Instead of considering the number of people in the exit and the entrance separately, there is a *Flow* quantity that gives the number of people in the exit minus the one in the entrance. So, there is a negative influence from *Flow* to *No_room*, i.e. the number of people in the room. Moreover, since this is a transition period between the meetings, the derivative of *Flow* should be negative, as initially people from the first meeting are exiting the room, but, afterwards, people starting the second meeting are entering. This behaviour is captured in the model fragment from Figure 3.10a.

For the initial scenario, we assume the number of people in the room and the flow are both positive, as in Figure 3.10b. According to the explanation before, an extrema inconsistent behaviour is expected within a path. By running the simulation in Garp3, this indeed turns out to be the case. As shown in Figure 3.11, there is a track with an extremum state, namely state 2, that continues with a value change on another quantity. More precisely, in state 2, both the flow and the derivative of the number of people in the room are zero, while immediately as the flow becomes negative, the derivative becomes positive, but the number of people in the room remains at zero. This leaves the impression that at some point the number of people in the room is zero and the flow is non-zero, which is



(a) The model fragment



(b) The initial scenario

Figure 3.10: A model fragment and a scenario for a meeting room transiting between two meetings.

never the case.

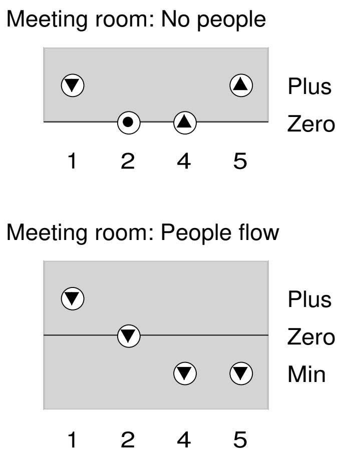


Figure 3.11: Value assertion for the states within the state graphs of the meeting room example transiting between two meetings.

To solve this, there are two possible approaches. One of them is to prioritise derivative terminations over value ones. This means that the epsilon ordering rule is enriched with another step that gives priority to terminations of parameters depending on the derivative order. For example, if there are from-point terminations that change the value of some quantity, the value of some derivative, and the value of a second order derivative, then the second order derivative change has the highest priority. By doing so, the track from Figure 3.11 would have a different state 4. More precisely, the number of people in the room would still be at zero and increasing, and the flow would still be decreasing, but this time with a zero value.

A more elaborate solution is to adapt the behaviour of derivatives. Instead of requiring a strict derivative inequality for a change in value, we could also allow for changes with zero derivative. This is in accordance with the mathematical form of derivatives, as, at any extremum, the derivative is zero, and this value changes only when the quantity moves away from the extremum and enters the interval. In the example before, this second solution eliminates state 4 entirely from the simulation.

This behaviour can be obtained with three changes. Firstly, for the to-point terminations, some constraints are added on the successor. More precisely, for each termination moving the value of some quantity Q to a landmark above (or below) and its derivative to 0, the constraint $val(dP) \leq 0$ (or $\geq$) is added to the successor. This means that if the derivative becomes zero, then, at the next state, it can either remain zero, that is to plateau, or go back to its value before, that is move away from the extremum. Secondly, Table 2.2 is adapted to allow value changes even under 0 derivative, as long as it is in accordance with the constraints. For example, the conditions for a termination with cause $(to\_interval\_above, Q)$ are $val(Q) = l_u$, and either $val(dQ) = +$, or $val(dQ) = 0$ and the constraint on the next state is $val(dQ) \geq 0$. Lastly, the results sets of the terminations should also be adapted. More precisely, any from-point change will have a strict condition on the value of the derivative. In the example before, instead of ensuring $val(dQ) \geq 0$, the required relation should be $val(dQ) > 0$.

3.3 Inexplicit Inequality Consistency

As mentioned in Chapter 2, each state has a set of entities, a valuation for the parameters of these entities, and (in)equality relations. However, not all (in)equalities inferable from the valuation are explicit in the relations within states. For example, if two quantities have the same value according to the valuation, the explicit relation that the valuation results at the two quantities are equal may not be in the state. If a state graph is also in accordance with the inexplicit inequalities we will say that it is *inexplicit inequality consistent*. This section investigates conditions for inexplicit inequality inconsistency within qualitative simulations, and suggests solutions for obtaining coherent behaviour.

To analyse this problem, let us first consider an example of inexplicit inequality inconsistency. Assume two runners are competing in a race. One of them is always faster than the other. To model this, there are two entities *Fast_Runner* and *Slow_Runner*, each of them with an associated progress quantity, and the consequence returned by the model fragment functions is $dProgress_{Fast\_Runner} > dProgress_{Slow\_Runner}$. This is shown in Figure 3.12. Moreover, they both compete on the same track, so the magnitude spaces of the two progresses quantities are equal. In Garp3, it is not necessary to make this explicit in the model, as there is a simulation preference, namely “Assume equal quantity spaces have equal points”, that does precisely this.

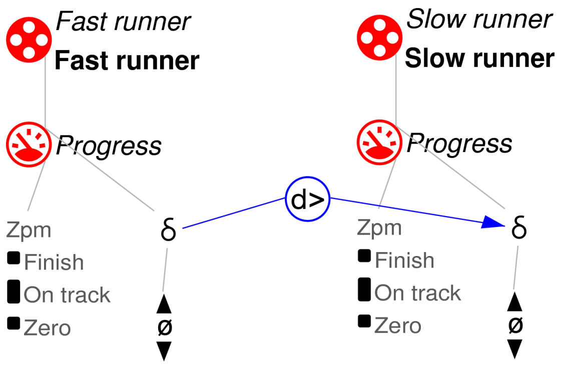


Figure 3.12: The model fragment for two competing runners, when one is faster than the other.

For the initial scenario, assume that the two runners start from the beginning of the track, that they have a progress equal to 0, but they have the tendency to increase it. With the formal notation, this means that in the first state $val(Progress_{Fast\_Runner}) = val(Progress_{Slow\_Runner}) = 0$, and moreover that $val(dProgress_{Fast\_Runner}) = val(dProgress_{Fast\_Runner}) = +$. The diagram for this scenario can be found in Figure 3.13. The expected simulation result is that the first runner reaches the end of the race, i.e. achieves maximum progress, first. However, this is not the case.

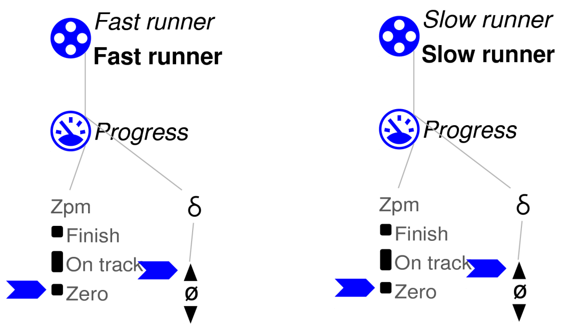


Figure 3.13: Initial scenario for two competing runners.

During simulation, since the initial equality between the two runners is not made explicit, the relation between the two quantities is not tracked. Therefore, the successor of the initial state has both runners with a positive progress, but with an unknown inequality between them, when in fact it should be inferred that the progress of the fast runner is greater than the one of the slow runner. In the case of adding the equality between value spaces to the initial scenario in Figure 3.13, then, according to Table 2.2, there would be an inequality termination on the added equality, and the correct behaviour would be inferred. This can be checked by performing the simulation in Garp3. Figure 3.14 shows a path in the resulting state-graph according to which the slow runner achieves maximum progress first, hence winning the race.

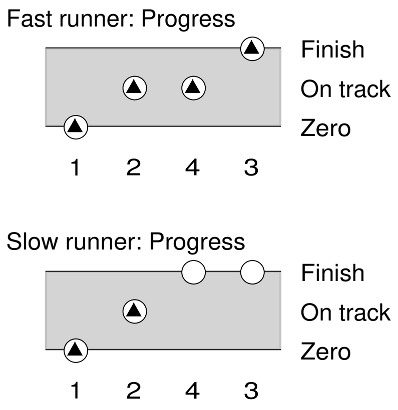


Figure 3.14: One track in the state graph of the simulation for two competing runners.

A solution for this was hinted in the analysis of the problem. As discussed before, making inequalities explicit would solve the issue, since then the inequality terminations can be applied. It is not necessary however to make all the inequalities explicit. Since the condition functions of inequality terminations also contain an inequality between the derivatives of the quantities, it is enough to make the relations explicit only between parameters with known inequality relations of their derivatives.

This section introduced a new consistency principle that states that inequalities must be accounted for even when they are not stated explicitly. Using the example of two competing runners, we presented a particular case where this principle is not valid. Furthermore, the analysis of the reason for this inconsistency suggested a solution for improving the simulation output.

3.4 Conclusion

To summarise, this chapter investigated whether sparse knowledge gives rise to incoherent simulation behaviour and if methods for solving these inconsistencies can be found. In this process, three novel consistency principles were introduced.

First, *model consistency* is the general principle saying that each state should be in accordance with the knowledge encoded by its active model fragments. In particular, two sub-types were discussed. According to *dependency consistency*, each simulation should be in accordance with the proportionality and influence relations. By means of an example, it was shown that this principle is not always respected without considering higher order derivatives, even though all the needed information is present. This is because the dependency relations, through the balances they impose, may give reason for particular next-state behaviours, i.e. simple termination. This observation was made possible by the axiomatisation and, based on it, a solution for this issue was proposed in the form of adding new simple terminations. According to *relation consistency*, each state should also be in accordance with the relations explicitly stated in the active model fragments or scenario. This principle was also not satisfied, as from an equality between terms on parameters, by taking derivatives, new inexplicit equalities can be inferred, which were sometimes not accounted for.

Second, *path consistency* states that each path should describe a plausible development of the system. One sub-type is *assumption consistency*, according to which, if an assumption about the system is needed for a state to be valid, then all of the future states developing from it must also be in accordance with this assumption. This turned out not to be satisfied either because of dependency relations, which might give assumptions about the amount of influence they offer at a specific point. To solve this, the solution of adding quantities for the dependency relations was suggested. Another sub-type of path consistency is *extrema consistency*, according to which, if an extremum point exists in some paths, the valuation should appropriately show this. Using an example, it was illustrated that the problem hinted by our axiomatisation was indeed present, and, while one value was at an extrema point, some other value could change as well. This is not possible as a from-extremum transition should be immediate. Two solutions were suggested. One was to change the priority of terminations, while the other was modelling the mathematical version of functions.

Third, *inexplicit inequality consistency* was discussed. According to this principle, a simulation should also be in accordance with the inequalities that are deducible from other information. An example of inequalities deducible from valuation was discussed. According to this, a race between a fast and a slow runner, both starting at the same time and running without interruptions, could finish with the slow runner winning. As a solution for this issue, it was suggested to find the inequalities that should be made explicit, and apply inequality terminations for these as well.

All the work carried out in this chapter brings two valuable contributions. First, this proves the axiomatisation to be highly valuable in investigating problems associated with sparse knowledge. Second, it shows five different methods for inferring more information from available sparse knowledge.

In the next section, the focus switches from analysing simulation results to analysing simulation processes. More precisely, a formal problem corresponding to a better use of the knowledge available in order to early identify inconsistencies between possible state developments is introduced and evaluated.

Chapter 4

Dynamic Systems of (In)Equalities

In the previous chapter, we have seen that, because data is sparse, the state-graph is sometimes not coherent. This is, however, not the only effect of sparse data. Because of it, incompatibilities between a model and a scenario are hard to determine. As a result, for each state, a large number of compound termination are found to be eligible, each of them then requiring multiple calls to the expensive inequality reasoning procedure for validation. This chapter aims to describe and analyse a formal problem corresponding to eligible compound termination selection.

To do this, Section 4.1 introduces two formal problems. Firstly, the *combining changes problem* takes as input a system of (in)equalities and a collection of changes that can be performed on that system, and finds a superset of the combinations of changes that give consistent developments of the original system. Secondly, the *inequality reasoning problem* decides the consistency of a set of (in)equalities, which may be linked with logical connectors. Thirdly, the *pair ordering problem*, which decides dependencies between occurrences of the pairs of changes, is discussed in Section 4.2. Sections 4.3 and 4.4 look at the theoretical complexity analysis of these problems. Section 4.5 presents a procedure improving the number of calls to the inequality reasoner, and hence better suited for practical use. This chapter ends with a discussion on the benefits of introducing these procedures.

4.1 Combining Changes and Inequality Reasoning

Consider a set of (in)equalities which is initially consistent. This set may develop into another consistent one by performing one or more changes of its elements. If only certain changes are allowed, but in any combination, we wish to identify which combinations of changes could preserve the consistency of the set. In this section, we formally define this problem. Since solving this problem requires performing calls to the inequality reasoning procedure, the latter will be formally defined first.

4.1.1 Inequality Reasoning Problem

This chapter considers dynamic sets of (in)equalities. The (in)equalities are defined as usual as two terms linked by one of the relations $<, \leq, =, \neq, \geq$ and $>$. A term is formed from constants and variables linked together with arithmetic operations and parenthesis. In addition to this, an *(in)equality relation* is obtained from (in)equalities linked together with logical connectives. A set of (in)equality relations will also be referred to as a *system of (in)equalities.*

When considering a system of (in)equalities, the problem of its consistency (i.e. is the system non-contradictory) can be introduced. For example, the system $\{x = 1, x + y < 2\}$ is non-contradictory, but adding the relation $y = 1$ makes it inconsistent, as the new relation with $x + y < 2$ implies $x < 1$, which in turn gives $1 < 1$ since $x = 1$. Of course, this is a contradiction. Hence, the problem of deciding if a system is consistent, which we will refer to as the *inequality reasoning problem*, is defined as follows:

$$IR := \{S | S \text{ is a consistent system of (in)equalities}\}$$

Given a system of (in)equalities, new relations can be deduced from it using arithmetic rules and natural deduction. Returning to the previous example, $y < 1$ can be deduced from $\{x = 1, x+y < 2\}$. For this, the notation $S \Rightarrow r$ will be used to signify that relation r can be inferred from system S. Hence, $\{x = 1, x + y < 2\} \Rightarrow y < 1$. The inconsistency of a set S is then simply a proof of $\perp$ from S, i.e. $S \Rightarrow \perp$. There is also a link between the deductibility of some relation r from S and the inconsistency of $S \cup \{\neg r\}$, which will be used later. Notice that a tautology is needed for pragmatic reasons, since, in practice, inequality reasoning engines check consistency of a system together with a relation (see Section 3.2).

Lemma 4.1.1. If S is a system of (in)equalities, and r is an (in)equality, then $S \Rightarrow r$ iff $S \cup \{\neg r\} \Rightarrow \perp$.

Proof. We prove each implication in turn:

($\Rightarrow$) If r is deducible from S this means that there is an arithmetic proof of r from S. The same proof holds of course from $S \cup \{\neg r\}$. But, of course, in $S \cup \{\neg r\}$, r is also deducible, so we have both r and $\neg r$ being true. This gives a contradiction, that is a deduction of $\perp$ and hence the inconsistency of $S \cup \{\neg r\}$.

($\Leftarrow$) To show this, we prove r from S. As the reasoning is done in classical logic, either r or $\neg r$ must hold. But, by hypothesis, adding $\neg r$ to S gives an inconsistent system. Hence, r holds. This gives a proof of r from S, and, therefore, $S \Rightarrow r$.

By the two implications we conclude that $S \Rightarrow r$ iff $S \cup \{\neg r\} \Rightarrow \bot$.

$\square$

Another property of the systems of (in)equality is the *monotonicity* of their inconsistency. To see this, consider some inconsistent set S. This means there is a proof of falsehood from S, i.e. $S \Rightarrow \bot$. But the same proof will hold when extending S with some other system S'. So, if $S \Rightarrow \bot$ then for any S', $S \cup S' \Rightarrow \bot$.

4.1.2 Combining Changes Problem

The problem of performing certain modifications to systems of (in)equalities will now be considered. A *change* on a given (in)equality relation r is an ordered pair where the first element, the source relation, is r and the second one, the destination relation, is r', i.e. $c = (r, r')$. The *update with a change* of a system of (in)equalities, S, results in a new system, S_c, obtained from S by replacing any occurrence of r with r'. Formally, if $r \notin S$ then $S_c = S$, otherwise, $S_c = S - \{r\} \cup \{r'\}$. For example, the system $S = \{x < y, x + z < y + z\}$ updated with the change $c = (x < y, x = y)$ forms $S_c = \{x = y, x + z < y + z\}$. Notice that we can also (simultaneously) update a system with a set of one or more changes as long as they do not change the same relation, that is if any two changes have different source relations. Moreover, $\mathcal{C}$ is a set of changes on S if for all $(r, r') \in \mathcal{C}$, $r \in S$. A subset of changes from $\mathcal{C}$ is called a *combination*. Such a combination C is considered *valid* for a system S if the resulting updated system S_C is consistent.

Using this, the search problem of combining changes, *COMB* can be formulated as follows:

Input: A consistent set of (in)equalities S, and a set of changes, $\mathcal{C}$, on S.

Output: Some set $Comb \subseteq \mathcal{P}(\mathcal{C})$ such that, for all $C \subseteq \mathcal{C}$, if C is valid for S, then $C \in Comb$.

An alternative, stricter formulation of this problem would be to demand the output set containing precisely the valid combinations, and no invalid ones. Notice however that, when using *IR* as an oracle, the two problems belong to the same time complexity class. To see this, firstly a solution for the strong version of the formulation is obviously also a solution for the weak version. For the converse, by updating S in turn with each member of the result set, and checking its consistency with *IR*, the output of the strong version can be produced linearly from the weak version's one. To avoid appending this last step of *IR* checks at the end, we choose to analyse the weak version.

Having introduced the combining changes problem, its complexity will be considered. The trivial case when all individual changes give consistent updates will have exponentially many valid combinations, so the problem *COMB* is clearly exponential. However, when the relations within the system are interdependent, the number of valid combinations decreases. In the remaining part of this chapter we investigate what are some conditions that would make the problem of combining changes polynomial in time when given access to constant time inequality reasoner checks.

4.2 Pair Ordering Problem

In the previous section, the problem of combining changes was introduced. So, given a consistent set and some changes, our aim is to find combinations of changes that might maintain the consistency of the set. Of course, some combinations can be ruled out immediately because the changes they contain are inconsistent with each other, so, a sub-problem of *COMB* is to look for logical dependencies between two changes. For instance, when two changes are inconsistent with each other, then if one appears the other one should not. In this section, we look at how these types of dependencies can be formally expressed, and how to formulate the problem of identifying them.

4.2.1 Pair Constraints

Given a consistent set of relations S and a set of changes $\mathcal{C}$, there may be certain constraints on the combinations of changes that make them valid for S. Even when only considering pairs of changes, dependencies may be found. For instance, if the system $S = \{x < y, x + z < y + z\}$ and changes $c_1 = (x < y, x = y)$ and $c_2 = (x + z < y + z, x + z = y + z)$ are considered, then in order to obtain a consistent updated set from S, the two changes must be applied together. This gives us the constraint that for any $C \subseteq \{c_1, c_2\}$ such that S_C is consistent, $c_1 \in C \iff c_2 \in C$. Similarly to Section 2.2.2, a *constraint* for S and $\mathcal{C}$ is formally defined as a logical relation that bounds the way combinations of changes in $\mathcal{C}$ can be used to update S in order to maintain the set's consistency.

The *pair constraints* for S and $\mathcal{C}$ are relations of the form $(\neg)c_1 \to (\neg)c_2$ with $c_1, c_2 \in \mathcal{C}$. This shorthand notation of $c_1 \to c_2$ is similar to the one used in Chapter 2, and signifies that for every $C \subseteq \mathcal{C}$, if S_C is consistent and $c_1 \in C$, then $c_2 \in C$. Similarly, $c_1 \to \neg c_2$ will be used when $c_1 \in C$ implies $c_2 \notin C$. We claim that these two types of relations are sufficient to capture all the restrictions on combinations of two changes, in the sense that considering relations of the form $\neg c_1 \to c_2$ and $\neg c_1 \to \neg c_2$ is redundant. This is shown by the following two lemmas, but more work is needed to find a canonical form for pair constraints.

Lemma 4.2.1. If S is a consistent set of (in)equality relations, and $\mathcal{C}$ is a set of changes on S, then there are no $c_1, c_2 \in \mathcal{C}$ such that $\neg c_1 \to c_2$.

Proof. We prove this by contradiction. Suppose that there are $c_1, c_2 \in \mathcal{C}$ such that $\neg c_1 \rightarrow c_2$. Making the notation explicit, this means that for every $C \subseteq \mathcal{C}$, if S_C is consistent and $c_1 \notin C$, then $c_2 \in C$. Let $C = \emptyset$. Then $S_C = S$, which by hypothesis is a consistent set. Then $c_1 \notin C$ but $c_2 \notin C$, giving a contradiction, and hence proving the lemma. $\square$

Lemma 4.2.2. If S is a consistent set of (in)equality relations, $\mathcal{C}$ is a set of changes on S, and $c_1, c_2 \in \mathcal{C}$, then $\neg c_1 \rightarrow \neg c_2$ iff $c_2 \rightarrow c_1$.

Proof. By definition, $\neg c_1 \rightarrow \neg c_2$ iff for all $C \subseteq \mathcal{C}$ with S_C consistent $c_1 \notin C \rightarrow c_2 \notin C$. But, $c_1 \notin C \rightarrow c_2 \notin C$ is the same as $\neg(c_1 \in C) \rightarrow \neg(c_2 \in C)$ which is logically equivalent with $c_2 \in C \rightarrow c_1 \in C$. This equivalence can be shown by first using the contrapositive rule and then the double negation one. This means that $\neg c_1 \rightarrow \neg c_2$ iff for all $C \subseteq \mathcal{C}$ with S_C consistent $c_2 \in C \rightarrow c_1 \in C$, which is the definition of $c_2 \rightarrow c_1$. $\square$

Those two results show that, in the above context, all pair constraints can be represented by relations of the form $c_1 \rightarrow (\neg)c_2$. However, this is not enough to assign a canonical form for pair constraints, as there might still be logically equivalent relations of the form $c_1 \rightarrow (\neg)c_2$. The next result underlines such an equivalence.

Lemma 4.2.3. If S is a consistent set of (in)equality relations, $\mathcal{C}$ is a set of changes on S, and $c_1, c_2 \in \mathcal{C}$, then $c_1 \rightarrow \neg c_2$ iff $c_2 \rightarrow \neg c_1$.

Proof. Analogous to the proof of lemma 4.2.2. $\square$

So, in order to identify a canonical form for the pair constraints, a fixed representative should be found for $\{c_2 \rightarrow \neg c_1\, c_1 \rightarrow \neg c_2\}$, which, by the lemma above, is an equivalence class with respect to logical equivalence. This can be solved by ordering the set of changes with some ordering rule. Then, if c_1 appears first in the set of changes, then $c_1 \rightarrow \neg c_2$ will be considered the representative of the given class. At this point, all the ingredients have been presented in order to construct a canonical form for the pair constraints.

Theorem 4.2.4. If S is a consistent system of (in)equality relations, $\mathcal{C}$ is a set of changes on S, $\prec$ is a total order on $\mathcal{C}$, and $Pair_{S,\mathcal{C}}$ is the set of pair constraints for for S and $\mathcal{C}$ then the function $can : Pair_{S,\mathcal{C}} \rightarrow Pair_{S,\mathcal{C}}$

$$can(pair) = \begin{cases} pair & \text{, if } pair = c_1 \rightarrow c_2 \\ c_2 \rightarrow c_1 & \text{, if } pair = \neg c_1 \rightarrow \neg c_2 \\ pair & \text{, if } pair = c_1 \rightarrow \neg c_2 \text{ and } c_1 \prec c_2 \\ c_2 \rightarrow \neg c_1 & \text{, if } pair = c_1 \rightarrow \neg c_2 \text{ and } c_2 \prec c_1 \end{cases}$$

is a canonicalisation of $Pair_{S,\mathcal{C}}$.

Proof. Firstly, *can* is indeed a function since, by lemma 4.2.1, $Pair_{S,\mathcal{C}}$ has no elements of the form $\neg c_1 \rightarrow c_2$, and the branching definition of *can* covers all the other cases.

For idempotence, we need to check that for any $pair \in Pair_{S,\mathcal{C}}$, $can(pair) = can(can(pair))$. This is easily checked: in the case *pair* is $c_1 \rightarrow c_2$ or $\neg c_2 \rightarrow \neg c_1$, $can(pair) = c_1 \rightarrow c_2$, which is a fixed point of *can*. Otherwise, $pair = c_1 \rightarrow \neg c_2$, so $can(pair)$ will be the same constraint but with the variables in accordance with the order $\prec$, which is again a fixed point of the function.

Lastly, we prove decisiveness: that $pair_1 \leftrightarrow pair_2$ iff $can(pair_1) = can(pair_2)$. The converse of this follows immediately from lemmas 4.2.2 and 4.2.3. For the direct implication, we assume $can(pair_1) \neq can(pair_2)$ and want to prove that $pair_1$ and $pair_2$ are not equivalent. By symmetry, and since the constraints in the image of *can* can be either of the form $c_1 \rightarrow c_2$ or of the form $c_1 \rightarrow \neg c_2$, we can assume w.l.o.g. that $can(pair_1) = c_1 \rightarrow c_2$ and $can(pair_2) \in \{c_2 \rightarrow c_1, c_1 \rightarrow \neg c_2, c_2 \rightarrow \neg c_1\}$. We show that $pair_1$ and $pair_2$ are not then necessarily logical equivalents by means of an example. Assume $S = \{x = 1, x > 0\}$ and $\mathcal{C} = \{c_1 = (x = 1, x = 0), c_2 = (x > 0, x \geq 0)\}$. Then, the changes that results in consistent updates are $\emptyset, \{c_2\}$, and $\{c_1, c_2\}$. Hence, c_1 only appears together with c_2, i.e. $c_1 \rightarrow c_2$, but c_2 may appear together with c_1 or alone, i.e. $c_2 \rightarrow c_1$ and $c_2 \rightarrow \neg c_1$ do not hold, and, c_1 does not imply the non-appearance of c_2, i.e. $c_1 \rightarrow \neg c_2$ does not hold. Therefore, $pair_1 \leftrightarrow pair_2$ iff $can(pair_1) = can(pair_2)$, which finalises the proof of *can* being a canonicalisation of $Pair_{S,\mathcal{C}}$. □

To conclude, a pair constraint is in canonical form if it is in the form $c_1 \rightarrow c_2$, or of the form $c_1 \rightarrow \neg c_2$ with $c_1 \prec c_2$. This will be used in the definition of the pair ordering problem. From now on, in the absence of an explicit ordering being mentioned, it will be assumed that the set of changes is ordered.

4.2.2 Definition of the Problem

In this context, it is natural to formally define the pair ordering decision problem, *PAIR*, as follows:

$$\begin{aligned} PAIR := \{\langle S, \mathcal{C}, pair\rangle \mid & S \text{ is a consistent set of relations}, \mathcal{C} \text{ is a set of changes} \\ & \text{on } S, \text{ and } pair \text{ is a pair constraint, in canonical form, for } S \text{ and } \mathcal{C}\} \end{aligned}$$

Hence, *PAIR* is the language deciding whether a certain pair constraint holds for S and $\mathcal{C}$. These types of constraints are very useful in lowering the number combinations of changes in the output of *COMB*. For example, consider the *co-occurrence graph*, $G_{S,\mathcal{C}} = (V, E)$, where the set of vertices V is precisely $\mathcal{C}$, and the set of edges, E, is $\{(c_1, c_2) | c_1 \rightarrow c_2 \in Pair_{S,\mathcal{C}}\}$. Notice that the cliques of this graph correspond to combination of changes that must always occur together. So, if there exist such a clique, of size k, then the number of eligible combinations of changes is reduced by 2^k.

In addition, the co-occurrence graph has a transitive set of edges. This is due to the transitivity of the implication relation. If a system S has three changes such as $c_1 \to c_2$ and $c_2 \to c_3$, by definition, for every set of changes C giving a consistent update of S, if $c_1 \in C$ then $c_2 \in C$, and if $c_2 \in C$ then $c_3 \in C$. Therefore, if $c_1 \in C$ then $c_2 \in C$, which in turn gives $c_3 \in C$, i.e. $c_1 \to c_3$. Hence, the co-occurrence graph has a transitive edge set.

Having defined the problem of deciding pair constraints for given (in)equality collections and sets of changes, the next section focuses on its complexity. As seen above, this will be useful for the time complexity analysis of *COMB*.

4.3 Complexity Analysis of PAIR

Pair constraints carry important information about how two constraints can appear together in a combination. Therefore, the focus of this section will be on finding the computational complexity of checking if a certain constraint in the form $c_1 \to (\neg)c_2$ is a pair constraint for a given system of (in)equalities S and a set of changes $\mathcal{C}$. This will be done under the assumption of constant time checks on the membership of systems in *IR*. In order to do this, the first subsection identifies the non-changing aspects of the current set of relations when transiting to any of its updates.

4.3.1 Common Relations

Given a consistent system of (in)equalities S and a set of changes $\mathcal{C}$, some knowledge can be inferred about the commonalities of all updates of S. For example, if $S = \{x < y, x+z < y+z\}$ and $\mathcal{C} = \{(x < y, x = y), (x+z < y+z, x+z = y+z)\}$, then it can be deduced that any update of S will have $x \leq y$ and $x + z \leq y + z$. Those statements that can be inferred to hold for any update of S will be referred to as the *common relations* for S and $\mathcal{C}$.

Formally, the set of common relations for S and $\mathcal{C}$ is

$$Common_{S,\mathcal{C}} = \{r \vee \bigvee_{(r,r')\in\mathcal{C}} r' | r \in S\}.$$

This says that, for every relation r in the system, after an update, the new system must have either r or one of its possible developments according to $\mathcal{C}$. So, for the example before, the set of common relations is $\{x < y \vee x = y, x + z < y + z \vee x + z = y + z\}$.

This concept will be used later, when analysing the complexity of *PAIR*. The construction of the set of common relations can be done in polynomial time; starting with the set equal to S, each $r \in S$ is replaced with the logical expression $e_r := r \vee \bigvee_{(r,r')\in\mathcal{C}} r'$. This expression is formed by starting with $e_r = r$, then looping through all elements of $\mathcal{C}$ and adding $\vee r'$ whenever some (r, r') is found. Therefore, the common set of constraints can be constructed in $\mathcal{O}(|S| \cdot |\mathcal{C}|)$ time.

Hence, we have identified a set of relations that remain invariant over updates. Note the important fact that the construction of this set is polynomial. In the next subsection, we use this set to check the validity of pair constraints.

4.3.2 Complexity of *PAIR*

At this point, we have all the ingredients needed to analyse the complexity of deciding whether a pair constraint is valid for a given system of (in)equalities and changes. Therefore, in this section, we prove the following result.

Theorem 4.3.1. *PAIR* $\in \mathbf{P}^{IR}$.

Since *IR* is **NP**-hard, and *PAIR* is clearly in **NP**, this theorem follows immediately. However, for pragmatic reasons, we opt for also providing a constructive proof, which will prove useful in the following chapter.

To show this, let's consider the following algorithm, P.

1. Check the validity of the input:
   - that S is a system of (in)equalities, $\mathcal{C}$ a set of changes on S, and *pair* is in the form $c_1 \to (\neg)c_2$ with $c_i \in \mathcal{C}$; the notation of $c_i = (r_i, r_i')$ will be used;
   - moreover, check that $S \in IR$.

   If the input is not valid, then output 0.

2. Construct the set of common relations for S and $\mathcal{C}$, $Common_{S,\mathcal{C}}$.

3. Next, proceed by cases:
   - First, suppose $pair = c_1 \to c_2$. In the particular case that $c_1 = c_2$, always output 1. Otherwise, output 1 iff for all $r \in \{r | (r_2, r) \in \mathcal{C}, r \neq r_2'\} \cup \{r_2\}$ $Common_{S,\mathcal{C}} \cup \{r_1', r\} \notin IR$.
   - If $pair = c_1 \to \neg c_2$, then output 1 iff $Common_{S,\mathcal{C}} \cup \{r_1', r_2'\} \notin IR$.

This procedure is polynomial with an oracle for language *IR*. The first step, checking the validity of the input, can be done in $\mathcal{O}(|S| \cdot |\mathcal{C}|)$. The second step builds the common set of relations, and, as shown in the previous section, is polynomial, while the last step consists of at most $|\mathcal{C}|$ calls to the oracle.

Therefore, it remains to show that the presented algorithm indeed outputs 1 iff *pair* is a pair constraint in canonical form for S and $\mathcal{C}$.

Lemma 4.3.2. If S is a consistent system of (in)equality relations, $\mathcal{C}$ is a set of changes on S, then procedure P outputs 1 iff *pair* is a pair constraint in canonical form.

Proof. We prove each implication in turn.

($\Rightarrow$) If P outputs 1, then, by the first step, $pair = c_1 \rightarrow (\neg)c_2$. Now we proceed by cases.

If $pair = c_1 \rightarrow c_2$, and $c_1 = c_2$, then $pair = c_1 \rightarrow c_1$ which is trivially true. However, if $c_1 \neq c_2$, since P produced output 1, for all $r \in \{r|(r_2, r) \in \mathcal{C}, r \neq r_2'\} \cup \{r_2\}$ $Common_{S,\mathcal{C}} \cup \{r_1', r\} \notin IR$. Informally, this means that any update of S that uses the change c_1, but either leaves r_2 as it is, or changes it to something different from r_2' is inconsistent.

For a formal proof, let $C \subseteq \mathcal{C}$ such that $c_1 \in C$, S_C is consistent, but $c_2 \notin C$. By definition, $S_C \Rightarrow Common_{S,\mathcal{C}} \cup \{r_1', \vee_{r \in \{r|(r_2,r) \in \mathcal{C} - \{c_2\}\} \cup \{r_2\}} r\}$. Since, $Common_{S,\mathcal{C}} \cup \{r_1', r\} \Rightarrow \perp$ for all $r \neq r_2'$, this means there is a proof by cases of $\perp$ from $Common_{S,\mathcal{C}} \cup \{r_1', \vee_{r \in \{r|(r_2,r) \in \mathcal{C} - \{c_2\}\} \cup \{r_2\}} r\}$, and, therefore, S_c is inconsistent, which is a contradiction. Hence, any set of changes containing c_1 and consistently updating S also contains c_2.

Otherwise, if $pair = c_1 \rightarrow \neg c_2$, since 1 is the output of P, $Common_{S,\mathcal{C}} \cup \{r_1', r_2'\} \notin IR$. By lemma 4.1.1, this is equivalent to $Common_{S,\mathcal{C}} \cup \{r_1'\} \Rightarrow \neg r_2'$. Let $C \subseteq \mathcal{C}$ such that $c_1 \in C$ and S_C is consistent. Then, $S_C \Rightarrow Common_{S,\mathcal{C}} \cup \{r_1'\}$ directly from proof theoretic axioms. Therefore, $S_C \Rightarrow \neg r_2'$, which means $c_2 \notin C$.

($\Leftarrow$) By hypothesis $pair = c_1 \rightarrow (\neg)c_2$ is a pair constraint. We consider each of the two cases.

If $pair = c_1 \rightarrow c_2$, then for all $C \subseteq \mathcal{C}$ such that S_C is consistent, $c_1 \in C \Rightarrow c_2 \in C$. In other words, for all $C \subseteq \mathcal{C}$ such that $c_1 \in C$ and $c_2 \notin C$, S_C is inconsistent. Hence, any update from S obtained by changing r_1 to r_1', r_2 with something different from r_2' and any other change of relations according to $\mathcal{C}$ results in an inconsistent set. Hence, there is a proof by cases that if the update contains r_1' but something different from r_2', then falsehood will result. That is, for all $r \in \{r|(r_2, r) \in \mathcal{C}, r \neq r_2'\} \cup \{r_2\}$ we have $Common_{S,\mathcal{C}} \cup \{r_1', r\} \Rightarrow \perp$, which means $Common_{S,\mathcal{C}} \cup \{r_1', r\} \notin IR$.

Otherwise, that is if $pair = c_1 \rightarrow \neg c_2$ is a pair constraint, then for all $C \subseteq \mathcal{C}$ such that S_C is consistent, $c_1 \in C \Rightarrow c_2 \notin C$. In other words, for all $C \subseteq \mathcal{C}$ such that $c_1 \in C$ and $c_2 \in C$, $S_C \Rightarrow \perp$. Similarly as above, we then have a proof by cases of $\perp$ from $Common_{S,\mathcal{C}} \cup \{r_1', r_2'\}$. That is, $Common_{S,\mathcal{C}} \cup \{r_1', r_2'\} \notin IR$.

This finalises the proof of the correctness of the output of P. $\square$

With the above result, P is known to be both polynomial with oracle *IR* and to decide *PAIR*. Therefore this concludes the proof of theorem 4.3.1. Next, we show how this result is used to solve the search problem $COMB$, and under which conditions it becomes polynomial.

4.4 Complexity Analysis of *COMB*

Construction of algorithms for solving *COMB* can be approached from two directions, depending on what information on pair constraints is used. On the one hand, one can only use the bidirectional co-occurrences which show changes that must always appear together. We already mentioned the benefits of this approach in Subsection 4.2.2, when the co-occurrence graph was introduced. On the other hand, one could also use the unidirectional co-occurrences. This section addresses the question of which assumptions are necessary to ensure the solving of *COMB* in polynomial time.

4.4.1 Bidirectional Perspective

The first approach to solving *COMB* is from the perspective of using bidirectional co-occurrences of changes. These are described by pair constraints of the form $c_1 \leftrightarrow c_2$, so they are changes that may only happen together. A set of changes for which its elements are pairwise bidirectional co-occurring for some given S and $\mathcal{C}$ forms a clique in the co-occurrence graph. The *clique cover number* of a graph is the minimum number of cliques that cover all vertices of the graph. For our problem, the lower this number is, the more dependencies are identified, which reduces the number of eligible combinations of changes. The next theorem looks at the complexity of *COMB* in terms of the clique cover number.

Theorem 4.4.1. If S is a consistent system of (in)equality relations, $\mathcal{C}$ is a set of changes on S, and the clique cover number of the associated co-occurrence graph is logarithmic in the size of $\mathcal{C}$, then *COMB* with oracle *IR* is polynomial.

Proof. Once again, we will opt for a constructive proof, so we consider the following procedure P:

1. Construct the bidirectional co-occurrence graph for S and $\mathcal{C}$, say $G = (\mathcal{C}, E)$. This is done by starting with the set of edges E being empty and for each $c_1, c_2 \in \mathcal{C}$ such that $\langle S, \mathcal{C}, (c_1, c_2) \in \mathit{PAIR} \rangle$ and $\langle S, \mathcal{C}, (c_2, c_1) \in \mathit{PAIR} \rangle$, adding (c_1, c_2) to E. Notice that this graph is undirected, so E is a set of unordered pairs.

2. For each c_i find the first c_j, which will be referred to as $f(c_i)$, with $j \leq i$ such that $(c_i, c_j) \in E$.

3. Find the image of f, i.e. $f[\mathcal{C}]$.

4. Construct the power set of $f[\mathcal{C}]$. This will be referred to as *Comb*.

5. Take in turn each set A in *Comb*. For all $c \in A, c' \in \mathcal{C}$ such that $f(c') = c$, add c' to set A.

6. Return the updated set *Comb*.

At this point, we claim that $\{\{c|f(c') = c\}|c \in f[\mathcal{C}]\}$, that is the collection of sets of changes with the same value of f is a minimum clique cover of G. Remember that the set of edges of the co-occurrence graph, and hence also of its bidirectional version, is transitive. Moreover, if $Cl_c = \{c|f(c') = c\}$, then, by construction, for all $c' \in Cl_c$ $(c, c') \in E$. Therefore, for each $c_1, c_2 \in Cl_c$, by construction, $(c, c_1) \in E$, and $(c, c_2) \in E$, and by the transitivity of E, $(c_1, c_2) \in E$. That is, each Cl_c describes a complete subgraph of G. Moreover, this subgraph is maximal since, if there was some other $c_3 \in V - Cl_c$ to be added to the clique, then $(c, c_3) \in E$, so $f(c) = f(c_3)$, which is not the case. Hence, $\{\{c|f(c') = c\}|c \in f[\mathcal{C}]\}$ is a clique cover. It is also the only one, as none of those cliques are connected by edges, due to the transitivity of E.

This procedure is clearly polynomial in the number of calls to *IR*. The first step uses $\mathcal{O}(|\mathcal{C}|^2)$ many calls to $PAIR$, which, by theorem 4.3.1, is in $\mathbf{P}^{IR}$. The next two steps also have the same complexity. Since the clique cover number is logarithmic in $|\mathcal{C}|$, constructing the power set will also be polynomial. This means in turn that step 5 is done in $\mathcal{O}(|\mathcal{C}|^3)$, while the output is again quadratic in $|\mathcal{C}|$.

Regarding the correctness of this program, by definition, the first step correctly constructs the bidirectional co-occurrence graph. The next steps identify, for each change c', the minimum labelled change c that is in the same maximal clique containing c'. Therefore, in any combination that results in a consistent update, c' appears iff c appears. Hence, only the clique representatives, i.e. members of $f[\mathcal{C}]$ can appear freely, the other changes depending on their respective representatives. Since, by assumption, their number is logarithmical with respect to the size of $\mathcal{C}$, the construction of the power set is done in polynomial time. Step 5 extends each subset of $f[\mathcal{C}]$ with the changes equivalent to its members according to f. The result is a set R. So, by construction, any combination set in $\mathcal{P}(\mathcal{C}) - R$ results in an inconsistent update, since there would be some $c_1 \leftrightarrow c_2$, such that the set contains c_1 but not c_2. In other words, R contains all the valid combinations of changes. $\square$

Therefore, the problem of finding some superset of the combination of changes resulting in consistent updates is polynomial when the clique cover number is logarithmic in the size of the input. The proof above is done by finding the maximal clique cover of the bidirectional co-occurrence graph, and allowing for all the combinations between the representative of each clique. Notice however that the clique cover problem is NP-complete (Karp, 1972) and we were able to solve it polynomially only because the co-occurrence graph has the special property of having a transitive set of edges. In the next section this problem will be viewed from the perspective of unidirectional co-occurrences.

4.4.2 Unidirectional Perspective

The approach presented in the previous subsection is useful in case there are many changes that only give consistent updates when applied together. We will now investigate whether it is possible to solve in polynomial time the combining changes problem even when there are only few such dependencies. We will answer this by considering paths in the co-occurrence graph. These correspond to sequences of implications. For example, given a sequence $c_1 \to c_2 \to c_3$, which says that if c_1 appears so does c_2, and if c_2 appears, so does c_3, the only valid combinations of changes can be $\emptyset, \{c_3\}, \{c_2, c_3\}$, and $\{c_1, c_2, c_3\}$. This reasoning lowers the number of valid combinations from 8 to 4.

First, let's introduce some useful definitions. A *(vertex-disjoint) path cover* of a directed graph $G = (V, E)$ is given by a set of paths (i.e. sequences of vertices) $Path_i = v^i_0, ..., v^i_{n_i}$ on vertex sets $V_i = \{v^i_j | j \leq n_i\}$ such that all $(v^i_j, v^i_{j+1}) \in E$, the vertex sets of the paths are pairwise disjoint, and $V = \dot\bigcup_i V_i$. So, a path cover of G is a set of vertex disjoint paths in G such that all vertices in V belong to some path. Moreover, a *minimum path cover* has the least number of paths, i.e. there is no path cover with a smaller number of paths. The size of a minimum path coverage is called the *path cover number* (Boesch and Gimpel, 1977).

In order to be able to use path covers for the current problem, it should be possible to find one in polynomial time. In general this is not possible, as this problem is known to be **NP**-hard. To see this, in the special case that the answer is a path coverage of size 1, this corresponds to solving the well known **NP**-hard problem of finding a Hamiltonian path (Arora and Barak, 2009). However, the co-occurrence graph has the special property of having a transitive set of edges. The next lemma uses this property to construct a polynomial time algorithm for finding a minimum path coverage.

Lemma 4.4.2. For any oriented graph $G = (V, E)$ with a transitive set of edges, a minimum path cover can be constructed in polynomial time.

Proof. To solve this, we will reduce the given problem to solving the same one on a directed acyclic graph using a similar procedure as for theorem 4.4.1, and then use the maximum-matching algorithm (Hopcroft-Karp) to solve the latter problem (Hopcroft and Karp, 1973). In short, the resulting procedure is as follows:

1. Order the vertices in V in some arbitrary order. So, $V = \{v_1, v_2, ..., v_n\}$.

2. Construct the function f for the representative of each vertex: for each $v \in V$ find the minimum labelled v' such that $(v, v') \in E$ and $(v', v) \in E$ and set $f(v) = v'$.

3. Construct the graph $G' = (V', E')$ from G by contracting the classes of vertices with the same value of f to their representative. More precisely, $V' = f[V]$ and $E' = \{(f(v), f(v')) | v, v' \in V, f(v) \neq f(v')\}$.

4. Form two subsets of vertices. The first one, O, contains the vertices in V' with at least one out-edge, and I contains the vertices in V' with at least one in-edge. That is, $O = \{\langle v, 0\rangle \,|v \in V', \exists v' \in V \ (v, v') \in E'\}$, and $I = \{\langle v, 1\rangle \,|v \in V', \exists v' \in V \ (v', v) \in E'\}$. Then, set $V_m = O \cup I$. In addition, form a set E_m the set of edges corresponding to E' for V_m, that is $E_m = \{(\langle v, 0\rangle \,, \langle v, 1\rangle)|(v, v') \in E'\}$. The new resulting graph is $G_m = (V_m, E_m)$.

5. Use the maximum matching algorithm for finding a maximum matching, M, in G_m.

6. Form a corresponding path cover in G'. The edges of the paths in the cover are (v, v'), such that $(\langle v, 0\rangle \,, \langle v, 1\rangle) \in M$.

7. Expand the cliques in any order. That is, given a path as a sequence of vertices in G', if some vertex v has multiple other vertices with the same f value, then put all of them, in any order (say in the order of their original label), between v and its successor in the path.

8. Return the resulting path coverage of G.

This procedure is polynomial, since the maximum matching algorithm has complexity $\mathcal{O}((|V_m| + |E_m|)\sqrt{|V_m|})$ (Hopcroft and Karp, 1973). Moreover, using König's Theorem (Konig, 1931), it can be proven that this algorithm outputs a minimum path coverage. □

This means, that we can decompose a graph into paths. To see how this is useful in the context of dynamic sets of (in)equalities, an analysis on the number of valid combinations for the changes in a given path is needed.

Lemma 4.4.3. If S is a consistent set of (in)equality relations, $\mathcal{C}$ is a set of changes on S, and *Path* is a path of length n in the associated co-occurrence graph, then at most $n + 1$ combinations of the changes in *Path* can be extended to a set of changes resulting in a consistent update of S.

Proof. Suppose $Path = c_1, c_2, ..., c_n$. Since this is a path in the co-occurrence graph, $c_i \to c_{i+1}$. So, if a set of changes contains c_i and gives a consistent update, then it also contains c_j for all $i \le j \le n$. That is, any set giving consistent updates and containing some c from *Path*, also contains all the changes that follow c in *Path*. By distinguishing the cases based on the first change that appears in the set, this means that $\{\{c_j|j > i\}|0 \le i \le n\}$ are the only subsets of $\{c_1, c_2, ..., c_n\}$ that can be extended to sets giving consistent updates. Hence, they are at most $n + 1$. □

Putting the above two results together, we find another case when the combining changes problem can be solved polynomially. This finding is captured in the theorem below.

Theorem 4.4.4. If S is a consistent set of (in)equality relations, $\mathcal{C}$ is a set of changes on S, and the path cover number of the associated co-occurrence graph is bounded by a constant k, then *COMB* with oracle *IR* is polynomial.

Proof. To show this constructively, we consider the following procedure P:

1. Construct the co-occurrence graph for S and $\mathcal{C}$, say $G = (\mathcal{C}, E)$. This is done similarly to the construction in the proof of theorem 4.4.1.

2. Form a minimal path cover of G, using the procedure in the proof of lemma 4.4.2. Denote the paths in the cover by $Path_1, ..., Path_k$.

3. For each $Path_i$ find the eligible combinations of changes, $Comb_i$, as shown in lemma 4.4.3.

4. Return the cross product between all $Comb_i$.

The procedure P is polynomial with oracle *IR*. As in the proof of theorem 4.4.1, the construction of the co-occurrence graph is done in polynomially many steps with calls to the *IR*. From lemma 4.4.2, a minimal path cover of G is constructed polynomially. Step 4 is done linearly, as shown by lemma 4.4.3. Lastly, since, by hypothesis, the number of paths is bounded by the constant k, the cross product is done in $\mathcal{O}(n^k)$.

To prove the last claim, check how many sets are in the cross product of $Comb_i$. By lemma 4.4.3, each such set has the size equal to 1 plus the length of the path $Path_i$. Say that $Path_i$ has length p_i. Then $|Comb_i| = p_i + 1$. In addition, since the vertices of the paths partition V, $\sum_i p_i = n$ and $k \leq n$. Putting these together, the cross product of all the $Comb_i$ has size $\prod_i (p_i + 1)$. By the inequality of arithmetic and geometric means, $\prod_i (p_i + 1) \leq \left(\frac{\sum_i (p_i+1)}{k}\right)^k$. Using the previously found observations, the right hand side is $\left(\frac{n+k}{k}\right)^k$, which is less than or equal to $(n+1)^k$.

The correctness of this procedure follows immediately from lemma 4.4.3, as any combination of changes not in the results set is not in accordance with the conditions imposed by some path i. $\square$

To conclude, the combining changes problem with oracle access to the inequality reasoning problem can be solved in polynomial time if the minimum path coverage number is constant. In the next section, focus shifts from solving *COMB* in polynomial time to reducing the size of the output as much as possible while using as few calls as possible to the oracle. A discussion on the benefits of the two algorithms presented in this section also follows.

4.5 Minimising *IR* Calls in Solving *COMB*

Until now, the complexity analysis was carried out under the assumption that checks to the inequality reasoning problem take constant time. However, in practice, this problem is computationally expensive, so the practical aim is to lower the number of calls to the oracle *IR*. In this section, we discuss the benefits of using pair constraints to lower this number.

The brute force algorithm takes all combinations of changes, updates the system of (in)equalities with each combination to generate new systems, and uses *IR* to check the consistency of each of these systems. This procedure uses an exponential number of calls to oracle *IR*, i.e. $2^{|\mathcal{C}|}$, where $\mathcal{C}$ is the set of changes. In the two approaches presented in the previous section, calls to *IR* were done only when constructing the (bidirectional) co-occurrence graph, which used $2 \cdot |\mathcal{C}|^2$ many calls, one for each ordered pair of changes, before calling *IR* again to check the validity of each combination in the output.

Since now we want to minimise the size of the output with the information from the pair constraints, the entire information on these is needed. Moreover, we want to do this minimisation with as few calls to the oracle as possible. The intuition on how to do this is to find co-occurring sets of changes as fast as possible, and afterwards determine the dependencies between these. Observe that, because of the transitivity of relation $\rightarrow$, the vertices of a cycle in the co-occurrence graph form a clique. In addition, since relations of the type $\neg c_1 \rightarrow c_2$ are not pair constraints and those of type $\neg c_1 \rightarrow \neg c_2$ are redundant (see lemmas 4.2.1 and 4.2.2), extending our reasoning on $\neg\mathcal{C}$ will not help identify cliques, as a change from $\neg\mathcal{C}$ can never be in the same clique with one from $\mathcal{C}$.

Using these observations, we suggest the following procedure for identifying the co-occurring sets of changes and the pair constraints between them. As before, we assume that the input is some consistent system of (in)equalities S and some (ordered) set of changes on it $\mathcal{C}$. For readability, we also assume that there is at most one change on each relation, but the generalisation is straightforward and does not change the results. By having this assumption, checks of membership in *PAIR* are done with only one call to *IR*. We will refer to the following procedure as $P_{\text{Co-occurrence}}$:

1. Start with the function $clique(c) = c$ for all $c \in \mathcal{C}$. This labelling function will return, for each possible c, which change is the representative one in its equivalence class.

2. Also, start with $Pair(c) = \emptyset$, where $Pair(c)$ is the set of all pair constraints of type $c \rightarrow c'$.

3. Construct the common set of relations, $Common_{S,\mathcal{C}}$.

4. Form the class of changes which could not be used for any consistent update.

Take the first change $(r, r') \in \mathcal{C}$ such that $Common_{S,\mathcal{C}} \cup \{r'\} \notin IR$, and, if it exists, denote it by $c_\perp$. Continue with the remaining changes, and for each $c = (r, r') \in \mathcal{C}$ such that $Common_{S,\mathcal{C}} \cup \{r'\} \notin IR$, make $clique(c) = c_\perp$.

5. Find other cliques while keeping notes on the pair constraints that were already discovered.
   (a) Let $visited(c) = 0$ iff $clique(c) \neq c_\perp$.
   (b) Take each $c_s = (r, r') \in \mathcal{C}$ such that $visited(c_s) = 0$. For each, start with $path = [c_s]$ and do the following DFS procedure:
      i. Let $c_l = (r_l, r'_l)$ be the last change in *path*.
      ii. Take, in order, each change $c \in \mathcal{C}$ for which $visited(c) = 0$ and $c_l \neq c$. First process the changes that are not in *path*, and then the other ones, in the order they appear in *path*, from first to last:
         A. Check if $c_l \to c$, i.e. if $Common_{S,\mathcal{C}} \cup \{r'_l, r\} \notin IR$. If true, proceed to step B, otherwise, process the next change. If all changes were processed, proceed to step iii.
         B. If c is in *path*, then for each c' appearing in *path* after c, make $clique(c') = c$, $visited(c') = 1$, and remove c' from *path*. Also remove c and make $visited(c) = 1$. Add to $Pair(c_n)$ the constraint $c_n \to c$, where c_n is the new last element in *path*. Otherwise, i.e. c is not in *path*, add c at the end of *path*.
         C. Continue recursively from step i.
      iii. Before backtracking, remove c from *path*, make $visited(c) = 1$, and add $c_l \to c$ to $Pair(c_l)$, where c_l is the current last change in *path*. Then backtrack and continue processing changes for c_l.
6. Use the algorithm from theorem 4.4.4 to find the valid combinations for the co-occurrence graph with vertices in $clique[\mathcal{C}] - \{c_\perp\}$ and edges from *Pair*. This will return a set of combinations *Comb* of representative changes.
7. Expand the representatives. In this step, for each combination set $\in Comb$, and any change $c \in Q$, add all the changes in the same equivalence class with c to Q, i.e. all c' such that $clique(c') = c$.
8. Return the updated set of combinations *Comb*.

Now, $P_{\text{Co-occurrence}}$ efficiently finds a super set of the valid combinations. Regarding the size of the final output, almost the same results hold as in the previous section. In case the number of cliques, i.e. $|clique[\mathcal{C}]|$, is logarithmic in $|\mathcal{C}|$, then the size of the output is polynomial. The same result is achieved when the minimum path cover number is constant.

The great benefit of this new procedure is that it does not check for deducible pair constraints. For example, for an existing path of the form $c_1 \to c_2 \to c_3$,

$P_{\text{Co-occurrence}}$ only checks for $c_1 \to c_2$ and $c_2 \to c_3$, but once these are found, $c_1 \to c_3$ is not verified. This means that we identify the minimum set of relations that, when taking their transitive closure, would give the full set of pair constraints. Moreover, observe that cliques are identified by finding only a cycle rather than all edges, and the relations $c_\perp \to c$, and $c_\perp \to \neg c$ (or its equivalent form) trivially hold and can be added without any call to the *IR*.

4.6 Conclusion

In this chapter, the problem of maximising the amount of information inferable from sparse knowledge is viewed from the perspective of a simulation. A theoretical problem correlated with this issue is introduced and evaluated. Consequently, three contributions are distinguished.

First, this chapter provides precise definitions of two relevant problems. One is the inequality reasoning problem, a problem deciding the consistency of qualitative systems of (in)equalities, i.e. *IR*. The discussion from Chapter 2 according to which the inequality reasoning problem is computationally expensive (**NP**-hard) motivated us to use *IR* as an oracle throughout the complexity analysis. The other one, is the problem of combining changes, which is a search problem that, given a set of (in)equalities and a set of changes on these, it returns at least all the combinations of changes giving consistent updates, i.e. *ELIG*.

Second, a complexity analysis of *COMB* with *IR* as an oracle was performed. Using pair constraints, which shows dependencies between occurrences of changes in combinations that give consistent updates, it was shown for two situations that *COMB* is polynomial when assuming constant time checks to *IR*. One such situation is if the number of cliques in the co-occurrence graph, that is in the graph given by the pair constraints on the set of changes, is logarithmic. Another such situation, is if the co-occurrence graph can always be covered by a number of paths smaller than some constant.

Third, an adapted algorithm better suited for practical use was presented. With that algorithm cliques and paths were identified while constructing the co-occurrence graph. In addition, only the pair constraints in canonical form were considered, hence not double checking for pair constraints validity.

As a result, in case of highly dependent changes, the method introduced in this chapter manages to infer more from sparse knowledge. More precisely, in these cases the problem becomes from exponential polynomial, and hence improving its theoretical complexity.

In the next chapter we look at how these results can be used for our original problem of reducing the number of eligible compound terminations.

Chapter 5

Reducing Eligible Compound Terminations

A qualitative simulation is complex and requires several steps, as outlined in Chapter 2. To compute the behaviour graph, for each state its set of simple terminations must first be computed. Because of the sparse nature of the data, identifying combinations of these terminations that do not produce valid updates usually requires a full analysis of the system of (in)equalities describing the model and the restrictions of the next state. As a result, the inequality reasoning procedure is used for almost all compound terminations in turn, even when only a few successors emerge. However, in the previous chapter we have introduced a method for efficiently selecting valid combination of changes. Importantly, we showed that the method only has polynomially many calls to the inequality reasoner in certain cases. The focus of this chapter is to investigate the extent to which this method can be applied for reducing the number of eligible compound terminations.

The current chapter is structured into three sections. The first section shows that the problem of finding eligible compound terminations can be reduced to combining changes, while also underlining the assumptions under which this reduction applies. In the second section we eliminate one of those assumption, and adapt the solution to work with changing model fragments. Lastly, the benefits of this method are illustrated by means of two examples.

5.1 Reduction to *COMB*

In this section, we will look at the correspondence between the eligible compound termination selection and the combining changes problem. In order to construct a reduction from the first to the second, a precise formulation of the two problems is needed. In addition, transformations that encode the input and decode output should be found. To perform the reduction, the input of the first problem is encoded into an input for the second one, the algorithm for *COMB* is used find a solution, which is decoded into an output for the original problem. For this to be a solution, the correctness of the output should be checked. Each of these issues are addressed in the next three subsections.

5.1.1 Eligible Compound Terminations Selection Problem

The problem we wish to express is part of the state ordering phase. Remember from Section 2.2 that this is done after a state is terminated. So, at that point, the system has a set of simple terminations that can be combined and applied in order to construct successor states. The aim of the ordering phase is to combine those simple terminations into compound ones such that none of the compound terminations giving valid successors is left out. Of course, this can be done by including all the compound terminations, but this would take exponential time.

The aim is then to identify the compound terminations leading to inconsistent states as early as possible. The reason for this is that each eligible compound termination found in the ordering phase will be analysed in the closing phase using multiple calls to the inequality reasoner. Much of the reasoning is then redone, because when two terminations cannot lead to a successor together, then, if no initial elimination is done, this combination will be tested with the inequality reasoner together with all possible choices of taking other terminations. This means exponentially many calls have to be made to the inequality reasoner. The better approach would be to have a method for identifying the incompatibility between the termination pairs with polynomially many checks.

Reducing the number of eligible compound terminations is of course not always possible. If the system has only few constraints, and almost all compound terminations give valid successors, this procedure would only perform a polynomial number of additional checks. Since, in that case the output is exponential, this does not affect the theoretical complexity. So adding this procedure is not detrimental. From a practical perspective, the systems that are usually modelled are ones with highly dependent quantities, and have only a few successors for each state. Simulations where most of the states can evolve to any possible combination are not interesting. Therefore, reducing the number of eligible compound termination with pair constraints should, in general, be useful.

In this chapter we will analyse the problem of eliminating compound terminations that cannot appear. Formally, this problem will be referred to as the *eligible compound termination selection*, *ELIG*, and is defined below.

Input:	Some state s, a model fragment function mf, the mathematical representation for s, S_s, the constraints of s according to the causal ingredients of mf, $Const$, and a set of simple terminations for s that are epsilon ordered, $\mathcal{T}$.
Output:	Some set $Comb \subseteq \mathcal{P}(\mathcal{T})$ such that all compound terminations resulting in a successor for s are in the set $Comb$.

Observe that the input and output are similar to the ones in the problem of combining changes. The epsilon order constraint on the set of simple terminations is added to ensure that combinations of terminations that give continuous and stable extensions of their transition scenario are indeed successors. The next subsection investigates the functions that transform the inputs and outputs of the two problems.

5.1.2 Definitions for the Reduction Functions

Since the aim is to construct a reduction from *ELIG* to *COMB*, the inputs and outputs of the two problems must be transformed into one another. That is, we need a (polynomial time-computable) function f that given an input of *ELIG* returns a corresponding one for *COMB*, and another (polynomial time-computable) function g that given an output of *COMB* returns the corresponding one for *ELIG*.

For the input function, assume there is an input $\langle s, mf, S_s, Const, \mathcal{T}\rangle$ for *ELIG*. This should be transformed to a pair where the first element is a system of (in)equalities, and and the second one is a set of changes on some (in)equalities of that set.

To do this, firstly, we adapt S_s into the system S. Since S_s is the mathematical model for state s, it is already a system of (in)equalities. From it, we construct a *continuous extension*, that is the mathematical model of s together with the constraints for continuity in the next state. That is, to form S, we start with it being equal to S_s and for each parameter with known valuation of its derivative, S is built according to Table 5.1.

The set of changes is obtained from the set of simple terminations. Table 2.2 outlines the possible simple terminations together with the conditions under which they were selected, and the results set, which shows the changes they produce in successors. Due to continuity, the results set of termination also contains a weak constraint on the derivative of the modifying parameter. However, since in the continuous extension of the state the continuity criterion was applied, the results equations on these weak constraints are not needed. Hence, the set of changes is constructed from the set of simple terminations by making pairs of the relation from the conditions that modifies and the new one that will appear in the new state if that simple termination is used. Table 5.2 shows the corresponding

Eliminated relation	Added relation
$val(dP) = 0$, or $val(dP) \leq 0$, or $val(dP) \geq 0$	$val(dP) = ?$
$val(dP) > 0$	$val(dP) \geq 0$
$val(dP) < 0$	$val(dP) \leq 0$

Table 5.1: Construction of the continuous extension for the state s with value assertion *val*. Initially the extension is the mathematical model describing s.

changes for a few examples of terminations. Finally, after changing the constraints of S_c, the relations in *Const* are added to the set S.

Simple termination	Modifying condition	Corresponding change
(to_point_above, Q)	$val(Q) = i_{u,u+1}$	$(val(Q) = i_{u,u+1},$ $val(Q) = l_{u+1})$
$(to\_interval\_below, Q)$	$val(Q) = l_{u+1}$	$(val(Q) = l_{u+1},$ $val(Q) = i_{u,u+1})$
$(derivative\_down\_$ $to\_stable, Q)$	$val(dQ) < 0$	$(val(dQ) \leq 0$ $val(dQ) = 0)$
$(from\_equal\_to\_$ $greater, (P_1, P_2))$	$P_1 = P_2$	$(P_1 = P_2, P_1 > P_2)$
$(from\_greater\_or\_$ $equal\_to\_equal, (P_1, P_2))$	$P_1 \geq P_2$	$(P_1 \geq P_2, P_1 = P_2)$

Table 5.2: Some simple terminations with their corresponding change.

Remember that in the mathematical model, only landmarks are used as constants, so $val(Q) = i_{u,u+1}$, is a shorthand notation for the relation $l_u < val(Q) \wedge val(Q) < l_{u+1}$. In addition, for the derivative terminations, the changes are done on the continuous extension of the condition, since the original condition which was in the mathematical model of s, S_s, is not in the same form in its continuous extension S. The values of the derivatives were changed according to Table 5.1.

The resulting set is obviously a set of changes. It is also a set of changes on S, as the first elements of the pairs are members of S. For the changes obtained from value and inequality terminations, the first elements in the pair are conditions of the termination, so they must be in the mathematical model of s. For the derivative and exogenous ones, we have used the continuous version of the constraint known to be in S_s, and by our construction of the continuous extension that must be in S_s. This describes a linear procedure for computing $f(x)$.

For the name of the changes, this is similar to the ones of the corresponding terminations. The simple terminations are labelled with increasing numbers in the order in which they were found in the termination phase of a state. The corresponding change then inherits the same label, i.e. t_i has the corresponding change c_i. Using this notation system, it is easy to decode an output of *COMB*. Given a set of combinations of changes, each change in each combination is replaced with its corresponding termination. This procedure is a description of the function g, and is linear in the size of its input.

5.1.3 The Reduction

In the previous section, the reduction functions for both the input and the output were defined. Hence, there is a function f encoding the input x of *ELIG*, then $f(x)$ is passed to *COMB* resulting in a set of combinations of changes *Comb*, and that set is decoded into an output for *ELIG* using function g. It remains to be shown that this procedure indeed solves the eligible combination selection problem. That is, we want to show that $\langle x, g(\mathit{Comb})\rangle$ is a pair where $g(\mathit{Comb})$ is a solution of *ELIG* given input x iff $\langle f(x), \mathit{Comb}\rangle$ is an input/output pair for *COMB*.

For notation purposes, since x is an input for *ELIG*, it is of the form $x = \langle s, \mathit{mf}, S_s, \mathit{Const}, \mathcal{T}\rangle$. Moreover, as described in the previous section, S is the continuous extension of S_s together with *Const*, while $\mathcal{C}$ is the set of changes obtained from the set of simple terminations $\mathcal{T}$. Then, the key observation to prove the equivalence is that the compound terminations in $\mathcal{P}(\mathcal{T})$ that do not result in a successor correspond with the combinations of changes that do not give consistent updates of S_s.

To prove the converse, notice that the relations in S must hold for all successors of the state s, as a successor needs to be stable and consistent. Moreover, if a certain simple termination is used for a successor, then a relation in S changes to the ones in the results set of that termination. So, if a combination of changes gives an inconsistent update on S then the corresponding compound termination cannot lead to a successor for that state. The direct implication is similar. If a compound termination is not valid, this can be because either the terminations are not at the same level according to the epsilon ordering, the resulting successor is not valid, or the resulting successor is not stable. The first cannot be the case as, according to the input restrictions, the set of simple termination is epsilon ordered. Both of the other restrictions are encoded into the set S, as the continuity is ensured when the continuous extension of S_s is computed, and the stability comes from the active constraints given by the model fragment function, so they are given by the set *Const*. To put these together, if a set of simple terminations does not give a successor, this must be because the results sets of those terminations are not consistent with S. Hence, the corresponding changes do not give a consistent updated of S. As a result, the problem *ELIG* reduces to *COMB*.

The results on the complexity of *COMB* presented in the previous chapters hence also extend to the problem *ELIG*. In particular, using the reduction functions, together with the procedure $P_{\text{Co-occurrence}}$, the size of the output becomes polynomial, performing polynomially many checks to the inequality reasoner in some cases: if either the number of classes of simple terminations that always appear together is logarithmic, or if the minimum cover with chains of implications of the set of terminations is bounded by a constant.

There are also two underlying assumption made within the procedure. The first is that the completeness of the inequality reasoning routine holds. We assumed that the inequality reasoner always give the correct answer, which in practice might not be the case. As explained by Bredeweg et al. (2009), this procedure is not complete, and inconsistencies might not be determined even though they might be derivable. However, the inequality reasoner is optimised so that in most cases the inconsistencies are identified. The second assumption is that the active model fragments are non-changing. For this proof, we assumed that all the active causal ingredients that are valid for the state s will always be valid in any possible successor. This is not always the case, even though, for most examples, there is little change of the active model fragments from one state or another. In fact, for the correspondence ordering rule that is currently applied in Garp3, the correspondences of the parent state are used for eliminating terminations, so this assumption already exists in the current implementation.

So far, the solutions for solving *COMB* were used for reducing the number of eligible compound terminations under the assumption of an ideal inequality reasoner, and of invariant active model fragments. The next section considers the case of dropping the second assumption and accounting for changing model fragments between states.

5.2 *ELIG* with Changing Model Fragments

Previously, the problem of eligible compound termination selection was considered by using the same active causal ingredients and relations for the successor states as for the parent state. This section investigates whether this is a correct assumption to make, and, if not, whether the ideas presented before can still be used to lower the number of combinations.

5.2.1 Stable and Unstable Constraints

Since model fragments are carrying information about the system behaviour, attention must be paid to their changes in each of their instances. As noted by Bredeweg et al. (2009), since the qualitative values and relations between parameters change from one state to another, the model fragment function might return a different set of causal ingredients. Therefore, if we label two simple combinations as being co-occurring in the context of the parent state, based on, say, an explicit equality given by a model fragment, this elimination might be incorrect because the constraint will be dropped in the next state.

For example, suppose there are two quantities, *Decreasing* and *Increasing*, both at their medium value. Suppose moreover that there is a model fragment asserting that if *Decreasing* is at the high value there is a bidirectional value correspondence between the two medium values, as shown in Figure 5.1. Assume now an initial scenario is given where the two functions are a high and low value, with negative and positive derivative respectively (see Figure 5.2). In this situation, there are two possible simple terminations, as each of the two quantities can go to their medium values. This results in three possible combinations of changes: either both, only the decreasing one, or only the increasing one become medium in value. From these only the last one is inconsistent with the system as the model fragment will be active and its constraint will not be satisfied. Therefore, two successor states are expected. However, only the successor state resulting from the first compound termination is found. The reason for this is the heuristic used by the old procedure which states that the set of constraints given by the model fragment function does not change over the states of the simulation. Using this assumption, the engine uses the value correspondence between the two medium values and, because of the correspondence ordering combination concept, infers that the simple terminations changing the values of the two functions to medium must be co-occurring.

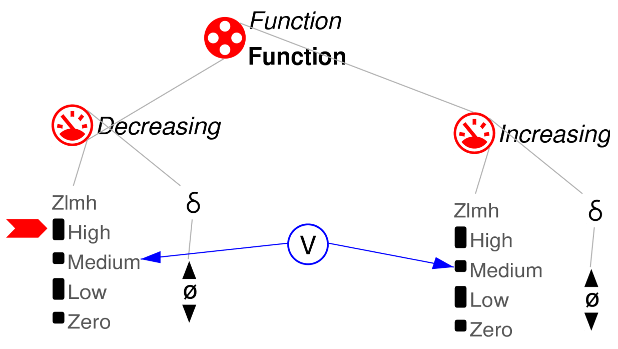


Figure 5.1: Model fragment showing the correspondence of the medium values of two functions, in the case the decreasing one is known to have a high value.

This suggests that a distinction needs to be made between the different types

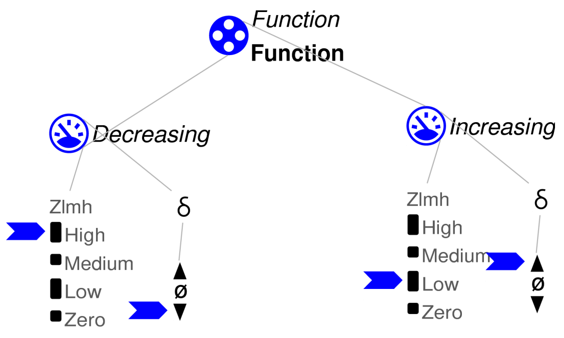


Figure 5.2: An initial scenario with two functions: a decreasing one at a high value, and an increasing one at a low value.

of constraints between parameters. Given a parent state s and a set of simple terminations $\mathcal{T}$, we say that an active constraint *cons* is *stable* if it will remain active in any transition scenario resulting from s with some compound termination in $\mathcal{P}(\mathcal{T})$. A constraint *cons* that is active in s, but could become inactive in some transition scenario is called *unstable*. For instance, the value correspondence in the example above is an unstable constraint for the initial state, as it might become not active in a following transition scenario.

Since, as shown in the example, assuming unstable constraints might result in eliminating valid successors, these should be used with caution. In Garp3, there are simulation preferences that eliminate the use of correspondence and mathematical orderings, and hence do not make any reductions based on active causal ingredients. However, this is a drastic approach, as some of the constraints are stable, and may therefore be used in the elimination of compound terminations. There are though two alternative solutions to this problem. One is to not use the unstable constraints at all. In this case, for any given state and set of simple terminations, the stable constraints are identified and are then the only ones used to invalidate combination of changes. Another solution is to group compound terminations by the (unstable) constraints active in the transition scenario that would result with that compound termination. This will result in a partition of the power set of simple terminations based on the constraints active in their resulting transition scenario. Hence, the constraints that will become active are known, and they can be used to identify illegal combinations of simple terminations within each set of the partition.

Initial analysis has shown the second solution to be too expensive for its benefits in most cases, as most model fragments active in the parent states will also remain active in the successor ones. Therefore, the next section focuses on the first solution.

5.2.2 Stable Constraints Extraction

In this section, we assume that there is a state s together with a set $\mathcal{T}$ of its simple terminations in epsilon order. Then, the program needs to identify the associated stable constraints. The resulting system of (in)equalities corresponds to the set of invariant relations used in Chapter 4. It is constructed in three steps.

Firstly, the *common scenario* for $s = \langle \mathcal{E}, val, <, \leq, = \rangle$ and $\mathcal{T}$ is found. This scenario is formed from s by keeping only the information known to be true in any possible successor state. More precisely, the common scenario is $c_{s,\mathcal{T}} = \langle \mathcal{E}, val_c, <_c, \leq_c, =_c \rangle$ where the valuation and relations are constructed from the ones in s according to Table 5.3. In this table we consider that, initially, $val_c = val$ and all (in)equalities $R_c = R$, while later these will be adapted through additions and eliminations of relations/function definitions. Notice that, since v and v' are adjacent qualitative values, the relations in the last column of the table can be written in terms of inequalities.

Cause	Eliminated relation	Added relation
Some t changes the value of Q from v to v'	$val(Q) = v$	$val_c(Q) \in \{v, v'\}$
Some t changes the value of dQ from v to v'	$val(dQ) = v$	$val_c(dQ) \in \{v, v'\}$
Some t changes the relation between Q_1 and Q_2 from R to R'	$(Q_1, Q_2) \in R$	$(Q_1, Q_2) \in R' \cup R$
Continuity on all derivatives	$val(dP) = +/$ $val(dP) = -/$ $val(dP) = 0$	$val_c(dP) \geq 0/$ $val_c(dP) \leq 0/$ $val(dP) = ?$

Table 5.3: Construction of the common scenario from a given state s and its set of simple terminations $\mathcal{T}$. Initially the valuation and relations of the common scenario are the same as in s. For each simple termination $t \in \mathcal{T}$, depending on its nature, some relations are eliminated and some are added. There is also a condition ensuring continuity on derivatives.

As an example, consider the first line of Table 5.3. This says that if in $\mathcal{T}$ there is a value termination on Q, say, $to\_point\_above(Q)$, that changes the value of the quantity, say, from $i_{0,1}$ to l_1, then the definition of the valuation $val(Q) = i_{0,1}$ is eliminated. This means that $val_c(Q)$ is undefined. Moreover, the constraint $val_c(Q) \in \{i_{0,1}, l_1\}$ is added. Formally, this means that the pairs (l_0, Q) and (Q, l_1) are added to the relation sets $<$ and $\leq$, respectively.

In the case there are multiple terminations on the same quantity, the disjunction of these will be taken. For the example before, if, besides $to\_point\_above(Q)$, $to\_point\_\ below(Q)$ is also a simple termination and both of them are assumed (i.e. the derivative of dQ is unknown, but it can have any qualitative value), then the added relations for these two terminations are $l_0 \leq val_c(Q)$ and $val_c(Q) \geq l_1$. Intuitively, the common scenario will contain all the quantities from the parent state, the valuations for the unchanging parameters, and the non-changing (in)equalities. In addition, the possible future valuation for the changing parameters are also specified.

Secondly, given the common scenario, the active model fragments are identified. This is done using the model fragment selection process from the closing phase of a state. Notice that this needs to not duplicate work. If in the ordering of the state some model fragments are found to be active for the common scenario, they will also be active for any possible successor state. Therefore, in the closing phase of that state, these model fragments will be known to be active and hence need not be re-checked.

Lastly, the stable constraints are identified. Using the active model fragments of the common scenario, there will be a resulting set of active constraints given by their consequences. These constraints might give (in)equalities between landmarks and/or parameters, as well as causal ingredients. In the case of using constrained next steps, as suggested in Chapter 3, the next-step constraints are added to this set.

At this point, a set of all the constraints that need to hold in the next step have been identified based on the model fragments that are known to be invariant. In the next subsection we investigate how this can be used in combination with the ideas for solving *COMB* in order to lower the number of eligible compound terminations.

5.2.3 Pair Constraint Ordering

Using the solution for *COMB*, as well as the stable constraints extracted as shown in the previous section, this section constructs a new combination concept. In addition, it investigates this rule's correctness and the extent under which the previously obtained results still hold.

Section 5.1 presented an algorithm for reducing the number of eligible compound termination under the assumption of non-changing model fragments. In the case this assumption is not true, given the input is $x = \langle s, mf, S_s, Cons, \mathcal{T} \rangle$, not all the constraints in $Cons$ necessarily hold in the successor states. Instead, only the stable constraints do. So, the procedure of solving *ELIG* using *COMB* should adapt the input encoding function, f, to use the set of stable constraints instead of $Cons$.

It remains to be investigated if the previous results generalise to this situation. The answer is no, as new model fragments that are valid in the successor states introduce additional constraints. Therefore, a compound termination with a corresponding set of changes giving a consistent update might give an inconsistent update after the new model fragment constraints are applied. This is problematic because if a pair constraint of the form $t_1 \to t_2$ holds, that is if for all compound terminations giving successors T, if $t_1 \in T$ then $t_2 \in T$, then it might not be identified by our procedure. This happens because c_1, the change corresponding with t_1, might lead to a new model fragment being applied, and the implication that c_2 must also be applied could now result from the constraints added by this new model fragment.

In the example with the two quantities *Decreasing* and *Increasing*, there are no stable constraints. Therefore, according to this information, no pair ordering constraint is discovered. However, $c_1 \to c_2$, where the two changes are $c_1 = (\mathit{val}(\mathit{Increasing}) = \mathit{Low}, \mathit{val}(\mathit{Increasing}) = \mathit{Medium})$ and $c_2 = (\mathit{val}(\mathit{Decreasing}) = \mathit{High}, \mathit{val}(\mathit{Decreasing}) = \mathit{Medium})$, since while the valuation of *Decreasing* is *High* the correspondence ordering in Figure 5.1 applies. This shows that, using this approach, not all existing pair constraints are guaranteed to be found. In practice though, most of the model fragments are not changing, and this means that most of the pair constraints should still be found.

By only using stable constraints it is ensured that valid compound terminations are not eliminated in this process, as constraint sets are inconsistency monotonous. The common scenario is an underspecification of the successor states, so it is not necessary that all model fragments valid in successors are found. However, if an inconsistency is derived in this underspecified system, this means there is a proof of falsehood and that proof will still be valid in an extended system. Consequently, if a pair constraint is valid for the common scenario together with the stable constraints, that pair constraint will still be valid with additional model fragments.

Combining all the observations from before, *ELIG* can be solved in the following way: firstly, the common scenario and the stable constraints are combined to make a system of (in)equalities. Secondly, the simple terminations are adapted, using function f from Section 5.2, to form a corresponding set of changes. Next, the *COMB* problem is solved, using the procedure $P_{\text{Co-occurrence}}$, to find a superset of the valid combination of changes. This is done by identifying cliques and a minimal path coverage in the co-occurrence graph. Finally, the result is translated back into sets of terminations using the procedure in g. This gives a new combination concept, which will be referred to as *pair constraint ordering.*

So, in this section, we have constructed a new ordering rule that gives a correct output even under changing model fragments. The next section showcases the benefits of this rule by means of two examples.

5.3 Using the Pair Constraint Ordering

After presenting the pair constraint ordering as a solution for reducing the number of eligible compound terminations, a natural question is how efficient this combination concept is in practice. It was already shown that if the quantities are sufficiently interdependent and there are no model fragment changes, then the number of calls to the inequality reasoner for identifying the successors of a state is reduced from exponential to polynomial.

We analyse such situations and the actual differences in the number of calls to *IR* by revisiting the two main models used in chapter 2. One is the continuous version of a two-bidder English Auction, and the other one is the large simulation investigated in section 2.4. For each of them some simulation steps are carefully considered in order to compare the performance with and without the use of pair constraint ordering.

5.3.1 English Auction Simulation with Pair Constraint Ordering

Returning to the English Auction example from Section 2.2, this section investigates what is the practical effect of adding pair constraint ordering. For this, let us consider state 7 in Figure 2.11. In that state, as shown in the figure, both quantities *Bid* have positive value and a positive derivative, as do both quantities *To_absolute*. The *Bid_difference* is at *Neg_small* with a negative derivative. The simple terminations associated with this state can be obtained from the termination validity criteria in Table 2.2, and are shown in Table 5.4. Notice that all five terminations are non-immediate, so the epsilon ordering rule is not eliminating any of them.

Label	Simple termination cause
1	$(\mathit{to\_point\_below}, \mathit{To\_abslolute_2})$
2	$(\mathit{to\_point\_above}, \mathit{Bid_1})$
3	$(\mathit{to\_point\_above}, \mathit{Bid_2})$
4	$(\mathit{to\_point\_below}, \mathit{Bid\_difference})$
5	$(\mathit{assummed\_derivative\_down\_to\_stable}, \mathit{Bid\_difference})$

Table 5.4: English Auction: simple terminations for state 7.

The first step in applying the pair constraint ordering is to construct the common scenario. To do this, the method outlined in Table 5.3 are applied, using the valuation in step 7 together with the simple terminations enumerated before. As a result, the scenario with the valuation described in Table 5.5.

Valuation in state 7	Valuation in the common scenario of state 7
$val(Bid_1) = +$	$val(Bid_1) > 0$
$val(Bid_2) = +$	$val(Bid_2) > 0$
$val(dBid_1) = +$	$val(dBid_1) \geq 0$
$val(dBid_2) = +$	$val(dBid_2) \geq 0$
$val(Bid\_difference) = Neg\_epsilon$	$0 > val(dBid_1) \geq Neg\_epsilon$
$val(dBid\_difference) = -$	$val(dBid_1) \leq 0$
$val(To\_abslolute_1) = +$	$val(To\_abslolute_1) = +$
$val(To\_abslolute_2) = +$	$val(To\_abslolute_2) \geq 0$
$val(dTo\_abslolute_1) = +$	$val(To\_abslolute_1) \geq 0$
$val(dTo\_abslolute_2) = -$	$val(To\_abslolute_2) \leq 0$

Table 5.5: English Auction: common scenario for state 7.

Let us explain the intuition behind this table. All the constraints on the derivatives of quantities are obtained from continuity; if the derivative was previously +, in the common scenario it will be greater or equal to 0. Similarly, for − and lower or equal to 0. This is because, due to continuity, in all possible successors of the state, the derivative should not have the opposite sign. Value changes can only be a result of terminations. Therefore, each quantity can either have its original value or one obtained from the termination. For example, since *Bid_1* can either remain at + or use the simple termination with cause (*to_point_above*, *Bid_1*) to change to *Max*, it is known to be greater than 0. However, as there is no simple termination for the quantity *$To\_abslolute_2$* its value is known to remain at +.

The second step is to identify the model fragments that are active for this common scenario. To do this, each model fragment is analysed and its conditions are checked. In the current example, by referring back to the model fragments presented in Subsection 2.2.2, it can be noticed that the only model fragments that are not active is the ones in figures 2.1 and 2.3. This is because a condition of that model is that the *Bid* is at *Max* which is not known to be the case for any of the two bidders.

Third, the constraints are extracted from the model fragments. Doing so means making the conditions explicit. For example, since the model fragment in Figure 2.4 is active, is known that $Bid\_difference = Bid_1 - Bid_2$. In addition, because of the proportionality relations, there is a proportionality balance constraints saying that $dBid\_difference = dBid_1 - dBid_2$. Similar reasoning gives additional relations according to the other model fragments. This, together with the landmark relations from the original scenario, gives the stable constraints.

Finally, the procedure form the combining changes problem, $P_{\text{Co-occurrence}}$, is used to identify the pair constraints and the eligible compound terminations. This results in the following steps:

- Each simple termination is checked to be consistent with the model. Consequently, termination 3 is found to be non-applicable. This uses five calls to the inequality reasoner.

  For each of the five simple terminations, the value change imposed by that termination is added to the common scenario. Then the resulting scenario together with the stable constraints are passed to the inequality reasoner. For example, for the third termination, $val(Bid_2) = Max$ is added. But, because, from the landmark relations in the scenario $Max_1 - Max_2 < Neg\_epsilon$, and $Bid\_difference \geq Neg\_epsilon$, this is not possible. Hence, $t_3 \rightarrow \neg t_3$, and the third termination is marked as contradictory and not considered.

- Construct the co-occurrence graph on the remaining terminations, while also checking for cliques. Consequently, $t_1 \leftrightarrow t_4$. This uses 7 calls to the inequality reasoner.

  This is done by adding in turn two results of simple termination to the common scenario and stable constraints, and checking for consistency. To do this the algorithm firstly takes t_1 and tries finding a $\rightarrow$ relation to some of the other three simple terminations. Only to t_4 this is possible (because of the value correspondence, see Figure 2.7). Next, it tries to extend the path, so it looks for edges from t_4 to t_2, t_5, and t_1 in this order. The only one found is to t_1. This closes a cycle; t_1, t_4, are not further considered in this step. One additional check finds no correlation between t_2 and t_5. This means that t_4 is in the class with representative t_1.

- Look for pair constraints of the type $t_i \rightarrow \neg t_j$. Consequently, $t_1 \rightarrow \neg t_5$. Done with 3 checks.

  From the 3 representative terminations check the relation of that type, with $i < j$. Because of the correspondence relation, $t_1 \rightarrow \neg t_5$ is found.

- Find the sets of eligible compound terminations using only representatives. The elements of this set are $\{t_1\}$, $\{t_1, t_2\}$, $\{t_2\}$, $\{t_2, t_5\}$, and $\{t_5\}$.

  Until now, the procedure has found that t_1 and t_4 must always appear together, and that t_3 can never be part of a compound termination giving a consistent update. This leaves us with 3 simple terminations that can be combined in any way, such that $t_1 \rightarrow \neg t_5$ holds.

- Extend these by adding the representative. This gives $\{t_1, t_4\}$, $\{t_1, t_2, t_4\}$, $\{t_2\}$, $\{t_2, t_5\}$, and $\{t_5\}$.

  For each set in in the previous step that contains t_1, the termination t_4 is also added to the set.

Therefore, this procedure returns 5 eligible compound terminations using 15 calls to the inequality reasoner. This is a very positive result taking into account that three of these eligible compound terminations result in a state. In contrast, the brute force procedure identifies 31 eligible compound terminations. Since the process of closing the state makes at least two inequality reasoning calls for each eligible compound termination, this means at least 62 calls to the inequality reasoning. Moreover, the current procedure, even when using all three combination concepts, finds 7 eligible compound terminations, requiring 14 additional calls. Those are additional since the mathematical ordering already uses the inequality reasoning. If deciding to drop this combination concept, 15 eligible compound terminations are identified.

To summarise, this subsection discussed an example of how pair constraint ordering is working in practice and that, even for a small simulation, it is beneficial. In the next subsection, the large simulation from Section 2.4, with pair constraint ordering is considered.

5.3.2 Large Simulation with Pair Constraint Ordering

In Section 2.4 the cellulose hydrolysis model of Kansou et al. (2017) was discussed from a simulation perspective. In particular, for state 3 the description (see Table 2.4) and the 11 simple terminations (see Table 2.5) were shown. It was also mentioned that the number of eligible compound terminations found by the current combination concepts is 511, while the state has only 3 successors. This is better, however, than a brute force procedure which would find 2047 compound terminations. In this section, the implications of using pair constraint ordering is discussed.

Because of the large gap between the number of possible combinations of simple terminations and the number of successors, it is obviously the case that there are dependencies between those terminations. Table 5.6 shows which simple terminations are used in the creation of each of the three successor states.

As shown in the table, terminations t_1, t_2, t_3, t_4, t_5 and t_7 are not used for the construction of any successor. Form the theoretical analysis, it should be expected that the reason for this is an inconsistency between the common scenario with the stable constraints, and each of these 6 terminations. By looking at the model fragments, one can check that this is indeed the case, and each of these is not

Successor	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
4	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓
5	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
6	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Table 5.6: Cellulose Hydrolysis: simple terminations applied to form each of the three successors of state 3.

aligned with the known commonalities of successors. This leaves only 5 states to be combined for making compound terminations. Hence, with only 11 calls for the inequality reasoner, one for each simple termination, the number of eligible compound ones was reduced from 511 to 31.

From Table 5.6, it should be derived that $t_8 \leftrightarrow t_9 \leftrightarrow t_{10}$ is also valid. However, because of the complexity of the model, we did not manage to pinpoint the precise model fragment giving each of this constraints. But, a model fragment giving the constraint $t_8 \to t_{11}$ was found. This will further reduce the number of compound terminations from 31 to 23.

In point of calls to the inequality reasoner the compression is immediate. Using the combining changes concept, 11 calls are used to find the inconsistent terminations, and after this at most $2 \cdot 5^2 = 50$ more calls are used for constructing the rest of the co-occurrence graph. Since the current procedure identifies 511, at least 1022 calls are due in the closing state of the state. Hence, this combination concept is beneficial for this example as well.

5.4 Conclusion

This chapter, using the axiomatisation constructed in Chapter 2, adapts the procedure for solving the combining changes problem (i.e. *COMB*, see previous chapter) in order to address the problem of lowering the number of eligible compound terminations (*ELIG*). That is, is done in the ordering phase of states, so it is assumed that a state together with its simple terminations is given. In doing so, three main contributions should be mentioned.

First, it was found that in the case of invariant active model fragments over states, there is a reduction from *ELIG* to *COMB*. This means that a procedure for solving *ELIG* can be found by encoding the input into one for *COMB*, then using the procedure from Chapter 4, and finally decode the result into one for *ELIG*. These two polynomial time-computable functions for encoding and decoding are presented in detail in the first section. This means that all the theoretical results found in the previous chapter are also valid for *ELIG* with a non-changing model.

Hence, in the case of systems with high dependencies between quantities, this problem can be solved with polynomially many calls to the inequality reasoner.

Second, this method was adapted to account for changing model fragments. It was firstly shown that in the case of changing model fragments the reduction is not possible any more. This motivated finding an alternative procedure. From the relations known to hold in all successors of a state, a common scenario is built. Using this together with the model fragment selection procedure, the model fragments known to be active in all future states are identified. These give a set of stable constraints. Next, the same procedure described in the previous paragraph is used, but this time it is performed only on the stable constraints instead of on the ones from the states active model fragments. This combination concept is referred to as *pair constraint ordering*. Using this routine, however, the theoretical results do not extend. But, since in practice systems are highly dependent, model fragments should not change much, and, therefore, good results are still expected.

Third, two examples are considered to test this hypothesis. One, which was discussed into much detail, was the English Auction model. Using this model, the number of eligible compound terminations for some state, namely 7, was 5, which is less than 32, the number of terminations found by the brute force procedure, and also less than 7, the number of ones found by the current procedure. It is also shown to use less calls to the inequality reasoner throughout the phases of this state. The other example was the large model of Cellulose Hydrolysis from Section 2.5. The results of this example were equally encouraging, as the number of eligible compound terminations was found to drop from 511, which is the number obtained by the current procedure, to 23.

Consequently, this chapter gives a method that uses the sparse knowledge about a system to infer more information. More precisely, it uses the known commonalities between successors in order to lower the number of eligible compound terminations, and hence faster identifying them. In addition, it further shows the value of the axiomatisation.

Chapter 6

Discussion and Conclusion

In its aim of addressing the research question of maximising the information inferable from sparse knowledge in process-oriented qualitative reasoning, this thesis made several advances in the use of incomplete information for qualitative models and simulations.

Chapter 2 provides partial axiomatisation of process-oriented qualitative reasoning. This axiomatisation is based on earlier formalisations, but differs in a few key aspects. For model construction, it follows the approach of Liem (2013), but still uses precise notation similar to the one of Weld (1988). The main difference from earlier attempts is the focus on simulation. Instead of opting for purpose oriented definitions of state graphs, our axiomatisation uses the work of Bredeweg et al. (2009) and Linnebank (2004) in order to give precise notions for simulation results. As a result, using these notions, a state graph for a given qualitative model and scenario can be constructed, but there are also formal notations allowing for theoretical problem analysis. In clarifying these newly introduced definitions, different examples were used, including a novel one from the field of Auction Theory, and a large one from the work of Kansou et al. (2017). This axiomatisation was extensively used throughout the remaining chapters of the thesis as a theoretical basis.

Chapter 3 uses the clarity derived from the axiomatisation to pinpoint some types of results and introduce three types of consistency principles. These principles were the following. First, all models need to be model consistent, that is each state and transition should respect the information provided by model fragment. This means both in terms of dependencies, i.e. dependency consistency, and relations, i.e. relation consistency. Second, all state graphs formed by qualitative simulations should be path consistent, meaning that each path in the graph should describe a coherent development of the system. In particular, each track should not contain contradictory assumptions, i.e. be assumption consistent, and should properly describe developments with local minima and maxima, i.e. be extrema consistent. Third, each behaviour graph should be inexplicit inequality consistent in the sense that coherent behaviour should result even when not all inequalities are explicitly stated. All of these principles were accompanied by particular examples where they occurred, and by suggested solutions.

Chapter 4 investigates the problem of combining changes in dynamic systems of (in)equalities as a generalisation of the problem of selecting the eligible compound terminations in the ordering phase of a state. In this chapter, it was considered that a consistent set of inequality relations together with a set of changes on these is given; a change was defined simply as an ordered pair of relations, with the meaning that, if the change is applied to a set, the first relation in the pair will be substituted with the second one. The combining changes problem (i.e. *COMB*) aims at finding some superset of combinations of changes that give consistent updates of the given system. Throughout this chapter, it was assumed that the inequality reasoning problem, which checks system consistency, is provided as an oracle. Under this assumption and using the concept of pair constraints, *COMB* was proven to be polynomial in two cases: if the number of cliques in the co-occurrence graph, which is formed from pair constraints, is logarithmic in the size of the input, or if the path cover number of the same graph is bounded by a constant. To prove these claims several classical results from complexity theory were used (Boesch and Gimpel, 1977; Arora and Barak, 2009). A procedure improving the complexity constants of the routine for solving *COMB* was presented at the end of the chapter.

Chapter 5 investigates the extent to which the previously introduced procedure for solving *COMB* can be applied for the problem of lowering the number of eligible compound terminations (i.e. *ELIG*). This is problem is difficult because data is sparse, so not all derivable relations in the system are made explicit. Consequently, if existing, contradictions are rarely obvious, and spotting usually require using the inequality reasoner. In the case of invariant model fragments, a reduction from *ELIG* to *COMB* was found. Moreover, an adaptation of the procedure for solving *COMB* was developed in order to account for simulations with changing model fragments. To illustrate these methods together with their practical benefits, the two main examples in Chapter 2 are used again and the changes in the numbers of eligible compound terminations is analysed.

Consequently, we made several contributions towards our original aim of maximising the amount of information derivable from sparse knowledge in process-oriented qualitative reasoning. First, we developed and showed the usefulness of a new partial axiomatisation providing the means for theoretical analysis of the simulation process. Second, we identified incoherences in simulation results, and suggested solutions for these. Third, we introduced the problem of *combining changes* which, in the context of a dynamic system of (in)equalities, aims at using the available incomplete knowledge in order to identify the combinations of changes that maintain the system's consistency. Fourth, we analysed this problem and showed that under some explicit assumption it is time polynomial. Fifth, we generalised the procedure found during the theoretical analysis to work for better using the knowledge available, and reducing the number of eligible compound terminations in the ordering phase of a state.

There are various directions for future research based on the work of this thesis. To begin with, as already mentioned, our axiomatisation is only partial. Extending it would be very useful as it might pinpoint new issues and help identifying novel solutions. In addition to this, the list of incoherences provided in Chapter 3 is by no means exhaustive, so further investigation may undercover additional consistency concepts. Moreover, from a practical point of view, implementing the ones already mentioned would be valuable. Regarding the analysis of *COMB*, its complexity was only analysed for pair constraints and with *IR* as an oracle, but it is natural to wonder what happens in other cases. For example, one could use the relations in the system to underpin more complex dependencies between multiple changes. Finally, it would be useful to provide a procedure for automated evaluation of the practical benefits for the methods in Chapter 5.

Bibliography

K. H. Abbott, P. C. Schutte, M. T. Palmer, and W. R. Ricks. Faultfinder: A diagnostic expert system with graceful degradation for onboard aircraft applications. 1988.

S. Alessi. Building versus using simulations. In *Integrated and holistic perspectives on learning, instruction and technology*, pages 175–196. Springer, 2000.

S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

F. T. Boesch and J. F. Gimpel. Covering points of a digraph with point-disjoint paths and its application to code optimization. *Journal of the ACM (JACM)*, 24(2):192–198, 1977.

B. Bredeweg and K. D. Forbus. Qualitative modeling in education. *AI magazine*, 24(4):35, 2003.

B. Bredeweg and P. Salles. Mediating conceptual knowledge using qualitative reasoning. *WIT Transactions on State-of-the-art in Science and Engineering*, 34, 2009.

B. Bredeweg, F. Linnebank, A. Bouwer, and J. Liem. Garp3workbench for qualitative modelling and simulation. *Ecological informatics*, 4(5-6):263–281, 2009.

J. De Kleer. Multiple representations of knowledge in a mechanics problem-solver. In *Readings in qualitative reasoning about physical systems*, pages 40–45. Elsevier, 1990.

S. L. Graham, P. B. Kessler, and M. K. McKusick. Gprof: A call graph execution profiler. *ACM SIGPLAN Notices*, 39(4):49–57, 2004.

J. E. Hopcroft and R. M. Karp. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

K. Kansou, C. Rémond, G. Paës, E. Bonnin, J. Tayeb, and B. Bredeweg. Testing scientific models using qualitative reasoning: Application to cellulose hydrolysis. *Scientific reports*, 7(1):14122, 2017.

R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

D. Konig. Gráfok és mátrixok. matematikai és fizikai lapok, 38: 116–119, 1931, 1931.

B. Kuipers. *Qualitative reasoning: modeling and simulation with incomplete knowledge.* MIT press, 1994.

J. Liem. *Supporting conceptual modelling of dynamic systems: A knowledge engineering perspective on qualitative reasoning.* Universiteit van Amsterdam [Host], 2013.

F. E. Linnebank. Common sense reasoning-towards mature qualitative reasoning engines. 2004.

R. P. McAfee and J. McMillan. Auctions and bidding. *Journal of economic literature*, 25(2):699–738, 1987.

P. R. Milgrom and R. J. Weber. A theory of auctions and competitive bidding. *Econometrica: Journal of the Econometric Society*, pages 1089–1122, 1982.

F. G. Pin, H. Watanabe, J. Symon, and R. S. Pattay. Autonomous navigation of a mobile robot using custom-designed qualitative reasoning vlsi chips and boards. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 123–128. IEEE, 1992.

F. van Harmelen, V. Lifschitz, and B. Porter. *Handbook of knowledge representation*, volume 1. Elsevier, 2008.

P. Veber, M. Le Borgne, A. Siegel, S. Lagarrigue, and O. Radulescu. Complex qualitative models in biology: A new approach. *Complexus*, 2(3-4):140–151, 2004.

W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

D. S. Weld. Theories of comparative analysis. Technical report, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1988.

D. S. Weld and J. De Kleer. *Readings in qualitative reasoning about physical systems.* Morgan Kaufmann, 2013.

Appendices

Appendix A

Axiomatisation for Process-Oriented Qualitative Reasoning

- *Entities* are the basic structural compounds of the system which do not change during the simulation phase. Formally, we define an entity as a finite set of quantities, $E = \{Q_1, ..., Q_n\}$.

- A *quantity* Q is defined as a variable, which has another associated variable named *(first order) derivative*, denoted by dQ. Moreover, if $\mathcal{Q}$ is a set of quantities, then the set of associated derivatives is denote by $d\mathcal{Q}$, i.e. $d\mathcal{Q} = \{dQ | Q \in \mathcal{Q}\}$.

- A *parameter* is defined inductively; any quantity is a parameter, and so are the derivatives of any parameter. For practical reasons, not all order derivatives are considered. In general, for a given set of quantities, the associated parameters are only these quantities together with their first order derivatives, that is $\mathcal{P}_{\mathcal{Q}} := \mathcal{Q} \cup d\mathcal{Q}$. Notice that this is different from the usual definition of a parameter (Weld and De Kleer, 2013).

- *Landmarks* are considered constants that can be partially ordered by the strict relation $<$. Two landmarks l_u, l_v within the set $\mathcal{L}$ of all landmarks are considered consecutive if $l_u < l_v$ and there is no $l_t \in \mathcal{L}$ such that $l_u < l_t < l_v$. As always, given a set of landmarks L, a minimal landmark is an l_1 such that there is no $l_0 \in L$ with $l_0 < l_1$ (and similarly for the maximal).

- *An interval* is also a constant which is associated either with two consecutive landmarks or with a minimal or maximal landmark. So, for the consecutive landmarks $l_u < l_v$, the associated interval will be $i_{u,v}$. If l_1 is minimal, then $i_{0,1}$ is its associated interval, while if l_u is maximal it will have $i_{u,u+1}$ as an associated interval. The partial order $<$ also extends over intervals, so $l_u < i_{u,v} < l_v$. However, the relation is not strict over intervals, as $i_{u,v} < i_{u,v}$ is valid.

- The *landmark space* associated with a quantity Q is defined as a set of landmarks totally ordered by $<$, $L_Q = \{l_1 < l_2 < ... < l_m\}$ that quantity Q can take, while the *interval space* of Q is the set of intervals associated with these landmarks, that is $I_Q = \{i_{1,2}, i_{2,3}, ..., i_{m-1,m}\}$ possibly together with one or both of $i_{0,1}$ and $i_{m,m+1}$. The *magnitude space* of Q is hence the set of landmark and interval spaces of Q, i.e. $M_Q = L_Q \cup I_Q$; e.g. for any parameter P, $L_{dP} = \{0\}$, $I_{dP} = \{-, +\}$, and $M_{dP} = \{- < 0 < +\}$.

- *Causal ingredients* are binary relation, which can be of three types:

  - *Proportionality* (which can be either positive or negative) models a correlation between the tendencies of two quantities; if Q_1 is positively proportionally influenced by Q_2, Q_1 has no other dependencies, and Q_2 is increasing, then Q_1 should also be increasing. The two types of proportionality are denoted by $P+$ or $P-$, and each of them is an antisymmetric binary relation on $\mathcal{Q}$.

  - *Influence* (which can be positive or negative) models the dependency between the magnitude of a quantity and the tendency of another; if Q_1 is positively (directly) influenced by Q_2, Q_1 has no other dependencies, and Q_2 is positive, then Q_1 should also be increasing. Formally, the two types of influences are denoted by $I+$ or $I-$, and each of them is an antisymmetric binary relation on $\mathcal{Q}$.

    We also say that a *dependency* is either an influence or a proportionality relation.

  - *Correspondence*, which can be directed or undirected, shows co-appearing magnitudes. These are elements of the *(value) correspondence set* C, which is such that

    $$C \subseteq \{((P_1, v_1), (P_2, v_2)) | P_i \in \mathcal{P}_{\mathcal{Q}}, v_i \in M_{P_i}, P_1 \neq P_2\}.$$

    Additionally, the undirected correspondence is symmetric. However, since the undirected correspondence can be expressed in terms of directed correspondences we will only use the latter, to which we will refer to simply as correspondences.

- *(In)equalities* are binary relations $R \in \{<, \leq, =, \geq, >\}$; for a given set of quantities $\mathcal{Q}$, the elements of the relations R are of the form:

  - (Q_1, Q_2), for $Q_1, Q_2 \in \mathcal{Q}$, i.e. comparing two quantities;
  - (Q, l), for $l \in L_Q$, i.e. comparing a quantity and a landmark;
  - (l, l'), for $l, l' \in L_Q$, $Q \in \mathcal{Q}$, i.e. comparing two landmarks;
  - (dQ_1, dQ_2), for $Q_1, Q_2 \in \mathcal{Q}$, i.e. comparing two derivatives;

  - $(dQ, 0)$, for $Q \in \mathcal{Q}$, i.e. comparing a derivative and zero.

  This is also extended to inequalities on terms, that is on expressions obtained from parameters together with arithmetic operators.

- An *assertion* attributes values to parameters:
  - A *magnitude assertion* for the set of quantities $\mathcal{Q}$ is a (partial) function $mag : \mathcal{Q} \to \cup_{Q \in \mathcal{Q}} M_Q$, such that $mag(Q) \in M_Q$ for all $Q \in \mathcal{Q}$ and $mag(Q)$ defined. Hence, a magnitude assertion for $\mathcal{Q}$ is a function that gives to (some) quantities in $\mathcal{Q}$ some magnitude within their associated magnitude space.
  - A *derivative assertion* for the set of quantities $\mathcal{Q}$ is a (partial) function $der : d\mathcal{Q} \to \{-, 0, +\}$.
  - A *(qualitative) value assertion* for the set of quantities $\mathcal{Q}$ is a (partial) function $val : \mathcal{Q} \cup d\mathcal{Q} \to \cup_{Q \in \mathcal{Q}} M_Q \cup \{-, 0, +\}$ such that $val_m := val|_{\mathcal{Q}}$ is a magnitude assertion, and $val_d := val|_{d\mathcal{Q}}$ is a derivative assertion.

- A *scenario* is a partial qualitative description of a situation. Formally, a scenario s is a tuple $\langle \mathcal{E}, val, <, \leq, = \rangle$, where $\mathcal{E}$ is a set of entities, val is a qualitative value assertion for $\mathcal{Q}$, and $<, \leq, =$ are inequalities on $\mathcal{E}$. In addition to this, if $val(P_1)$ and $val(P_2)$ are defined, and $P_1 \; R \; P_2$ for some $R \in \{<, \leq, =\}$, then the relation between P_1 and P_2, if specified, should be $val(P_1) \; R \; val(P_2)$, where P_1, P_2 are parameters in $\mathcal{E}$, formally in $\mathcal{P}_{\cup_{E \in \mathcal{E}} E}$.

- The *behaviour ingredients* of a scenario are the relations and the value assertion function. That is, the behaviour ingredients of $s = \langle \mathcal{E}, val, <, \leq, = \rangle$ are $val, <, \leq, =$.

- The *structural ingredients* of a scenario are the set of its entities (which also contains information about quantities), together with agents, configurations, attributes, and assumptions. The last four types are beyond the scope of this thesis, so we will not insist on these.

- The *conditionals* of a scenario are its structural and behavioural ingredients.

- A *model fragments* (MF) is used to describe how parts of the system behave under certain conditions. These are then rules that link a set of conditionals to a set of causal ingredients and possibly (in)equality restrictions. Hence, they can be determined by a function, say the *model fragment function* $mf : \mathcal{P}(Cond) \to \mathcal{P}(Caus)$, where $Cond$ is the set of all possible conditionals, and $Caus$ is the set of causal ingredients together with (in)equality relations.

- Therefore, for every state there are associated proportionality, influence and correspondence relations, plus possible (in)equality restrictions, that are given by the MF function. Those will be referred to as the set of *active causal ingredients*, which, for a scenario s, is given by $\bigcup_{Co \subseteq Cond_s} mf(Co)$.

- A *state* represents a complete version of a scenario, i.e. the qualitative value assertion restricted on quantities (val_m) is a total function. Furthermore, a state s extends a scenario s' if all the elements of s are supersets of the respective ones in s'. A scenario is called *ambiguous* if it can have more than one state extension.

- A *simple termination* for a state s is a pair (c, Re), where c is a cause, and Re is a set of results. Moreover, the pair needs to satisfy the termination validity criteria which will be defined later. The set of results is a collection of relations on the parameters of s, while the different types of causes are defined below.

- A *cause* c of a termination for a state $s = \langle \mathcal{E}, val, <, \leq, = \rangle$ is a pair that gives the type of change (i.e. a constant from Table 2.1) and a quantity (or a pair of quantities) from $\cup_{E \in \mathcal{E}} E$, i.e. $c = (cause\_name, Q)$, for some Q *in* $\cup_{E \in \mathcal{E}} E$. Table A.1 presents the causes grouped into three possible categories. Formally, there are four categories, as there are also exogenous terminations on derivatives (Bredeweg et al., 2009), but these necessitate additional constraints, and are outside the scope of this thesis. In additon, for the derivative and inequality causes, there are also assumed versions of the constants, e.g. ($assumed\_derivative\_stable\_to\_down$, Q).

- As stated before, in order for a pair $t = (c, Re)$ to be a termination of state $s = \langle \mathcal{E}, val, <, \leq, = \rangle$, t needs to satisfy the *termination validity criteria.* This means that only certain pairs of causes and results are valid. Moreover, these are valid for the state s only when a certain set of conditions (constraints) of s is valid. The *condition function* of s, denoted by *cond*, is a function from the set of simple terminations to the power set of constraints of s. Hence, $cond(t)$ is the set of constraints for the termination t. Table A.2 outlines the termination validity criteria. This is based on the work of Linnebank (2004).

- The set of terminations is divided into two parts, namely *immediate terminations*, which are from a point or from equality, i.e. the ones marked with * in Table A.1, and *non-immediate terminations*, which are to a point or to equality. Hence, the set of simple terminations $\mathcal{T}_s$ is the disjoint union of the set of immediate terminations $\mathcal{I}_s$ and the set of non-immediate terminations $\mathcal{N}_s$. That is, $\mathcal{T}_s = \mathcal{I}_s \dot{\cup} \mathcal{N}_s$.

- A *compound termination* for a state s is defined to be a set of simple terminations of s. Since immediate terminations take priority over non-immediate

Class of causes	Type of cause
Value causes	$(\mathit{to\_point\_above}, Q)$
	$(\mathit{to\_point\_below}, Q)$
	$(\mathit{to\_interval\_above}, Q)$*
	$(\mathit{to\_interval\_below}, Q)$*
Derivative causes	$(\mathit{derivative\_stable\_to\_down}, Q)$*
	$(\mathit{derivative\_stable\_to\_up}, Q)$*
	$(\mathit{derivative\_down\_to\_stable}, Q)$
	$(\mathit{derivative\_up\_to\_stable}, Q)$
Inequality causes	$(\mathit{from\_equal\_to\_greater}, (Q_1, Q_2))$*
	$(\mathit{from\_equal\_to\_smaller}, (Q_1, Q_2))$*
	$(\mathit{from\_smaller\_to\_equal}, (Q_1, Q_2))$
	$(\mathit{from\_greater\_to\_equal}, (Q_1, Q_2))$
	$(\mathit{from\_smaller\_or\_equal\_to\_smaller}, (Q_1, Q_2))*$
	$(\mathit{from\_smaller\_or\_equal\_to\_equal}, (Q_1, Q_2))$
	$(\mathit{from\_smaller\_or\_equal\_to\_greater}, (Q_1, Q_2))*$
	$(\mathit{from\_greater\_or\_equal\_to\_greater}, (Q_1, Q_2))*$
	$(\mathit{from\_greater\_or\_equal\_to\_equal}, (Q_1, Q_2))$
	$(\mathit{from\_greater\_or\_equal\_to\_smaller}, (Q_1, Q_2))*$

Table A.1: Types of causes

one, a compound terminations only has elements from the immediate terminations, if such elements exist, otherwise it is composed by non-immediate terminations. Formally,

$$T \subseteq \begin{cases} \mathcal{I}_s\texttt{, if } \mathcal{I}_s \neq \emptyset \\ \mathcal{N}_s\texttt{, otherwise} \end{cases},$$

where $\mathcal{I}_s$ is the set of immediate terminations of s, $\mathcal{N}_s$ is the set of non-immediate ones, and T is a compound termination. This rule of prioritising immediate terminations will be referred to as the *epsilon ordering rule.*

The set of compound terminations of s is denoted by $\overline{\mathcal{T}_s}$. The set of causes of the compound termination T is the set of causes that belong to the simple terminations within T, that is $\{c|(c, Re) \in T\}$. Similarly, the results of T is the union of the result sets of the simple terminations, i.e. $\cup_{(c,Re)\in T} Re$. Moreover, the condition function naturally extends over compound terminations by letting $\mathit{cond}(T) = \cup_{t\in T}\mathit{cond}(t)$.

- The physical world is continuous, in the sense that there are no jumps in

Cause type	Elements of $cond(t)$	Elements of results set
(to_point_above, Q)	$val(Q) = i_{u,u+1}$, for $i_{u,u+1} \in I_Q$ $val(dQ) = +$	$val(Q) = l_{u+1}$ $val(dQ) \geq 0$
(to_point_below, Q)	$val(Q) = i_{u,u+1}$, for $i_{u,u+1} \in I_Q$ $val(dQ) = -$	$val(Q) = l_u$ $val(dQ) \leq 0$
$(to\_interval\_above, Q)$	$val(Q) = l_u$, for $l_u \in L_Q$ $val(dQ) = +$	$val(Q) = i_{u,u+1}$ $val(dQ) \geq 0$
$(to\_interval\_below, Q)$	$val(Q) = l_{u+1}$, for $l_{u+1} \in L_Q$ $val(dQ) = -$	$val(Q) = i_{u,u+1}$ $val(dQ) \leq 0$
$(derivative\_stable\_to\_down, Q)$	$val(dQ) = 0$ $val(ddQ) = -*$	$val(dQ) = -$ $val(ddQ) \leq 0*$
$(derivative\_stable\_to\_up, Q)$	$val(dQ) = 0$ $val(ddQ) = +*$	$val(dQ) = +$ $val(ddQ) \geq 0*$
$(derivative\_up\_to\_stable, Q)$	$val(dQ) = +$ $val(ddQ) = -*$	$val(dQ) = 0$ $val(ddQ) \leq 0*$
$(derivative\_down\_to\_stable, Q)$	$val(dQ) = -$ $val(ddQ) = +*$	$val(dQ) = 0$ $val(ddQ) \geq 0*$
$(from\_equal\_to\_greater, (Q_1, Q_2))$	$Q_1 = Q_2$ $dQ_1 > dQ_2*$	$Q_1 > Q_2$ $dQ_1 \geq dQ_2*$
$(from\_equal\_to\_smaller, (Q_1, Q_2))$	$Q_1 = Q_2$ $dQ_1 < dQ_2*$	$Q_1 < Q_2$ $dQ_1 \leq dQ_2*$
$(from\_smaller\_to\_equal, (Q_1, Q_2))$	$Q_1 < Q_2$ $dQ_1 > dQ_2*$	$Q_1 = Q_2$ $dQ_1 \geq dQ_2*$
$(from\_greater\_to\_equal, (Q_1, Q_2))$	$Q_1 > Q_2$ $dQ_1 < dQ_2$	$Q_1 = Q_2$ $dQ_1 \leq dQ_2*$

Table A.2: Termination validity criteria. Note that the inequality transitions from a weak constraint to a strong one were omitted, but these are similar to the mentioned ones. In addition, if the constraints marked with star do not appear explicitly, but could be added while maintaining consistency of the state, then the cause is assumed.

tendencies between consecutive states of the same system. So, the *continuity criterion* states that the tendency of a quantity cannot jump from the positive to the negative interval or vice-versa without first passing through zero. Another way of saying this is that a pair of states (s_1, s_2) satisfies the continuity criterion iff for every parameter P resulting from the entity spaces of the two states the following holds:

– if $val_1(dP) = -$, then $val_2(dP) \leq 0$
– if $val_1(dP) = +$, then $val_2(dP) \geq 0$

- A *transition scenario* from state s with the compound termination T is then defined as the scenario s' obtained from s by changing with the results in T, that is with $Re_T = \cup_{(c,Re)\in T} Re$.

- A state $s = \langle \mathcal{E}, val, <, \leq, = \rangle$ with its active causal ingredients $P+, P-, I+, I-, C$ is *stable* if:

  – There is an influence balance. This happens when the qualitative value assertion *val* is plausible with the influence relation of s. Formally, that is when for every $Q \in \cup_{E\in\mathcal{E}} E$, the relation

  $$dQ = \sum_{(Q_1,Q)\in I+} f_1(Q_1) - \sum_{(Q_2,Q)\in I-} f_2(Q_2),$$

  where the functions f_i are strictly increasing and with zero as a fixed point, holds.

  – There is a proportionality balance. Similarly, this happens when the qualitative value assertion *val* is plausible with the proportionality relation of s. Formally, that is when for every $Q \in \cup_{E\in\mathcal{E}} E$, the relation

  $$dQ = \sum_{(Q_1,Q)\in P+} g(dQ_1) - \sum_{(Q_2,Q)\in P-} g(dQ_2).$$

  where the functions g_i are strictly increasing and with zero as a fixed point, holds. Notice that in this case, because of the possible values of the derivatives, the relation is equivalent with $val(dQ) = \sum_{(Q_1,Q)\in P+} val(dQ_1) - \sum_{(Q_2,Q)\in P-} val(dQ_2)$.

  – There is a correspondence fit. This happens when the qualitative value assertion *val* is plausible with the correspondence relation of s. Formally, that is when for every $P_1, P_2 \in \cup_{E\in\mathcal{E}} E$ and $v_i \in M_i$, such that $((P_1, v_1), (P_2, v_2)) \in C$, if $val(P_1) = v_1$, then $val(P_2) = v_2$.

Notice that the influence and proportionality balances are separate, and not mixed. The reason for this is that in qualitative reasoning, the dependencies are interpreted as causalities, so a quantity cannot have both incoming influences and proportionalities.

- A *successor* of a state s is a continuous and stable state s', extending the transition scenario of s and some compound termination T.

- A *transition* is a pair of successor states (s, s').
- A *state graph* (or behaviour graph) is a directed graph $G = (V, E)$ where V is a set of states and E is the set of transitions between them. Moreover, any behaviour graph must be maximal, in the sense that there is no state in V with a successor outside V.
- A path in this state graph, which shows a possible development of the system, is referred to as a *track*.
- A *simulation* is a function that for a given model fragment function and initial scenario returns a representation of the possible developments of the system, i.e. a state-graph.