Compositional Ecological Modelling via Dynamic Constraint Satisfaction with Order-of-Magnitude Preferences

Jeroen Keppens

Doctor of Philosophy

Centre for Intelligent Systems and their Applications
Division of Informatics
University of Edinburgh

2002

Abstract

Compositional modelling is one of the most important knowledge-based approaches to automating domain model construction. However, its use has been limited to physical systems due to the specific presumptions made by existing techniques. Based on a critical survey of existing compositional modellers, the strengths and limitations of compositional modelling for its application in the ecological domain are identified and addressed.

The thesis presents an approach for effectively building and (re-)using repositories of models of ecological systems, although the underlying methods are domain-independent. It works by translating the compositional modelling problem into a dynamic constraint satisfaction problem (DCSP). This enables the user of the compositional modeller to specify requirements to the model selection process and to find an appropriate model by the use of efficient DCSP solution techniques.

In addition to hard dynamic constraints over the modelling choices, the ecologist/user of the automated modeller may also have a set of preferences over these options. Because ecological models are typically gross abstractions of very complex and yet only partially understood systems, information on which modelling approach is better is limited, and opinions differ between ecologists. As existing preference calculi are not designed for reasoning with such information, a calculus of partially ordered preferences, rooted in order-of-magnitude reasoning, is also devised within this dissertation.

The combination of the dynamic constraint satisfaction problem derived from compositional modelling with the preferences provided by the user, forms a novel type of constraint satisfaction problem: a dynamic preference constraint satisfaction problem (DPCSP). In this thesis, four algorithms to solve such DPCSPs are presented and experimental results on their performance discussed.

The resulting algorithms to translate a compositional modelling problem into a DCSP, the order-of-magnitude preference calculus and one of the DPCSP solution algorithms constitute an automated compositional modeller. Its suitability for ecological model construction is demonstrated by applications to two sample domains: a set of small population dynamics models and a large model on Mediterranean vegetation growth. The corresponding knowledge bases and how they are used as part of compositional ecological modelling are explained in detail.

Acknowledgements

This dissertation, and the work that led up to it, would not have been possible without the help and support of many people.

First and foremost, I would like to thank my principal supervisor, Dr. Qiang Shen, for encouraging me to take up this project, for his friendship and guidance throughout this project, and for always being prepared to provide detailed and constructive comments on my work. I am also much indebted to my second supervisor, Dr. Robert Muetzelfeldt, for introducing me to ecological modelling and for his advice and help during this project.

The work presented herein has been shaped by many useful discussions within the Approximate and Qualitative Reasoning group. I would like to thank all the group members, past and present, for the supportive environment they have helped to create and for the inspiration provided. I am particularly grateful to Dr. Ian Miguel for proofreading this thesis and his invaluable advice.

Many thanks to the Faculty of Science and Engineering at the University of Edinburgh, for funding this work by means of a faculty scholarship. Without their financial support, this work would not have been possible.

Last but not least, I would like to extend my gratitude to my family and friends, for their encouragement and support throughout this project.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jeroen Keppens*)

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ATMS	assumption-based truth maintenance system
ALTMS	assumption literal based truth maintenance system
B&B	branch and bound
BPQ	basic preference quantity
CM	compositional modelling
CMF	composite model fragment
CML	compositional modelling language
CSP	constraint satisfaction problem
DCSP	dynamic constraint satisfaction problem
DFS	depth first search
DME	Device modelling environment
DPCSP	dynamic preference constraint satisfaction problem
DVCSP	dynamic valued constraint satisfaction problem
GDE	general diagnostic engine
GoM	graph of models
MPB	maintaining preference boundaries
MPB-FC	maintaining preference boundaries by forward checking
ODE	ordinary differential equation
OMP	order of magnitude preference
OMR	order of magnitude reasoning
QDE	qualitative differential equation
QID	qualitative influence diagram
QPE	qualitative process engine
QPT	qualitative process theory
QSIM	qualitative simulation

SQPC	semi-quantitative physics compiler
SQUID	semi-quantitative system identification
TRIPEL	tailoring relevant influences for predictive and explanatory leverage
VCSP	valued constraint satisfaction problem

Chapter 1

Introduction

Modelling plays a central role in intelligent problem solving. Theoretical developments in symbolic artificial intelligence since as early as 1968 have established that modelling is at least as important as building problem solving techniques themselves [2]. Before a problem solver can be applied to a particular problem, one is first confronted with the task of reformulating the problem at hand, such that the chosen problem solver can be applied.

More specifically, intelligent problem solving is usually accomplished through three steps. First, it is necessary to commit to the most efficient and effective problem solver by choosing the problem representation formalism. An appropriate representation of the problem at hand can then be constructed by means of the chosen formalism. This second step is called model construction and the resulting problem representation is usually termed the model of the problem. Finally, an adequate problem solving algorithm is applied to the model. In the broad sense, modelling involves the first two issues: choice of representation and model construction. Research in automated modelling to date focuses on model construction however, since this is a prerequisite to resolving modelling as a whole. Therefore, the remainder of this thesis focuses on model construction or modelling in the narrow sense.

Model construction is itself a complex problem. Because there is in general no single correct model [105], model construction involves more than merely syntax transformation. Modelling also requires reasoning about the aims and intentions of the problem solver, and the most effective means of applying the decisions about these to model construction, whilst being economical with the available resources. Thus, mod-

elling is far from a trivial task and forms an inherent part of intelligent problem solving. Automating this process would provide considerable opportunities for the associated problem solvers. The main advantage of such automation is the increased versatility of the resulting systems. Firstly, an automated modeller takes over most of the model construction effort, allowing the user to concentrate on problem specifications and alternative solutions. Secondly, a more gradual decrease in performance with problems beyond the original specifications can be achieved because the problem solvers may be able to adapt the model with minimal user intervention when circumstances change. This may involve recoding the model to reflect different user requirements or working environments, or revising the model itself with respect to changes in the original system configuration.

Several different classes of automated modellers have been proposed, specialised with regard to different types of task and architecture. This work employs one of the most successful paradigms of automated model construction: compositional modelling [45, 96]. A compositional modeller is essentially a knowledge based approach to model construction. Its input consists of a formal high-level description of the system and a specification of the requirements of the model to be constructed, and its knowledge base contains composable pieces of knowledge called model fragments. Each of these model fragments includes a model for a certain part of the problem domain. This part may constitute a sub-system or component of the system being modelled, or a process that occurs within the system. The inference engine instantiates these model fragments and searches for the most appropriate combination of them. The appropriateness of a particular combination depends on relevance and resource economics, of course.

Most work in automated model construction is aimed at modelling physical systems. Although this area is comprised of a wide variety of different types of system, there are a number of features that they all have in common. Physical systems consist of a number of components and/or processes. The laws that govern the behaviour of these components/processes are well-understood because the underlying first principles have been derived by means of careful experimentation. The use of experiments in controlled laboratory conditions also implies that it is known under which assumptions and operating conditions the aforementioned first principles are valid.

Ecological systems are macroscopic aggregations of physical systems. Although

an ecological system has an underlying physical reality, it is generally too large and too complex to study by means of first principles. Instead, ecological models describe the behaviour of populations of components and of aggregate processes. The behaviour of the components and processes is little understood and aggregate behaviours are rarely the sum of their parts. In addition, ecological theory is mostly based on empirical studies of observed behaviour. Uncertainties about climate models aiming to support the ongoing debate about the significance and causes of global warming are a good illustration of this [82].

As such, the corresponding models are necessarily gross abstractions of reality. For example, modelling the day-to-day activities of each individual in a population of thousands of individuals is virtually impossible. Instead, ecologists either build models on the evolution of aggregate metrics, such as population size of a species, or consider extremely simplified descriptions of the behaviour of the individuals. Other areas studying macroscopically large systems, such as macro economics, are confronted with the same issues.

As a result of these differences between physical systems modelling and ecological modelling, automated model constructors that have been successfully applied to a wide variety of different types of physical system have proven to be of limited use in the ecological domain. However, because ecological systems are more difficult to understand than physical systems, models are even more important in the ecological domain than they are in the physical domain [54]. Yet, the intrinsic complexities of ecological model construction introduce a number of new challenges to automated model constructors that have to be addressed. In this dissertation a novel compositional modelling approach is developed to support the automated construction of ecological models.

1.1 Challenges in automated eco-model construction

To better reflect the background of this work, important challenges facing automated ecological modelling are reviewed here.

1.1.1 Abductive modelling

Most automated modellers are either deductive, inductive or a hybrid between both [96]. In [151], the case for an abductive approach to ecological model construction is presented. As ecological models are not necessarily mathematical expressions of physical laws by which nature abides, an ecologist often chooses a particular approach to model part of a system for the reason that the properties of that part are desirable to be established.

An abductive modeller aims to construct models that possess certain prescribed properties. The properties formally describe special qualities or features that may belong to a model. For example, the fact that a model contains an endogenous variable describing the size of some population could be defined as a feature. Because properties belong to a model (and not the other way around), it is possible to deduce which properties a given model has. It is not possible to deduce a model from a set of properties. Instead, properties merely impose restrictive criteria upon which model selection is based.

However, many existing compositional modellers, including those presented in [45, 47], reason deductively. That is, they contain knowledge that prescribes when certain assumptions are satisfied in a given situation and rules that describe under which conjunctions of assumptions each partial model is accurate. As such, these compositional modellers are aimed at problems where a model can be deduced from a given compositional modelling problem.

Other compositional modellers, such as those presented in [135, 150], are hybrid. Similar to deductive compositional modellers, they derive models from assumption sets, but the models produce prescribed types of behaviour. The latter are a type of property that the constructed models must satisfy, and as such, these compositional modellers reason abductively. Yet, the knowledge representation and inference mechanisms employed by these approaches do not generalise to search for models with any type of property that describes a feature of a model. Instead, the properties imposed upon models are restricted to certain types of behavioural requirement. Therefore, this work aims to produce a more generic approach to abductive compositional modelling.

1.1.2 Granularity and disaggregation

One of the more difficult choices that must be made in model construction in general is the granularity of the model [27]. The granularity of a model describes the level of detail at which the behaviour is described. Conventional automated model constructors provide various facilities to choosing the grain size of the model. Often, this choice is based on knowledge about the structure of the system, that is, how components are composed of subcomponents and how processes consist of subprocesses.

Granularity choices play a central role in reasoning about ecological models [173, 174]. In particular, ecological descriptions are often required to reflect the aggregated behaviour of collections of components and processes. These collections can obviously be partitioned into sets of subcollections. For example, a population of species can be disaggregated into subpopulations that group individuals that have the same age, gender or physical location. This type of grain choice in models differs from conventional ones in several ways:

- Disaggregation of a collection of components/processes into a set of smaller collections does not generate an entirely new set of components/processes. Instead it generates new collections of components/processes that can be disaggregated again based on different criteria.

- Many automated modellers, e.g. [25], employ a knowledge base that describes the behaviour of the smallest components/processes. If, in such an approach, the behaviour of a larger component/process is needed, it is constructed by composing the behaviours of the parts and then simplifying the resulting description. Yet, ecological systems are too complex to be described in their most disaggregate form. Hence, the feasibility of such an approach does not extend to ecological systems, even though it is an appropriate one in the physical systems domain.

- An ecological model usually has much in common with an equivalent model where one or more collections of individuals are described as having been partitioned into subcollections. It would be beneficial if the granularity selection mechanism could employ this feature such that existing “aggregate” knowledge could be reused as much as possible in the generation of corresponding “disag-

gregate" models.

Existing compositional modelling techniques for representing and inferring grain choices are inadequate for dealing with these issues related to disaggregation. Therefore, this work introduces a novel approach to model granularity that enables the automatic synthesis of important types of disaggregate models.

Note that component-subcomponent and disaggregation of populations relations are fairly broad types of granularity choice, in modelling in general as well as in ecological modelling in particular. A significant number of variations on these types granularity choice exist and the reader is referred to [173, 174] for a detailed overview.

1.1.3 Reasoning with subjective preferences

Thus far, the discussion has focussed on the structural restrictions and content requirements in the construction of ecological models. In addition to these, the ecologist, i.e. the user of the automated modeller, may also have a set of personal preferences for the available modelling choices. Such preferences stem from specific knowledge and opinions about the suitability of existing types of ecological model in adequately describing an existing situation.

It may seem that such knowledge or opinions could be formalised also by a knowledge base and that various decision or choice theories exist to resolve this question(see for example [14, 20, 21]). However, because of the complexity of the behaviour of ecological systems, most ecological models are necessarily created on the basis of partial information and educated guesses. Any formalisation of knowledge about the adequacy of ecological models will be specific to only a few experts, and other experts may disagree with it. It is not obvious how to acquire such knowledge either, as it involves deep insights in the motivation behind ecological model construction and because it is often a source of significant debate amongst ecologists. For these reasons, this work makes no attempt to abstract the underlying decision process of choosing a particular set of modelling approaches. Instead, the expert is allowed to specify preference orderings over the options available for model construction. Such preference orderings may, in turn, motivate the construction of different ecological models, even if the problem and the requirements are the same.

As an example, figure 1.1 shows a possible preference ordering of models appli-

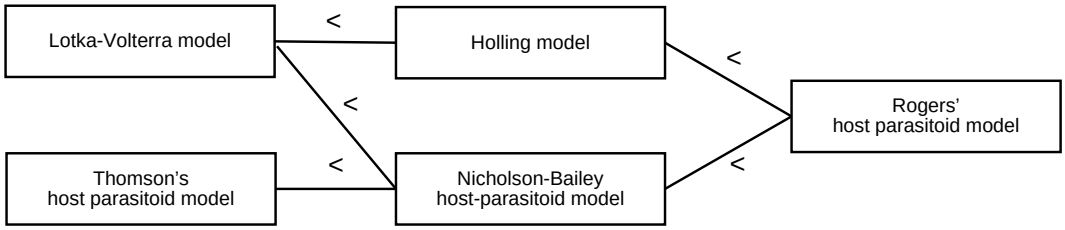


Figure 1.1: Sample partial preference ordering

cable to a host-parasitoid scenario (i.e. a situation where one species acts as a host of a species of parasites). These preferences are supposed to express the adequacy of alternative types of ecological model describing the host-parasitoid relation. Different types of host-parasitoid model will be more or less adequate at depicting the scenario, depending on the features of this scenario and the different types of model. The available knowledge about these features will, typically, be incomplete. Also, the features of the different types of model may not always be comparable. Therefore, any ordering of preferences may be partial. Furthermore, deriving an ordering of preferences is not obvious and hence, it may be the subject of some debate. As a consequence, some ecologists may agree with the sample ordering of figure 1.1, whilst others may disagree with it.

In this preference ordering, Rogers' host-parasitoid model [152] is preferred over that by Holling [79] and that by Nicholson and Bailey [138]. Yet, it may be difficult to compare the adequacy of the underlying assumptions of the latter two models in certain scenarios, and hence their respective preferences can not be compared with one another. Because both types of model are deemed to be more appropriate than the Lotka-Volterra model [110, 179], the preferences of the Holling and Nicholson-Bailey models are greater than the preference of the Lotka-Volterra model. Finally, Thompson's host-parasitoid model [167] is less preferred than that of the Nicholson-Bailey model, but the adequacy of the underlying assumptions is considered incomparable with that of the Lotka-Volterra and Holling models.

Conventional preference calculi are ill-equipped to express such partial orderings. They either employ a conventional numeric calculus or some form of qualitative reasoning that is itself rooted in a numeric one. A novel preference calculus is therefore required for computing with partially ordered preferences and combinations thereof.

1.2 Overview of the compositional ecological modeller

Figure 1.2 presents an overview of the automated modelling system that is designed to tackle the issues raised in section 1.1. The overall aim of this system is to translate a given *scenario* into a mathematical model, called the *scenario model*, by means of a knowledge base and a set of preferences. To this end, four inference procedures are performed.

First, the knowledge base is instantiated with respect to a given scenario. This is done by constructing a hypergraph, called the *model space*, of all possible partial models that can be derived from the scenario and modelling assumptions via instantiating the knowledge base.

The partial models in the model space typically describe the behaviour of aggregated components and processes. The knowledge base may contain model fragments describing disaggregate components and processes, but as explained in section 1.1.2, such an approach often requires an overly complex knowledge base. Models of aggregate components and processes could be made to describe a higher level of detail via a process called disaggregation. A disaggregation is a model transformation that involves replacing variables and/or equations in a model by sets of variables and/or equations, each representing a partition of parts of the concepts/processes described by the original variables/equations. The second inference procedure involves applying such model transformation to the partial models in the model space, such that it incorporates disaggregated models as well as the original aggregate ones.

The model that is to be composed for the provided scenario must not be inconsistent and it may be required to possess certain properties. This will be achieved by selecting and combining a set of partial models from the model space that meets these requirements. By means of rules defining what constitutes an inconsistent model and rules prescribing under which conditions the properties are satisfied by a model, it is possible to determine which partial models in the model space are inconsistent and which ones have some of the predefined properties. As such, these rules are used to impose restrictions upon the combinations of partial models of the model space that are allowed to become part of the eventual scenario model.

The selection of a set of partial models that is consistent and possesses the required properties is achieved by the third inference procedure: reformulating this problem as

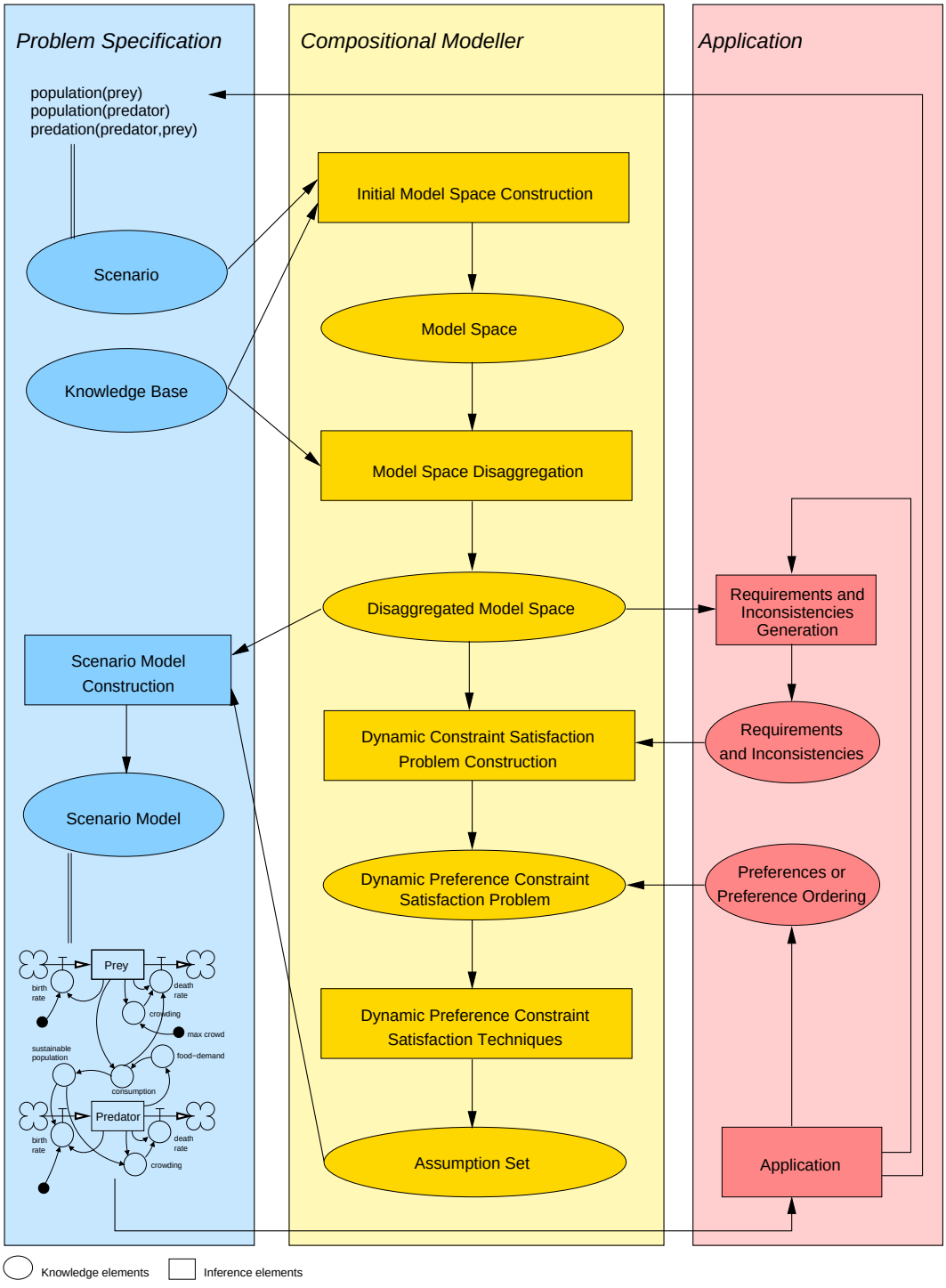


Figure 1.2: The proposed compositional modeller

a dynamic constraint satisfaction problem (DCSP). A DCSP [119, 123] extends the notion of conventional constraint satisfaction problem [169], by adding so-called activity constraints which are able to introduce or remove attributes and their corresponding constraints to/from the CSP. Note that the type of DCSP referred to in this dissertation is in fact a specific type of DCSP. In [117], a more generic framework for DCSPs is presented and any reference to DCSP in this thesis corresponds to an activity constraint based DCSP (or aDCSP) in the aforementioned work.

This approach enables the use of proven constraint satisfaction algorithms for searching for a solution that meets all the imposed criteria. There are three benefits that this approach has over existing techniques. Firstly, the use of DCSP algorithms is more generic than most purpose built search algorithms employed by compositional modellers, which are often based on specific assumptions about the organisation of the knowledge base (see section 2.2 for a more detailed discussion). Secondly, a significant amount of research into efficient CSP and DCSP algorithms, which was previously unusable, is now available to help solve compositional modelling problems [119]. Thirdly, this approach allows for future work into analysing the complexity of different types of compositional modelling problem and the efficiency and suitability of solution algorithms.

The scenario model should not only be consistent and meet the model requirements, but it also meet the preferences of the user. The DCSP is therefore enriched with information derived from user preferences to form a dynamic preference constraint satisfaction problem (DPCSP). This is the fourth and final procedure required to generate the eventual ecological model for the given problem. The preferences may be produced manually by the user or by some problem solver, but that is beyond the scope of this thesis. The solutions of the DPCSP are the solutions of the corresponding DCSP that have an optimal overall preference.

Each such solution of the DPCSP corresponds to a set of modelling assumptions in the model space. Every partial model that follows from the scenario and from these assumptions forms a scenario model that is consistent and satisfies the required properties, whilst maximising the user's preferences.

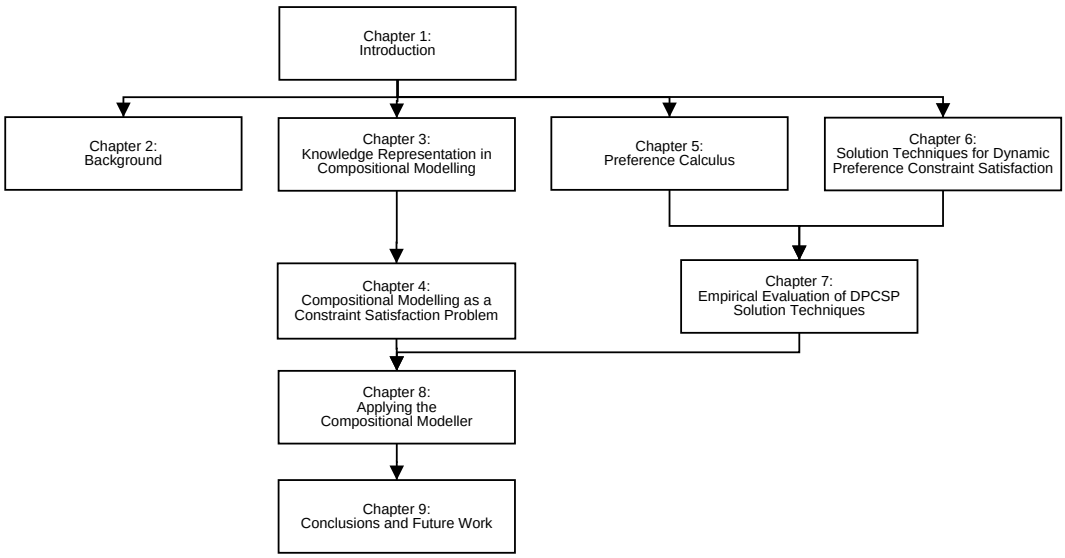


Figure 1.3: How to read this thesis

1.3 Thesis structure

This section lists the rest chapters of this dissertation and summarises their content. As it is understood that readers of this dissertation may have very different backgrounds and interests, a flow chart depicting the order in which the chapters may be read is shown in figure 1.3.

- *Chapter 2: Background*. In this chapter, a systematic overview of compositional modelling is presented. It is a considerably shortened and revised version of the work presented in [96]. First, a general framework for existing compositional modellers is presented. Based upon this common framework, important approaches to compositional modelling are then surveyed and compared. Finally, a summary of applications of compositional modelling is discussed.

- *Chapter 3: Knowledge Representation in Compositional Modelling*. This chapter presents the knowledge representation formalism employed to express model construction knowledge in the ecological domain. It is a significantly extended version of the representational framework introduced in [94, 95]. Although a standardised language exists in compositional modelling (the Compositional Modelling Language (CML) [13]), it is not suitable for tackling the challenges set in section 1.1. In particular, CML does not allow goals to be attached to the

model construction process and it does not provide a representational framework for disaggregating models (which is a new problem to compositional modelling). The representation formalism developed herein tackles these issues, but leaves open the possibility of using conventional CML representations where necessary.

- *Chapter 4: Compositional Modelling as a Constraint Satisfaction Problem*. The algorithms that translate a compositional modelling problem into a CSP are presented in this chapter. This chapter contains a more detailed description of the algorithms presented in [94, 95, 98]. They are divided in three sets of algorithms to be applied in sequence. First, the knowledge base is instantiated with respect to a given scenario, resulting in a space of all possible models. This model space is then extended to incorporate disaggregates of selected models in the original model space. Finally, the model space is translated into a dynamic constraint satisfaction problem. In this way, the complex problem of finding a consistent model of a given scenario that meets all the requirements can be solved by employing efficient dynamic constraint satisfaction techniques rather than by less efficient ad-hoc search algorithms.

- *Chapter 5: Preference Calculus*. In addition to hard requirements, the ecologist/user of the compositional ecological modeller may have a set of preferences for the available modelling choices. In this chapter, a novel preference calculus is developed to combine preferences drawn from a partially ordered set and to compare such preferences. This preference calculus is based on ideas drawn from order of magnitude reasoning (OMR) [145], and therefore, an overview of existing OMR calculi and their limitations is presented first. In the remainder of the chapter, an efficient algorithm to perform the comparison of preferences is developed and justified. The contents of this chapter have been published in [99, 97, 100].

- *Chapter 6: Solution Techniques for Dynamic Preference Constraint Satisfaction*. This chapter formalises the concept of a DPCSP and presents four different algorithms to find solutions for this type of problem. The first algorithm applies a conventional best-first search to the dynamic structure of the CSP, and employs heuristics that prevent the search from getting stuck in local maxima. This algorithm and illustrations of its applications to configuration and compositional

modelling tasks is presented in [97]. The second algorithm extends the first forward checking [70]. A brief discussion of the aforementioned two algorithms is discussed in [99]. The third algorithm extends the first by learning upper boundaries of reachable preferences. This information is utilised to improve the accuracy of the heuristics, thereby producing a more informed algorithm and avoiding some redundancies in the search. Finally, the fourth algorithm combines the improvements of the second and third algorithm. The chapter ends with a discussion of applications of the DPCSP.

- *Chapter 7: Empirical Evaluation of DPCSP Solution Techniques*. The performance of the DPCSP solution algorithms is evaluated in this chapter, by testing them on batches of randomly generated problems. Although the results show that, generally speaking, finding optimal DPCSP solutions is computationally very expensive, beneficial effects of an integrated dynamic and preference CSP scheme are demonstrated. The use of a dynamic preference CSP is found to be significantly more efficient than a preference CSP without dynamicity. The results also indicate the advantages of using partial preference orderings over total preference orderings.

- *Chapter 8: Applying the Compositional Modeller*. This chapter illustrates the usefulness of the automated ecological model construction techniques by means of two large examples. The first shows how a small body of ecological modelling knowledge (in the area of population dynamics) can be employed to engineer a realistic knowledge base, and how an ecological model can be derived from the resulting knowledge base. The second example illustrates the important considerations in the engineering of a large knowledge base, by means of the presented techniques, that has been designed from an existing sophisticated ecological model, which is being used to solve real-world problems.

- *Chapter 9: Conclusions and Future Work*. This chapter concludes the dissertation and presents a number of issues that are important to address in future. In particular, based on the experience gained throughout the course of this research, it is postulated that significant efficiency gains may be achieved by developing algorithms for the decomposition of DCSPs and by employing genetic algorithms for solving DPCSPs. The future work section contains detailed proposals

for such work.

1.4 Major technical contributions

This section summarises the main contributions of this thesis.

- *Literature review*: This thesis includes a systematic review of compositional modelling literature, its strengths and weaknesses and its applications. To date, this is the first such review of this important family of approaches to automated model construction.

- *An algorithm to translate a compositional modelling problem into a DCSP*: The relation between compositional modelling problems and dynamic constraint satisfaction problems was first reported in [123]. However, that approach required that the CSP be encoded in the knowledge base. As a consequence, subsequent work in compositional modelling did not employ CSP techniques, thus ignoring the large number of efficient search algorithms developed by the constraints research community. Here, an approach is presented to translate a compositional modelling problem into a dynamic constraint satisfaction problem automatically, incorporating activity constraints, inconsistent combinations of partial models and external requirements imposed upon the desired model.

- *A method to construct disaggregate models from aggregate ones*: This offers a novel approach to handle granularity with respect to models describing the behaviour of collections of individuals and processes. The technique presented herein differs from existing work in that grain choices are made by transforming an emerging model's level of detail through disaggregation. The result is a means of choosing a model's level of detail that is sufficiently flexible for eco-modelling and that allows grain choices to be described in terms of scenario-level concepts. Thus, different disaggregations of a model can be composed themselves when necessary. As such, a small number of disaggregations may encompass a much larger selection of models at different levels of detail.

- *A calculus of partially ordered order-of-magnitude preferences*: This is a calculus that defines basic preferences in a network of ordering relations and that

supports comparisons between combinations of such preferences. In order to assume that the comparison of combined preferences does not increase the complexity of the search algorithms, an algorithm is developed to maintain the efficiency of performing order-of-magnitude inferences.

- *Solution algorithms for DPCSPs*: A DCSP in which the domain values are assigned preferences results in a novel type of CSP: a DPCSP. This DPCSP formalism is shown to fit not only compositional modelling but also a variety of important types of synthesis problem, such as configuration and planning. In this work, four algorithms are designed and implemented to find a solution to the DPCSP that is both consistent with the constraints, and optimal with respect to the preference assignments.

As a whole, this thesis presents a compositional modelling approach to automated ecological model construction. It differs from earlier work in that it is capable of handling alternative modelling approaches and combining them based on structural restrictions, hard requirements and user preferences. The utility and efficiency of this work are evaluated using both randomly generated problems and realistic large-scale example problems.

Chapter 2

Background

This chapter provides the background for compositional ecological modelling. Section 2.1 presents a formal description of the compositional modelling problem in general. Based on this, a survey of specific compositional modellers is described in section 2.2. Then, section 2.3 introduces the representational formalisms typically employed in ecological modelling. This leads to an analysis of the issues that need to be addressed, in section 2.4. Finally, section 2.5 summarises this chapter.

2.1 Outline of compositional modelling

Compositional modellers are a class of knowledge-based modellers that, based on an initial model and a task specification (e.g. an initial state specification) instantiate a knowledge base of composable pieces of models and combine an appropriate subset of these pieces into an adequate model. The term compositional modelling was introduced in [45], and is herein used to refer to a much wider class of similar modellers as is common in the literature [103].

This section presents a general overview of compositional modelling. First, the modelling task of compositional modellers and the framework within which it operates is explained. Then, the most essential knowledge representation formalisms that distinguish compositional modellers from other types of automated modeller are introduced in section 2.1.2. Finally, section 2.1.3 shows how model composition is achieved by this approach to automated modelling. The concepts that are defined in this discussion are illustrated by means of a small and simple example and are primarily based

on the seminal work in the field presented in [45].

2.1.1 The compositional modelling task

A compositional modeller is either a deductive or a hybrid modeller that constructs a mathematical or conceptual model based on a technical level input. Figure 2.1 presents a generic architecture for compositional modellers. A compositional modeller takes a scenario and a task specification as its input. The scenario constitutes the technical level input, and the task specification is a formal description of the criteria imposed upon the model to be composed.

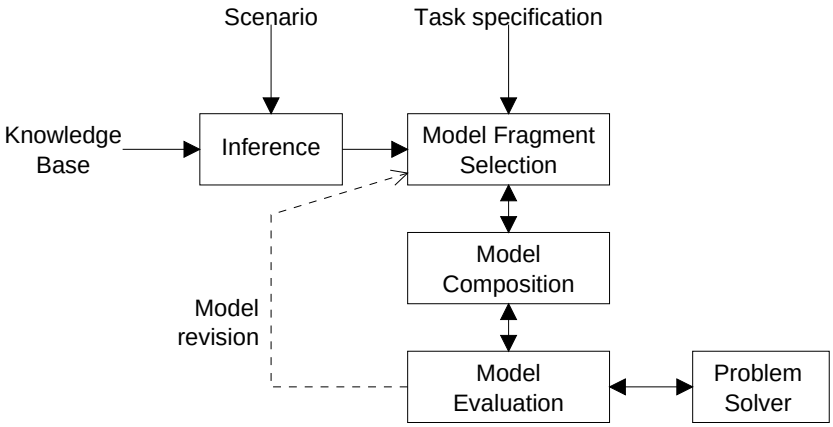


Figure 2.1: Generic architecture of compositional modellers

Model composition occurs through several stages. First, an *inference mechanism* instantiates the constructs of the knowledge base, such as model fragments and rules, that apply to the scenario. The model fragments describe how certain components, processes or concepts can be modelled. Different model fragments may have the same scope, representing the same component, process or concept.

Model fragment selection chooses a subset of the instantiated model fragments resulting from the inference phase by means of the *task specification*. Task specifications come in a variety of forms and are normally specific to each implementation since many compositional modelling based systems are specialised to cope with one particular type of task (although the underlying ideas may well be generalised for other types of application). They are usually represented either as a *query*: a request to describe the relationship between certain model components or to specify the value and

direction/rate of change of a variable, or as an *initial state* of a model from which the problem solver must extrapolate future behaviour. In compositional modellers that perform a hybrid modelling task, the task specification includes a description of the expected behaviour of the resulting model.

The selected instantiated model fragments are then composed into a model. This *model composition* phase may use various techniques. In particular, *consistency checking* determines whether the underlying assumptions of the model fragments are compatible with one another and whether the set of equations can be combined with one another (i.e. the resulting set of equations is not overconstrained) [106]. *Causal ordering techniques* may be used to establish cause and effect relations over the variables in the models. Causal relations are typically required by certain problem solvers, such as explanation generators and diagnostic engines [135]. An *equation processor* transforms an intermediate conceptual model into a mathematical model (e.g. translating QPT influences into QSIM constraints [46]). Problem solvers that perform simulation, for example, need a mathematical model of the scenario.

The models that are generated during the model composition phase are to be used by the problem solver. However, not all models are equally suitable. The quality of a model depends upon the adequacy of the underlying assumptions, the necessity of the components, processes and/or concepts that are included in the model and the overall complexity of the model. In the *model evaluation* phase, alternative models are assessed and the best alternative is passed on to the problem solver.

During the model evaluation and problem solving stages, new information may be derived that contradicts earlier assumptions. For example, certain variables may not remain within the operating ranges assumed by the model [46]. This information is fed back to the model fragment selection phase, which replaces the affected model fragment and hence revises the model accordingly.

Compositional modelling is conceptually close to certain formal theories of modelling that study how mathematical models can be derived from technical or conceptual representations. Compartmental modelling, for example, conceptually represents systems as compartments representing amounts of homogeneous materials and flows between them. Formal theories have been developed to translate such descriptions into mathematical models and to analyse the represented systems [180]. Incidentally, the compositional modelling techniques mentioned in [165, 166] use compartmental mod-

elling as the underlying paradigm for their ontology and model fragments. A related paradigm is system dynamics [57], which organises a system as flows between stocks that are controlled by a causal network of influences.

2.1.2 Essential knowledge representation formalisms

All compositional modellers employ a number of basic constructs in its knowledge representation formalism: scenarios and model fragments. These constructs are discussed in this section.

2.1.2.1 Scenario

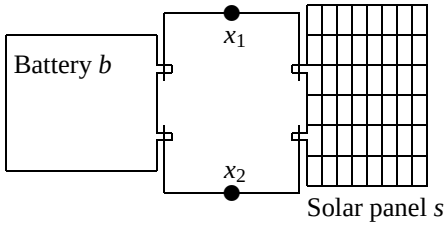
The compositional modeller's task is to transform a general, user-friendly representation of a system, e.g. a component-connection model, into another more specific representation of the same system that can be manipulated by the problem solver, e.g. a mathematical model. In the remainder of this chapter, a *model* is represented as a pair $\langle O,C\rangle$ where O is a set of object constants $\{o_1,\ldots,o_v\}$ and C is a set of relations over these object constants $\{c_1(o_1,\ldots,o_v),\ldots,c_w(o_1,\ldots,o_v)\}$.

A *scenario* is a model that is given as part of the task description. It is usually (but not necessarily) specified using a representation formalism at the technical abstraction level, e.g. via a component-connection representation. Note that, throughout this dissertation, it is presumed that a scenario contains the *necessary* and *sufficient* object constants and relations to describe the system of interest. As such, every scenario employed herein provides an accurate representation of the real-world system that needs to be modelled.

An example of a scenario, which formalises the situation depicted in figure 2.2(a), is given in figure 2.2(b). This particular scenario represents a rechargeable battery that is hooked up to a solar panel.

2.1.2.2 Model fragments

As section 2.1.1 shows, a compositional modeller essentially performs the complex transformation of a very simple and abstract representation of a system (a scenario) into a conceptual or mathematical model that contains sufficient detail for problem solving purposes. It is therefore necessary to differentiate between a source-model



(a) Physical system

battery(b), solar-panel(s),
rechargeable(b),
electrical-
connection(b, x_2, x_1),
electrical-
connection(s, x_1, x_2).

(b) Specification

Figure 2.2: A sample scenario

$M_s = \langle P_s, C_s \rangle$ and a target-model $M_t = \langle P_t, C_t \rangle$. In this thesis, the participants in the sets P_s and P_t are called source-participants and target-participants respectively, and the constraints in the sets C_s and C_t are called source-constraints and target-constraints respectively. As mentioned earlier, the knowledge base of a compositional modeller consists of composable pieces of sub-system models, called model fragments. Each of these model fragments relates a particular scenario of a subsystem to its equivalent representation in the target modelling language, under certain conditions. Definition 2.1 formalises the content of a model fragment.

Definition 2.1 (*Model fragment*). A model fragment μ is a tuple $\langle P^s, P^t, C^s, C^o, C^t, A \rangle$ where

- $P^s(\mu) = \{p_1^s, \ldots p_m^s\}$ is a set of *source-participants*,
- $P^t(\mu) = \{p_1^t, \ldots, p_n^t\}$ is a set of *target-participants*,
- $C^s(\mu) \cup C^o(\mu)$ is a set of *preconditions*, where $C^s(\mu) = \{c_1^s, \ldots, c_v^s\}$ is the set of *structural condition* that apply over the vector of source-participants $\overrightarrow{p_s}(\mu) = (p_1^s, \ldots, p_m^s)$ and $C^o(\mu) = \{c_1^o, \ldots, c_w^o\}$ is the set of *operating conditions* that apply over the vector of target-participants $\overrightarrow{p_t}(\mu) = (p_1^t, \ldots, p_n^t)$.
- $C^t(\mu) = \{c_1^t, \ldots, c_u^t\}$ is a set of *postconditions* which are constraints that apply over $\overrightarrow{P_t}(\mu)$, and
- $A(\mu) = \{a_1, \ldots, a_s\}$ is a set of *assumptions*,

such that for $i = 1, \ldots, u$

$$\forall p_1^s, \ldots, p_m^s, \exists p_1^t, \ldots, p_n^t$$
$$c_1^s \wedge \ldots \wedge c_v^s \rightarrow (a_1 \wedge \ldots \wedge a_s \rightarrow (c_1^o \wedge \ldots \wedge c_w^o \rightarrow c_i^t))$$

In most compositional modellers, the target modelling language consists of some type of (qualitative) constraints over variables. Therefore target-participants are usually called *quantities* and source-participants are known as *participants* or *individuals*.

Source-participants: B, X_1, X_2

Structural conditions: battery(B), electrical-connection(B, X_1, X_2)

Target-participants: V, V_0, CL, CL_0

Assumptions: ¬damaged(B), binary-voltage(B)

Operating conditions: $CL \geq CL_0$

Postconditions: voltage(B, V), nominal-voltage-level(B, V_0), charge-level(B, CL), charge-level-threshold(B, CL_0), $V = V_0$

Figure 2.3: Sample model fragment

An example of a model fragment is given in figure 2.3. This model fragment applies to a battery B that is electrically connected to the connection points X_1 and X_2. If applied, it introduces four new target-participants to the scenario model: voltage V generated by the battery, a nominal voltage V_0 of the battery, the charge level CL of the battery and a charge level threshold CL_0 for the battery. The implication that is formalised by this model fragment states that, if the battery B is electrically connected, and under the assumptions that B is not damaged, the binary voltage model for B is required and appropriate and the charge level CL of B is above its threshold CL_0, then $V = V_0$

2.1.3 Model composition

Model fragments are stored and organised in what is called a *domain theory* or a *model fragment library*. By using the standard operations of substitution and modus ponens, new knowledge can be derived from existing knowledge through the implication defined by the model fragments. In this respect, the usage of a model fragment can be defined as follows.

Definition 2.2 (*Usage of a model fragment*). A model fragment μ is *applicable* with respect to a scenario $\langle P_s, C_s \rangle$ if a substitution σ exists such that $P^s(\mu)\sigma \in P_s\sigma$, the structural conditions μ, are logically entailed by the source-constraints of the scenario: $C_s \models C^s(\mu)$. A model fragment is *applied* with respect to μ if it is applicable with respect to μ and its assumptions $A(\mu)$ are consistent and are assumed true. A model fragment is *active* with respect to μ if it is applied with respect to μ and its operating conditions hold and are consistent.

Determining the applicability of a model fragments is achieved by pattern matching in much the same way as in production systems. However, a compositional modeller aims at constructing different models depending on the task. As mentioned in section 2.1.1, the task specification can take various forms, e.g. a query or an initial state, depending compositional modelling approach. It is therefore not feasible to provide a general definition of the task specification, but section 2.2 will describe the task specifications of individual compositional modellers in more detail. The scenario model that is most appropriate for a given task must be discovered under incomplete knowledge, that is made explicit by committing to the assumptions and operating conditions of certain model fragments. The eventual scenario model can be defined as follows:

Definition 2.3 (*Scenario model*). A set of model fragments $\Phi = \{\mu_1, \ldots, \mu_n\}$ defines a scenario model $M_t = \langle P_t, C_t \rangle$ of a scenario $M_s = \langle P_s, C_s \rangle$ in which $P_t = \cup_{i=1}^{n} P^t(\mu_i)$ and $C_t = [\cup_{i=1}^{n} C^o(\mu_i)] \cup [\cup_{i=1}^{n} C^t(\mu_i)]$ if $\forall \mu \in \Phi : \mu$ is applicable with respect to M_s, $P_s = \cup_{i=1}^{n} P^s(\mu_i)$ and $C_s = \cup_{i=1}^{n} C^s(\mu_i)$.

Consider for example the domain theory presented in figure 2.4. Based on this domain theory, various scenario models can be constructed. The following model can be derived via the inference shown in figure 2.5:

Model fragments μ_i with $P^s(\mu_i) = B, X_1, X_2$; $C^s(\mu_i)$ = battery(B), electrical-connection(B, X_1, X_2); $P^t(\mu_i) = V$ and $C^t(\mu_i)$ = voltage(B, V).	
μ_{11}	$\mathcal{C}(\text{constant-voltage}(B)) \rightarrow V = (0, \infty)$
μ_{12}	$\mathcal{C}(\text{normal-degrading-voltage}(B)) \rightarrow \frac{d}{dt}V = (-\infty, 0))$
μ_{13}	$\mathcal{C}(\text{binary-voltage}(B)) \rightarrow \text{charge-level}(B, CL) \wedge \text{charge-threshold}(B, C_0)$
μ_{14}	$\text{charge-level}(B, CL) \wedge \text{charge-threshold}(B, C_0) \wedge CL < C_0 \rightarrow V = 0$
μ_{15}	$\text{charge-level}(B, CL) \wedge \text{charge-threshold}(B, C_0) \wedge CL \geq C_0 \rightarrow V = (0, \infty)$
μ_{16}	$\mathcal{C}(\text{charge-level-sensitive}(B)) \wedge \text{rechargeable}(B) \rightarrow \text{charge-level}(B, CL) \wedge V = M^+(CL)$
μ_{17}	$\mathcal{C}(\text{temperature-sensitive}(B)) \wedge \text{rechargeable}(B) \wedge \text{charge-level}(B, CL) \wedge \text{temperature-of}(B, T) \rightarrow V = M^+(CL, T)$

Model fragments μ_i with $P^s(\mu_i) = \{B, I, X_1, X_2, CL\}$; $C^s(\mu_i)$ = battery(B), rechargeable(B), charge-level(B, CL),electrical-connection(B, X_2, X_1).	
μ_{21}	$\mathcal{C}(\text{constant-charge-level}(B)) \rightarrow CL = (0, \infty)$
μ_{22}	$\mathcal{C}(\text{normal-accumulation}(B)) \wedge \mathcal{C}(v(X_1, X_2) > 0) \wedge (i(X_1, X_2)) \rightarrow CL = \int i(X_1, X_2) dt$
μ_{23}	$\mathcal{C}(\text{accumulation-with-aging}(B)) \wedge \mathcal{C}(v(X_1, X_2) > 0) \wedge (i(X_1, X_2)) \rightarrow CL = \int i(X_1, X_2) dt - (k \times DOD + l \times TSLC)$

Other model fragments.	
μ_{31}	$\text{solar-panel}(S) \wedge \text{electrical-connection}(S, X_1, X_2) \rightarrow \text{voltage}(S, V) \wedge \text{solar-energy}(E_l) \wedge V = M^+(E_l)$
μ_{32}	$\text{electrical-connection}(C, X_1, X_2) \wedge \text{voltage}(C, V) \rightarrow v(X_1, X_2) = C^+(V)$
μ_{33}	$\text{electrical-connection}(C, X_2, X_1) \wedge \text{voltage}(C, V) \rightarrow v(X_1, X_2) = C^-(V)$
μ_{34}	$\text{battery}(B) \wedge \text{electrical-connection}(B, X_2, X_1) \wedge \mathcal{C}(v(X_1, X_2) > 0) \rightarrow \text{resistance}(B, R) \wedge \frac{1}{r(X_1, X_2)} = C^+(\frac{1}{R})$
μ_{35}	$\frac{1}{r(X_1, X_2)} \wedge v(X_1, X_2) \rightarrow i(X_1, X_2) = \frac{v(X_1, X_2)}{r(X_1, X_2)}$

Figure 2.4: Model fragments of a sample domain theory

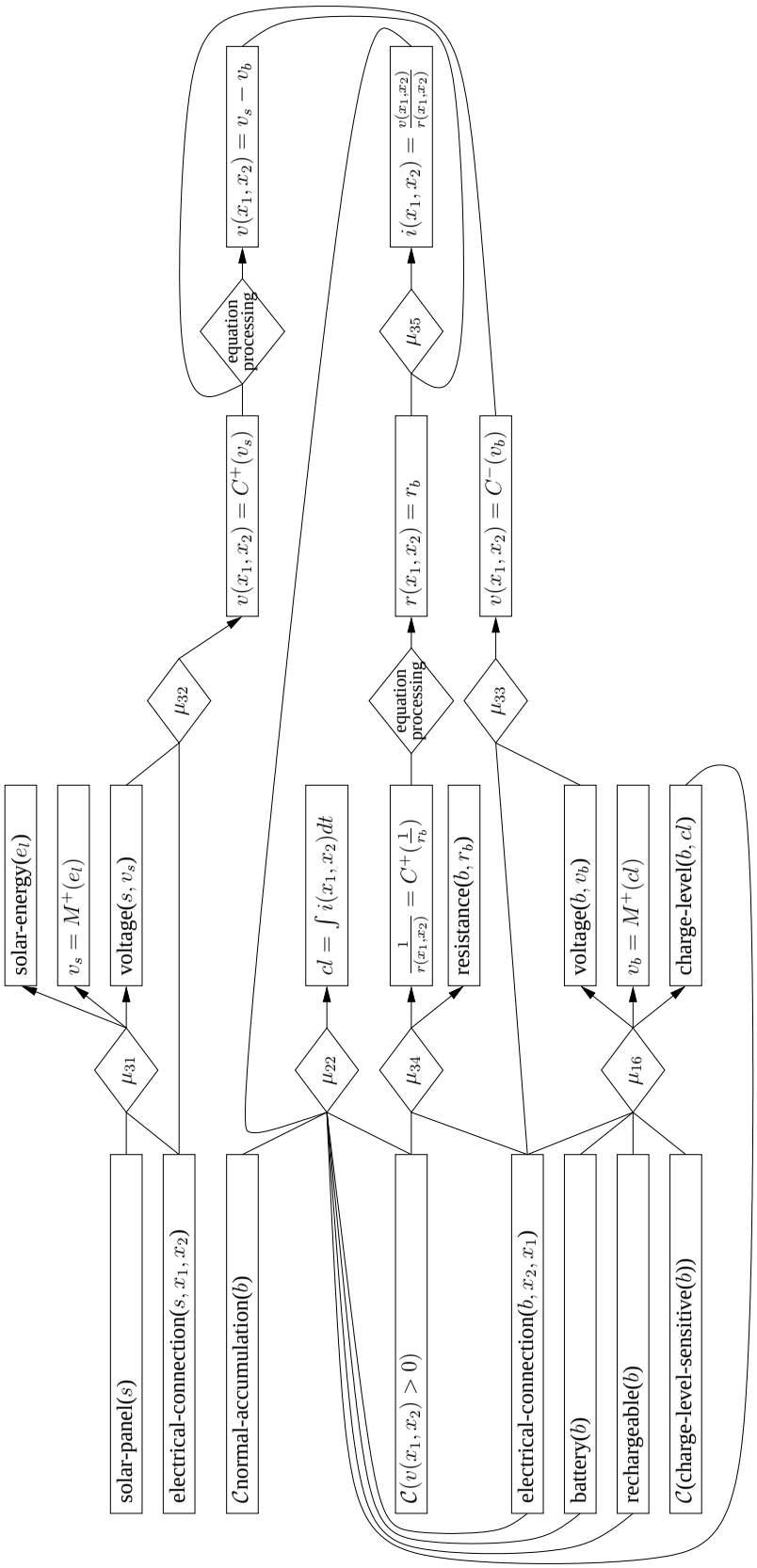


Figure 2.5: Model inference example

$$
\begin{aligned}
cl &= \int i(x_1,x_2)dt \\
v(x_1,x_2) &= v_s - v_b \\
i(x_1,x_2) &= \frac{v(x_1,x_2)}{r_b} \\
v_s &= M^+(e_l) \\
v_b &= M^+(cl) \\
r(x_1,x_2) &= r_b
\end{aligned}
$$

The model fragments, in figure 2.4, from which this model has been deduced, are axioms that describe how partial models can be derived. For example, model fragment μ_{31} states that an electrically connected solar panel S generates a voltage V_s over two points that is proportional to the available solar energy E_l. Instances of such inferences are depicted in figure 2.5. Here, the rectangles denote instances of predicates and the diamonds, annotated with the name of a model fragment, correspond to instantiated axioms. For instance, figure 2.5 contains an application of model fragment μ_{31} where S is instantiated to s, V_s to v_s and E_l to e_l.

Note that two of the diamonds in figure 2.5 contain "equation processing" instead of a reference to a model fragment. These inferences perform domain independent equation manipulations instead of domain dependent axioms. One of these inferences combines the two composable addition relations $v(x_1,x_2) = C^+(v_s)$ and $v(x_1,x_2) = C^-(v_b)$ into a fully specified equation $v(x_1,x_2) = v_s - v_b$. The other makes the algorithm transformation from $\frac{1}{r(x_1,x_2)} = C^+(\frac{1}{r_b}$ to $r(x_1,x_2) = r_b$. Composable addition relations contain a variable in the left-hand side and a term of a sum in the right-hand side. Different composable addition relations with the same variable on the left-hand side can be integrated with one another to form a fully specified equation, by combining all the terms on their right-hand sides into a sum. Section 3.3.1 will explain composable relations, such as composable addition, in more detail.

Finally, it is important to point out that the way in which the most appropriate assumptions and operating conditions are discovered is still an important research issue. Currently, compositional modellers tend to use approaches that are specific to a class of well-defined tasks. Therefore, each compositional modeller should be examined with respect to the specific model construction task it tackles. Section 2.2 presents a

survey of several important contributions to compositional modelling.

2.2 Survey of compositional modelling approaches

The inference mechanisms of a compositional modeller search for consistent and complete models for a specific task description. Since most compositional modellers are designed with a specific model construction task in mind, there are considerable differences between the inference engines of specific compositional modellers. For example, some compositional modellers search for any model that meets basic consistency requirements (e.g. [46]) whereas others search for a sufficiently optimal, i.e. adequate, model (e.g. [150]). In the latter case, the degree of optimisation of a model is dependent on the compositional modeller's notion of relevance. In this section, a number of important approaches to compositional modelling are examined and compared. All of these approaches have been implemented and tested on real-world or sample problems, as reported in the literature. This discussion will focus on the inference mechanisms, based on the specific implementation of the representational framework and the typical tasks that the target models are designed to accomplish.

2.2.1 CM: an ATMS based compositional modeller

The compositional modeller presented in [45] lends its name to the entire class of automated modellers that form the subject of this thesis. This automated modeller is based on an approach to qualitative reasoning, called *qualitative process theory* (QPT). It extends the original work on GIZMO [50] and QPE [51]. To avoid confusion between compositional modelling in general, and Falkenhainer and Forbus' latest extension of QPE, the latter will be identified as CM in the remainder of this paper.

The task CM is designed to do is to construct a model of a particular subset of a scenario, to be identified based on a query, and to produce a total envisionment for that model. A total envisionment is a description of the behaviour of a (single) model under all possible operating conditions. CM uses three sets of algorithms to perform this task. The first represents the space of models that meet the structural conditions imposed by the scenario. The second identifies the subspace of models that are relevant to the query. And finally, the third searches for the simplest set of

applicable model fragments. This set constitutes the output of QPE/CM since this compositional modeller produces total envisionments.

CM follows the general framework of section 2.1 relatively closely, but its associated inference mechanisms are quite different from those employed by most compositional modellers. The scenario $M_s = \langle P_s, C_s \rangle$ is an instance of a component-connection formalism that allows "part-of" relations. These transitive part-of relations imply that one component forms a subsystem of another and organise all components in a hierarchy. Before the actual problem solving, this scenario is instantiated by the compositional modeller in its entirety. A model fragment in CM is a tuple $\langle P^s, P^t, C^s, C^o, C^m, C^t, A \rangle$ where P^s,P^t,C^s,C^o,C^t, and A follow definition 2.1 and C^m is a set of conditions on model fragment usage (e.g. "is applied"). The operating conditions $C^o(\mu)$ (the term has a slightly different meaning than in [45]) are boundary conditions on variables and are not resolved by CM.

Assumptions are used in a slightly different way from other compositional modellers. Some assumptions are $\mathcal{C}(\text{exists}(p_i^s))$, p_i^s is a source-participant. Such assumptions represents the presumption that consideration of the source-participant is necessary to use the associated model fragment to model a query. Some assumptions that apply to source-participants are organised in a knowledge representation unique to CM: the assumption class. An is a tuple $\langle c, a_1, \ldots, a_n \rangle$ where c is a condition and the $a_i, i = 1 \ldots n$ represent individual assumptions. An assumption is logically equivalent to $c \rightarrow [(a_1 \vee \ldots \vee a_n) \wedge (\forall a_i, a_j, i \neq j, \neg a_i \vee \neg a_j)]$ [45]. This means that if the condition of the assumption class is true, then one and only one of its assumptions is true. The model fragment may also contain rules. Rules are typically used to instantiate the condition of an assumption class under well-specified circumstances or to constraint certain combinations of individual assumptions belonging to different assumption classes. Model fragments, assumption classes and rules are instantiated with respect to a scenario and stored in an assumption-based truth maintenance system (ATMS) (see section 4.1.1 for a formal specification of the ATMS).

The ATMS enables the compositional modeller to store economically all applicable model fragment instantiations simultaneously whilst maintaining consistency or inconsistency of all possible conjunctions of assumptions. All model fragments applicable to the scenario are represented in the ATMS. For each individual model fragment assumption, an ATMS assumption is instantiated. For each applicable model fragment

μ, an ATMS node is created for each different possible usage of μ (CM distinguishes between a model fragment that is applied with respect to the assumptions regarding source-participants and application with respect to all assumptions). These ATMS nodes are justified by ATMS assumptions and ATMS nodes (representing usages of model fragments) as is implied by the assumptions and model fragment usage conditions of these model fragments. Finally, an ATMS node is created for each target-participant and postcondition in each model fragment. The latter ATMS nodes are justified by an ATMS node indicating that the model fragment containing these target-participants and postconditions is applied. The resulting ATMS network enables CM to select a set of postconditions C_t and associated target-participants P_t by selecting a number of (ATMS) assumptions. The latter set of assumptions is called the *modelling environment* of $\langle P_t, C_t \rangle$.

Model selection in the compositional modeller is driven by the task specification, which consists of a query facility. A query is a question about specific properties of source and/or target participants. For example, a request to list the factors that influence the efficiency of a physical system's component consists of relating a number of energy inputs and outputs to and from that component. With the assistance of a domain and problem solver specific component, the query facility generates a set of expressions with respect to the instantiated domain theory. These expressions refer to a specific set of nodes in the ATMS that instantiates the domain theory with respect to the scenario. A node for the query is created with this set of expressions as its justification. The label of this query node is a set of candidate modelling environments.

These candidate environments imply consistent (guaranteed by the ATMS) but incomplete models and must therefore be extended. Additional target-participants and constraints are selected by adding assumptions to the candidate environments. First, all source-participants of relevance must be considered. The set of source-participants considered relevant by a candidate environment E_i is $P_s(E_i) = \{p_j \mid \exists e \in E_i,\ e = \mathcal{C}(\text{exists}(p_j))\}$. CM assumes that no component or subsystem (i.e. source-participant) can be modelled in isolation from other components or subsystems of the same covering system. A source-participant p is said to be a covering system of a set of source-participants P_s if $\forall p_j \in P_s$, part-of(p_j, p). For each E_i, the smallest possible covering system of $P_s(E_i)$ and all source-participants that are part of it are added as assumptions to E_i.

Second, the partial candidate modelling environments produced by the previous computations are completed by applying domain specific rules in order to account for all phenomena relevant to the completed object of the query. These domain specific rules are the conditions of the assumption classes. A candidate modelling environment must contain an assumption from at least each assumption class whose condition is true as a result of the implications of this environment. This problem is resolved by the dynamic constraint satisfaction algorithm presented in [123]. The resulting candidate modelling environments are then ordered by means of some simple heuristics measuring their complexity. The model defined by the "simplest" candidate is used and validated. When inconsistencies are detected, an alternative candidate is chosen to define a new model, depending on the specific inconsistencies that occurred during previous trial runs.

The models produced by the compositional modeller are passed on to a simulator that generates a total envisionment of the model. One of the main problems with this compositional modeller and its predecessors is that these models consist of a set of instantiated constraints from the active model fragments. The representation of time employed by QPT does not require the resulting model fragments to hold at the boundaries of their operating conditions. Consequently, a variable continuously changing over time can cause the set of model fragments that hold at time instant t to be different from those that hold at $t-\varepsilon$ or $t+\varepsilon$, with ε denoting an infinitesimally small amount of time. A model can therefore be valid for only a single instant of time. This prevents QPT from being expressed in terms of ODEs and gives rise to various problems with the associated qualitative simulator such as stutter: a sequence of situations which are represented by a model that is only valid for an infinitesimally small instant of time. In the first instance, this problem is due to the qualitative representation employed within QPT and the corresponding simulator. As such, it is not inherent to the principles underlying the compositional modelling approach, as will be clarified in the next section.

2.2.2 QPC: qualitative physics compiler

An alternative approach to compositional modelling is QPC [29, 46, 47]. QPC's main task is to extrapolate the behaviour of the system described in a scenario, from a (pos-

sibly incomplete) set of initial conditions using a domain theory of model fragments. It aims at producing the *attainable envisionment* of a domain system from a certain state. An attainable envisionment is a description of all distinct sequences of states that could be observed from a system, from a limited set of possible initial states. To that end, a QPC task specification consists of a (partial) initial state of the system. QPC's goal contrasts with CM's purpose to find a *total envisionment* with respect to a given scenario and query. A scenario in QPC is specified by a set of source-participants and structural conditions, which typically, though not necessarily, expresses a component-connection model.

The model fragments of a QPC model fragment library are tuples of the form $\mu = \langle P^s, P^t, C^s, C^o, C^t \rangle$. Note that the model fragments do not contain a set of assumptions $A(\mu)$. Consequently, for a given structural setting, operating conditions are the only means to differentiate between models. In itself, this is not a major drawback since assumptions could be reformulated in terms of operating conditions. However, incompleteness of knowledge with respect to true operating conditions is not resolved. Instead, all possible models under operating conditions that are not disproved are constructed and simulated. This is achieved as follows. QPC first determines the set of model fragments that is applicable to the scenario.

Model fragments that can not be determined active or inactive are considered ambiguous. Suppose that there is a set $\Phi_{ambiguous}$ of n ambiguous model fragments. QPC resolves this by considering all so-called refinements of this set of model fragments. A refinement is computed as follows. QPC considers a way to partition the set of ambiguous model fragments into two sets Φ_{active} and Φ_{inactive} (there are 2^n ways of doing this and, in principle, all must be considered). Now, all ambiguous operating conditions of the model fragments in Φ_{active} are assumed to be true and one ambiguous operating condition of each model fragment in Φ_{inactive} must be assumed to be false (again, there is an exponential number of possible assignments). If one of these assignments is consistent with the unambiguous operating conditions, then the model resulting form this refinement must be considered and used for simulation.

The general architecture of QPC also differs from CM. In addition to a model processor responsible for model fragment selection and instantiation, QPC also utilises an equation processor. The postconditions of the model fragments in QPC are QPT constraints as in the compositional modeller. A QPT constraint relates a small number

of variables by means of one of limited set operator. Most types of QPT constraints formed in this way, can be combined with others to form more complex mathematical relations, such as QSIM QDEs, and this makes them well suited for compositional modelling. However, as mentioned in section 2.2.1, QPT constraints are not ideal for simulation. By having an equation processor to translate the set of QPT constraints (produced by the model processor) into QDEs, QPC resolves these problems. This translation is possible since, in this specific implementation, the influences are expected to hold at the limit of their operating conditions. The resulting QDEs can then be used for qualitative simulation via QSIM.

2.2.3 Quantitative information-aided compositional modelling

The qualitative states considered by both CM and QPC tend to be very imprecise. This prevents them from exploiting whatever quantitative information is available. However, quantitative information can enable modellers to focus the search for model fragments within narrower operating bounds. It also allows an associated simulator to produce more precise predictions which in turn narrow potential future operating conditions and enable more specific answers to the possible queries.

SIMGEN [52] is a quantitative variant of QPC. Using a domain theory of fully specified numeric equations, it produces a total envisionment of the behaviour of a system when given a scenario and precise initial and boundary conditions. There are two obvious drawbacks, however, with this strict numeric approach. SIMGEN applications need to be able to construct a fully specified numeric procedure for every foreseeable combination of influences and this typically requires a large number of model fragments. Also, it makes the system incapable of coping with imprecise knowledge.

Between these extremes lies the *Semi-Quantitative Physics Compiler* (SQPC) [48], a semi-quantitative extension to QPC. SQPC represents both the qualitative magnitudes of QPC and precise numeric intervals within the same framework. In so doing, SQPC can take advantage of quantitative information available in addition information embedded in conventional qualitative QPC input. However, because the boundaries of the intervals are crisp and any value within an interval is treated equally, SQPC can not reason about the relative appropriateness of applied model fragments, nor can it give an indication of the relative likelihoods of different attainable behaviours.

2.2.4 Causal explanation-oriented compositional modelling

In [134, 135, 136] a compositional modeller is presented that searches the for simplest model of causal relations amongst the variables of a system which is sufficiently detailed to explain an expected behaviour of the system. Consequently, the task specification of this compositional modeller consists of an expected behaviour. This expected behaviour consists of a set of causal orientations between variables. The scenario is again a component-connection model. However, a rich representation may be used to express the source-participants, which model the components, and the structural conditions, which model the connections. This is possible because specific representation formalisms and inference techniques are available to process such knowledge. The target-participants of the model fragments are variables, and the postconditions are equations relating these variables together with the allowed causal orientations between the variables within each equation.

The target model is the set of direct causal dependencies between variables (the target-participants) generated from the onto-causal mappings over the union of postconditions of all selected model fragments. The onto-causal mappings are one-to-one functions from a set of equations to a set of causal orientations, such that all causal orientations are allowed by the model fragments and such that for each of the endogenous variables there is at least one equation which is causally oriented towards it. Such onto-causal mappings can be generated from a set of equations extracted from the postconditions of the selected model fragments, provided that the number of independent equations equals the number of endogenous variables, i.e. the set of equations is not overconstrained. In this case, the model defined by the set of selected model fragments is said to be complete.

The model fragments $\mu = \langle P^s, P^t, C^s, C^o, C^t, A \rangle$ represent classes of components of physical systems. Similar to object oriented systems, these model fragments are organised in an inheritance hierarchy. A model fragment μ_i that specialises some other model fragment μ_j inherits all information contained in μ_i and may add to it, such that $P^s(\mu_j) \subseteq P^s(\mu_i) \wedge P^t(\mu_j) \subseteq P^t(\mu_i) \wedge C^s(\mu_j) \subseteq C^s(\mu_i) \wedge C^o(\mu_j) \subseteq C^o(\mu_i) \wedge C^t(\mu_j) \subseteq C^t(\mu_i) \wedge A(\mu_j) \subseteq A(\mu_i)$. Some of the participants in the subclass μ_i may be physically the same as participants already in μ_j, but used for different functions. Such relations are modelled by so-called articulation rules. The components of the scenario are not

necessarily instances of the classes of components described by the model fragments. The components represented by the model fragments are objects that perform a certain function (e.g. magnetic coil) whereas the components of the scenario describe the physical appearance of the objects in the system (e.g. metallic wire coiled around a magnetic material). The structural conditions $C^s(\mu)$ represent preconditions on the structural setting that are required for a set of objects to be an instance of the component represented by the model fragment μ.

In addition to the structural conditions within model fragments, the knowledge base may also contain so-called structural coherence constraints. These are rules that impose a certain model fragment (or one of its subclasses) when given certain structural conditions. When these structural conditions are encountered, the structural coherence constraints prevent the compositional modeller to match them with any of the superclasses of the imposed model fragment, thereby focusing the inference mechanisms. Similar to the aforementioned structural coherence constraints, the operating conditions $C^o(\mu)$ are enhanced by so-called behavioural coherence constraints. These are rules that impose a certain model fragment (or one of its subclasses) when given certain operating conditions, thereby preventing the inference mechanisms from searching through any of the superclasses of this imposed model fragment.

In order to differentiate between different modelling perspectives, model fragments are also organised in so-called assumption classes (assumption class has a different meaning here than in CM). Here, an *assumption class* is defined by a set of binary approximation and contradictory relations between the model fragments involved. The *approximation* relation orders the model fragments with respect to their complexity. It is strictly domain-dependent and must, therefore, be imposed by the designer of the model fragment library. The approximation relation is irreflexive, anti-symmetric and transitive, and therefore, it defines a partial ordering on a set of model fragments [134]. Such a partial ordering of model fragments is called an assumption class. Two model fragments are *contradictory* if their respective operating conditions or assumptions are inconsistent. This relation is also transitive. It is assumed that two model fragments are contradictory only if one is an approximation of the other. Hence, all model fragments in an assumption class are contradictory with one another. Since the only source of inconsistency lies in the approximation relation, a model based on a set of model fragments Φ is inconsistent if and only if there exists an assumption class Φ^{AC} and two

model fragments $\mu_i, \mu_j \in \Phi$ such that $\mu_i \in \Phi^{AC} \wedge \mu_j \in \Phi^{AC}$.

The model fragment selection algorithm is based on a specific version of the approximation relation explained earlier, called *causal approximation*. A model fragment μ_i is an approximation of μ_j if $\langle P^s(\mu_i), C^o(\mu_i) \cup C^t(\mu_i) \rangle \subset \langle P^s(\mu_j), C^o(\mu_j) \cup C^t(\mu_j) \rangle$. In other words, μ_i is an approximation of μ_j if μ_i contains a subset of the target-participants of μ_j and each constraint relating target-participants in μ_i has its equivalent in μ_j. It can be proven that, when a set of model fragments does not imply a consistent and complete causal model, a causally approximate set of model fragments does not imply a consistent and complete causal model either. This property is called the upward failure property and it enables a considerable reduction in the complexity of the search algorithm.

The search algorithm underlying this compositional modeller essentially works as follows. First, the most complex valid model is constructed. This is achieved by checking whether the structural conditions of the most complex model fragment of each assumption class matches the scenario and substituting the source-participants of the scenario if the match succeeds. The validity of the resulting model is then checked. A model is valid if an onto-causal mapping can be constructed from it that matches the expected behaviour. An onto-causal mapping matches the expected behaviour if the causal relations implied by the expected behaviour are also implied by the onto-causal mapping. Additional constraints on the onto-causal mapping may be included in the operating conditions of certain model fragments. A subset of each $C^o(\mu)$ may consist of causal orderings between target-participants, representing the function of the component described by the model fragment. Next, the model constructed in this initial phase is subsequently simplified by searching through possible combinations of causal approximations of the originally selected model fragments. During this search, a set of model fragments Φ_1 defines a simpler model than a set of model fragments Φ_2, if $\forall \mu_i \in \Phi_1, \exists\ \mu_j \in \Phi_2,\ (\mu_i = \mu_j) \vee (\text{causal-approximation}(\mu_i, \mu_j))$. From the simplest models found in this way, the final model may be selected using preference constraints or some other form of additional processing and selection.

2.2.5 TRIPEL: relevant influences enabled compositional modelling

Tailoring relevant influences for predictive and explanatory leverage (TRIPEL) [149, 150] is a compositional modeller designed for predicting a system's behaviour with respect to certain variables of interest, under hypothetical operating conditions. The task specification of this compositional modeller consists of a set of variables of interest (target-participants) and a set of driving conditions. Driving conditions are a partial initial state description that may include a value or a rate of change for certain variables of interest.

TRIPEL's inference mechanisms then search for an adequate model. In the general framework of compositional modelling as given in section 2.1, the inference mechanisms consist of searching adequate model fragments and the model is defined by the resulting set of model fragments. In TRIPEL, a model fragment $\mu = \langle P^s, P^t, C^s, C^o, c^t, A \rangle$ where c^t is a single QPT influence relating the target-participants P^t. Each element of $P^t(\mu)$ is a variable. The model fragment selection process considers the adequacy of each individual target-participant and influence in the emerging model rather than model fragments as a whole. The model fragments, however, contain part of the information on which adequacy of target-participants and influences are based, such as the operating conditions $C^o(\mu)$ and assumptions $A(\mu)$.

In TRIPEL, a model is considered to be *adequate* if no less complex model exists that contains influences and variables at the relevant level of detail. Influences and variables can be aggregated into less detailed alternatives and the permitted aggregation is defined in terms of the time scale of a problem. A *time scale* represents the relative speed at which phenomena occur, e.g. seconds, hours, months, etc. Additionally, TRIPEL formalises what approximations, i.e. simplifications, are allowed using time scale information. This enables TRIPEL to resolve much ambiguity in model fragment selection by means of choosing the appropriate time scale for the problem. Time scale information is integrated in the model fragment library as follows. On the one hand, if $c^t(\mu)$ is a functional or indirect influence then it is considered instantaneous. On the other hand, if $c^t(\mu)$ is a differential or direct influence then it represent the effect of a process. In the latter case, there must be an assumption $a_i \in A(\mu)$ that represents the fastest time scale on which the effect of the differential influence is significant.

Other knowledge used in the determination of the adequacy of target-participants

and influences with respect to the task description consists of aggregation hierarchies. An aggregation hierarchy of target-participants defines how sets of target-participant add up to another target-participant. Because target-participants in TRIPEL represent numeric properties of source-participants, this aggregation hierarchy forms a more detailed variant of the "part-of hierarchy" of the compositional modeller. In an aggregation hierarchy of influences, child-nodes represent the set of influences that explain the influence represented by their parent node. Each influence that is explained represents the same phenomenon as the set of influences that explain it. The explained influence, however, is more approximate or simpler.

For a given time scale of the problem, the aggregation hierarchies and time-scale information jointly determine the simplifications that are allowed. For example, the modeller may treat influences that occur on a slower time scale than that of the problem as insignificant. Influences that occur on a time scale that is faster than that of the problem may be replaced by the so-called quasi-static approximations[102, 88]. Separate target-participants may be replaced by their aggregates when they equilibrate on a time scale at least as fast as the time scale of the problem itself.

The simplicity of a model is expressed by the total number of variables in the model. This notion of simplicity contrasts with that of most other compositional modellers. In [134, 135, 136], for example, a model M_1 if simpler than a model M_2 if for each model fragment that constitutes M_1, M_2 contains a model fragment with equal or greater complexity or detail. Although this criterion may render model fragment selection more efficient, it is less generally applicable. The latter concept of simplicity can not, for example, compare a model that represents a small subsystem in great detail with a model that represents a larger system containing that subsystem with little detail. Consequently, the latter criterion of simplicity limits the associated compositional modeller's ability to determine the appropriate scope of a system automatically.

The inference mechanism of TRIPEL performs a best-first search for the simplest model that is valid. Allowed simplifications are determined as described above. Validity is defined by a number of rules, called adequacy constraints. These constraints define which variables must be included in the model, which variables may be exogenous and which must be endogenous, etc. The simplest set of target-participants and influences that satisfies these adequacy constraints is considered the most adequate model. The presence of adequacy constraints may require considerable domain context

dependent knowledge, of course.

2.2.6 DME: device modelling environment

DME [87] is a system that provides an environment for the design of physical, especially electro-mechanical devices. DME is built on the compositional modeller described in [86, 106]. The task specification in DME is an initial state description and a query that consists of a set of quantities whose values must be predicted, the set of target-participants that are considered exogenous (as opposed to TRIPEL, DME does not determine exogenous variables automatically), a set of assumptions that must hold in all simulated states and a set of constraints over target-participants that must be enforced. DME aims at constructing the simplest model that meets the requirements of the query and then simulating it from the initial state specification.

The model fragment library in DME is organised as follows. The structure of the model fragments closely follows the general framework of section 2.1. However, model fragments are assumed to represent a physical system's component or physical phenomenon. Different model fragments representing the same component or phenomenon must have the same set of source-participants and structural conditions and are organised as follows. A Composite Model Fragment (CMF) Φ^{CMF} is a set of model fragments such that $\forall \mu_i, \mu_j \in \Phi^{CMF}$, $[P^s(\mu_i) = P^s(\mu_j) = P^s(\Phi^{CMF}] \wedge [C^s(\mu_i) = C^s(\mu_j) = C^s(\Phi^{CMF})] \wedge [A(\mu_i) = A(\mu_j) = A(\Phi^{CMF})]$. CMFs are DME's way to represent the same phenomenon under the same assumptions, but under different operating conditions. The operating conditions typically describe mutually exclusive ranges of variables. DME also assumes that operating conditions can not be used to differentiate between operating conditions. As such, assumptions are used (similar to the $\mathcal{C}(\text{exists}(p_i^s))$ assumptions used in CM) to introduce new target-participants. Consequently, $\forall\ \mu_i, \mu_j \in \Phi^{CMF}, P^t(\mu_i) = P^t(\mu_j)$.

Different CMF's that represent the same physical component or phenomenon under different assumptions (including target-participants), are grouped in assumption classes (this is yet another definition of assumption class). More formally, an assumption class Φ^{AC} is a set of composite model fragments such that $\forall \Phi_i^{CMF}, \Phi_j^{CMF} \in \Phi^{AC}$, $[P^s(\Phi_i^{CMF} = P^s(\Phi_j^{CMF}] \wedge [C^s(\Phi_i^{CMF}) = C^s(\Phi_j^{CMF})]$. The CMFs are partially ordered with respect to the underlying assumptions. This ordering of CMFs is defined by a

directed acyclic graph in which the links are labelled by sets of literals. These literals represent the differences in assumptions between CMF. Positive (or negative) relevance literals represent the addition (or removal) of assumptions, with respect to source-participants or target-participants, when switching between model fragments in the direction of the associated arc.

Similar to the approach in [134, 135, 136], DME's concept of model adequacy assumes a model fragment library in which replacing a model fragment by a more relevant model fragment involves replacing a relatively simpler model fragment by a more complex and detailed one. Within this framework, positive relevance literals denote addition of complexity and detail whilst negative relevance literal denote removal of complexity and detail. Other literals represent changes in assumptions, other than relevance claims, between model fragments of the same assumption class.

Model fragment selection is achieved by means of two procedures of relevance reasoning. The first determines what target-participants are relevant to the posed problem. For each of the initial variables in the query, the selection algorithm backward chains through all assumption classes in which model fragments exist with equations (postconditions) that may causally explain the query variables. These causal explanations are produced by the postconditions, which are equations that define functional relations between variables. For each set of assumption classes that explains one of the query variables, the second procedure identifies the simplest CMF that is consistent with respect to the assumptions stated in the query and the assumption classes from which a CMF is already instantiated. This choice of assumption class and CMF must be revised each time a new CMF and associated assumption class is instantiated. The assumptions of each instantiated CMF are then added to the modelling assumptions of the current model. Its inputs, i.e. the variables of this CMF that have not yet been explained by the current model, are added to the backward chaining queue of the first procedure. Finally, this CMF is added to the current model as a direct causal explanation for the variable that triggered its instantiation. As mentioned earlier, this procedure may have added modelling assumptions to the current model, and hence previously selected CMFs may no longer be valid. Therefore, the first procedure discards any CMF whose assumptions are no longer valid from the model and replaces it by the next simplest CMF in the associated assumption class that is consistent with the rest of the model being constructed.

This backward chaining mechanism performs the same function as the combination of the object completion algorithm in CM [45], which selects the model components to be added to the scenario using a "part-of hierarchy" and uses domain specific rules to complete the candidate modelling environments. However, DME presents a more uniform approach that relies directly on the domain theory and does not depend on any specific rules nor on the specific decomposition of the scenario. Nevertheless, the independence between assumptions and model fragments in CM allows for more flexibility in the modelling formalism.

Similar to the compositional modeller described in [134, 135, 136], DME follows causal influence paths through the domain theory. Whereas the approach in [134, 135, 136] first constructs the most complex model and simplifies it once it is found, DME maintains the simplest CMF of each assumption class and only adjusts it when necessary. DME's method may, therefore, result in considerable savings when complex assumption classes are involved, although its criterion of simplicity is also similar to that in [134, 135, 136] and has the same limitations as explained in section 2.2.5.

DME is also similar to TRIPEL in that the model fragment selection procedure searches through the causal relations between target-participants covered by the model fragments. However, in TRIPEL, the knowledge used to determine the appropriate trade-off between detail and simplicity is formalised in different forms that are specific to the type of simplifications. In this way, TRIPEL deals with aggregation hierarchies of influences and target-participants, time-scale information associated with influences, etc. This permits TRIPEL to use generic (at least with respect to a domain) meta-level knowledge to determine the allowed simplifications. In DME, all of this information must be contained within CMFs and assumption classes. Consequently, DME's model fragment library must be more carefully constructed such that each consistent combination of model fragments can produce a model.

2.2.7 Diagnostic process based compositional modelling

In its most general form, a diagnostic system determines the actual configuration of a system and establishes how this configuration differs from an appropriate one. Model based diagnosis [69] is a research area in which techniques are developed to predict

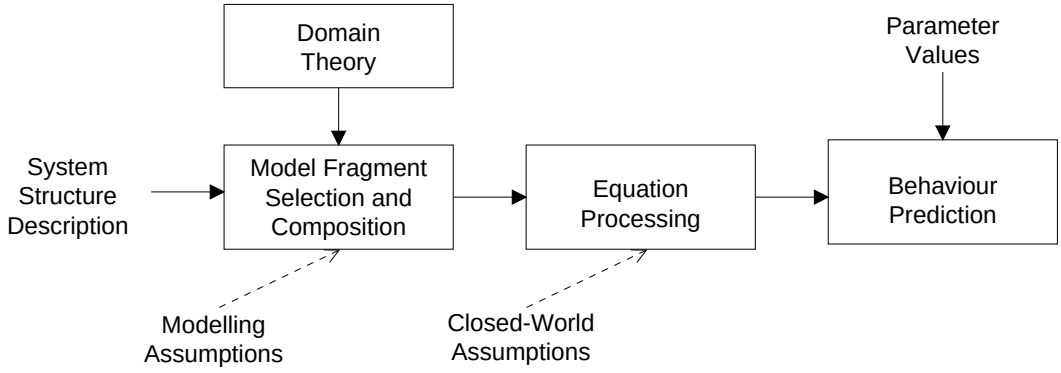


Figure 2.6: General architecture of process-based diagnosis [77]

a system's behaviour from a technical level description, often called the structure or model of the system, and to trace inconsistencies with the observed behaviour back to deviations from that model.

Early model based diagnostic engines employed a single component-connection model of the system, where the components may exhibit different behaviours. By means of observations, the sets of potential behaviours of the components are reduced in order to ultimately find the components that are malfunctioning. Appropriate descriptions of ecological systems, however, typically involve processes that occur due to the combined presence of a number of prerequisite elements under certain conditions or constraints. For example, the process of ozone formation in the lower atmosphere may occur as a result of high solar radiation and the presence of certain substances. Therefore, different presumed configurations of the system require different models instead of a single one.

In process-based diagnosis [165, 166], such models are generated by means of compositional modelling. As shown in figure 2.6, the initial scenario is translated by a compositional modeller into a conceptual representation (more specifically, qualitative influence diagrams (QID) as discussed in section B.1.2). As with any compositional modeller, this involves making modelling assumptions. The resulting representations (QIDs) are then translated into constraints by applying closed-world assumptions to the sets of influences that causally affect the same variable. These constraints, combined with the quantity specifications for certain parameters, are used to extrapolate the system behaviour.

If this system behaviour is inconsistent with given observations, diagnostic can-

didates (i.e. minimal sets of assumptions that may contradict the observations) are generated to explain the discrepancy between actual observations and model predictions. These candidates consist of modelling assumptions and closed-world assumptions [35]. Candidates are tested by creating a new model without them, by retracting all the assumptions they contain. Modelling assumptions are retracted by searching for a consistent scenario model that is not based on a superset of the contradictory assumptions. A closed-world assumption is false only if an unanticipated influence affects the respective variable. Therefore, a model fragment must be found that introduces such an additional influence and a new closed-world assumption must be formulated to construct the new constraint. This search can be focused with respect to the model fragments that causally affect the variable in question. An ATMS is used to trace which new assumptions must be added and which assumptions must be removed to/from the model.

This approach to diagnosis is more general than many others because it can identify new elements, external to those expected to be part of the system, that cause the unwanted behaviour, provided that model fragments are defined in the domain theory that describe the effect of these external elements. For this reason, the approach is potentially suitable for diagnosing ecological systems [164]. An ecological system does not consist of components that malfunction nor subsystems that operate according to a specific failure mode. Instead, there usually are disturbances, i.e. unanticipated external influences that interrupt the natural equilibrium, that account for the unwanted behaviour. If present in the domain theory, this diagnostic engine can help to identify candidate sets of such disturbances.

2.2.8 Key benefits

Compositional modelling has been part of useful application in many different domains [96]. As an automated modelling approach, it has been found to provide the following key benefits over alternative approaches:

- *Generality*: Although compositional modelling stems from research efforts in the domain of qualitative reasoning on physical systems, its application area need not be limited to physical systems or mathematical models. The basic concept behind compositional modelling consists of substituting a set of model

fragments that comply with the format of definition 2.1. Some of the existing compositional modellers are indeed suitable for many different domains [46]. They use model fragments to describe the generic processes and utilise no domain specific representation formalisms. These processes are identifiable parts of a system that can be modelled by a number of variables and by constraints imposed over these variables. Most domain systems could be perceived as composed of such processes.

- *Rich representational framework*: Developments in compositional modelling enable the designer of a model fragment library to organise domain knowledge in representational languages with useful feature. These include: inheritance hierarchies [135], level of detail orderings on component models [136, 150, 106], postconditions that can be both discrete event rules and continuous functional relations [85], varying degrees of accuracy vs. generality by taking advantage of quantitative and semi-quantitative information [52, 48, 136] and time-scale based abstractions [150].

  Of course, tools must exist that can apply the resulting models to solve given problems. This is enabled by simultaneous developments in qualitative and semi-quantitative simulators [103], the predominant problem solvers created and used in conjunction with compositional modellers. These research efforts improve the richness of the representational framework of compositional modelling and can express a great variety of aspects and problems of a carefully selected domain.

- *Composable domain theory*: The knowledge base of a compositional modeller consists of composable fragments of theoretical knowledge regarding a domain. Specific applications of such knowledge need not be known at the time of implementation of the model fragment. Only the types of variable and equation that the problem cases will require must be known a priori. In general, the range of different systems in most domains are constructed from a much smaller number of classes of system components or processes. Therefore a reasonably small number of model fragments can model an exponentially larger range of systems. The component structure of the domain theory itself eases the maintenance and extension of this knowledge base, which is generally a difficult task in knowl-

edge based applications. However, central to compositional modelling is a decomposed knowledge base. This decomposition allows component knowledge to be verified and validated independently from unrelated knowledge. In other words, model fragments can be adapted or added without much intervention to the remainder of the knowledge base.

- *Different perspectives with a single domain theory*: A compositional modeller can compose a range of target models for a given scenario. It does not define a functional relation between the domain of scenarios and the domain of target models. Instead, a compositional modeller can use information on user preferences and abilities [10], explicit time scale specifications [150] or explicit assumptions to guide its search for a suitable model. As such, compositional modelling provides an efficient and economical means of storing modelling knowledge of different types of perspective. When different perspectives affect different aspects of a system, they can be modelled by separate sets of model fragments. Model fragments of these different sets may be combined to produce a consistent and complete system model. A relatively small model fragment library can therefore be used to represent a large number of models, representing different combinations of perspectives on the same system.

2.3 Ecological modelling

In order to tackle the intrinsic uncertainty and incomplete knowledge in the ecological domain, ecologists have tried and tested an enormous variety of modelling paradigms. To illustrate this issue, this subsection presents a number of important paradigms and related issues.

2.3.1 Representation formalisms

The Holy Grail of modelling languages is a representation formalism that enables the user to express all possible situations in a succinct and easily understandable way. Until such a language is found, ecologists are limited by the modelling formalism and the modelling tools they use.

2.3.1.1 Mathematical models

Mathematics is the language employed by most sciences. A system of ordinary differential equations (ODEs), for example, provides a concise method of expressing continuous behaviour. Because of its representational power, many other languages are based on ODEs.

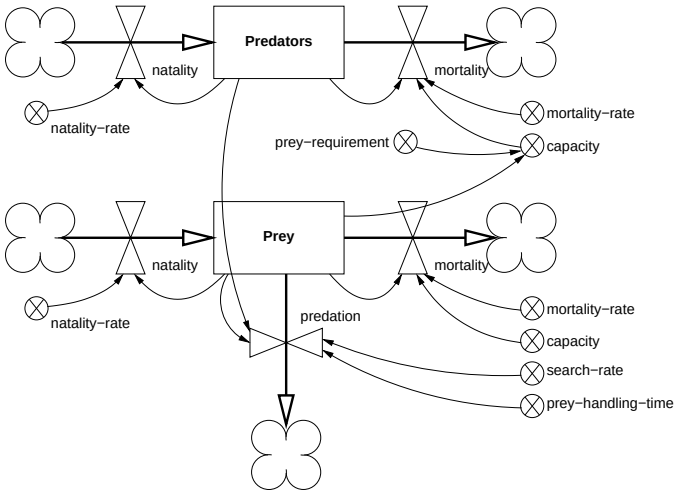


Figure 2.7: A system dynamics model of the predator prey scenario

System dynamics [57] provides a useful interface to ODEs, simplifying model design, construction and maintenance. Figure 2.7 shows a sample ecological model, using the system dynamics framework. It has been formulated using the ecological principles presented in [79] and [178]. This predator prey representation is a typical component of ecological models of related population sizes and growth. In this model, the squares are stocks representing population sizes (N_{prey} and N_{predator}) and the thick arrows are flows representing birthrates (B_{prey} and B_{predator}), death-rates (D_{prey} and D_{predator}) and a rate of predation (P_{prey}). The circles denote the remaining variables and parameters of the model whose influence is depicted via the thin arrows. As a further illustration, a more precise mathematical formulation of this model is (for $i \in \{\text{prey}, \text{predator}\}$):

$$\frac{d}{dt}N_{\text{predator}} = B_{\text{predator}} - D_{\text{predator}} \quad \frac{d}{dt}N_{\text{prey}} = B_{\text{prey}} - D_{\text{prey}} - P_{\text{prey}}$$

$$B_i = b_i \times N_i$$

$$D_i = d_i \times N_i \times \frac{N_i}{C_i}$$

$$P_{\text{prey}} = \frac{s_{(\text{prey,predator})} \times N_{\text{prey}} \times N_{\text{predator}}}{1 + s_{(\text{prey,predator})} \times N_{\text{prey}} \times t_{(\text{prey,predator})}}$$

$$C_{\text{predator}} = r_{(\text{predator,prey})} \times N_{\text{prey}}$$

2.3.1.2 Programming languages

Programming languages are a type of conceptual representation of the behaviour of an ecological system. As such, they can be perceived as models. Of course, conventional *procedural programming languages* are difficult to understand and maintain as they include a specification of how the behaviour must be simulated and of other implementational details that detract from the actual ecological system behaviour.

In recent decades, *object-oriented programming* (OOP) languages have become more widely used for implementing complex (ecological) software systems. They differ from conventional procedural languages in that they encapsulate the attributes and the behavioural descriptions of each type of component or process in a single class definition. These class definitions can be instantiated with respect to individual components and processes to form objects. These objects interact with one another via simple interfaces and message passing mechanisms without knowing about the actual implementation of the other. As such, OOP models of ecological systems are somewhat easier to design and maintain than their procedural equivalents [109].

Despite these drawbacks, programming languages are commonly used as a modelling paradigm because they can symbolically describe very complex behaviour. To facilitate the design and maintenance of such models, techniques from software engineering are being ported to the ecological modelling domain and this is still an area of ongoing research [80].

2.3.1.3 Knowledge-based approaches

A growing number of successful paradigms from AI research are being ported to the ecological modelling domain. These include:

- *Rule-based models*, which employ sets of rules of the form

  if *antecedent* then *consequent* else *alternative*

  to describe system behaviour. Such rules have been employed successfully in knowledge-based systems to formalise expert knowledge. In ecological modelling, rules are employed to propagate the effect of structural changes and different operating conditions in the system [92].

- *Cellular automata*, which are essentially one or multi-dimensional arrays of cells. Each cell has a state and contains a number of rules or equations that prescribe the cell's future state based on its own current state and those of its neighbours. As such, the predominant application of cellular automata is the simulation of changes in the spatial distribution of some phenomenon over time, such as population migration [26], vegetation growth [184] and fire [71, 9].

- *Artificial neural networks* (ANNs) are networks of simple units, called neurons, that take one or more inputs (possibly from other neurons) to produce a single output (possibly to other neurons). A given set of inputs can be propagated through this network to produce a set of outputs. Typically, the connections between neurons are weighted in order to increase or decrease the value transferred via that connection during propagation. By employing a learning algorithm to adapt the weights of the network, the ANN can be trained to mimic the behaviour of a real-world system. As such, ANNs have been a powerful machine learning approach [121]. ANNs are relevant to ecological modelling because they can be trained to mimic the behaviour of an ecological system. They have been used as an alternative to statistical regression techniques [113] or to describe the behaviour of complex components, such as schools of fish, in larger ecological simulation models [30, 42]. Of course, as ANNs are constructed by pure induction and are hardly decomposable, they do not seem to suit acting as the knowledge representation tool for compositional modelling.

2.3.1.4 Hybrid approaches

The growing amount of interdisciplinary research in ecological modelling does not only lead to the introduction and exploration of new approaches, but it also results in the integration of different formalisms in a consistent paradigm. The idea behind such work is to produce a hybrid modelling paradigm that possesses the strengths of multiple complementary approaches.

Simile, for example, is a tool [129] that is essentially based on system dynamics principles, but extends it with additional representational formalisms, including object oriented design, cellular automata and rule-based approaches. For example, a Simile model may contain one or more submodels (e.g. representing population growth) and relations between such generic submodels (e.g. representing predation). These submodels can be instantiated several times (e.g. once for each species the modellers want to incorporate), with different sets of parameters, to match a given scenario. As such, Simile already incorporates a rudimentary version of a model fragment.

2.3.2 Uncertainty handling techniques

As explained earlier, knowledge of how ecological systems work is incomplete. Models that predict a single precise behaviour for a set of variables of interest are bound to be inaccurate. To tackle this issue, ecological modellers have also employed techniques for reasoning explicitly about uncertainty.

2.3.2.1 Statistical/stochastic models

Conventional systems of differential equations exemplify deterministic models because they extrapolate a single behaviour from a given state. Stochastic ecological models concern random events that occur in ecological systems. Two particular approaches are:

- *Markov chains*, which describe a system by means of a finite set of states of the system and a set of probabilities dictating the likelihood of transition from one state to another over time [141]. As such, a Markov chain can reason about the probability distribution the state can be in. A typical application of Markov

chains is modelling vegetation dynamics to predict future distributions of vegetation species [175, 7, 108].

- *Bayesian networks*, which offer an alternative sophisticated representation of a stochastic system. Essentially, a Bayesian network is a directed graph in which the nodes correspond to observations or hypothesised properties of the system being modelled and the arcs are associated with a table of conditional probabilities relating the likelihood of the antecedent to the likelihood of the consequents. By applying Bayes law over connected nodes in a Bayesian network, known probabilities can be propagated throughout the network [141]. Bayesian networks are primarily used as an ecological modelling approach for analysing the potential impact of decisions [114].

2.3.2.2 Approximate/qualitative models

The recommended usage of precise numerical models is to perform multiple simulations with them, employing different sets of parameters covering the most optimistic and pessimistic scenarios [54]. This is because what ecologists require is a family of behaviours rather than a single behaviour. To increase computational efficiency (and the ability to cope with imprecise knowledge) approximate and qualitative representation formalisms have been devised. The following are two well known examples of such modelling methods:

- *Fuzzy ecological models*, which are based on *fuzzy differential equations* [91]. Instead of precise numbers, fuzzy numbers, which are fuzzy sets describing to what a extent certain precise numbers belong to the fuzzy number, are employed. Techniques for solving fuzzy differential equations are devised to propagate such fuzzy numbers and extrapolate an imprecise behaviour for the system. The resulting imprecise behaviour description better reflects the intrinsic uncertainty of the knowledge about the values of parameters and variables [154, 8].

- *Qualitative ecological models*, which employ qualitative differential equations, instead of ordinary differential equations, upon which a qualitative simulation algorithm is applied. They produce a family of behaviours of the system under

simulation, each containing a description focusing solely on the qualitatively distinct changes in the behaviour. Qualitative ecological models have been found to be useful in decision support systems [76, 75] and in eco-educational applications [153, 15].

2.3.3 Suitability of compositional modelling for eco-modelling

This work aims to produce a compositional modelling approach for ecological modelling. However, given the plethora of approaches to ecological modelling, it can be difficult to claim and test this feature in general. In this work, the system dynamics formalism will be adopted. Because it encompasses concepts from multiple formalisms, such as differential equations, rules, influences, stocks and flows, it offers a very rich representational framework for the theoretical developments herein. It is expected that the techniques developed in this thesis will generalise to alternative knowledge representation formalisms as well. In [74], for example, the early compositional ecological modelling ideas presented in [94] have been applied to OOP models of ecological systems.

Compositional modelling lends itself more to some ecological modelling approaches than others. Generally speaking, compositional modelling requires an ecological modelling approach (1) to generate models that consist of clearly identifiable components and (2) to construct these models from a body of ecological knowledge. Most of the approaches discussed in this section meet these requirements, although there are some obvious exceptions, such as ANNs.

2.4 Issues of compositional ecological modelling

This section highlights a number of issues important to compositional modelling for eco-model construction.

2.4.1 Non-monotonicity due to abductive reasoning

For ecological modelling, the model construction may involve abductive reasoning, where the antecedent, in an implication from which a fact or a required property can

Compositional modeller	Mode of reasoning	Requirements
CM	partly non-monotonic	query, simplest model
QPC, SIMGEN, SQPC	monotonic	operating conditions
Causal approximations	monotonic	expected behaviour, simplest model
TRIPEL	monotonic	operating conditions, simplest model
DME	monotonic	query, simplest model
Process-based diagnosis	non-monotonic	observed behaviour

Table 2.1: Mode of reasoning of compositional modellers

be derived, is hypothesised to be true. That antecedent may prove to lead to inconsistencies, and hence, the earlier hypothesis may need to be retracted. Thus, abductive reasoning usually requires some form of non-monotonic reasoning.

However, most compositional modellers employ a monotonic mode of reasoning. Given a scenario and a task specification, they apply the available knowledge to produce a consistent model. In addition to consistency, other requirements may be imposed upon the scenario model. However, these requirements are specified such that the search for a scenario model that satisfies them can be implemented in a monotonic fashion.

Consider the existing compositional modellers, their mode of reasoning and imposed requirements are summarised in table 2.1. The absence of non-monotonic reasoning in most conventional compositional modellers can be explained by considering each type of requirement individually:

- *Query*: A query specifies a set of variables in a mathematical model which need to be related to one another. That is, certain variables in the query need to be explained in terms of other variables in the query. Because the knowledge base consists of model fragments containing influences that relate one set of variables to another, finding a scenario model that satisfies the query involves searching for a set of paths from one set of variables to another. This can be implemented by a conventional backtracking search algorithm.

- *Operating conditions*: An operating condition is a range of allowed values for one or more variables. Operating conditions are only employed in conjunction with a simulator, e.g. QSIM [101]. Given an initial state (from which it can be determined whether the operating conditions are satisfied) an initial model is

constructed and employed for simulation. The simulation proceeds until at least one of the operating conditions is no longer satisfied. At that point, a new model is constructed to satisfy the new operating conditions.

- *Expected behaviour*: Behaviours expected from a model generated by a compositional modeller are usually specified by a set of causal influences that are expected to be part of the model. These causal influences are contained in model fragments that are included in or excluded from the model by the search algorithm. As such, the required expected behaviours can again be found by a conventional backtracking search algorithm.

- *Simplest model*: Model simplicity is measured by means of a heuristic for which the knowledge representation is optimised. In some approaches, model fragments are grouped in partially ordered sets, called assumption classes. Increasing or reducing simplicity involves replacing a model fragment by another one, taken from the same assumption class. The search algorithms employed by these compositional modellers make modelling choices by selecting model fragments from these assumption classes, and hence, maximised simplicity can be achieved as part of this search algorithm. Other approaches employ a similar approach but decide on simplicity at the level of individual influences between variables or at the level of assumptions in assumption classes.

The compositional modeller developed in this thesis can not make any of these simplifying assumptions. Therefore, it employs a truth maintenance system to enable the abductive mode of reasoning.

2.4.2 Granularity selection

Previous work on compositional modelling has not addressed the issue of disaggregation of collections of individuals. Grain choices are made with respect to components and processes of physical systems and it is implicitly assumed that the resulting decomposed components or processes are entirely distinct from the original component or process. The compositional ecological modeller will have to be able to generate models of populations of identical individuals. Obviously, any part of such a collection of individuals is still a collection of individuals.

In this thesis, partitioning the collection of individuals is called disaggregation. Disaggregation is different from the aforementioned decomposition of components and processes in physical systems because disaggregate models are similar to their originals and disaggregations can be combined. To illustrate these points, consider the disaggregations of prey into age-classes and with respect to gender. Both disaggregations can be combined to form populations such as “young males”. The model of this group of prey is similar to that of prey, because young male prey may die or be eaten by predators (just like the original prey population).

2.4.3 Preferences for subjective model fragment selection

All existing compositional modellers impose formal requirements upon the scenario model. However, this is not always sufficient in ecological modelling. Because ecological models are typically gross abstractions of very complex and yet only partially understood systems, information on which modelling approach is better is limited, and opinions differ between ecologists. To reflect this reality, some means of specifying subjective user preferences for different modelling approaches is necessary and a calculus to reason about such preferences is devised in this work.

2.4.4 Representation formalism

Although the principles of compositional modelling do not depend a specific representation formalism, most compositional modellers employ inference mechanisms that are dependent upon a knowledge representation formalism that is specific to the physical systems domain. The work presented in [45] and [86, 106] requires that the scenario or the knowledge base be structured in some form of component hierarchy or network. Such formalisms are clearly not suitable to represent ecological systems. Indeed, in general, the decisions involved in ecological model construction can not be reduced to selecting component or process models for each component and process specified in the scenario. Some approaches, for instance QPC [29, 46, 47] and the approach presented in [135, 136] are embedded in qualitative representations such as qualitative process theory [50] and causal approximations [134]. Although these representation formalisms have proven their benefit in areas such as qualitative physics, ecologists usually prefer alternative modelling formalisms they are more familiar with.

2.4.5 Model fragment structure

Several compositional modellers employ a flat model fragment structure. They use model fragments that translate objects or processes indicated in the scenario directly into the model. The work presented in [86, 106] (i.e. the compositional modeller for the device modelling environment (DME) [87]) is an example following this approach. For each component in the scenario, it selects a component model from a set of alternatives.

A similar approach can be used to select which phenomena are to be incorporated in a model, assuming that for each type of phenomenon, only one model is available in the knowledge base. For example, process-based diagnosis [165, 166] also employs a flat model, and for the application domains discussed in that work, this was a well justified presumption.

In general, however, one chosen model for a certain process may introduce new sets of modelling choices for other processes. In the example of section 2.3.1.1, part of the Holling predation model (equation 3.16) depends on the existence of a capacity variable (C_{predator}) introduced by the logistic growth model of the predator. If this occurs, the latter modelling choices must by represented by model fragments that are only instantiated if the former is selected. Because these two types of modelling choice are orthogonal to each other, the flat model fragment structure is unsuitable for the needs of the present work.

2.5 Summary

This chapter provided the background to this thesis. Section 2.1 presented a formal problem specification of compositional modelling, defined as an automated modelling approach that relies on a knowledge base containing alternative models of different components or processes of domain systems. It distinguishes between alternative models for the same component or process by means of operating conditions and assumptions.

Based on the general specification, a survey of individual compositional modellers was presented in section 2.2. From this survey, it can be concluded that whilst the methods employed for knowledge representation and inference are similar, the model

fragment selection, model composition and model evaluation techniques can be very different. The differences are largely due to the variety of the types of problem specification, which depends on the situations that the respective compositional modellers are devised to address.

Naturally, it follows that the ecological domain comes with its own set of issues that need to be considered. For this, section 2.3 presented an overview of common ecological modelling formalisms. Of these formalisms, those that consist of identifiable component models and that primarily require knowledge for their construction were deemed suitable for compositional modelling. For the purpose of illustration throughout this dissertation, the system dynamics formalism was selected. In section 2.4, the issues that need to be addressed in the remainder of this thesis were discussed.

Chapter 3

Knowledge Representation in Compositional Modelling

As any other knowledge based approach, compositional modelling requires a knowledge representation formalism for the specification of its inputs, its outputs and its knowledge base. This chapter introduces the formalism employed in this work. It is loosely based on a proposed standard knowledge representation formalism known as the compositional modelling language (CML) [13]. However, CML itself is not adopted since it is geared too much towards engineering applications. The formalism introduced herein is not intended to be a standard for compositional ecological modelling work, but merely a means to test and illustrate the inference mechanisms and search algorithms discussed in the following chapters. With the exception of the work presented in [165, 166], this work is the first application of compositional modelling for the purpose of an ecological and system dynamics model repository. However, the compositional modeller discussed in [165, 166] employs compositional modelling as a means of generating models for different system configurations. It is not equipped to produce alternative models for the same configuration. For these reasons, only the most essential constructs have been used and the formalism is open-ended to allow future extensions.

This chapter is organised as follows. Section 3.1 presents the most basic constructs of participants, relations and assumptions. Section 3.2 builds on that and discusses scenarios and scenario models. The important problem of model disaggregation is also introduced in this section. Next, section 3.3 formalises and illustrates all of the

constructs that are employed in the knowledge bases that are employed in the following chapters. Finally, section 3.4 summarises the contents of this chapter.

3.1 Compositional modelling primitives

The most primitive constructs in a compositional modeller are participants, relations and assumptions. This section describes each of these concepts.

3.1.1 Participants

Participants refer to the objects of interest, which are involved in the scenario or scenario model. Some of the previous work in compositional modelling refers to these as *individuals* and *quantities*. These names would be quite unfortunate for the present application. Ecological models typically describe the behaviour of populations rather than individuals and it is often hard to distinguish quantities. Also, it is often hard to distinguish quantities from other kinds of participants because the present approach does not translate scenario level constructs directly into variables (quantities) and equations.

The objects of interest that are represented by participants may be *real-world objects*, i.e. objects the modeller is interested in, or *conceptual objects*, such as variables that express features of real-world objects in a mathematical model. When applying compositional modelling to ecological systems, a population of a species constitutes a typical example of a real-world object. A variable that expresses the number of individuals of this species would be an example of the conceptual object, which may eventually appear in a mathematical model.

It is natural to group objects that share something in common into classes. Therefore participants are grouped into *participant classes*, which are defined to be sets of participants that share certain features in common. What participant classes a participant belongs to affects how that participant may be utilised. In what follows, a participant class will be referred either as a set (of the participants that belong to it) or by means of a name assigned to that class. In particular, the name of a participant class is the type of the participants that are members of that participant class. For example, the aforementioned participant that represents the size of a population could be

referred to as a “variable”.

3.1.2 Relations

Relations describe how the participants are related to one another. As with participants, some relations represent a real-world relationship, such as

$$\text{predation(frog, insect)} \tag{3.1}$$

Other relations may be conceptual in nature, such as the mathematical relationship

$$\text{deaths} = \text{d} \times \text{size} \times \frac{\text{size}}{\text{capacity}} \tag{3.2}$$

In general, relations consist of a *functor* and a set of *arguments*. An argument is a constant, a variable or a term. Constants and variables refer to participants. A *constant* names a specific participant (i.e. a single instance) whereas a variable can be instantiated by a number of participants. In what follows, a *variable* will be denoted by a name that begins with a question mark (e.g. `?population-size`). *Terms* are themselves constructs that consist of a functor and a set of arguments.

Depending on what is being expressed by the relation, different notations may be utilised. When using predicate logic to describe the world (for example (3.1)), the default notation is:

$$\text{functor}(\text{argument}_1, \ \ldots, \ \text{argument}_n) \tag{3.3}$$

When a relation expresses a mathematical equation (for example (3.2)), an infix notation is usually preferred.

$$\text{argument}_1 \quad \text{functor} \quad \ldots \quad \text{functor} \quad \text{argument}_n \tag{3.4}$$

Both these notations will be utilised when discussing the mechanisms of compositional modelling and explaining sample modelling problems. In implementation, the compositional modeller employs a single notation (list notation) for all relations based on LISP style lists.

$$(\text{functor} \quad \text{argument}_1 \quad \ldots \quad \text{argument}_n) \tag{3.5}$$

All sample inputs and outputs of the compositional modeller (including the knowledge bases listed in the appendices) employ the notation as shown in (3.5). For example, relations (3.1) and (3.2) can be rewritten as follows using this notation:

```
(predation frog insect)
(== deaths (* size (/ size capacity)))
```

3.1.3 Assumptions

Assumptions are a special type of relations. They are hypotheses or presumptions that can be made in the construction of a scenario model. As a scenario does not provide an assumption set, it is up to the compositional modeller to find a consistent and appropriate set of assumptions upon which to base the scenario model.

Each assumption has a type, a subject and a specificity. The *type* of an assumption indicates the kind modelling choice that it describes. The *subject* of an assumption is a participant, a relation or a set of participants or relations about which the assumptions presume certain features. The *specificity* of an assumption identifies what hypothesis is made about the subject.

Definition 3.1. Given an assumption type and an assumption subject, set of *all* assumptions that have that type and that subject is said to be the *assumption class* for that type and subject.

The current version of the implemented modeller employs three types of assumption: relevance, model and disaggregation assumptions [94], [95]. *Relevance assumptions* state what phenomena are to be included in or excluded from the scenario model. The general format of a relevance assumption is shown in (3.6). The phenomenon that is incorporated in the scenario model when describing a relevance assumption is identified by ⟨*name*⟩ and is specific to the subsequent participants or relations. For example, relevance assumption (3.7) states that the growth of participant `?population` is included in the model.

`(relevant` ⟨*name*⟩ [{⟨*participant*⟩} | ⟨*relation*⟩]`)` (3.6)

```
(relevant growth ?population)
```
(3.7)

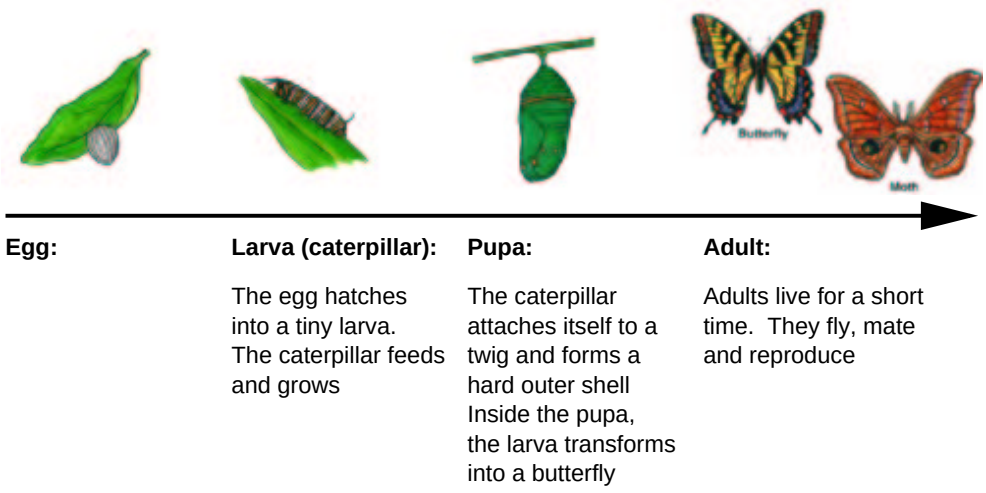


Figure 3.1: The life of a butterfly

As all types of assumption, relevance assumptions have a subject and a specificity. The *subject of a relevance assumption* is the combination of the phenomenon name and the participants or the relation in the assumption. The *specificity of a relevance assumption* consists of the types of the participants (or the participants in the argument terms of the relation). The reason for this is that a phenomenon may involve multiple meanings depending on the type of the participants. For example, the growth phenomenon for the average population of a species involves changes in population size through births and deaths of individuals. As figure 3.1 shows, the butterfly population is divided over four different incarnations, each with its own dynamics.

Model assumptions specify which type of model is utilised to describe the behaviour of a certain participant or relation. The formal specification of a model assumption is given in (3.8). Often the ⟨*name*⟩ in (3.8) corresponds to the name of a known (partial) model of the phenomenon or process being described. The example in (3.9) states that the population `?population` is being modelled using the logistic approach (see [178]).

`(model` [⟨*participant*⟩ | ⟨*relation*⟩] ⟨*name*⟩`)` (3.8)

`(model ?population logistic)` (3.9)

The subject and specificity of a model assumption are different from those of a

relevance assumption. The participant or relation constitutes the *subject of a model assumption* and the model name is the *specificity of a model assumption*. Note that the example may give rise to the question of what participant or relation forms the most appropriate subject of the model assumption. This question is implicitly circumvented by leaving it to the knowledge engineer. For simple problems, this may work fine. In more complex situations, however, one may require sets of models for various variables that describe different attributes of the population (e.g. population size, percentage carriers of a disease, etc). This can be achieved by making the subject more specific and employing the assumption `(model ?population-size logistic)`. A more complex, but potentially clearer way, involves adding a relation `(growth-of ?population)`, which means "growth of population `?population`", and making it the subject of the assumption.

Disaggregation assumptions describe the way in which participants corresponding a piece of information on a population of individuals should be decomposed. As opposed to conventional engineering models, ecological models do not always describe the behaviour of the system at the level of each individual component. If the reader imagines a mathematical model of an ant colony that formalises the behaviour of each ant, each piece of food and each piece of soil, (s)he will understand why population models may be necessary. In order to improve the level of detail of the model, the populations (e.g. the ants) may be decomposed according to certain dimensions (e.g. function or role in the colony, gender, physical location, age, etc.) and the specific behaviour of each subpopulation can then be considered.

More generally, a disaggregation assumption is a kind of grain assumption. *Grain assumptions* indicate the degree of detail at which certain participants are being modelled. In engineering applications, grain assumptions are typically made by selecting components at the right level of detail from a network defined by the so-called part-of relations [45]. In such a network, components are related to the components it consists of. Within this work, no type of assumption is defined to describe the way in which components are decomposed because such knowledge can easily be described by means of a set of model assumptions. In section 8.2.2.1, it will be shown how this is done.

With improvements in qualitative reasoning research [103], grain choices can also be made at the level of the variables in the model [27]. In addition to conventional

equations defined over variables denoting pure numeric quantities, models in terms of qualitative influences [50], qualitative differential equations [101], order of magnitude relations [145], or fuzzy relations [160] may also be used. Again, such grain choices can be made by means of a model assumption.

As mentioned earlier, ecological modelling requires a third type of grain assumptions to express the grain of a participant describing information on a population. Because populations are groups of individuals, they can be disaggregated into a number of subgroups which are still populations (and hence, may be disaggregated further). However, this has not been considered in earlier compositional modelling work, and therefore, a new type of assumption and a novel theory is developed in this thesis.

Disaggregation assumptions are employed to express this new grain choice [95]. They take the general form shown in 3.10, where the ⟨*name*⟩ names the disaggregation and the ⟨*enumerator*⟩ refers to the domain of values used to index the subclasses. The example shown in 3.11 is the assumption for disaggregating a population `?population` into subpopulations according to age classes. The participant `?n` refers to a domain of values, e.g. a set of integers $\{1,\ldots,m\}$ or something more descriptive such as $\{\text{child}, \text{adult}, \text{elderly}\}$.

`(disaggregation` [⟨*participant*⟩ | ⟨*relation*⟩] ⟨*name*⟩ {⟨*enumerator*⟩}`)` (3.10)

`(disaggregation ?population age-classes ?n)` (3.11)

Note that a disaggregation assumption may contain multiple enumerators, each referring to a domain of index values, in order to to represent multi-dimensional disaggregations. An typical example of such a multi-dimensional disaggregation is disaggregation with respect to a spatial grid, where two or three dimensions are needed for the, say, x, y and z coordinates.

3.2 Scenarios and scenario models

Ultimately, the aim of compositional modelling is to translate a scenario into a scenario model in order to perform a certain model-based reasoning task (see figure 1.2). Both a given scenario and a scenario model are described by a set of participants and their

relations. The following definition reflects this observation formally.

Definition 3.2. A *model* μ is a tuple $\langle P, R \rangle$ where P is a set of participants and R is a set of relations holding over these participants.

In general, the main input to the compositional modeller is a representation (which is itself a model) that formally describes the system of interest by means of an accessible formalism. This model, which normally consists of (mainly) real-world participants and their interrelationships is called the *scenario*. A typical example of a scenario is a physical specification of the system as provided by the component connection representation formalism.

The output of the compositional modeller is another model that describes the system in a more complex formalism, which the model-based reasoner can employ readily. Such a model, which normally contains conceptual participants and interrelationships, is called a *scenario model*. Within this work, both systems dynamics [57, 58] and ordinary differential equations (ODEs) will be employed as the modelling formalisms.

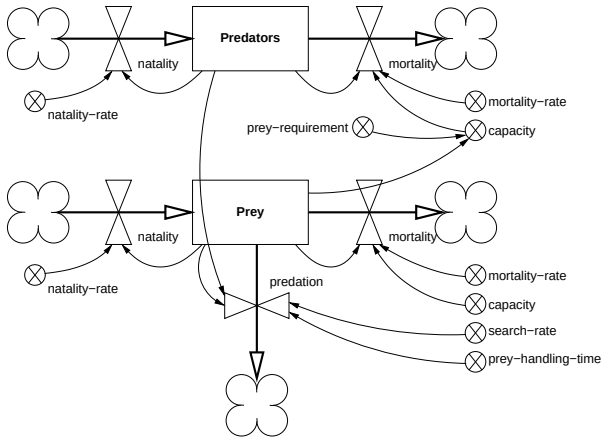
Note that most of the previous work in compositional modelling implicitly assumed some hard distinction between a scenario and a scenario model. However, apart from the difference in syntax and semantics of the corresponding formalism, both are models and the task of the modeller is translation from one formalism to the other. As a result of this approach, the rules in the knowledge base do not have to translate parts of the scenario directly into the scenario model. This facilitates the representation of ecological modelling knowledge, which is incremental in nature. That is, ever more detailed models can be constructed by adding new participants and relations over the new and existing participants.

When the intermediate models can not be considered as such, additional constraints on the inference mechanism must be added to restrict the use of the more detailed knowledge when the prerequisites have not been fulfilled (see for example [45]). In domains where the model may consist of lengthy sequences of influences, e.g. ecological modelling, this unnecessarily inhibits the knowledge representation.

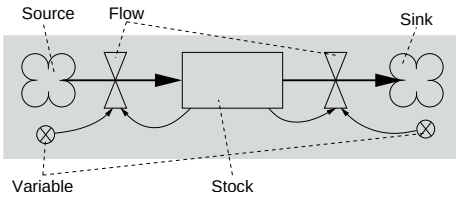
3.2.1 Ecological scenarios and ecological scenario models

Consider an example of a simple ecological scenario, representing predation between two species, is given below:

Participants: `predator`
`prey`
Relations: `(predation predator prey)`



(a) Stock flow diagram of the predator prey scenario



(b) How to read a stock flow diagram

Figure 3.2: Sample system dynamics model using the stock flow formalism

Figure 3.2(a) is a stock-flow diagram showing a sample scenario model corresponding to the above ecological scenario, within the system dynamics framework. Figure 3.2(b) depicts the names of the different symbols found in a typical stock-flow diagram. The model in figure 3.2(a) has been formulated using the ecological principles presented in [79] and [178]. This predator prey model is a typical component of ecological models of related population sizes and growth. In this model, the rectangles are stocks representing population sizes (N_{prey} and N_{predator}) and the thick arrows are flows representing births or natality (B_{prey} and B_{predator}), deaths or mortality (D_{prey} and D_{predator}) and predation (P_{prey}). The $\otimes$ symbols denote the remaining variables and parameters of the model whose influence is depicted via the thin arrows. The more

precise mathematical formulation of this model is (for $i \in \{\text{prey}, \text{predator}\}$)

$$\frac{d}{dt}N_{\text{predator}} = B_{\text{predator}} - D_{\text{predator}} \quad \frac{d}{dt}N_{\text{prey}} = B_{\text{prey}} - D_{\text{prey}} - P_{\text{prey}} \tag{3.12}$$

$$B_i = b_i \times N_i \tag{3.13}$$

$$D_i = d_i \times N_i \times \frac{N_i}{C_i} \tag{3.14}$$

$$P_{\text{prey}} = \frac{s_{(\text{prey},\text{predator})} \times N_{\text{prey}} \times N_{\text{predator}}}{1 + s_{(\text{prey},\text{predator})} \times N_{\text{prey}} \times t_{(\text{prey},\text{predator})}} \tag{3.15}$$

$$C_{\text{predator}} = r_{(\text{predator},\text{prey})} \times N_{\text{prey}} \tag{3.16}$$

The participants in this model are $N_i, B_i, D_i, C_i, P_{\text{prey}}, b_i, d_i, s_{(\text{prey},\text{predator})}, t_{(\text{prey},\text{predator})}$, and $r_{(\text{prey},\text{predator})}$. The equations are known to the compositional modeller as relations written in the notation as defined in 3.5.

More complex scenarios consist of larger sets of participants and different types of relations between them. Subsets of these participants and relations will correspond to different parts of the scenario model. For example, the above scenario and scenario model is part of many complex population models.

3.2.2 Disaggregate models

Unlike physical systems, which are comprised of individual components, ecological systems contain populations of individuals, and hence, component networks are insufficient a representation for making all potential grain choices. However, populations of individuals can be divided into subpopulations according to certain criteria, each of which is a population in its own right that can be divided further. This process is hereafter referred to as *disaggregation into classes* or the construction of a disaggregate model.

Definition 3.3. A model $\langle P^d \cup P^c \cup P', R^d \cup R \cup R' \rangle$ is said to be a *disaggregate* of another $\langle P^a \cup P^c, R^a \cup R \cup R'' \rangle$ if there exits a surjection $\sigma : P^d \rightarrow P^a$ and an injection $\rho : \text{dom}(\rho) \rightarrow R^a$ (with $\text{dom}(\rho) \subset R^d$), such that $\forall p^a \in P^a, \exists \{p^d_1, \ldots, p^d_n\} \subset P^d, (\sigma(p^d_1) = p^a \wedge \ldots \wedge \sigma(p^d_n) = p^a)$ and $\forall r(p^d_q, \ldots, p^d_r, p^c_v, \ldots, p^c_w) \in R^d$ - $\text{dom}(\rho)$, $r(\sigma(p^d_q), \ldots, \sigma(p^d_r), p^c_v, \ldots, p^c_w) \in R^a$. The participants in P^a and the relations in R^a are said to be *disaggregated* into the participants of P^d and the disaggregate relations of R^d, respectively.

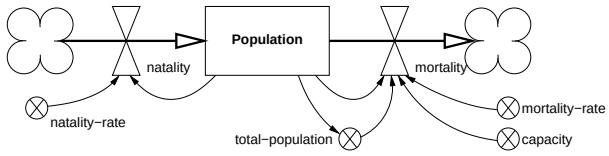


Figure 3.3: Aggregate logistic population growth model

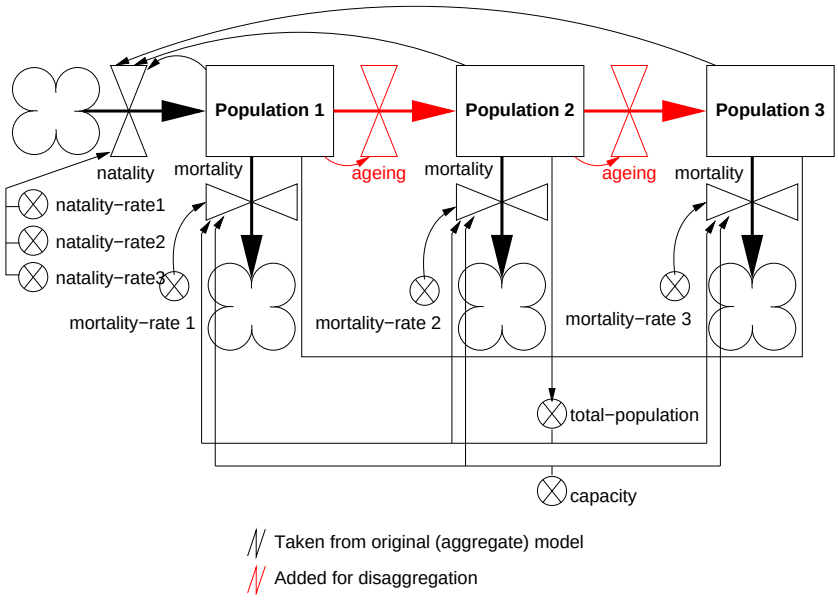


Figure 3.4: The logistic population growth model disaggregated into age classes

To illustrate the concept of a disaggregate model consider a simple scenario of logistic population growth [178]. The standard (i.e. aggregate) model, as shown in figure 3.3, can be formalised by means of ODEs such that:

$$\frac{d}{dt}N = C^+(B), \frac{d}{dt}N = C^-(D) \tag{3.17}$$

$$B = r \times N, D = d \times N \times \frac{T}{C}, T = C^+(N) \tag{3.18}$$

where N is the population size, B is the number of births within a given time interval, D is the number of deaths within the same time interval, r is the reproduction rate, d is the death rate, T is the total relevant population and C is the population capacity. A disaggregate model of this logistic growth model with respect to, say, n age classes may be:

$$\frac{d}{dt}N_0 = C^+(B_i), \frac{d}{dt}N_i = C^-(D_i) \tag{3.19}$$

$$B_i = r_i \times N_i, D_i = d_i \times N_i \times \frac{T}{C}, T = C^+(N_i) \tag{3.20}$$

$$\frac{d}{dt}N_i = C^-(M_i), M_{i+1} = \text{delay}(N_i, t_i) \tag{3.21}$$

A graphical illustration of this model (for $n = 3$) is shown in figure 3.4. In the aggregate model, $P^a = \{N, D, r, d\}$, $P^c = \{B, T, C\}$ and equations (3.17) and (3.18) are relations in R^a. In accordance with definition 3.3, the participants of P^a are mapped to $P^d = \{N_0, \ldots, N_n, D_0, \ldots, D_n, r_0, \ldots, r_n, d_0, \ldots, d_n\}$, those in P^c are retained and the equations in R^a are mapped onto (3.19) and (3.20) respectively, which are the relations in R^d. Migration from one class to another is described by the set of participants $P' = \{M_1, \ldots, M_n\}$ and by the set of relations containing the equations expressed by (3.21).

3.3 The knowledge base

To construct scenario models from a given scenario, a compositional modeller relies on a knowledge base. This section describes the important constructs in the knowledge base that will be employed by the compositional modelling techniques introduced in the subsequent chapters.

3.3.1 Composable relations

In ecological model building, sometimes relations can not be created in one go. When multiple phenomena affect how several participants are related to one another, the resulting relations may be different for each combination of phenomena. For example, the model on vegetation growth to be discussed in section 8.2 contains an eradication event switch that, when activated (i.e. the boolean is set to true), causes the remaining population of the species to die. Multiple phenomena may trigger this event switch, such as frost, drought, insufficient energy reserves, fire, etc. In the system dynamics formalism, the models of these phenomena influence a single switch, representing the eradication event, that has a single if-then-else statement associated with its value, which contains a term for each of the phenomena affecting it. Because the combination of phenomena is not known at the time the knowledge base is created, composable relations are created for each of the phenomena. And from these composable relations, a single equation can be constructed at the time of the knowledge base is utilised.

More generally, compositional modelling tackles the problem of combinations of phenomena affecting a single relation by postponing the construction of the relation until the model is ready. The knowledge base generates smaller, composable relations, which can be combined into the required relation at a later stage. The use of composable relations enables the knowledge base to consider all combinations of the phenomena that may affect a relation, by representing each phenomenon individually rather than everything being precompiled altogether. Because only the component parts (i.e. the composable relations) of relations need to be represented rather than all possible combinations of them, this leads to a smaller and more effective knowledge base.

Composable relations are those containing composable functors and for which a method of composition exists. Here, a method of composition describes how a complete set of composable relations can be composed, provided composition is possible. The composable functors employed are essentially taken from [13] with a new addition: composable selection. A summary of such composable relations is presented in table 3.1.

The composable relations taken from [13] are easy to understand. The formulae f in $v = C^{+}(f)$ and $v = C^{-}(f)$ represent terms (respectively f and $-f$) of a sum. The formulae f in $v = C^{\times}(f)$ and $v = C^{\div}(f)$ represent factors (respectively f and $\frac{1}{f}$)

Name	Syntax (infix notation)	Syntax (prefix notation)
Addition	$?\text{var} = C^{+}(\text{formula})$	`(== ?var (C-add formula))`
	$?\text{var} = C^{-}(\text{formula})$	`(== ?var (C-sub formula))`
Multiplication	$?\text{var} = C^{\times}(\text{formula})$	`(== ?var (C-mul formula))`
	$?\text{var} = C^{\div}(\text{formula})$	`(== ?var (C-div formula))`
Selection	$?\text{var} = C^{\text{if},p}(\text{antecedent}, \text{formula})$	`(== ?var (C-if` antecedent formula `:priority` p`))`
	$?\text{var} = C^{\text{else}}(\text{formula})$	`(== ?var (C-else formula)`

Table 3.1: Composable functors and composable relations

of a product. A more formal representation of the methods of combination for these composable relations follows shortly.

Ecological models often contain selection statements which declare that one equation must be employed when a condition is satisfied and another if the condition is not satisfied. A selection statement can be composed from a set of composable "if" relations $v = C^{\text{if},p}(a, f)$ and a single composable "else" relation $v = C^{\text{else}}(f_{\text{else}})$. The composable "if" relations consist of a priority p, an antecedent a and a formula f. The semantics of a composable "if" relation $v = C^{\text{if},p}(a, f)$ are:

$$(\neg\exists v = C^{\text{if},p_i}(a_i, f_i), p_i \prec p \wedge a_i) \rightarrow (a \rightarrow f) \tag{3.22}$$

That is, if none of the antecedents of composable "if" relations, which have a higher priority (i.e. lower rank) and which assign the same variable v are true, then formula f follows from the antecedent a. The semantics of composable "else" are similar to composable "if", but the priority is (implicitly) lower than the composable "if" relations and no antecedent can be satisfied. More formally, the semantics of $v = C^{\text{else}}(f)$ are

$$(\neg\exists v = C^{\text{if},p_i}(a_i, f_i), a_i) \rightarrow f \tag{3.23}$$

To be concise, table 3.2 summarises what composable relations can be joined to form compounded relations. Composable relations can be joined together if they can be rewritten to form terms of a mathematical operation such that the order in which the terms must be considered is uniquely defined or it does not affect the result. Composable additions and the negations of composable subtractions can be written as terms of

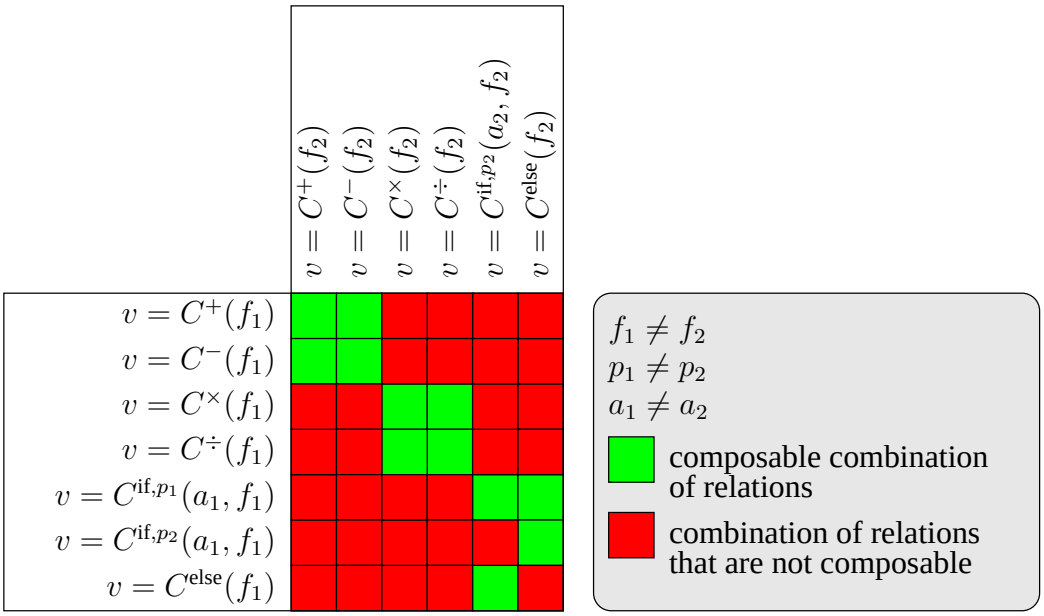


Table 3.2: Composable combinations of relations

a sum. Composable products and the inversions of composable divisions can be written as terms of a product. Because addition and multiplication are commutative and associative operators, the order in which terms are considered is irrelevant. The order in which the composable selections must be considered is defined by the priorities (or is implicit in the case of C^{else}). Therefore, composable selections can be combined with one another provided no two composable "if" relations have the same priority.

In order to formally define the actual rules of composition, the sets of all composable relations with the same functor for a given model $\langle P, R \rangle$ are defined first.

$$R(v, C^{+}) = \{v = C^{+}(f_i) \mid (v = C^{+}(f_i)) \in R\} \tag{3.24}$$

$$R(v, C^{-}) = \{v = C^{-}(f_i) \mid (v = C^{-}(f_i)) \in R\} \tag{3.25}$$

$$R(v, C^{\times}) = \{v = C^{\times}(f_i) \mid (v = C^{\times}(f_i)) \in R\} \tag{3.26}$$

$$R(v, C^{\div}) = \{v = C^{\div}(f_i) \mid (v = C^{\div}(f_i)) \in R\} \tag{3.27}$$

$$R(v, C^{\text{if}}) = \{v = C^{\text{if},p_i}(a_i, f_i) \mid (v = C^{\text{if},p_i}(a_i, f_i)) \in R\} \tag{3.28}$$

$$R(v, C^{\text{else}}) = \{v = C^{\text{else}}(f_i) \mid (v = C^{\text{else}}(f_i)) \in R\} \tag{3.29}$$

The rules of composition (see equations 3.30, 3.31 and 3.32) state how a given set of composable relations can be rewritten as a single compound relation. Each of

these rules contains a complete set of all composable relations in the antecedent. The antecedent of rule (3.30) contains the sets of all composable addition and subtraction relations with the same participant v in the left-hand side. According to table 3.2, only these can be combined with one another. It is self-evident that not additional relation (i.e. $w = C^+(f)$, $w = C^-(f)$ or a relation $v = C^x(f)$ where $x \notin \{+,-\}$) can be added to this antecedent. Also, no composable relation $v = C^+(f_x)$ or $v = C^-(f_x)$ can be removed from the antecedent as f_x should clearly be added to or subtracted from the other terms for the equation to be correct.

Similarly, the antecedent rule (3.31) contains the complete sets of for composable multiplication. Finally, the antecedent of rule (3.32) is satisfied for the complete sets of composable if and else relations with the same left-hand participant v provided the priorities are strictly ordered (i.e. no two priorities are equal), and there is only a single composable else relation. The latter two conditions are added because two composable if relations with the same priority or two composable else relations can not be compounded. The consequents of the rules of composition explain how these complete sets of composable relations can be joined. This is simply a matter of applying the appropriate mathematical operation to the provided terms.

$$\begin{aligned} R(v,C^+) = \{v = C^+(f_{1+}), \ldots, v = C^+(f_{m+})\} \wedge \\ R(v,C^-) = \{v = C^-(f_{1-}), \ldots, v = C^-(f_{n-})\} \rightarrow \\ v = f_{1+} + \ldots + f_{m+} - (f_{1-} + \ldots + f_{n-}) \end{aligned} \tag{3.30}$$

$$\begin{aligned} R(v,C^\times) = \{v = C^\times(f_{1\times}), \ldots, v = C^\times(f_{m\times})\} \wedge \\ R(v,C^\div) = \{v = C^\div(f_{1\div}), \ldots, v = C^\div(f_{n\div})\} \rightarrow \\ v = \frac{1 \times f_{1\times} \times \ldots \times f_{m\times}}{f_{1\div} \times \ldots \times f_{n\div}} \end{aligned} \tag{3.31}$$

$$\begin{aligned} R(v,C^{\text{if}}) = \{v = C^{\text{if},p_1}(a_1, f_1), \ldots, v = C^{\text{if},p_m}(a_m, f_m)\} \wedge \\ R(v,C^{\text{else}}) = \{v = C^{\text{else}}(f_{\text{else}})\} \wedge p_1 \prec \ldots \prec p_m \rightarrow \\ v = \text{if } a_1 \text{ then } f_1 \\ \text{else } \ldots \\ \text{if } a_m \text{ then } f_m \\ \text{else } f_{\text{else}} \end{aligned} \tag{3.32}$$

An example of the need for compositional addition is shown in figure 3.5. Here, the

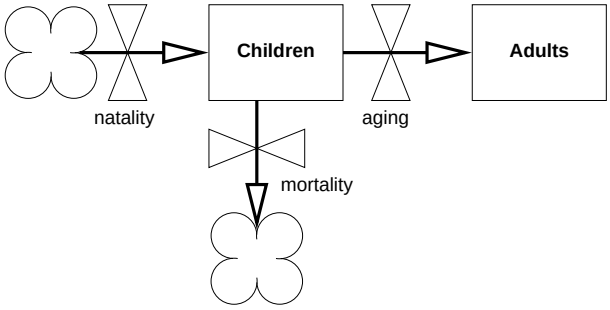


Figure 3.5: Example of compositional addition

size of the "Children" stock is governed by three flows: flow(natality,source,Children), flow(mortality,Children,sink) and flow(ageing,Children,Adults). Flows are translated by relationship type declarations (see 3.3.5) into compositional additions and subtractions as follows:

$$\text{flow(natality,source,Children)} \rightarrow \text{Children} = C^{+}(\text{natality})$$
$$\text{flow(mortality,Children,sink)} \rightarrow \text{Children} = C^{-}(\text{mortality})$$
$$\text{flow(ageing,Children,Adults)} \rightarrow \text{Children} = C^{-}(\text{ageing}) \wedge \text{Adults} = C^{+}(\text{ageing})$$

Rule 3.30 translates these three compositional relations into:

$$\text{Children} = +\text{natality} - (\text{mortality} + \text{ageing})$$

Examples of compositional multiplication and compositional selection will be discussed in chapter 8.

3.3.2 Model fragments

The model fragments are the rules that are utilised to derive a scenario model from a given scenario. They can be defined as follows.

Definition 3.4. A model fragment μ is a tuple $\langle P^s, P^t, \Phi^s, \Phi^t, A\rangle$ where

- $P^s(\mu) = \{p_1^s, \ldots p_m^s\}$ is a set of variables, called *source-participants*,
- $P^t(\mu) = \{p_1^t, \ldots, p_n^t\}$ is a set of variables, called *target-participants*,
- $\Phi^s(\mu) = \{\phi_1^s, \ldots, \phi_v^s\}$ is a set of relations, called *structural conditions*, whose free variables are elements of P^s,

- $\Phi^t(\mu) = \{\phi_1^t, \ldots, \phi_s^t\}$ is a set of relations, called *postconditions*, whose free variables are elements of $P^s \cup P^t$,
- $A(\mu) = \{a_1, \ldots, a_t\}$ is a set of relations, called *assumptions*,

such that for $i = 1, \ldots, s$:

$$\forall p_1^s, \ldots, \forall p_m^s, \exists p_1^t, \ldots, \exists p_n^t \; \phi_1^s \wedge \ldots \wedge \phi_v^s \rightarrow (a_1 \wedge \ldots \wedge a_t \rightarrow \phi_i^t) \tag{3.33}$$

The source-participants and structural conditions (which may be instantiated by other relevant model fragments in describing the given scenario) describe the conditions under which the model fragment is applicable, the assumptions express the presumptions that are deemed appropriate for the problem at hand, and target-participants and postconditions represent the new objects/variables and relations that extend the model under these conditions. Note that the participants, relations (i.e. structural conditions and postconditions) and the assumptions correspond to the definitions provided in section 3.1. Hence, a model fragment fits in the template: (source-participants $\wedge$ structural conditions) $\rightarrow$ (assumptions $\rightarrow$ (target-participants $\wedge$ postconditions)).

Below is an example of a model fragment from which equations (3.13) and (3.14) are instantiated on the conditions that a stock, birth-rate flow and death-rate flow exist describing a population and on the assumption that the logistic model for population size is used.

```
(defModelFragment logistic
  :source-participants
    ((?population :type population)
     (?size :type stock)
     (?birth-flow :type flow)
     (?death-flow :type flow))
  :structural-conditions
    ((flow ?birth-flow source ?size)
     (flow ?death-flow ?size sink)
     (size-of ?size ?population))
  :assumptions
    ((model ?size logistic))
  :target-participants
    ((?birth-rate :type variable :name birth-rate)
     (?death-rate :type variable :name death-rate)
     (?density :type variable :name total-population)
```

```
    (?capacity :type variable :name capacity))
 :postconditions
  ((== ?birth-flow (* ?birth-rate ?size))
   (== ?death-flow (* ?death-rate ?size ?density))
   (== ?density (C+ (/ ?size ?capacity)))
   (population-density-of ?density ?population)
   (population-capacity-of ?capacity ?population)))
```

This model fragment states the following. Given a population `?population`, a stock `?size` and two flows `?birth-flow` and `?death-flow`, such that

- `?birth-flow` is a flow from a source into `?size`,
- `?death-flow` is a flow from `?size` into a sink, and
- `?size` describes the size of population `?population`,

and assuming that `?size` is described using the logistic model, then the following must be created:

- four new variables `?birth-rate`, `?death-rate`, `?density`, and `?capacity`,
- three new relations describing the equations:

$$\texttt{?birth-flow} = \texttt{?birth-rate} \times \texttt{?size}$$

$$\texttt{?death-flow} = \texttt{?death-rate} \times \texttt{?size} \times \texttt{?density}$$

$$\texttt{?density} = C^{+}\left(\frac{\texttt{?size}}{\texttt{?capacity}}\right)$$

- two new relations denoting that `?density` represents the density of population `?population` and `?capacity` represents the capacity of population `?population` that is sustainable by the environment.

3.3.3 Model properties

Properties are features of interest to the application or person requiring a scenario model. In practice, it is possible to define any feature that can be described a set of participants and a set of relations as a property. More formally,

Definition 3.5. A model property Π is a tuple $\langle P^s, \Phi, \pi \rangle$ where $P^s(\mu) = \{p_1^s, \ldots p_m^s\}$ is a set of *source-participants*, a predicate calculus sentence Φ whose free variables are

elements of P^s, and a relation π, whose free variables are also elements of P^s, such that

$$\forall p_1^s, \ldots, p_m^s \forall \Phi \rightarrow \pi$$

For example, the model property below describes when a "math-object" `?m` is *endogenous*. It represents the situation whenever either `?m` $=$ `*` or $\frac{d}{dt}$`?m` $=$ `*` is true (where `*` matches any constant or formula), `?m` is deemed to be endogenous.

```
(defproperty endogenous
  :source-participants ((?m :type math-object))
  :structural-condition ((or (== ?m *) (d/dt ?m *)))
  :property (endogenous ?m))
```

Alternatively, the property describing that a "math-object" is *exogenous* can be defined by means of the endogenous property: i.e. a "math-object" is exogenous when such an object `?m` exists and it is not endogenous (i.e. `(not (endogenous ?m))`).

```
(defproperty exogenous
  :source-participants ((?m :type math-object))
  :structural-condition ((not (endogenous ?m)))
  :property (exogenous ?m))
```

Required model properties can be specified in two different ways: either globally as goals for the scenario model construction or locally as a required purpose of a certain model fragment (by means of the keyword `:purpose-required`). The example below is an alternative version of an earlier model fragment which introduces a variable representing population size under the condition that the growth phenomenon is relevant to a given population. The `:purpose-required` field states that any instantiation of this model fragment (and hence, the creation of a variable for `?p-size`) requires the new `?p-size` to be endogenous.

```
(defModelFragment
  :source-participants ((?p :type population))
  :assumptions         ((relevant growth ?p))
  :target-participants ((?p-size :type stock))
  :postconditions      ((size-of ?p-size ?p))
  :purpose-required    ((endogenous ?p-size)))
```

3.3.4 Participant class declaration and participant type hierarchies

In general, participant classes need not be defined. However, certain types of participants may be described in terms of other interesting participants (e.g. important quantities), irrespective of the modelling choices. This feature provides syntactic sugar for important relations between participants, making it easier to declare required properties for a scenario model in terms of scenario level participants. For example, the behaviour of a population may be described in terms of population size and growth rate variables:

```
(defEntity population
  :participants (size growth-rate))
```

Such participant class declarations are utilised to deal with participants that describe an important aspect of another participant that is of interest. The participant class declaration in the above example states that size and growth-rate are important features of a population.

Participant class declarations are employed within model fragments to provide a more specific definition of the meaning of the source-participants and the target-participants. In this way, participant specifications are constrained to be a feature of another participant by means of the `:entity` statement, as the following example illustrates:

```
(defModelFragment define-population-growth-phenomenon
  :source-participants ((?p :type population))
  :target-participants ((?pn :type stock :entity (size ?p))
                        (?pg :type variable :entity (growth-rate ?p))
                        (?pb :type flow)
                        (?pd :type flow))
  :assumptions ((relevant growth ?p))
  :postconditions ((== ?pg (- ?pb ?pd))
                   (flow ?pb source ?pn)
                   (flow ?pd ?pn sink)))
```

Participant class declarations may define one class to be an immediate subclass of another. For example, the population participant class of holometabolous insects (e.g. the aforementioned butterfly population) may be defined as a subclass of the population participant class:

```
(defEntity holometabolous-insect-population
  :subclass-of (population)
  :participants (larva-number pupa-number adult-number))
```

In this way, a participant type hierarchy is defined. Each subclass inherits all participants its superclasses (i.e. its immediate superclass and superclasses of superclasses). The concepts introduced above can be defined more formally:

Definition 3.6. A *participant class declaration* is a tuple $\Pi = \langle \Pi_S, P \rangle$ where Π_S is a participant class, called the immediate superclass of the participant class and P is a set of participants classes that describe important features of the participant class.

In general, the participant class declaration defines an assymetric, irreflexive and transitive relation $\mapsto_{\text{is\_a}}$. A pair of participant classes $(\Pi_{\text{super}}, \Pi_{\text{sub}})$ belongs to the relation, denoted $\Pi_{\text{sub}} \mapsto_{\text{is\_a}} \Pi_{\text{super}}$ if Π_{super} is the immediate superclass of Π_{sub} or of a participant class Π_i that is itself a superclass of Π_{sub}. More formally:

$$[\Pi_{\text{sub}} \mapsto_{\text{is\_a}} \Pi_{\text{super}}] \leftarrow \Pi_{\text{sub}} = \langle \Pi_{\text{super}}, P \rangle$$
$$[\Pi_{\text{sub}} \mapsto_{\text{is\_a}} \Pi_{\text{super}}] \leftarrow \exists \Pi_i, ((\Pi_{\text{sub}} \mapsto_{\text{is\_a}} \Pi_i) \wedge (\Pi_i \mapsto_{\text{is\_a}} \Pi_{\text{super}}))$$

3.3.5 Relationship type declaration

A relationship type declaration describes how one generic class of relations can be translated into other relations. This is useful to translate models described using more elaborate representation formalisms into more basic ones. The following relationship type declaration, for example, translates a system dynamics description of a flow, from one stock to another, into differential equations by means of composable relations.

```
(defRelation flow1
  :<= ((flow ?flow ?stock1 ?stock2))
  :=> ((d/dt ?stock1 (C- ?flow))
       (d/dt ?stock2 (C+ ?flow))))
```

Thus, relationship type declarations provide syntactic sugar for a model fragment $\langle \{\}, \{\}, \Phi^s, \Phi^t, \{\} \rangle$. That is, a relationship type declaration is a model fragment without participants or assumptions.

3.3.6 Representing disaggregation

In theory, disaggregate models (see 3.2.2) can be constructed by the use of the normal model fragments. Unfortunately, this would require a significant portion of the model fragments to be rewritten with respect to a required disaggregation. Consider for example a knowledge base that contains model fragments for population growth and predation. To incorporate disaggregation into age-classes in this knowledge base, model fragments must be added for the population growth models that are disaggregated into age-classes and for the predation models where the predator population, the prey population or both are disaggregated.

Matters are further complicated because a subpopulation resulting from a disaggregation could be disaggregated itself. Specifying such chained disaggregations only by means of model fragments would involve writing a set of model fragments for each specific combination of disaggregations. For example, if three possible types of disaggregation would need to be considered (e.g. disaggregation according to age, gender and social function in a society), then 2^3 different versions of the population models would have to be created.

However, as figures 3.3 and 3.4 illustrate, disaggregate models are not so different from the models they are disaggregated from. Normally, in disaggregating a model, some participants and relations stay the same, some are replaced by arrays and some new participants and relations need to be added. By exploiting this observation, potential disaggregations can be specified more concisely. To facilitate the specification of disaggregations and their combinations, the concepts of disaggregation mapping and disaggregation fragments are introduced here.

3.3.6.1 Disaggregation mapping

Since a disaggregate model usually has much in common with the model it is disaggregated from, a knowledge representation is introduced that makes the relations between them explicit. It captures the disaggregation mapping that specifies which participants and relations are the same, and which participants and relations are mapped to arrays of participants and relations:

Definition 3.7. A *disaggregation mapping M* is a tuple $\langle N, \delta_P, \delta_R \rangle$ where

- $N = N_1 \times \ldots \times N_l$, with N_i being a set of domains of indices representing the

names of classes,

- δ_P is a bijection $N_1 \times \ldots \times N_l \times \text{dom}(\delta_P) \rightarrow \text{range}(\delta_P)$ ($\text{dom}(\delta_P), \text{range}(\delta_P) \subset \mathcal{P}$ and $\mathcal{P}$ is the set of all participants),

- δ_R is a bijection $N_1 \times \ldots \times N_l \times \text{dom}(\delta_R) \rightarrow \text{range}(\delta_R)$ ($\text{dom}(\delta_R), \text{range}(\delta_R) \subset \mathcal{R}$ and $\mathcal{R}$ is the set of all relations),

such that $\forall p \in \text{dom}(\delta_P), (\forall i \neq j, \delta_P(\ldots, i, \ldots, p) \neq \delta_P(\ldots, j, \ldots, p))$,

In terms of definition 3.3, the participant mapping δ_P states what participant classes belong to P^a and how they are mapped onto P^d of the disaggregate model. For what follows, a transformation called the generalised participant mapping $\delta'_P : \mathcal{P} \rightarrow \mathcal{P}$ is defined such that for all $p \in \text{dom}(\delta_P), \delta'_P(\ldots, n_i, \ldots, p) = \delta_P(\ldots, n_i, \ldots, p)$ and all $p \in \mathcal{P} - \text{dom}(\delta_P), \delta'_P(\ldots, n_i, \ldots, p) = p$. The bijection δ_R describes how the relations of R^a in definition 3.3 which do not disaggregate according to the defined classes map onto R^d.

For the above example of disaggregating a population into age classes, a suitable disaggregation mapping is:

$$\delta_P(\overrightarrow{i}, N) = N_{\overrightarrow{i}}, \delta_P(\overrightarrow{i}, D) = D_{\overrightarrow{i}}, \tag{3.34}$$

$$\forall p, \text{parameter}(p) \rightarrow \delta_P(\overrightarrow{i}, p) = p_{\overrightarrow{i}}, \tag{3.35}$$

$$\delta_R(\overrightarrow{i}, (\frac{d}{dt}N = C^+(B))) = (\frac{d}{dt}N_0 = C^+(B)) \tag{3.36}$$

This disaggregation mapping states that N, B and D and the parameters b and d are mapped into n age classes. It furthermore specifies a particular mapping for $\frac{d}{dt}N = C^+(B)$ in (3.17) to become $\frac{d}{dt}N_0 = C^+(B)$ in (3.19). As such, the disaggregation mapping describes the explicit links between a disaggregate model and the model it disaggregates.

3.3.6.2 Disaggregation Fragment

By themselves, disaggregation mappings contain insufficient information to enable the construction of disaggregate models from a knowledge base of model fragments that are not disaggregated. As indicated earlier, in addition to the participants and relations that are mapped to themselves or an array of similar participants and relations, disaggregation may involve the creation of new participants and relations. In the ongoing

sample application of disaggregation into age-classes, the disaggregate model contains a number of participants and relations to explain the migration from one age-class to the next. These participants and relations are specific to the age-class disaggregate model and are to be introduced when disaggregation is applied. The compound construct containing this additional information and the disaggregation mapping is called a disaggregation fragment. More formally:

Definition 3.8. A disaggregation fragment is a tuple $\langle P^s, P^t, A, \Phi^s, \Phi^t, M\rangle$ where

- $P^s = \{P^s_1, \ldots, P^s_m\}$ is a set of source-participants,
- $P^t = \{P^t_1, \ldots, P^t_n\}$ is a set of target-participants, called *target-participants,*
- A is the set of assumptions that a disaggregation fragment depends on. All assumption specifications in A refer to assumption instances other than those referred to in the (normal) model fragments in the knowledge base (see below).
- Φ^s is a set of structural conditions (i.e. relations defined over $P^s_q \times \ldots \times P^s_r$),
- Φ^t is a set of postconditions (i.e. relations defined over $P^s_q \times \ldots \times P^s_r \times P^t_v \times P^t_w$), and
- M is a disaggregation mapping

such that

$$\forall \langle P^a, R^a\rangle \in \{\mu_M \mid \mu_S, A \cup \{\neg \text{disaggregation}(P(M))\} \vdash \mu_M\},$$
$$(\mu_S, A \cup \{\text{disaggregation}(P(M))\} \vdash \langle P^d, R^d\rangle)$$

where

$$P^d = \{p^d \mid p^d = \beta(n_1, \ldots, n_l, p^a), p^a \in P^a\} \cup P^t \text{ and}$$
$$R^d = \{r(\delta'_P(n_1, \ldots, n_l, p^a_q), \ldots \delta'_P(n_1, \ldots, n_l, p^a_r)) \mid$$
$$(r(p^a_q, \ldots, p^a_r) \in R^a) \wedge (r(p^a_q, \ldots, p^a_r) \notin \text{dom}(\delta_R))\} \cup$$
$$\{\delta_R(r(p^a_q, \ldots, p^a_r)) \mid (r(p^a_q, \ldots, p^a_r) \in \text{dom}(\delta_R))\} \cup \Phi^t$$

A disaggregation fragment is shown below for the example of splitting up a population into n age classes. It indicates that this disaggregation is applicable to a population `?p` for which a stock `?pn` and two flows `?pb` and `?pd` are known, such that `?pb` is the flow into `?pn` and that `?pd` is the flow out of `?pn`.

```
(defDisaggregationFragment population-age-classes
  :source-participants ((?p :type population)
                        (?pn :type stock :unit population)
                        (?pb :type flow :unit population)
                        (?pd :type flow :unit population))
  :structural-conditions ((stock ?p ?pn)
                          (flow ?pb source ?pn)
                          (flow ?pd ?pn sink))
  :meta-participants ((?n :type integer))
  :assumptions ((disaggregation ?p age-classes ?n))
  :mapping-types (((age-classes ?t) :type (array (0 ?n) ?t)))
  :target-participants
   ((?pn* :type (age-classes stock) :mapped-from ?pn)
    (?pd* :type (age-classes flow) :mapped-from ?pd)
    (? :type (age-classes parameter) :mapped-from (? :type parameter))
    (?pm :type (array (1 ?n) flow))
    (?ts :type (array (0 (1- ?n)) variable)))
  :postconditions
   (((flow ?pb source (?pn 0)) :mapped-from (flow ?pb source ?pn))
    (for (?i 1 ?n) (flow (?pm (1- ?i)) (?pn (1- ?i)) (?pn ?i)))
    (for (?i 1 ?n) (== (?pm i) (delay (?pn (1- ?i)) (?ts ?i))))))
```

The target-participants of this disaggregation fragment specify the new participants to be introduced into a disaggregated model. The first three target-participant specifications describe how individual participants of aggregate model are to be transformed into arrays of participants in the disaggregate model. For example,

```
(?pn* :type (age-classes stock) :mapped-from ?pn)
```

specifies how an individual stock `?pn` in the aggregate model should be transformed into an array of participants of type stock, i.e. `(array (0 ?n) stock)`, in the disaggregate model (where `?pn` refers to one of the source-participants). Similarly,

```
(? :type (age-classes parameter) :mapped-from (? :type parameter))
```

denotes that any participant of type parameter of aggregate model must be transformed into an array of parameters in the disaggregate model. The last two target-participants refer to participants in the disaggregate model that have no equivalent in the aggregate model. For instance,

```
(?pm :type (array (1 ?n) flow))
```

states that a population growth model that is disaggregated into n age-classes must contain $n-1$ (or `(1- ?n)` in LISP notation) new flows. These flows will refer to the ageing flows depicted in figure 3.4.

Similar to the target-participants, the postconditions of the disaggregation fragment describe what new relations need to be introduced into a disaggregated model. Again, there are two types of *new* postconditions in any disaggregate model, those that are translations of postconditions in the aggregate models, and those that are entirely unique to the disaggregate model. An example of the first type is:

```
(flow ?pb source (?pn 0)) :mapped-from (flow ?pb source ?pn))
```

This postcondition states that the original postcondition `(flow ?pb source ?pn)`, denoting a flow of births from a source into the population size stock, in the aggregate model must be transformed into a flow of births from a source into population stock 0 (i.e. the first population stock or the stock containing the youngest members of the population) in the disaggregate model. An example of the second type of postcondition is:

```
(for (?i 1 ?n) (flow (?pm (1- ?i)) (?pn (1- ?i)) (?pn ?i)))
```

This postcondition introduces the aforementioned array of $n-1$ ageing flows depicted in figure 3.4.

Note that this disaggregation fragment contains all of the information introduced in the sample disaggregation mapping shown in section 3.3.6.1. The actual mappings of participants and relations are the target-participants and postconditions employing the `:mapped-from` keyword, followed by a reference to a participant/relation from the aggregate model. Additionally, the disaggregation fragment contains new participants to express migration between age-classes. A graphical illustration of these has been shown in figure 3.4.

3.4 Summary

This chapter has introduced the knowledge representation formalism to be utilised by the compositional modeller presented in this thesis. The basic constructs of this formalism are participants, relations and assumptions. *Participants* are objects and may

refer to real-world objects (e.g. entities in the scenario) or conceptual objects (e.g. variables in the scenario model). *Relations* are predicates that describe how the participants are related to one another. *Assumptions* are a special kind of relation that denote the presumptions that are made in the construction of a required scenario model.

The essential constructs in the knowledge base are model fragments and properties. *Model fragments* are generic partial models that can be instantiated and combined in different ways to suit varying scenarios and the needs of the modeller. Model fragments may make use of composable relations to generate partial equations that can be consolidated after the initial scenario model has been constructed. *Model properties* are formal descriptions of important features of a scenario model, representing conditions that a scenario may be required to satisfy. A number of additional constructs that facilitate the application of the compositional modeller to important types of problem have also been presented. In particular, a *disaggregation fragment* expresses the relationships holding between an aggregate model and its disaggregated alternatives, and *participant class hierarchies* describe ways in which classes of participants are related. Finally, *relationship type declarations* dictate how certain relations must be translated into something more useful when the generated participants and relations are intermediate constructs that require some further processing for further use in the model construction process or to meet the requirements of the problem solver.

Together, these constructs provide a formalism that can express what partial models can be derived from a scenario. The next chapters will show how scenario models are actually constructed, using the knowledge representation scheme introduced here.

Chapter 4

Compositional Modelling as a Constraint Satisfaction Problem

Within this work, a scenario model is constructed by translating the model construction problem to a constraint satisfaction problem (CSP) and then by solving the resulting CSP. This chapter discusses how such a CSP can be constructed. Figure 4.1 shows the overall approach. Basically, the model construction CSP is created by means of two phases. First, the knowledge base is instantiated with respect to a specific scenario. This procedure is discussed in section 4.1. A hypergraph, called the model space, is constructed that links initial or intermediate participant, relation and assumption instances to new participant and relation instances via nodes representing model fragments (see section 3.3.2). This hypergraph is then expanded to incorporate partial disaggregate models by applying disaggregation fragments (see section 3.3.6.2). In the second phase, the model space is translated into a dynamic CSP (DCSP) by exploiting additional requirements and model properties. This procedure is discussed in section 4.2. Other knowledge elements and knowledge-processing phases shown in the figure will be explained in the next chapters.

4.1 Constructing model spaces

The instantiation of part of the knowledge in the knowledge base, with respect to a given scenario, is required to determine which model fragments can be combined with one another. To this end, a model space is constructed which contains the participants

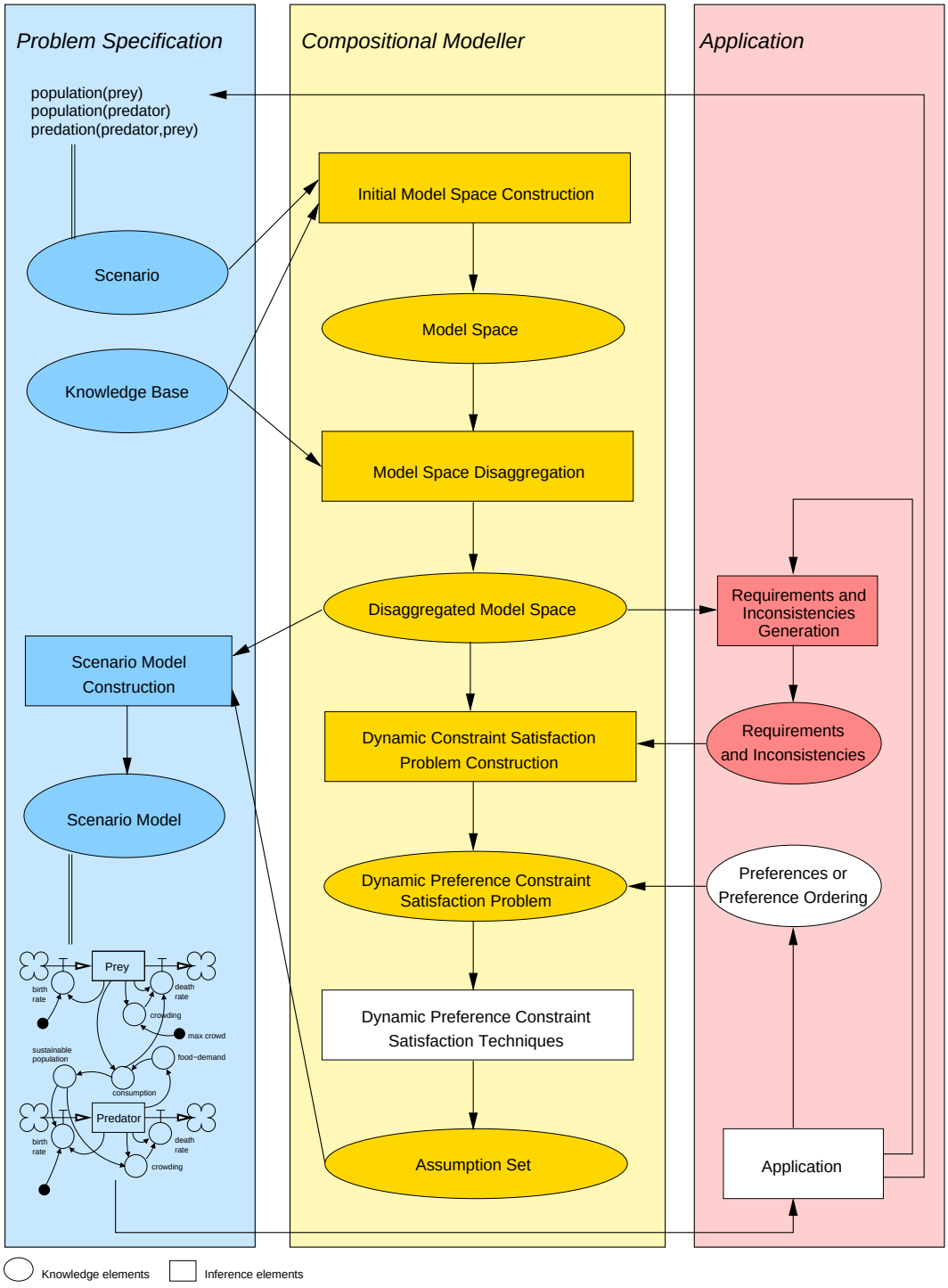


Figure 4.1: Role and methodology of CSP construction

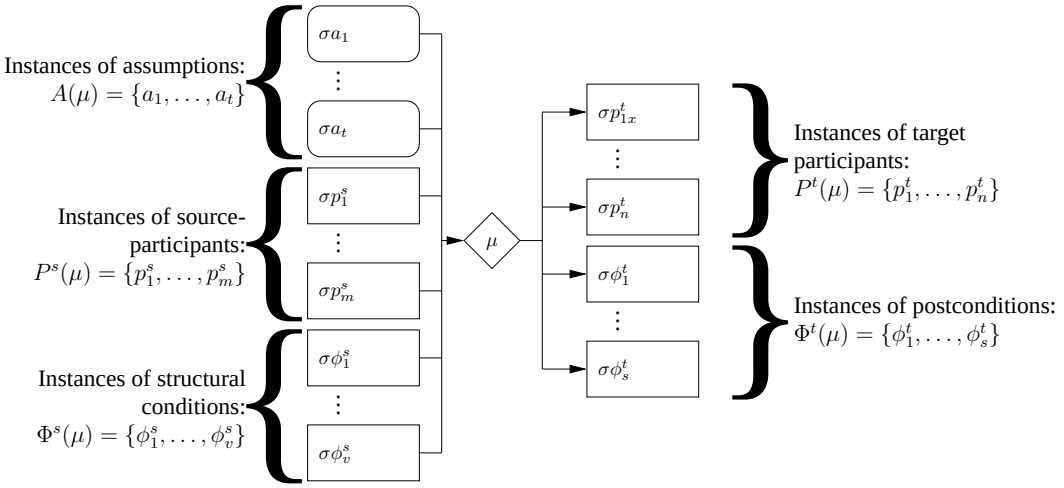


Figure 4.2: The model space

and relations that can be deduced from the given scenario, under each consistent set of assumptions.

The information contained in the model space is stored in an assumption literal based truth maintenance system (ALTMS), as discussed in section 4.1.1. An ALTMS is an extension of the existing *assumption-based truth maintenance system* (ATMS). The next two subsections then describe how the model space is constructed from a given knowledge base and scenario. In particular, section 4.1.2 shows the algorithm for model space construction and section 4.1.3 presents an extension to this algorithm to cover the construction of model spaces that involve disaggregate models.

4.1.1 The model space

As explained in the last chapter, a model fragment represents a rule indicating how new participants and relations can be deduced from existing ones and a conjunction of certain assumptions. Reasoning with these rules involves instantiating them to participant instances. Let σ represent the substitution of variables representing participants in model fragments to participant instances. Figure 4.2 then shows a graphical representation of an instantiation of a model fragment μ, in which σ represents the substitution of participant classes by participant instances. The combination of participants, relations and assumptions $\sigma p_1^s, \ldots, \sigma p_m^s, \sigma \phi_1^s, \ldots, \sigma \phi_v^s, \sigma a_1, \ldots, \sigma a_t$ justifies an instantiation of μ, which in turn justifies the corresponding generation of instances

$\sigma p_1^t, \ldots, \sigma p_n^t, \sigma \phi_1^t, \ldots, \sigma \phi_s^t$.

In the model space, such justifications are linked together in a *hypergraph* . The root nodes in this hypergraph are the participants and relations explicitly given by the scenario and the assumptions instantiated by the model fragments. Within a model space, it is possible to determine which sets of assumptions are inconsistent, which participants and relations may be derived from a set of assumptions, and whether some model property can be satisfied under a set of assumptions. Because it is required within this work to reason hypothetically with assumptions that may be established to be inconsistent when given others, it is desirable to maintain the model space by a truth maintenance system.

A truth maintenance system (TMS) stores the dependencies between inferred information and given assumptions, and aids a problem solver to avoid inconsistent inferences. Of particular interest here is the idea of assumption based TMS or ATMS [34], which achieves the goal of a TMS by computing for each piece of information the consistent sets of assumptions that justify it. To that end, it takes a set of data $\mathbb{D} = \{d_1, \ldots, d_m\}$ that the problem solver is interested, and a set of Horn clauses J

$$d_i \wedge \ldots \wedge d_j \rightarrow d_k$$

where $\{d_i, \ldots, d_j, d_k\} \subset \mathbb{D}$, and $d_k \notin \{d_i, \ldots, d_j\}$. An ATMS translates this information into a hypergraph. That is, for each problem solver datum $d \in \mathbb{D}$, it creates a node $n(d)$, and for each Horn clause $d_i \wedge \ldots d_j \rightarrow d_k$, it creates a hyperarc from $n(d_i), \ldots, n(d_j)$ to $n(d_k)$ (denoted $\{n(d_i), \ldots, n(d_j)\} \rightarrow n(d_k)$.

As a TMS, the ATMS also takes conjunctions of datums $d_i \wedge \ldots \wedge d_j$ that are inconsistent, i.e.

$$d_i \wedge \ldots \wedge d_j \rightarrow \perp$$

where $\{d_i, \ldots, d_j\} \subset \mathbb{D}$. To handle such inconsistencies, an ATMS contains a special purpose node, called the nogood node $n_\perp$ and it represents these inconsistencies by hyperarcs $\{n(d_i), \ldots, n(d_j)\} \rightarrow n_\perp$.

Some of the problem solver datums in $\mathbb{D}$ are assumptions. They are datums from which the problem solver deduces new datums, but which are not known to be true. In the compositional modelling application, for example, instances of assumptions in model fragments correspond to ATMS assumptions. By means of the hypergraph, the ATMS computes a minimal set of conjunctions for each node, including $n_\perp$, that

support it. Such sets are called *labels*, and the conjunctions within them are called *environments*. The environments in the label of $n_\perp$ are called nogoods, because they are conjunctions of assumptions that lead to inconsistencies. All supersets of nogoods in the labels of other nodes are removed since it is known that they can not provide a consistent justification.

Unfortunately, the use of *Horn clauses* restricts the usefulness of an ATMS for the present work as it may be required to compute negations of conjunctions, disjunctions and assumptions explicitly. Consider, for example, the following model properties defined in section 3.3.3:

```
(defproperty endogenous
  :source-participants ((?m :type math-object))
  :structural-condition ((or (== ?m *) (d/d ?m *)))
  :property (endogenous ?m))


(defproperty exogenous
  :source-participants ((?m :type math-object))
  :structural-condition ((not (endogenous ?m)))
  :property (exogenous ?m))
```

On the one hand, the endogenous property can be derived for instances of relations that match `(== ?m *)` or `(d/d ?m *)`. Within a standard ATMS, instances of the Horn clause `(== ?m *)` $\rightarrow$ `(endogenous ?m)` can be utilised to derive that `(endogenous ?m)` is deducible from the same assumption sets as `(== ?m *)`. On the other hand, the property `(exogenous ?m)` can not be computed with a conventional ATMS because the algorithm can only propagate disjunctions and conjunctions of data, not negations thereof. Having taken notice of this, an assumption literal based truth maintenance system (ALTMS) is suggested herein to handle negations.

4.1.1.1 The assumption literal based truth maintenance system

Similar to an ATMS, an ALTMS is a hypergraph of nodes denoting problem solver datums. However, instead of only using Horn clauses, the problem solver datums may be justified by propositional calculus expressions. That is, a problem solver datum $d \in \mathbb{D}$ may be deduced via rules of inference of the form $j \rightarrow d$, where the *justification* j is either:

- a problem solver datum: $j = d \in \mathbb{D}$,
- a conjunction of other justifications: $j = j_1 \wedge \ldots \wedge j_m$,
- a disjunction of other justifications $j = j_1 \vee \ldots \vee j_m$, or
- the negation of another justification: $j = \neg j'$.

As such, the benefit of an ALTMS over an ATMS is that it can handle negations of nodes in the antecedent of a justification.

The purpose of an ALTMS is to relate problem solver data to the assumptions that justify them. To that end, for each node n, a label $\mathcal{L}(n)$ is computed.

Definition 4.1. A *label* $\mathcal{L}(n)$ is a set of environments. Each *environment* E is a set of assumptions and negations of assumptions $\{a_{1\top}, \ldots, a_{p\top}, \neg a_{1\bot}, \ldots, \neg a_{q\bot}\}$ such that

$$(a_{1\top} \wedge \ldots \wedge a_{p\top}) \wedge \neg a_{1\bot} \wedge \ldots \wedge \neg a_{q\bot} \rightarrow n$$

A node's label can be computed from the labels of the nodes that justify it. Given complete and sound labels for the nodes of two problem solver datums d_1 and d_2:

$$\mathcal{L}(n(d_1)) = \{E_{11}, \ldots, E_{1p_1}\}$$
$$\mathcal{L}(n(d_2)) = \{E_{21}, \ldots, E_{2p_2}\}$$

a label can be computed for the conjunction, disjunction and negation of these problem solver datums, as follows:

$$\mathcal{L}(n(d_1 \wedge d_2)) = \{E_{1i} \cup E_{2j} \mid E_{1i} \in \mathcal{L}(n_1), E_{2j} \in \mathcal{L}(n_2)\}, \tag{4.1}$$

$$\mathcal{L}(n(d_1 \vee d_2)) = \mathcal{L}(n_1) \cup \mathcal{L}(n_2), \text{ and} \tag{4.2}$$

$$\mathcal{L}(n(\neg d_1)) = \{\{\text{negate}(a_{1i}), \ldots, \text{negate}(a_{p_{1j}})\} \mid a_{1i} \in E_{11}, \ldots, a_{p_{1j}} \in E_{1p_1}\} \tag{4.3}$$

$$\text{where negate}(a_{k\top}) = \neg a_{k\top} \text{ and negate}(\neg a_{k\bot}) = a_{k\bot}$$

Based on these principles, labels can be computed that are *complete* $(\forall E_i, (E_i, \theta \vdash n, \exists E_j \in \mathcal{L}(n), (E_j \subseteq E_i)))$ and *sound* $(\forall E_i \in \mathcal{L}(n), E_i, \theta \vdash n)$, as formalised in the following theorems:

Theorem 4.2. If the labels $\mathcal{L}(n(d_1))$ and $\mathcal{L}(n(d_2))$ are complete and sound, the labels $\mathcal{L}(n(d_1 \wedge d_2))$, $\mathcal{L}(n(d_1 \vee d_2))$ and $\mathcal{L}(n(\neg d_1))$, as respectively computed in (4.1), (4.2)

and (4.3) are complete.

Proof: See appendix A, page 279.

Theorem 4.3. If the labels $\mathcal{L}(n(d_1))$ and $\mathcal{L}(n(d_2))$ are complete and sound, the labels $\mathcal{L}(n(d_1 \wedge d_2))$, $\mathcal{L}(n(d_1 \vee d_2))$ and $\mathcal{L}(n(\neg d_1))$, as respectively computed in (4.1), (4.2) and (4.3) are sound.

Proof: See appendix A, page 280.

Some conjunctions of nodes and/or negations of nodes may be inconsistent. As with the nogood conjunctions of nodes in a standard ATMS, such conjunctions justify a special node, called the nogood node and denoted $n_\perp$. The environments $E \in \mathcal{L}(n_\perp)$, i.e. the environments in the label of the nogood node are called nogood environments. To make a label $\mathcal{L}(n)$ *consistent* ($\forall E \in \mathcal{L}(n), E, J \nvdash n_\perp$, where J is the set of all justifications that have been provided for the nodes in the ALTMS) all inconsistent environments must be removed. An environment E is inconsistent ($E \rightarrow n_\perp$), and hence removed from a label if

$$\exists E_\perp \in \mathcal{L}(n_\perp), (E_\perp \subset E), \text{ or} \tag{4.4}$$

$$(a \in E) \wedge (\neg a \in E) \tag{4.5}$$

Finally, the labels must be made *minimal* ($\forall E_i, E_j \in \mathcal{L}(n), (E_i = E_j) \vee \neg(E_i \subset E_j)$). This is achieved by removing environments from the label that are supersets of other environments in the label.

4.1.1.2 Example

These concepts can be illustrated by means of the endogenous and exogenous properties. Suppose that a knowledge base is given which contains the following implications:

- a variable x is created if an assumption $\text{relevant}(\text{growth}, p)$ is true:

$$\text{relevant}(\text{growth}, p) \rightarrow \text{exists}(x), \text{exists}(N)$$

- $x = r \times N$ if x exists and an assumption $\text{model}(x, \text{exponential})$ is true, and

$$\text{exists}(x) \wedge \text{exists}(N) \wedge \text{model}(x, \text{exponential}) \rightarrow \text{exists}(r), = (x, \times(r, N))$$

- $x = r \times N \times \frac{N}{C}$ if $\text{model}(x, \text{logistic})$

$$\text{exists}(x) \wedge \text{exists}(N) \wedge \text{model}(x, \text{logistic}) \rightarrow \text{exists}(r), \text{exists}(C) = (x, \times(r, N, \div(N, C)))$$

Under these conditions, the labels of the properties exists(x) and endogenous(x) can be computed as follows (noting that $\text{endogenous}(x) \leftarrow \text{exists}(x) \wedge = (x, *)$):

$$\mathcal{L}(\text{exists}(x)) = \{\{\text{relevant}(\text{growth}, p)\}\}$$
$$\mathcal{L}(\text{endogenous}(x)) = \{\{\text{relevant}(\text{growth}, p), \text{model}(x, \text{exponential})\}, \{\text{relevant}(\text{growth}, p), \text{model}(x, \text{logistic})\}\}$$

As a variable is exogenous if $\text{exists}(x) \wedge \neg\text{endogenous}(x)$, the label of exogenous(x) is computed as follows:

$$\mathcal{L}(\neg\text{endogenous}(x)) = \{\{\neg\text{relevant}(\text{growth}, p)\}, \{\neg\text{model}(x, \text{exponential}), \neg\text{model}(x, \text{logistic})\}\}$$
$$\mathcal{L}(\text{exogenous}(x)) = \{\{\text{relevant}(\text{growth}, p), \neg\text{relevant}(\text{growth}, p)\}, \{\text{relevant}(\text{growth}, p), \neg\text{model}(x, \text{exponential}), \neg\text{model}(x, \text{logistic})\}\}$$

Naturally, $\{\text{relevant}(\text{growth}, p), \neg\text{relevant}(\text{growth}, p)\}$ is an inconsistent environment, and therefore, the complete, sound, consistent and minimal label of exogenous(x) becomes

$$\mathcal{L}(\text{exogenous}(x)) = \{\{\text{relevant}(\text{growth}, p), \neg\text{model}(x, \text{exponential}), \neg\text{model}(x, \text{logistic})\}\}$$

4.1.1.3 ALTMS and GDE candidate generation

Readers from the model-based reasoning community will undoubtedly note that there is a very close relationship between an ALTMS and candidate generation in the General Diagnostic Engine (GDE) [37] which is itself rooted in an ATMS. Indeed, the ALTMS employs the same types of set operations as the GDE and hence, the ALTMS incorporates the mechanism for candidate generation that was added to GDE.

Generally speaking, in GDE a model of a system is represented by a set of constraints and a constraint propagator is employed to compute values for unknown variables. The constraints are imposed by potentially faulty components and when applying a constraint, it is explicitly assumed that it is always valid. As such the following

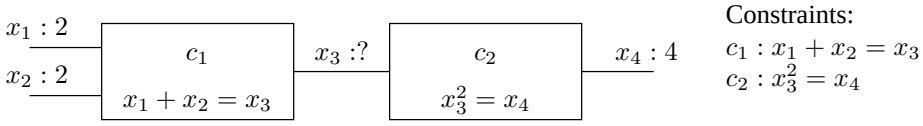


Figure 4.3: Sample GDE model

inferences are made, where $c_1, \ldots, c_n$ refer to constraints imposed by components and $x : i$ represents the assignment of value i to variable x in the constraint system:

$$\text{valid}(c_1) \wedge \ldots \wedge \text{valid}(c_n) \rightarrow x : i$$

Theorem 4.4. Given that the constraint propagator of GDE has generated a set of nodes in an ALTMS that corresponding to hypothesised values of connections $x : i$ and these nodes have been justified by the sets of components that must function correctly for the hypothesis to be true, then:

1. the ALTMS can produce the minimal set of fault candidates in the same way as the GDE by computing the label of a node hypotheses, which has been justified such that:
$$\neg(n_\perp) \rightarrow \text{hypotheses}$$
2. the ALTMS can produce the minimal set of fault candidates that remain if a measurement of component x yields i, in the same way as the GDE, by computing the label of a node hypotheses($x : i$), which has been justified such that:
$$\neg \wedge_{n, n=n(x:j), j \neq i} n \rightarrow \text{hypotheses}(x : i)$$

Proof: See appendix A, page 281.

Take for example the model shown in figure 4.3. This model has two constraints that correspond to components $c_1 : x_1 + x_2 = x_3$ and $c_2 : x_3^2 = x_4$ and invoking one unknown variable x_3. Two possible values can be computed for x_3:

$$\text{valid}(c_1) \rightarrow x_3 : 4$$
$$\text{valid}(c_2) \rightarrow x_3 : 2$$

However, each variable can only have one value at a time and multiple plausible values under different assumption sets indicate a malfunction in the system. Therefore,

such inconsistencies are reported to the ATMS as follows:

$$x_3 : 4 \wedge x_3 : 2 \rightarrow \bot$$

This yields the following label for the nogood node $\{\{c_1, c_2\}\}$.

The GDE has a purpose built procedure to translate a set of nogood nodes into a set of hypothesis that is equivalent to computing the label of a node hypotheses in an ALTMS which is justified such that:

$$\neg(n_\bot) \rightarrow \text{hypotheses}$$

In the ongoing example, $\mathcal{L}(\text{hypotheses}) = \{\{\neg c_1\}, \{\neg c_2\}\}$, which means that the behaviour can be explained by a fault in c_1 *or* by a fault in c_2. This is the same set of hypotheses that GDE would generate.

The measurement proposer of GDE requires that the set of remaining candidates for all measurement outcomes are computed and it then applies a heuristic to compute the next best measurement. The ALTMS can be employed to determine the sets of remaining candidates for a measurement outcome $x : i$, by computing the label of a node hypotheses$(x : i)$, which is justified such that:

$$\neg \wedge_{n, n=n(x:j), j \neq i} n \rightarrow \text{hypotheses}(x : i)$$

In the ongoing example, $\mathcal{L}(\text{hypotheses}(x_3 : 4)) = \{\{\neg c_2\}\}$.

4.1.2 Inference from a scenario

By means of the model fragments in the knowledge base, new information can be derived from the scenario. To that end, let a database δ be a tuple $\langle O, R, A \rangle$, where O is a set of object constants, R is a set of ground relations that are known to be true in δ, and A is set of ground relations expressing assumptions. Before any inference is done by the compositional modeller, a scenario $\langle O, R \rangle$ is provided by the user, which corresponds to a database $\langle O, R, \{\} \rangle$.

The set of relations R in a database $\delta = \langle O, R, A \rangle$ may be pattern matched with the structural conditions of some model fragment μ by a substitution $\sigma = \{p_1^s / o_i, \ldots, p_m^s / o_j\}$, which maps each source-participant in $P^s(\mu)$ onto an object constant of O. Also, σ instantiates the assumptions $A(\mu) = \{a_1, \ldots, a_t\}$ to a set of ground relations. If

Algorithm 4.1: GENERATEMODELSPACE($\langle O, R \rangle$)

$\theta \leftarrow$ new hypergraph;
for each $o \in O$, add-node(θ, o);
for each $r \in R$, add-node(θ, o);
for each $\mu \in$ model_space, match(μ, θ, σ)
do {
  justification $\leftarrow \emptyset$;
  for each $a \in A(\mu)$
    do {
      newnode $\leftarrow$ add-node$(\theta, (\sigma a))$;
      justification $\leftarrow$ justification $\cup \{$newnode$\}$;
    }
  for each $p \in P^s(\mu)$
    do justification $\leftarrow$ justification $\cup \{$find-node$(\theta, (\sigma p))\}$;
  for each $\phi \in \Phi^s(\mu)$
    do justification $\leftarrow$ justification $\cup \{$find-node$(\theta, (\sigma \phi))\}$;
  add-node$(\theta, n_{(\sigma,\mu)})$;
  add-justification$(\theta, n_{(\sigma,\mu)}, \wedge_{n \in \text{justification}} n)$;
  for each $p \in P^t(\mu)$
    do {
      $\sigma \leftarrow \sigma \cup \{p/\text{gensym}()\}$;
      $o \leftarrow$ add-node$(\theta, (\sigma p))$;
      add-justification$(\theta, o, n_{(\sigma,\mu)})$;
    }
  for each $\phi \in \Phi^t(\mu)$
    do {
      $o \leftarrow$ add-node$(\theta, (\sigma \phi))$;
      add-justification$(\theta, o, n_{(\sigma,\mu)})$;
    }
}
for each $n_1, \ldots, n_m$, inconsistent$(\{n_1, \ldots, n_m\})$
  do add-justification$(\theta, n_\perp, n_1 \wedge \ldots \wedge n_m)$;

$\sigma a_1 \wedge \ldots \wedge \sigma a_t$ is consistent with δ, a new database $\delta' = \langle O', R', A' \rangle$ logically follows from δ, such that:

$$
\begin{aligned}
&\sigma' = \sigma \cup \{p_1^t/o_{z+1}, \ldots, p_n^t/o_{z+n}\} && \text{with } P^t(\mu) = \{p_1^t, \ldots, p_n^t\} \\
&O' = O \cup \{o_{z+1}, \ldots, o_{z+n}\} && \\
&R' = R \cup \{\sigma'\phi_1^t, \ldots, \sigma'\phi_u^t\} && \text{with } \Phi^t(\mu) = \{\phi_1^t, \ldots, \phi_u^t\} \\
&A' = A \cup \{\sigma' a_1, \ldots, \sigma' a_t\} && \text{with } A(\mu) = \{a_1, \ldots, a_t\}
\end{aligned}
$$

In other words, in the new database $\delta' = \langle O', R', A' \rangle$, O' contains every $o \in O$ and a new object constant for each target-participant $p \in P^t(\mu)$, thereby forming a new substitution σ'. R' is the union of R and the instantiated postconditions of μ, and A' is the union of A and the set of the instantiated assumptions μ.

In this way, a database $\delta = \langle O, \Phi, A \rangle$ can be constructed such that $\langle O, R \rangle$ corresponds to a scenario model for the given scenario. However, such a scenario model should also be consistent. Inconsistencies may arise from inconsistent assumptions or relations and from failure to meet the requirements imposed by the user. Also, multiple alternative models may be consistent and all of them potentially meet the requirements. Therefore, a model space is generated which stores all possible models that are consistent. The pseudo-code in the GENERATEMODELSPACE() presents an algorithm to compute the model space.

4.1.2.1 Instantiating the knowledge base

Most of the function in GENERATEMODELSPACE() are self-explanatory, with perhaps the exception of match(μ, θ, σ) and gensym(). The former returns a boolean value of true if the source-participants and structural conditions of the model fragment μ are matched by participant and relation instances in θ via a substitution σ, and it returns false otherwise. More formally,

$$\begin{aligned} \text{match}(\mu, \theta, \sigma) = \text{true} \leftarrow & \sigma = \{p_1^s/o_1, \ldots, p_m^s/o_m\} \wedge P^s(\mu) = \{p_1^s, \ldots, p_m^s\} \wedge \\ & o_1 \in \theta \wedge \ldots \wedge o_m \in \theta \wedge \forall \phi \in \Phi^s(\mu), \sigma\phi \in \theta \\ \text{match}(\mu, \theta, \sigma) = \text{false} \leftarrow & \quad \text{otherwise} \end{aligned}$$

The function gensym() returns a new and unique symbol for each new participant.

In GENERATEMODELSPACE(), the model space is first initialised with the contents of the scenario. That is, given a scenario $\langle O, R \rangle$, a node is added to the model space θ for each participant instance $o \in O$ and for each relation instance $r \in R$. Then, for each model fragment whose source-participants and structural conditions match the objects and relations already in θ, a set of new nodes and hyperarcs are created to describe the instantiation of the implications defined by the model fragment (see (3.33) in definition 3.4). That is, given a model fragment μ and a substitution σ such that match(μ, θ, σ) is true then, the knowledge that can be potentially inferred is registered in the ALTMS as follows:

- Each assumption in the set $A(\mu)$ is instantiated by means of the substitution σ and added as an assumption node in the ALTMS, provided it does not exist already.

- A node $n_{(\sigma,\mu)}$ is created. This node denotes the application of a model fragment based on the match described by substitution σ. A new justification is added to the ALTMS, such that the node $n_{(\sigma,\mu)}$ is proved by:

$$(\wedge_{p\in P^s(\mu)}\sigma p)\wedge(\wedge_{\phi\in\Phi^s(\mu)}\sigma\phi)\wedge(\wedge_{a\in A(\mu)}\sigma a)$$

- For each of the target-participants $p_i^t \in P^t(\mu)$, a new object constant o_i and a corresponding node in the ALTMS are created. Each of these new nodes is justified by $n_{(\sigma,\mu)}$. Then, a new substitution σ', that includes σ and maps the target-participants to the newly created object constants, is created. That is,

$$\sigma' = \sigma \cup \{p_1^t/o_1,\ldots,p_n^t/o_n\}$$

  Finally, for each of the postconditions $\phi_i \in \Phi^t(\mu)$, a new node that corresponds to $\sigma'\phi_i$ is added to θ and justified by $n_{(\sigma,\mu)}$.

4.1.2.2 Inferring inconsistencies

At the end of GENERATEMODELSPACE(), the justifications for the nogood node $\perp$ are added. This is done for all sets of nodes $\{n_1,\ldots,n_m\}$ for which inconsistent$(n_1,\ldots,n_m)$ holds. In this work, two sources of inconsistencies are considered: (i) relations that can not be composed according to the modelling paradigm and (ii) model properties that are not satisfied.

4.1.2.2.1 Non-composable relations As mentioned in section 3.3.1, there are restrictions on what assignment relations are composable. In a system dynamics model, two equations $v = f_1(\ldots)$ (or $\frac{d}{dt}v = f_1(\ldots)$) and $v = f_2(\ldots)$ (or $\frac{d}{dt}v = f_1(\ldots)$) can only be part of the same model if $f_1(\ldots) \equiv f_2(\ldots)$ or if f_1 and f_2 are composable relations. Otherwise, the joint presence of $v = f_1(\ldots)$ and $v = f_2(\ldots)$ in the model make this model inconsistent as it can not be used for simulation. Therefore, for every two nodes n_1 and n_2 that contain such inconsistent assignment relations, inconsistent(n_1,n_2) holds.

If a modelling paradigm, other than system dynamics, is employed in the domain theory, nodes will be deemed inconsistent for different reasons. In that case, different rules must be implemented to define the sets of nodes $\{n_1,\ldots,n_m)$ for which inconsistent$(n_1,\ldots,n_m)$ holds. However, these different rule sets do not affect the

technique employed to register inconsistencies. Also, with certain extensions to the existing knowledge representation, model properties (see section 3.3.3) could be employed to define inconsistent$(n_1,\ldots,n_m)$. But as mentioned in section 9.2.5, this is left as a future extension on this work.

4.1.2.2.2 Required properties The model properties are instantiated in the model space. In essence, a node is created for each model property that can be instantiated, and the instantiation source-participants and structural conditions, taken from the property definition, is given as its justification. The example discussed in 4.1.1.1 has already illustrated how this is implemented for the endogenous and exogenous properties.

Properties instances that are required impose additional constraints upon the model construction process. That is, models that do not satisfy the required properties must be deemed nogood. There are two different ways of constraining model construction by means of properties. A property instance p is either a global property required in all possible models or it is a purpose-required property of a model fragment (see section 3.3.3). If p represents the node corresponding to a global property instance, the following nogood justification is added to the ALTMS:

$$\neg p \rightarrow n_{\perp}$$

If p corresponds to the purpose-required property of a model fragment instance $\sigma\mu$, then every set of assumptions under which $\sigma\mu$ is justified, but p is not, should not be allowed. Hence, for each purpose-required property p of each model fragment instance $\sigma\mu$ the following nogood justification is added to the ALTMS:

$$\sigma\mu \wedge \neg p \rightarrow n_{\perp}$$

The model space for the running example contains parts of models representing natural reproduction in the absence of other species for the `predator` and `prey` species, parts of models describing the predation of the `prey`, and parts of models on predation by the `predator`. Figure 4.4 shows the model space that is generated from this sample scenario. From this model space, two types of model of natural reproduction of the `predator` species can be obtained. This figure also illustrates part of the possible predation models involving the predator and one of the prey species.

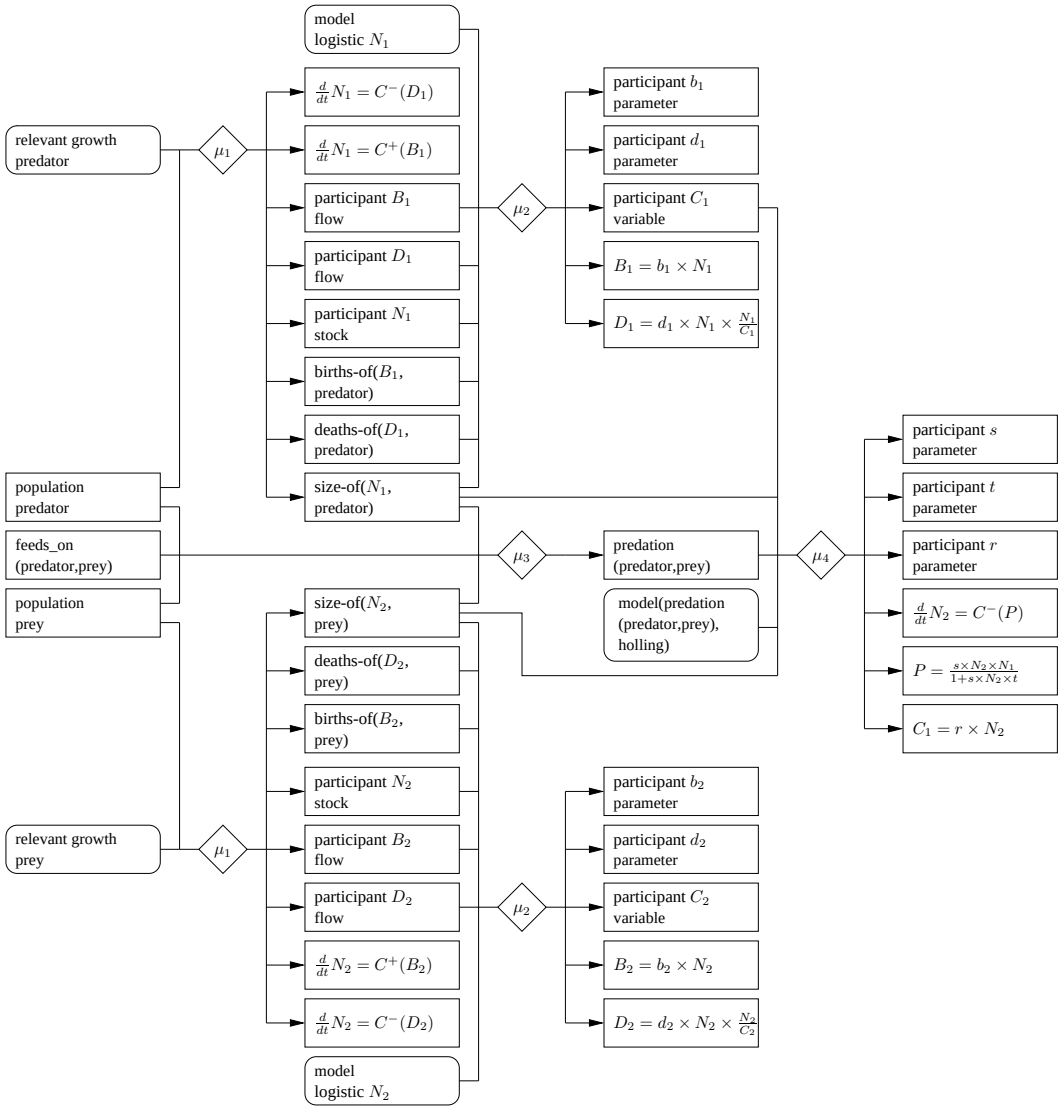


Figure 4.4: Partial model space for the predator-prey scenario

4.1.3 Disaggregating the model space

The purpose of disaggregating an existing model into a more detailed one is to provide a concise way of expressing different phenomena, i.e. the participants and relations that describe them, with varying granularities. The construction of disaggregate models (see section 3.2.2) is enabled by expanding the model space to incorporate the disaggregate alternatives of the models already existing in the model space.

In general, disaggregating a model into n classes involves one or more of the following:

1. Replacing a certain participant by a set of n participants.

2. Mapping the relations of the disaggregated participants onto disaggregate relations. That is, each relation $r(p_1^d,\ldots,p_m^d,p_1,\ldots,p_n)$, where $p_1^d,\ldots,p_m^d$ are the disaggregated participants, is replaced by n relations $r(p_{i1}^d,\ldots,p_{im}^d,p_1,\ldots,p_n)$, $i=1,\ldots,n$.

3. Adding additional participants and relations. These are typically used to describe migration between the resulting disaggregate classes.

This section formally discusses how these can be achieved by extending a model space by means of a disaggregation fragment (see section 3.3.6.2).

4.1.3.1 Applying disaggregation fragments

Similar to the normal model fragments, disaggregation fragments are applicable with respect to a set of instances of source-participants and structural conditions. Also similar to normal model fragments, a disaggregation fragment adds new instances of target participants and postconditions to the model space when it is applied. However, the application of a disaggregation fragment requires copying and transforming all of the participants and relations that depend upon a set of nodes to which the disaggregation fragment is applicable.

Algorithm 4.2: APPLYDF(Δ, d, M)

$d_\theta(\Delta,d,M) \leftarrow \Delta$; $\quad d_M(\Delta,d,M) \leftarrow \{\}$; $\quad Q \leftarrow M$; $\quad S \leftarrow \{\}$;
repeat
 $m \leftarrow$ dequeue(M); $\quad m' \leftarrow$ add-node$(d_\theta(\Delta,d,M), \text{content}(m))$;
 $d_M(\Delta,d,M) \leftarrow d_M(\Delta,d,M) \cup \{m'\}$;
 if $m \in M$
  then { add-justification$(d_\theta(\Delta,d,M), m', \mathcal{L}(d_\theta(\Delta,d,M),m) \wedge d(A))$;
  justification$(m) \leftarrow \mathcal{L}(d_\theta(\Delta,d,M),m) \wedge \neg d(A)$;
  else justify$(m', \text{substitute}(A(m), S))$;
 for each $p \in \text{consequents}(m), \text{participant}(p)$
  do {
   if $p \in \text{dom}(\delta_P)$
    then $S \leftarrow S \cup (p, \delta_P(p))$; $\quad n \leftarrow$ add-node$(d_\theta(\Delta,d,M), \delta_P(p))$;
    else $n \leftarrow$ add-node$(d_\theta(\Delta,d,M), p)$;
   add-justification$(d_\theta(\Delta,d,M), n, m')$;
   for each $m'', (\text{successor}(m'', p))$
    do enqueue(m'', Q);
 for each $\phi(\overrightarrow{p}) \in \text{consequents}(m), \text{relation}(\phi(\overrightarrow{p}))$
  do {
   if $\phi(\overrightarrow{p}) \in \text{dom}(\delta_R)$
    then $n \leftarrow$ add-node$(d_\theta(\Delta,d,M), \delta_R(\phi(\overrightarrow{p})))$;
    else $n \leftarrow$ add-node$(d_\theta(\Delta,d,M), \text{disaggregate}(\phi(\overrightarrow{p}), S))$;
   justify(n, m');
   for each $m'', (\text{successor}(m'', p))$
    do enqueue(m'', Q);
 if $m \in M$
  then {
   for each $p \in P^t(d)$
    do { $n \leftarrow$ add-node$(d_\theta(\Delta,d,M), p)$;
    add-justification$(d_\theta(\Delta,d,M), n, m'))$;
   for each $\phi(\overrightarrow{p}) \in \Phi^t(d)$
    do { $n \leftarrow$ add-node$(d_\theta(\Delta,d,M), \phi(\overrightarrow{p}))$;
    add-justification$(d_\theta(\Delta,d,M), n, m'))$;
until $M = \{\}$;
return $(d_\theta(\Delta,d,M), d_M(\Delta,d,M))$;

Definition 4.5. Given a disaggregation fragment d with $P^s(d) = \{p_1^s, \ldots, p_m^s\}$ and $\Phi^s = \{\phi_1^s, \ldots, \phi_v^s\}$, a set of participants instances $O = \{o_1, \ldots, o_m\}$, a set of instantiated relations $R = \{r_1, \ldots, r_v\}$, a substitution $\sigma = \{p_1^s/o_1, \ldots, p_m^s/o_m\}$ and a model space Δ

such that

$$(\forall o \in O, o \in \Delta) \wedge (\forall r \in R, r \in \Delta) \wedge (\forall i = 1, \ldots, v, \sigma\phi_i^s = r_i)$$

set of nodes to which the disaggregation fragment is *applicable* with respect to σ is the set of nodes M such that

$$\Delta, M \vdash o_1, \ldots, o_m, r_1, \ldots, r_v$$
$$\neg\exists M' \subset M\Delta, M' \vdash o_1, \ldots, o_m, r_1, \ldots, r_v$$

In other words, the set of nodes to which a disaggregation fragment is applicable is the smallest set of nodes which justify the nodes that match the source-participants and structural conditions of the disaggregation fragment. The algorithm APPLYDF(Δ, d, M) describes the procedure of extending and transforming the model space Δ for an application of a disaggregation fragment d, when it is deemed applicable to a set of nodes M with respect to a substitution σ (see definition 4.5).

The algorithm APPLYDF(Δ, d, M) returns a new, extended, model space $d_\theta(\Delta, d, M)$ which replaces the original Δ, and the set of new nodes $d_M(\Delta, d, M)$, which replaces the nodes from which the newly added parts of the model space are derived. The set of nodes $d_M(\Delta, d, M)$ corresponds to the root nodes subgraph that has been added to the original model space (this is relevant to the discussion on combining disaggregations in section 4.1.3.2).

As discussed in section 4.1.3, the application of a disaggregation fragment also involves the creation of new target-participant instances and corresponding instances of postconditions. INSTANTIATEDF(Δ, d, M, σ) is the procedure for adding these participants and relations and it returns $P^t(d, M)$, the set of newly created participant instances, and $\Phi^t(d, M)$, the set of newly created postcondition instances.

The application of a disaggregation fragment then consists of running algorithms APPLYDF() and INSTANTIATEDF(). The resulting consists of the union of the original model space, the newly disaggregated model space and the newly added participants and relations. Definition 4.6 presents a formal definition of this.

Definition 4.6. Given a model space Δ, a disaggregation fragment d and a set of nodes M (representing model fragment instances) to which d can be applied, the model space $D_\theta(\Delta, d, M)$ that is constructed by applying d to M is

$$D_\theta(\Delta, d, M) = d_\theta(\Delta, d, M) \cup P^t(\Delta, d, M) \cup \Phi^t(\Delta, d, M)$$

Algorithm 4.3: INSTANTIATEDF(Δ, d, M, σ)

$P^t(\Delta,d,M) \leftarrow \{\}; \Phi^t(\Delta,d,M) \leftarrow \{\};$
for each $p \in P^t(d)$
do $\begin{cases} \sigma \leftarrow \sigma \cup \{p/\text{gensym}()\}; \\ n \leftarrow \text{add-node}(d_\theta(\Delta,d,M),(\sigma p)); \\ \text{add-justification}(d_\theta(\Delta,d,M),n,d_M(d,M)); \\ P^t(\Delta,d,M) \leftarrow P^t(\Delta,d,M) \cup \{n\}; \end{cases}$
for each $\phi \in \Phi^t(d)$
do $\begin{cases} n \leftarrow \text{add-node}(d_\theta(\Delta,d,M),(\sigma \phi)); \\ \text{add-justification}(d_\theta(\Delta,d,M),n,d_M(d,M)); \\ \Phi^t(\Delta,d,M) \leftarrow \Phi^t(\Delta,d,M) \cup \{n\}; \end{cases}$
return $(P^t(d,M),\Phi^t(d,M));$

In less abstract terms, the example given in figure 4.5 presents the result of applying APPLYDF() to the disaggregation fragment shown above to model space Δ (indicated as the original model space in figure 4.5), with respect to an instance of a model fragment for population growth. APPLYDF() essentially copies the subtree of consequents of $M = \{\mu_1\}$, disaggregates all participants and relations according to the bijections δ_P and δ_R of the disaggregation mapping respectively, and INSTANTIATEDF() adds the new participants $P^t(\Delta,d,M)$ and relations $\Phi^t(\Delta,d,M)$ for migration. The entire model space shown in figure 4.5 is $D_\theta(\Delta,d,M)$

More specifically, the left half of figure 4.5 shows the model space generated using the model fragments for the population growth phenomenon (μ_1) and the one for the logistic growth model (μ_2) with respect to a single population. In particular, the model specified in equations (3.17) and (3.18) follows under the set of assumptions of "relevance of growth of p", "logistic growth model for N" and "no disaggregation of p in age classes". The right half of the figure is the disaggregated version of the model space that is added due to the application of the aforementioned disaggregation fragment. In fact, the model specified in equations (3.19), (3.20) and (3.21) follows from the assumptions "relevance of growth of p", "logistic growth model for N" and "disaggregation of p into n age classes".

Theorem 4.7. If a model M can be derived from a model space Δ, M can also be

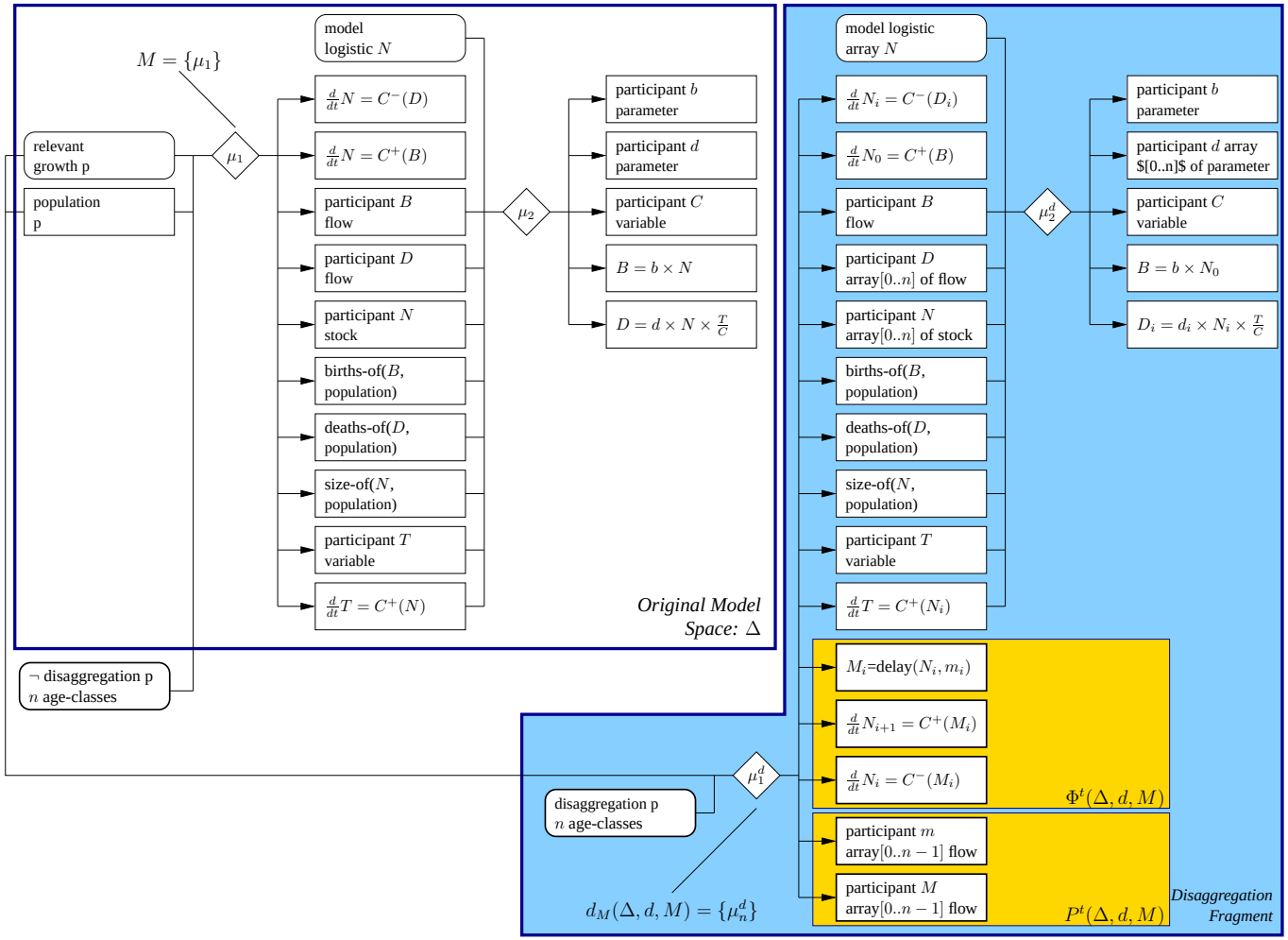


Figure 4.5: Model space expansion via disaggregation fragment

derived from $D_\theta(\Delta, d, M)$

Proof: See appendix A, page 283.

Theorem 4.8. Given that

- Δ is a model space,
- $\Delta' = d_\theta(\Delta, d, M)$, i.e. the model space resulting from extending Δ by the application of a disaggregation fragment d to a set of model fragment nodes M,
- A is a set of assumptions such that $A, \Delta \nvdash \bot$, and
- M_A is a model such that $A, \Delta \vdash M_A$,

then the model M_D obtained by $A \cup A_{df}, \Delta' \vdash M_D$ is a disaggregate model of M_A.

Proof: See appendix A, page 283.

From theorem 4.7, all models that may be deduced from a model space Δ, can be deduced from the model space that is computed by APPLYDF(Δ, d, M). Further, theorem 4.8 shows that for each set of assumptions from which a scenario model can be deduced in a non-expanded model space with regards to a set of assumptions, a disaggregate version of that model can be deduced from the expanded model space by extending the original set of assumptions with instantiated grain assumptions taken from the disaggregation fragment.

4.1.3.2 Combining disaggregations

Up to now, only individual disaggregations have been discussed. There are, however, many scenarios where it may be necessary to apply different disaggregations to the same participants. For example, in addition to disaggregating a population into age classes, a population could be disaggregated according to sex, physical location or subspecies. The effects of these disaggregations must therefore be combined. The combined application of two disaggregation fragments d_1 and d_2 to a set of model fragment instances involves (i) applying d_2 to the set of model fragments generated by applying d_1 to M and (ii) applying the disaggregation mapping of d_1 to the instances of the target-participants and postconditions introduced into the model space by applying

d_2. Definition 4.9 formalises this concept of combined disaggregation of the model space:

Definition 4.9. Given two disaggregation fragments d_1 and d_2, a model space Δ and a partition $\{M_{12}, M_1, M_2, M\}$ of those nodes in Δ which represent model fragment instances, where $M_{12} \cup M_1$ and $M_{12} \cup M_2$ are minimal sets for model fragments to which d_1 and d_2 can be respectively applied, the result of the *combined disaggregation of the model space,* denoted $D_\theta(\Delta, d_2 \circ d_2, M_1 \cup M_2 \cup M_{12})$ is:

$$\begin{aligned} D_\theta(\Delta, d_2 \circ d_1, M_1 \cup M_2 \cup M_{12}) = & D_\theta(d_\theta(\Delta, d_1, M_1 \cup M_{12}), d_2, d_M(\Delta, d_1, M_{12}) \cup M_2) \cup \\ & D_\theta(d_\theta(\Delta, d_1, M_1 \cup M_{12}), d_1, P^t(d_2, M_2 \cup M_{12})) \cup \\ & D_\theta(d_\theta(\Delta, d_1, M_1 \cup M_{12}), d_1, \Phi^t(d_2, M_2 \cup M_{12})) \end{aligned}$$

Consider, for example, disaggregation a population of a particular species into q populations of subspecies. This requires a disaggregation mapping that disaggregates all participants other than T (total population) and K (maximal sustainable population). The application of this disaggregation mapping to equations (3.17) and (3.18) results in (with $j = 0, \ldots, q$):

$$\frac{d}{dt} N_j = C^+(B_j), \frac{d}{dt} N_j = C^-(D_j) \tag{4.6}$$

$$B_j = r_j \times N_j, D_j = d_j \times N_j \times \frac{T}{K}, T = C^+(N_j) \tag{4.7}$$

Now consider the combined application (see definition 4.9) of disaggregation into n age-classes and disaggregation into q subspecies to (4.6) and (4.7). This involves:

1. applying the subspecies disaggregation fragment,
2. applying the age-class disaggregation fragment, and
3. applying the subspecies disaggregation fragment to the target-participants and postconditions introduced by the application of the age-class disaggregation fragment.

Figure 4.6 shows a graphical illustration of this procedure. More specifically, step 2

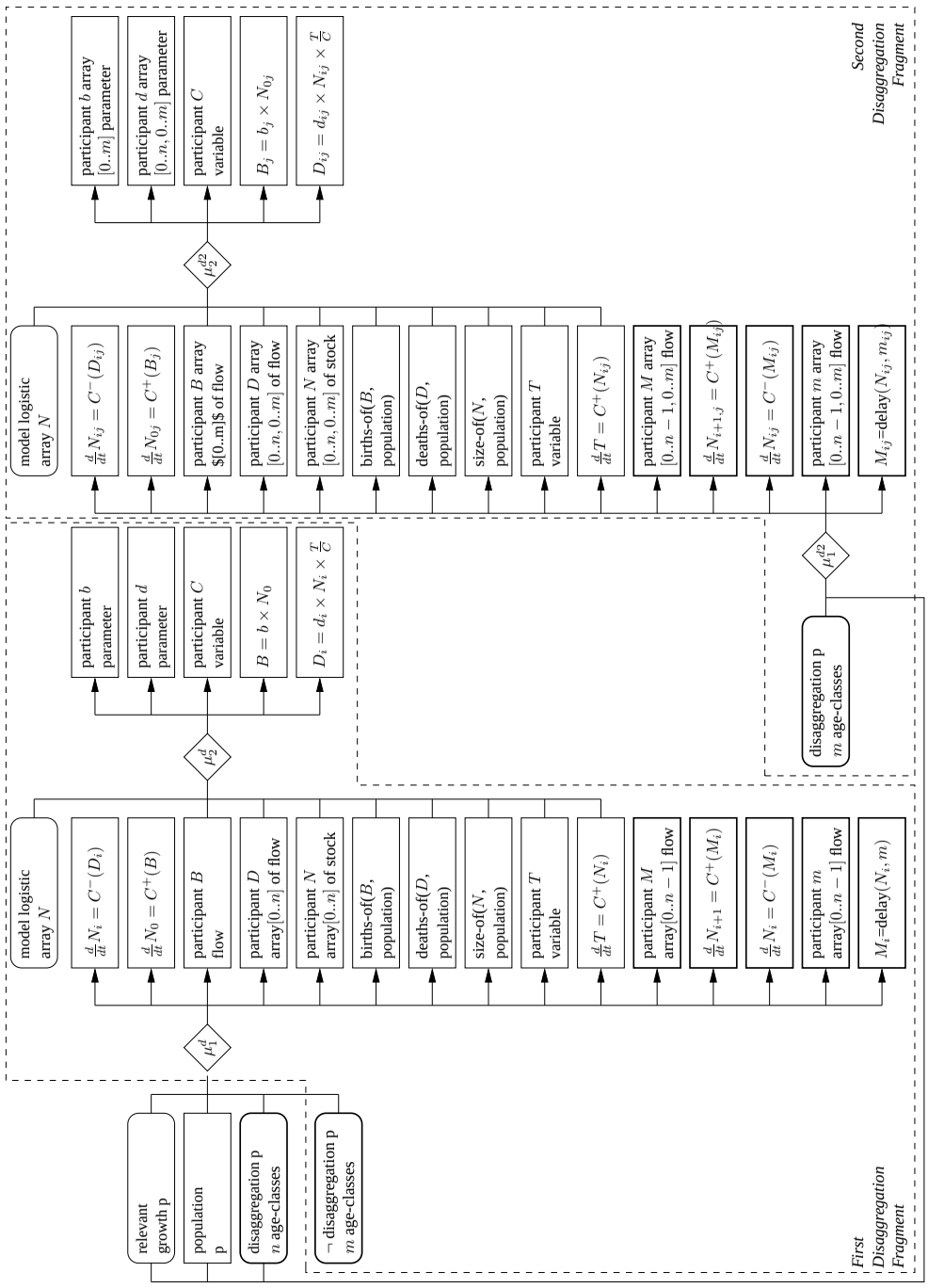


Figure 4.6: Combined application of disaggregation fragments

results in:

$$\frac{d}{dt}N_{0j} = C^+(B_{ij}), \frac{d}{dt}N_{ij} = C^-(D_{ij}) \tag{4.8}$$

$$B_{ij} = r_{ij} \times N_{ij}, D_{ij} = d_{ij} \times N_{ij} \times \frac{T}{K}, T = C^+(N_{ij}) \tag{4.9}$$

$$\frac{d}{dt}N_i - 1 = C^-(M_i), M_i = \text{delay}(N_i, t_i) \tag{4.10}$$

with $i = 0,\ldots,n$ and $j = 0,\ldots,q$ in (4.8) and (4.9) and with $i = 0,\ldots,n$ in (4.10). Step 3 applies to (4.10) and results in:

$$\frac{d}{dt}N_{i-1,j} = C^-(M_{ij}), M_{ij} = \text{delay}(N_{ij}, t_{ij}) \tag{4.11}$$

Theorem 4.10. The combined disaggregation of the model space is a commutative operation. In other words,

$$D_\theta(\Delta, d_2 \circ d_1, M_1 \cup M_2 \cup M_{12}) = D_\theta(\Delta, d_1 \circ d_2, M_1 \cup M_2 \cup M_{12})$$

Theorem 4.10 ensures that the combined application of disaggregation fragments is commutative, even if the disaggregation fragments describe different types of disaggregation. This result is due to the definition of combined application of disaggregation fragments. Definition 4.9 requires that in the case of combined application of, say, two disaggregation fragments, the participants and relations that are newly introduced by disaggregation fragment applied last (i.e. the participants and relations that were not mapped from pre-existing participants and relations), are processed by the disaggregation fragment applied first.

Consequently, the combined application of a number of disaggregation fragments yields a unique result, irrespective of the order in which the combination is implemented. Therefore, in order to combine different ways of disaggregating a model, it may be sufficient that only the individual ways of disaggregating it are represented by means of disaggregation fragments.

The latter requires that the disaggregation fragment are designed to correctly disaggregate all participants and relations that can be introduced by other disaggregation fragments. For example, the combined application of two disaggregation fragments, one describing disaggregation into age-classes and one describing disaggregation according to gender, can only yield a correctly disaggregated model if the latter can

correctly disaggregate the new participants and relations introduced by the former (e.g. flows and variables describing aging) and vice versa. More generally speaking, it is important from a disaggregation fragment design perspective that that the disaggregation fragments are defined in a sufficiently general way. This means that the disaggregation fragment must be designed to correctly disaggregate all participants and relations that can be introduced by model fragments as well as other disaggregation fragments.

The feasibility of the combined application of disaggregation fragments may significantly reduce the size and complexity of the knowledge base. If model fragments were used to specify disaggregations a different set of model fragments would be necessary for each combination of disaggregations. This is because each combination implies a different, whilst similar, set of participants and relations. As disaggregation fragments can be composed, only one is needed for each type of disaggregation instead of one per combination of disaggregations.

4.2 Constructing dynamic CSPs (DCSPs)

Having derived a model space from a given scenario and disaggregated it, individual models can be created by selecting a set of assumptions. To support this selection, information contained in this model space is translated into the description of a dynamic constraint satisfaction problem (DCSP). This translation allows the use of the solution techniques developed for DCSPs to be employed for the selection of a consistent model, to gain computational efficiency in the search for a consistent model that meets the imposed requirements.

4.2.1 The approach

The algorithm CREATEDCSP() for translating a model space into a DCSP is presented and discussed in this subsection. First, this algorithm generates the attributes and the domains of the attributes in the DCSP. As explained in section 3.1.3, an assumption class is a set of assumptions with the same subject and type and the assumptions in an assumption class correspond to mutually exclusive modelling decisions that can be made with respect to the assumption subject. Therefore, a DCSP attribute is created for each assumption class and a domain value (for that attribute) is generated for each

assumption in the assumption class.

Activity constraints define the conditions under which attributes are active. The relevance of a phenomenon at a certain specificity can only be considered with respect to a set of objects to which the phenomenon is applied. Also, a model type of a certain object constant or relation can only be considered under the conditions that the object constant or the relation exists. Therefore, an attribute representing an assumption is active only if its associated subject is instantiated. The conditions under which an attribute x_i is instantiated are hereafter denoted by $\gamma(x_i)$. If x_i represents an assumption with a subject that contains the relations or participants $\phi_1, \ldots, \phi_n$ then,

$$\gamma(x_i) = \gamma(\phi_1) \wedge \ldots \wedge \gamma(\phi_n).$$

From section 4.1, it is known that the combinations of assumptions under which ϕ_i is part of a scenario model are summarised in the corresponding node's label. Thus,

$$\gamma(\phi_i) = \mathcal{L}(\phi_i)$$

The compatibility constraints dictate what combinations of individual assumptions do not satisfy the constraints imposed on the scenario model. In this work, these constraints stem from the composability of assignment relations and from global and purpose-required model properties. In section 4.1.2.2, it is explained how all these restrictions are registered as justifications of the nogood node $n_\perp$. The compatibility constraints are therefore already part of the model space, and they can be extracted from the label of the nogood node $\mathcal{L}(n_\perp)$.

4.2.2 Example

Using this method, the model space of figure 4.4 can be translated into a DCSP. A graphical representation of the constraints resulting from the partial model space given in figure 4.4 is shown in figure 4.7. Assumptions with the same subject are grouped into a single attribute. For example, the assumptions "model N_1:simple", "model N_1:exponential" and "model N_1:logistic" of figure 4.4 are grouped into a single attribute "model N_1" with domain {"simple", "exponential", "logistic"}. The activity constraints over the attributes representing assumptions are defined by the conditions that activate the subject. The activity constraints of attributes are defined by tracing

Algorithm 4.4: CREATEDCSP()

comment: σ is the set of substitutions

$\sigma \leftarrow \{\}$;
comment: For each assumption class, create an attribute and a corresponding domain

for each A, assumption-class(A)
do
- $x \leftarrow$ create-attribute;
- $D(x) \leftarrow \{\}$;
- $\sigma \leftarrow \sigma \cup \{A/x\}$;
- **comment:** Fill the domain with the individual assumption instances:
- **for each** $a \in A$
  - **do**
    - $v \leftarrow$ create-value;
    - $D(x) \leftarrow D(x) \cup \{v\}$;
    - $\sigma \leftarrow \sigma \cup \{a/x : v\}$;

comment: Generate the activity constraint from the labels of the subjects

for each A, assumption-class(A)
do
- $s \leftarrow$ subject(A);
- **for each** $\{a_{1\top}, \ldots, a_{p\top}, \neg a_{1\bot}, \ldots, \neg a_{q\bot}\} \in \mathcal{L}(s)$
  - **do** add-constraint$(\sigma a_{1\top} \wedge \ldots \wedge \sigma a_{p\top} \wedge \sigma \neg a_{1\bot} \wedge \ldots \wedge \sigma \neg a_{q\bot} \rightarrow \text{active}(\sigma A))$;

comment: Generate the compatibility constraints from the label of the nogood node

for each $\{a_{1\top}, \ldots, a_{p\top}, \neg a_{1\bot}, \ldots, \neg a_{q\bot}\} \in \mathcal{L}(n_\bot)$
do add-constraint$(\sigma a_{1\top} \wedge \ldots \wedge \sigma a_{p\top} \wedge \sigma \neg a_{1\bot} \wedge \ldots \wedge \sigma \neg a_{q\bot} \rightarrow \bot$;

the necessary justification in the model space. For example, x_2 represents a set of assumptions and therefore depends on its subject, which is in this case a model variable that has another CSP variable associated with it. Hence, the activity of the latter CSP variable is a necessary condition for the activity of x_2 as illustrated in figure 4.7.

Non-composable postconditions are prohibited by compatibility constraints. For example, figure 4.4 shows that the assignments x_4 : Lotka-Volterra (denoting the Lotka-Volterra model of predation [110, 179]) and x_2 : logistic in the CSP both result in an assignment postcondition with respect to R, which has been filtered by a compatibility constraint in figure 4.7.

Given a DCSP that represents the original model space, each solution of it, that is each consistent assignment to the attributes in the DCSP, represents a consistent scenario model that meets all the requirements. For example, the assignment $(x_1$:

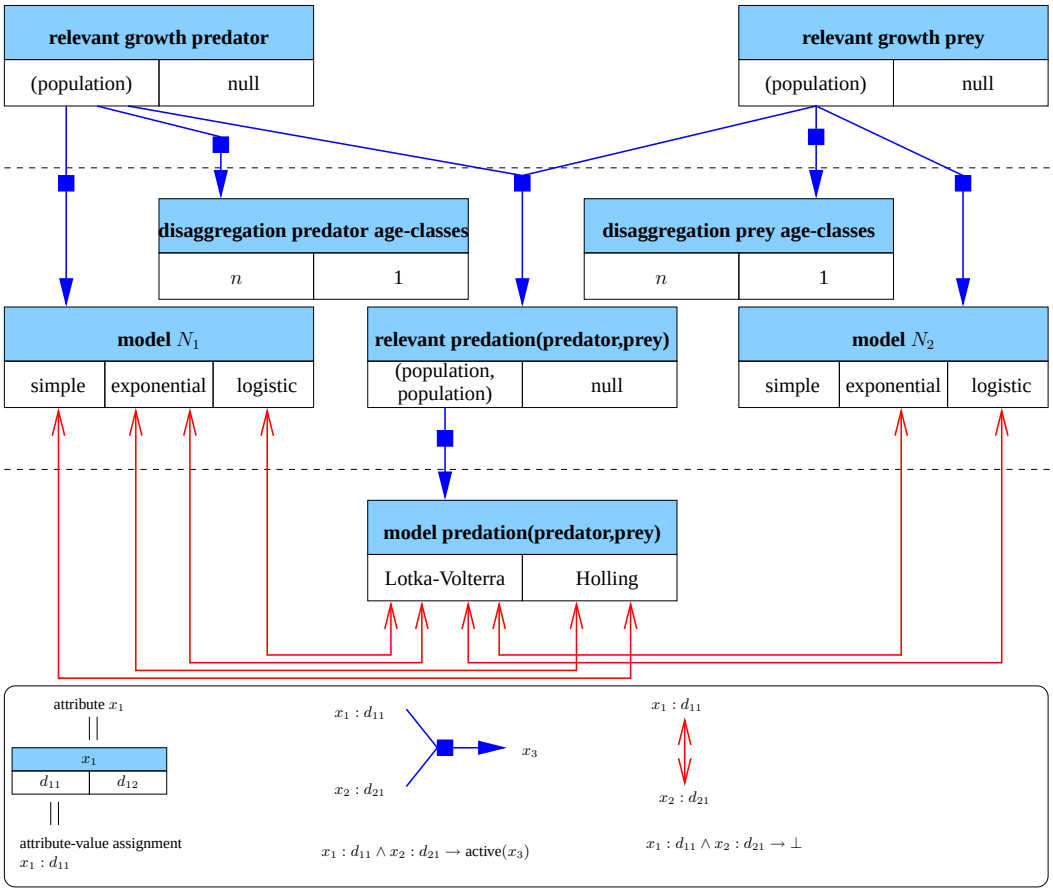


Figure 4.7: Sample DCSP derived from the model space of figure 4.4

population, R : endogenous, N_1 : endogenous, x_2 : logistic, r_0 : exogenous, K : exogenous) is consistent with the CSP of figure 4.7. By considering the assumptions associated with these attribute assignments as facts, and by following the justifications defined through the hyperarcs in the model space, a model is then constructed. In particular, the following statements can be inferred: $\frac{d}{dt}N_1 = R$, $R = r_0 N_1(1 - \frac{N_1}{K})$, exogenous($r_0$), exogenous($K$).

To perform a search for DCSP solutions, a conventional algorithm, such as the one presented in [123], can be employed. In chapter 6, novel algorithms will be introduced to solve an extension of the the DCSP.

4.3 Algorithm complexity

This section briefly discusses the overall time and space complexity of the algorithms devised for the initial construction of the model space, for the application (and combined application) of disaggregation fragments, and for the translation from model space to dynamic constraint satisfaction.

4.3.1 Initial model space construction

The algorithm that constructs the initial model space will instantiate a model fragment for each possible match of its source-participants and structural conditions. In general, pattern matching is a complex issue, but due to the nature of model fragments, an efficient implementation is possible based on the following three important observations:

- Model fragments are not allowed to be cyclic. In other words, model fragments should not instantiate new participant and relation instances are used to perform the instantiation in the first place. In practice, such circular structures would make little sense (if not non-sense at all) since a model fragment expresses a way to translate one type of (higher level) model to another type of (lower level) model. If lower level information would affect higher level choices, then model fragments should not be the mechanism to implement this. Instead model properties should be used to express that a necessary feature can or can not be realised.

- A model fragment, without purpose-required properties, can be treated as a Horn clause, as shown in definition 3.4. However, as discussed in section 4.1.2, the purpose-required model properties that may be included in some model fragments are treated separately from the remainder of the model fragment.

- GENERATEMODELSPACE() employs a forward chaining mode of reasoning.

As a result of these observations, the pattern matching procedure can be organised by traversing an ordered list of model fragments, thereby processing all the instantiations of each model fragment in one go without the need for a conflict resolution mechanism. If there are m model fragments and n sets of participants and relations matching the source-participants and structural conditions, the time complexity of this algorithm is $O(m \times n)$. Note that n is very hard to establish, but in general, it depends

on the number of similar objects and relations in the given scenario and the degree of reusability of model fragments, which are both domain dependent.

As for the actual matching of source-participants and structural conditions of the model fragment, time efficient algorithms such as the RETE pattern matching procedure [56] can be employed. This algorithm organises the rules (i.e. model fragments) in a net (i.e. rete) that allows the instantiated participants and grounded relations already in the model space to be matched incrementally. In this way, a significant amount of computationally expensive join operations of partial matches can be avoided.

Of course, storing the space of all possible models may impose significant storage requirements. However, when compared to other approaches that store a space of models such as the Graph of Models [1], the approach presented herein is far more economical. Because only model fragments are instantiated rather than entire models, the communal parts of different models need only be stored once. In theory, if a model space represents m different models that have c% in common, and each model has a space requirement of s, then the space complexity can be as low as $O(c \times s + (1-c) \times m \times s)$, rather than $O(m \times s)$ when all models are stored explicitly. For large values of m and when c is not too close to 0, $c \times s + (1-c) \times m \times s \ll m \times s$.

4.3.2 Truth maintenance

Although the construction of the model space is in itself not particularly complex, the operations of adding justifications and nogoods to an ATMS or ALTMS may significantly increase the complexity of the algorithm.

The time and space complexity of an ATMS or ALTMS primarily depends on the size of the largest labels. If, as suggested in [34], the environments are stored as bit-strings, the time complexity of individual set operations is relatively constant. Similarly, the storage space required to store an ALTMS is directly proportional to the number of environments in the labels. Hence, the time and space complexity of the ALTMS can be measured by the (space) complexity of the labels. To discuss this complexity of labels, two different types of justification operations (which are the only way of increasing the size of labels) need to be distinguished: (1) justifications added because of the instantiation of a model fragment and (2) justifications added because of the detection of inconsistencies (due to non-composable relations or required proper-

ties).

4.3.2.1 Model fragment instantiations

The former type of justification does not tend to significantly increase the complexity of the algorithm. Each instantiation of a model fragment involves creating a set of nodes containing the new instance of the model fragment and new instances for the target-participants and the postconditions. The model fragment node is then justified by the conjunction of the nodes containing the instances of the source-participants, the structural conditions and the assumptions to which the model fragment was matched. This label computation involves conjoining the labels of the source-participant and structural condition nodes. The new target-participant and postcondition nodes are then justified by the model fragment node. Because only one node is in the antecedent of these justifications, the label of the antecedent node (i.e. the model fragment node) can simply be copied to the consequent node.

It is clear that each time a model fragment instantiation depends on the instantiation of other model fragments (because the source-participants and structural condition of the former are created by the latter), the label of the former may increase exponentially with respect to the latter. Let the length of the longest sequence of such dependencies between model fragments be defined as the maximum number of levels of model fragments in the knowledge base and denoted l. Then, it can be said that the complexity of labels increases exponentially with l.

Generally speaking, however, l tends to be very small. In conventional compositional modellers designed for engineering applications, model fragments map scenario level components directly to equations, and therefore, $l = 1$. Multiple levels of model fragments were introduced in this work as a tool to represent natural dependencies between decision in ecological modelling. For example, a decision on how to model predation of one population by another only becomes relevant when it has been decided to include such a model in the first place. The number of levels of dependency between such decisions is necessarily restricted by the domain. In the large example described in section 8.2, only six levels ($l = 6$) were necessary (see figure 8.13).

4.3.2.2 Inconsistencies

The complexity of adding inconsistencies to the model space is more significant than that due to model fragment instantiations. Firstly, the label of the nogood node is likely to be more complex than that of other nodes, for two reasons:

- Because the specifications of required properties may consist of negations as well as conjunctions and disjunctions, l is not the only parameter affecting the complexity of these justifications. When a node is negated, the cross-product of the environments of the label of that node must be computed. The time and space complexity of this operation is exponential with respect to the number of environments in the label.

- Each pair of non-composable relations and each instance of a required property will expand the label of the nogood node with an addition set of nogood environments. Therefore, the size of the label of the nogood node is directly proportional to the number of matches of non-composable relations and required properties.

It is difficult to formalise the complexity of the nogood node in general because it depends on the definition of the required properties in each application. Therefore, more experimental work would be needed to establish the effect of the different factors influencing the complexity of ALTMSs in general. For the purpose of model space construction, the potential complexity of the ALTMS did not seem to be problematic. In the practical applications presented in chapter 8, the time and space required to construct the ALTMS was very small compare to the time and space requirements of the constraint satisfaction algorithms. This is mainly due to limited length l of any sequence of justifications (which is suspected to be a general feature of compositional modelling problems).

Secondly, for the ALTMS network to be sound, supersets of environments in the label the nogood node need to be removed from the other labels. As this involves comparing each nogood environment with each other environment, it is the single most time-consuming operation of the ALTMS. However, because a constraint satisfaction algorithm will be employed to find a consistent set of assumptions (i.e. a set of assumptions that is not a superset of an environment in the nogood node), it is not necessary for the ALTMS to be sound. Hence, this step can be omitted altogether.

4.3.3 Model space disaggregation

As shown in 3.3.6, the knowledge contained in a disaggregation fragment can be modelled by means of a set of model fragments as well, but this would introduce significant duplication in the knowledge base. Nevertheless, applying a disaggregation fragment involves copying and transforming part of the model space and hence, in itself, the use of disaggregation fragments does not provide any time and space savings over the use of model fragments. Therefore, the discussion of the previous subsection still applies.

Because a single disaggregation fragment replaces a large set of model fragments and because disaggregation fragments can be combined, the application of just a few disaggregation fragments can significantly expand the model space. This is illustrated in figure 4.8, where it is shown how two disaggregation fragments d_1 and d_2 (containing a single grain assumption disaggregation(x, t_i, n_i) each) can build four (no disaggregation, d_1, d_2 and $d_1 \wedge d_2$) alternatives on the same model space.

It is possible, however, to postpone the application of disaggregation fragments until it is deemed necessary. More than engineers, ecologists and system dynamics modellers tend to lack hard rules on the specification of modelling options. It is a common practice amongst ecological modellers to start with a small model first, and to make it more detailed later [54]. One way of doing this is by postponing the consideration of ways to disaggregate the model space. Postponing disaggregation also allows incremental disaggregation. This allows the user, interested in a specific partitioning of a number of subjects according to various dimensions to introduce the associated disaggregation, one at a time, and for just one subject. This enables preliminary experiments before large sets of alternative models are added to the final model space.

As figure 4.8 illustrates, disaggregation fragments do not change the original model space, with exception of the extension of the activation conditions of the original model space. Instead, new assumptions, nodes and justifications are added to the model space. The new assumptions correspond to new attributes and domains. The new nodes and justifications constrain the existing (and new) attributes in different (but similar) ways as the existing constraints. As a result of this, repairing an existing solution to meet the specifications of the new model space and corresponding DCSP requires minimal work. Indeed, it can be as simple as changing a grain assumption. Existing CSP techniques, such as those presented in [39, 118], can be utilised to repair an existing

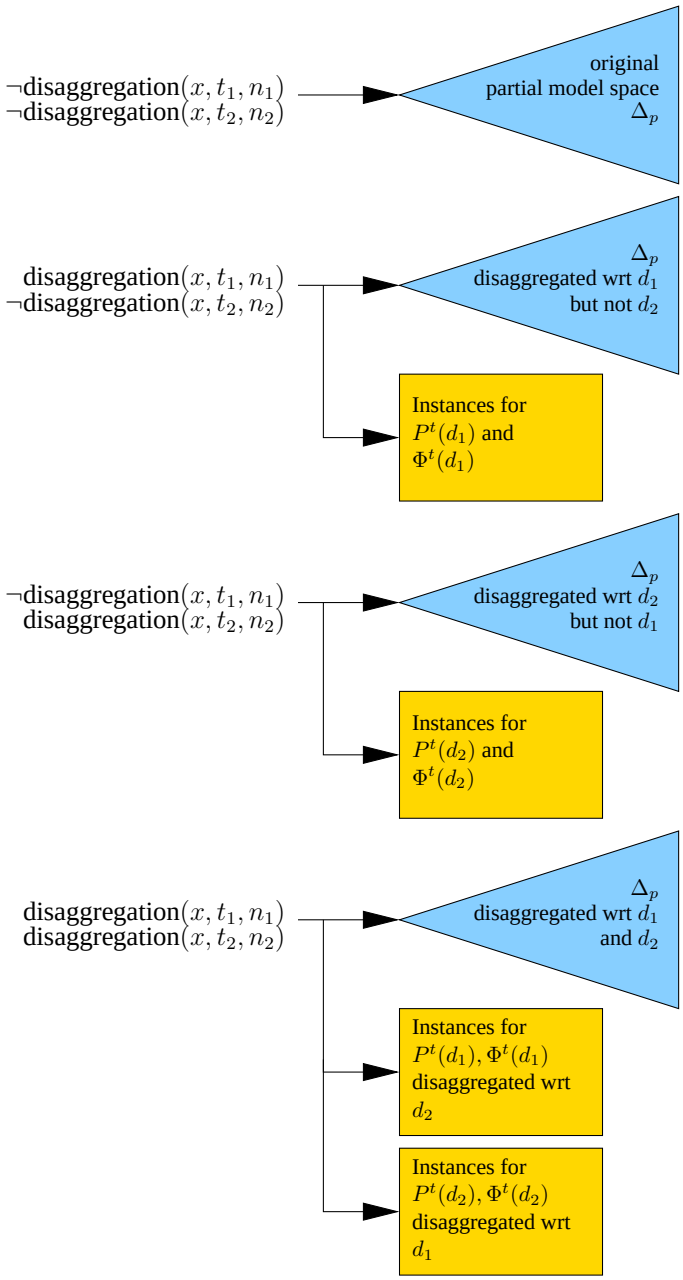


Figure 4.8: Disaggregation and the model space

solution. The incorporation of solution repair techniques for the dynamic preference CSP algorithms presented in this work remains as a future extension.

4.3.4 DCSP construction procedure

Once the model space is constructed, the DCSP can be read from the ALTMS labels that contains the model space. As shown in section 4.2, the attributes and domains are straightforwardly derived from the assumption sets, the compatibility constraints are contained in the nogood node of the ALTMS and the activity constraints are contained in the labels of nodes corresponding to model fragments. Thus, the construction of DCSPs for the specification of a given model space does not incur much time and space complexity. The computational complexity is largely associated with the solving of the resulting DCSPs. This will be discussed later. Nevertheless, getting compositional models via finding solutions to DCSPs itself will help increase the overall efficiency of the compositional modeller.

4.4 Summary

This chapter has shown how a compositional modelling problem, employing the knowledge representation formalism discussed in chapter 3 can be translated to a dynamic constraint satisfaction problem. This translation procedure consists of two phases. In the first phase, the knowledge base is instantiated with respect to a given scenario into a model space. The model space is stored using a novel type of assumption based truth maintenance system. It is constructed by first instantiating all the model fragments, then extending it by means of disaggregation fragments, and finally adding the model properties. This model space, combined with the required properties is then translated in to a dynamic constraint satisfaction problem via a straightforward mapping mechanism.

Chapter 5

Preference Calculus

5.1 Introduction

The previous chapter showed how the compositional modelling problem can be translated into a dynamic constraint satisfaction problem. However, in addition to finding a consistent model, the compositional modeller also needs to take into account potential user preferences for the assumptions upon which the scenario model will be built.

For this purpose, a generic approach is developed in this chapter to assign preferences to assumptions (and their corresponding attribute value assignments). The overview of this approach is shown in figure 5.1. It can be argued that for modelling with preferences, conventional numeric uncertainty calculi [140] may not be suitable because they implicitly assume that the user can provide a total ordering of his/her preferences and express them on a ratio or an interval scale [90, 171]. However, it is often not realistic to expect the user to generate anything more than a partial ordering of preferences expressed on an ordinal scale. Therefore, an order of preference scale is introduced that can efficiently propagate combinations of such preferences.

5.2 Background

In compositional modelling, not all consistent scenario models are equally suitable for a problem specification and hence, not all DCSP solutions may be considered as such. All models are based on a large number of underlying presumptions, which may or may not be appropriate for the application at hand [54]. Because ecological models

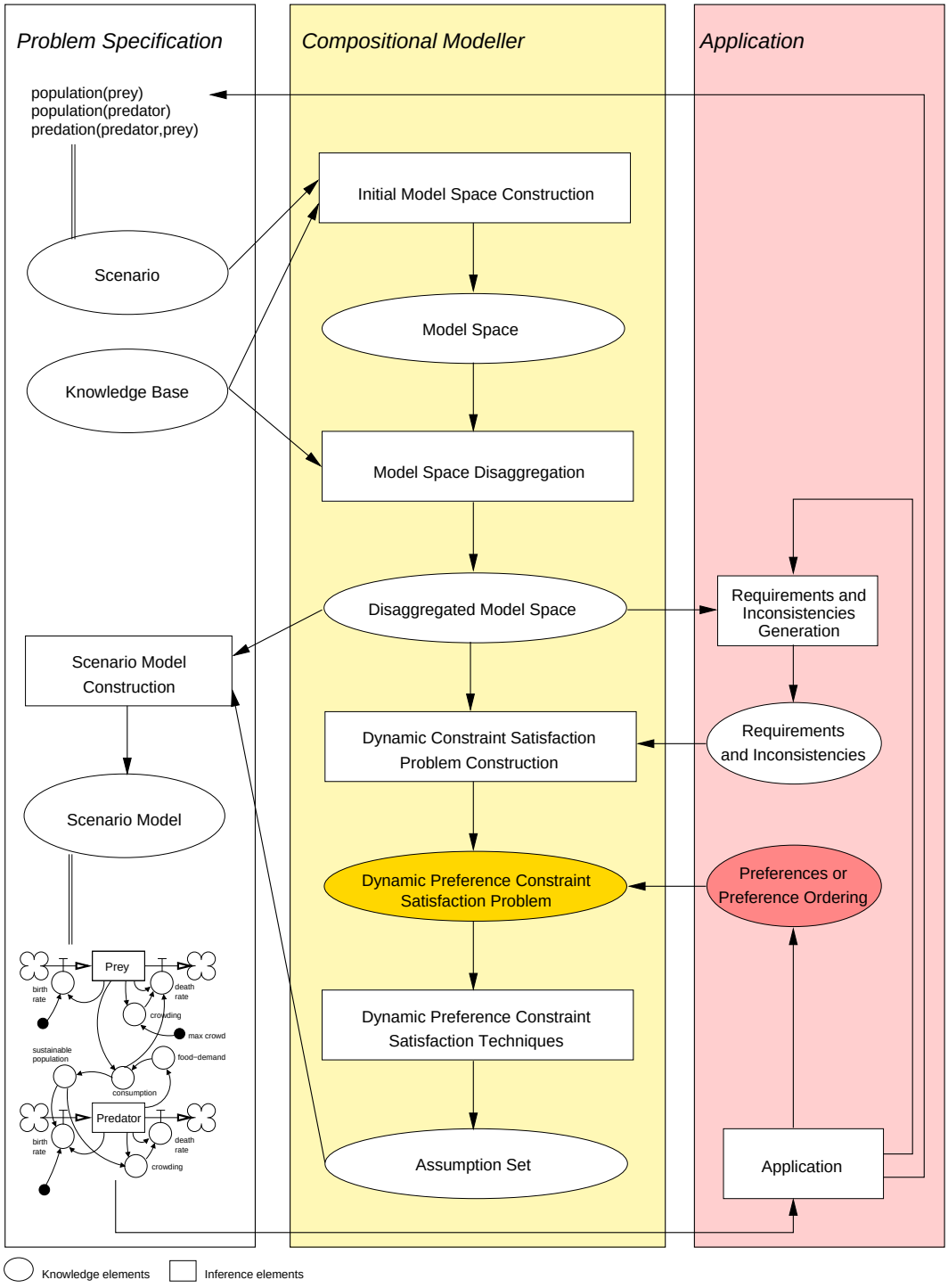


Figure 5.1: The role of the preference calculus in compositional modelling

are abstractions of very complex systems that are only partially understood [68], ecologists will have different ideas of what assumptions are appropriate. The approach taken herein is to enable the ecologist/user to order his/her personal preferences for the modelling choices that are available in the model space for a particular problem.

To enable the ecologists/users of the compositional modeller to specify their personal preferences for these modelling choices implicitly assumes that they are able to do so. In micro-economics and decision theory there exist established utility calculi to express preferences associated with rational choices numerically. However, such calculi are based on number of assumptions that would cause difficulties in their potential application to ecological modelling problems [124, 44, 83].

A utility calculus defines a utility function that assigns a numeric valuation of an individual's preference for his/her available options. This presumes:

1. that the individual can judge whether one option is preferred over the other or if both options are equally preferable,

2. that preferences are transitive, and

3. that the individual can make a trade-off between preferences.

Although such a framework is neat and appealing from a computational point of view, assumption 1 and 3 are very difficult to enforce. In particular, making a trade-off between preferences for options is extremely difficult. Although most ecologists are likely to prefer the logistic population growth model over the exponential population growth model, they would find it impossible to quantify that preference. Empirical validation of the assumptions that such a trade-off can be made by experts in other domains has generally failed [67].

To alleviate this difficulties, modelling choices could be ranked instead of being assigned utilities. Although this approach does not make the aforementioned assumption 3, assumption 1 can also not be guaranteed in general. Indeed, research has shown that human experts can not always define a strict and consistent ordering of rational options [89, 90, 171]. For example, it would not be feasible to expect an ecologist to decide whether he/she prefers one particular type of population growth model over a type of water evaporation model. Because both models describe entirely different phenomena, any comparison between them would be artificial.

Nevertheless, the combination of preferences, required to tackle the compositional modelling problem (and indeed many other problems such as design and configuration), is still required. For a given scenario and problems specification that requires an ecological model, the ecologist/user will be able to provide partial knowledge on his/her preferences. For this reason, a preference calculus that is based on *order of magnitude reasoning* [145] is suggested herein. This calculus allows preferences in a partial ordering, and at various degrees of precision, to be combined with one another.

5.2.1 Overview of order of magnitude reasoning

Clearly, *order of magnitude reasoning* (OMR) is employed in this work as a means to express and manipulate expert/user/preferences. There have been a number of important OMR calculi developed in the qualitative reasoning area [103], though they all have deficiencies for reasoning about preferences.

In general, as summarised in [145] OMR employs a calculus of coarse values that can be formally defined by:

Definition 5.1. Given a domain $\mathbb{D}$ of quantities (e.g. the set of real numbers) that are totally ordered by an operator $\prec$, a *coarse value* Q is a subset of consecutive quantities taken from $\mathbb{D}$, that is

$$\forall q_1, q_2, q_3 \in Q, q_1 \prec q_2 \prec q_3 \wedge q_1 \in Q \wedge q_3 \in Q \rightarrow q_2 \in Q.$$

Depending on the approach taken to define a domain, called the *quantity space* $\mathbb{S}$, of the coarse values that are related to one another, different OMR calculi can be devised. Normally, mathematical operations in OMR are extended from conventional mathematical operations as follows [145]:

$$f(Q_1, \ldots, Q_n) = \{ f(q_1, \ldots, q_n) \mid q_1 \in Q_1, \ldots, q_n \in Q_n \}$$

In the seminal work on OMR, *FOG* [144] and *Estimates* [145], coarse values are

defined with reference to precise values $v \in \mathbb{R}$, such that:

$$\text{identity}(v) = \{v\} \tag{5.1}$$

$$\text{close}(v) = \{q \mid |q| \leq |v \times (1 + small)|\} \tag{5.2}$$

$$\text{comparable}(v) = \{q \mid |q| \leq |v \times rough|\} \tag{5.3}$$

$$\text{sign}(v) = \begin{cases} \mathbb{R}^+ & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ \mathbb{R}^- & \text{if } v < 0 \end{cases} \tag{5.4}$$

In these definitions, *small* and *rough* are domain dependent constants respectively representing fine grain and large grain distinctions.

Two coarse values Q_1 and Q_2 are deemed qualitatively equal ($Q_1 \approx Q_2$) if $Q_1 \cap Q_1 \neq \emptyset$. In general, this form of equality differs from conventional mathematical equality in that it is not transitive. In order to avoid the problems of reasoning with the non-transitive operator $\approx$, the equations are usually rewritten employing the inclusion relation $\subseteq$ (which is transitive) instead of $\approx$. Systems of such rewritten equations can now be solved by means of resolution of scales (i.e. replacing coarse sets by their definition given in (5.1), (5.2), (5.3) and (5.4)) and of substitution by coarser values (which amounts to $A \subseteq B, B \subseteq C \rightarrow A \subseteq C$).

A number of variations on the work presented in [144, 145] have been proposed. For example, O[M] [115, 116] employs 7 (instead of 4, including identity) basic ordering relations and allows for the use of disjunctions of consecutive primitive ordering relations. The parameters controlling the definition of the quantities that belong to a coarse values (e.g. small and rough in FOG), do not have to be defined by a single value. Instead, the ranges describing the coarse values are reduced when used in the antecedent of an inference and they are expanded when used in the consequent of an inference. In this way, coarse values do not have the tendency to expand in size over a sequence of inference, thus more closely approximating human reasoning (even though this is not mathematically correct when repeated over a number of inferences).

The further extension of FOG and O[M], *ROM(K)* [32] and ROM($\mathbb{R}$) [31] are motivated by a need to tie the OMR calculus more closely to those defined on $\mathbb{R}$. To that end, these OMR calculi have associated themselves with a novel theoretical foundation of axioms and derived theorems, and with additional primitive ordering relations

and disjunctions thereof. The latter feature also allows for a smoother scale of coarse values.

A very different approach is taken in *NAPIER* [132, 133], which groups different numerical quantities into one coarse value provided their absolute values have the same logarithm rounded to the nearest integer. That is, the order of magnitude of a quantity q is computed by:

$$om(q) = \text{round}(\log_b(\mid q \mid))$$

The value b is presumed to be the lowest number that is deemed significantly greater than 1. A coarse value is then defined by the order of magnitude om and its sign (or the sign of the underlying quantities). The rules of inference for this calculus have been derived from interval calculus [125] and include constraints such as:

$$om(q_1) + om(q_2) \leq om(q_1 \times q_2) < om(q_1) + om(q_2) + 1$$

$$\begin{cases} om(q_1) \leq om(q_1 + q_2) \leq om(q_1) + 1 & \text{if } om(q_1) = om(q_2) \\ om(q_1 + q_2) = \max(om(q_1), om(q_2)) & \text{if } om(q_1) \neq om(q_2) \end{cases}$$

For a given system of equations, the rules of inference instantiate into a system of constraints that, when solved, results in upper and lower boundaries on the orders of magnitude of the different quantities in the system.

Other OMR work has aimed at developing a theory to link qualitative reasoning, over a series of increasingly less coarse OMR calculi, to numerical reasoning. In [131], four OMR calculi are related to one another: (1) qualitative reasoning with confluences [36], (2) FOG and O[M], (3) a logarithm based OMR calculus, and (4) a calculus of rounded numbers. In the fourth calculus, quantities are expressed as $x \times b^{om}$, where x is a number of n significant digits and b and om have the same meaning as in NAPIER. Clearly, this calculus approximates the calculus of real numbers as n approximates infinity.

In [168], a theory is developed to link sign algebra to conventional numeric algebra, over the so-called *Q-algebrae* containing quantities of various granularities. The coarse values $Q \in \mathbb{\$}$ are related to one another via an "is more precise than" relation $\subseteq$. Qualitative equality is defined as:

$$\forall Q_1, Q_2 \in \mathbb{\$}, Q_1 \approx Q_2 \leftarrow (\exists Q \in \mathbb{\$}, Q \subseteq Q_1 \wedge Q \subseteq Q_2)$$

Here, a Q-algebra is defined by a quantity space and with two operators $\oplus$ and $\otimes$, corresponding to qualitative addition and qualitative multiplication. These operators are assumed to be commutative and associative, and $\otimes$ is assumed to be distributive with respect to $\oplus$.

The OMR calculi outlined above aim to provide a means of reasoning with incomplete knowledge of numeric values. As such, these techniques are employed for applications in areas such as commonsense reasoning [145] and model-based reasoning [69]. Because the underlying domain $\mathbb{D}$ upon which coarse values are built usually equals the set of real numbers $\mathbb{R}$ or a subset of $\mathbb{R}$, their suitability for reasoning about preferences is limited. In particular, the aforementioned disadvantages of a numeric utility calculus or a total symbolic ordering of preferences still hold.

5.2.2 Reasons for new OMR calculus

In many application problems of OMR, including the work of this thesis, the actual "values" of preferences do not matter, all that is useful is that they can be compared and combined with one another to derive an overall most preferred solution. This task requires a means of expressing and evaluating the relative magnitudes of preferences with respect to one another and a means of combining them in a totally consistent manner. For the present work, the required representation, valuation and combination must suit the description and solution of a complex CSP. For these reasons, this work adopts the basic ideas of OMR and proposes and a novel OMR calculus.

In conventional OMR, the available ordering information is projected to the real number line (or a similar totally ordered set of precise values) because all coarse values considered to be abstractions of real numbers. As opposed to these OMR calculi, the coarse values representing preferences are not abstractions of valuations taken from a totally ordered set, such as $\mathbb{R}$ or $\mathbb{N}$. Instead, they should be taken from a partially ordered set. This is because, although in the problem domain concerned, subsets of totally ordered values may exist, the values in one subset may be totally unrelated to values in another.

Take for example the partially ordered values in figure 5.2. Here, very high is greater than high, extra large is greater than large and all of these values are greater than medium. However, high and very high are incomparable to large and very large as

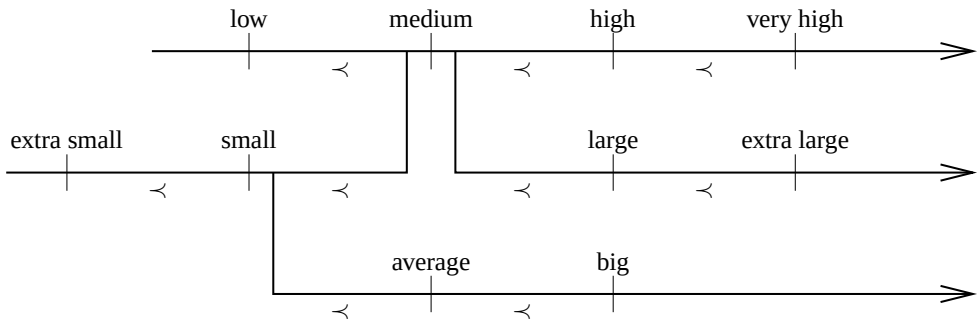


Figure 5.2: Partially ordered values

they represent different concepts. Conventional OMR calculi would attempt to make a comparison between all such values to come up, with a least a range of ordering relations between them. Yet, an ordering between high/very high and large/very large would make no sense. That is, whilst conventional OMR calculi may deem $n \times \text{high} \succ$ large for an integer n that is much greater than 0, the expert/user in the problem domain may well specify in figure 5.2, that high and large things can not be compared with one another.

5.2.3 Order of magnitude preference scales

This subsection presents a theory of representing and comparing preferences by means of orders of magnitude. Each order of magnitude preference (see later for the precise definition of these OMPs) is a combination of basic preference quantities (BPQs). BPQs are the smallest units of preference valuation and they are predefined as part of an order of magnitude scale. The set of BPQs $\mathbb{B}$ of such an order of magnitude scale are partially ordered with respect to one another. In general, the implicit value of an OMP P equals the combination $p_1 \oplus \ldots \oplus p_n$ of its constituent BPQs $p_1, \ldots, p_n$.

Definition 5.2. An *order of preference scale* definition is a tuple $\langle \mathbb{B}, O_<, O_\ll \rangle$ where $\mathbb{B}$ is *the set of BPQs of the scale*, and $O_<$ and $O_\ll$ are *partial orders* defined over $\mathbb{B} \times \mathbb{B}$. The pairs $(x,y) \in O_<$, where $x,y \in \mathbb{B}$, represent that $(x < y) \wedge \neg(x \ll y)$ whereas the pairs $(x,y) \in O_\ll$, where $x,y \in \mathbb{B}$, represent that $x \ll y$.

Here, a partial ordering of BPQs is provided by the "order of magnitude smaller than" relation $\ll$ and the "equivalent order of magnitude as" relation $\sim$. BPQs that have the same order of magnitude can be compared with one another using the "smaller

than within the same order of magnitude" relation $<$. Therefore,

$$\begin{aligned}&\forall p_1, p_2 \in \mathbb{B}, (p_1, p_2) \in O_< \rightarrow (p_1 \sim p_2) \wedge (p_2 \sim p_1)\\&\forall p_1, p_2, p_3 \in \mathbb{B}, ((p_1, p_2) \in O_< \vee (p_2, p_1) \in O_<) \wedge (p_2 \sim p_3) \rightarrow p_1 \sim p_3\end{aligned} \tag{5.5}$$

Naturally, the order of magnitude smaller than relation is shared by all BPQs within the same order of magnitude. That is,

$$\forall p_1, p_2, p_3 \in \mathbb{B}, p_1 \sim p_2 \wedge p_2 \ll p_3 \rightarrow p_1 \ll p_3$$
$$\forall p_1, p_2, p_3 \in \mathbb{B}, p_1 \sim p_2 \wedge p_3 \ll p_2 \rightarrow p_3 \ll p_1$$

Definition 5.3. Given an order of preference scale definition $\langle \mathbb{B}, O_<, O_\ll \rangle$, every combination of BPQs $q_1 \oplus \ldots \oplus q_n$, where $q_1, \ldots, q_n \in \mathbb{B}$, is said to be an *order of magnitude preference* (OMP) defined in $\langle \mathbb{B}, O_<, O_\ll \rangle$.

It is assumed that $\oplus$ is commutative and associative. Thus, an order of magnitude preference $P = P_1 \oplus P_2$ where $P_1 = q_{11} \oplus \ldots \oplus q_{1n_1}$ and $P_2 = q_{21} \oplus \ldots \oplus q_{2n_2}$ equals $q_{11} \oplus \ldots \oplus q_{1n_1} \oplus q_{21} \oplus \ldots \oplus q_{2n_2}$. Because an order of preference magnitude may contain multiple instance of the same quantity (e.g. $P = q \oplus q$), the following shorthand notation is introduced:

$$P = 1 \times q = q$$
$$P = n \times q = (n-1) \times q \oplus q$$

Definition 5.4. Given an OMP $P = p_1 \oplus \ldots \oplus p_n$, a function representing that OMP can be defined by:

$$f_P : \mathbb{B} \rightarrow \mathbb{N} : p \mapsto f_P(p)$$

where $\mathbb{B}$ is the set of BPQs, $\mathbb{N}$ is the set of natural numbers and $f_P(p)$ calculates the number of occurrences of p in $p_1, \ldots, p_n$.

Because these OMPs correspond to the preference of a possible solution (e.g. a scenario model in compositional modelling or a set of attribute-value assignments in a DPCSP), the ultimate aim of the calculus is to compare OMPs with one another. In what follows, an approach will be presented to compute a partial ordering relation $\prec$ over the OMPs, based on the constituent BPQs of the OMPs. Generally speaking, the calculus is based on the following assumptions:

- *Properties of* $\oplus$: The combination operator $\oplus$ is assumed to be commutative, associative and strictly monotonic ($P \prec P \oplus P$). The latter assumption is made to better reflect the ideas underpinning conventional utility calculi.

- *Prioritisation*: A combination of BPQs is never an order of magnitude greater than its constituent BPQs. That is, given the following ordering of BPQs $p_1 \sim p_2 \sim \ldots \sim p_n \ll p$,

$$p_1 \oplus p_2 \oplus \ldots \oplus p_n \prec p$$

  Also, distinctions at higher orders of magnitude are considered to be more significant than those at lower orders of magnitude. That is, given an ordering of BPQs $p_1 \sim \ldots \sim p_{m-1} \sim p_m \sim \ldots \sim p_n \ll p_a < p_b$,

$$p_1 \oplus \ldots \oplus p_{m-1} \oplus p_a \prec p_m \oplus \ldots \oplus p_n \oplus p_b$$

  In terms of OMPs, it means that the DPCSP algorithm will prioritise the optimisation associated with preferences of higher order of magnitude.

- *Strict monotonicity*: Even though distinctions at higher orders of magnitude are more significant, distinctions at lower orders of magnitude are not negligible. That is, given an ordering of BPQs $p_1 < p_2$ and an OMP P, $p_1 \oplus P \prec p_2 \oplus P$, irrespective of the orders of magnitude of the BPQs that constitute P. This is a departure from conventional OMR. If the OMPs associated with two (partial) DPCSP solutions contain equal BPQs at a higher order of magnitude, it is usually desirable to compare both solutions further in terms of the (less important) constituent BPQs at lower orders of magnitude.

- *Partial ordering maintenance*: Conventional OMR is motivated by the need for abstract descriptions of real-world behaviour, whereas the OMP calculus is motivated by incomplete information. As opposed to conventional OMR, OMPs do not map onto the real number line. This implies that, when the user states, for example, that $p_1 < p_2 < p$ and that $p_3 < p_4 < p$, the explicit absence of ordering information between the BPQs in $\{p_1, p_2\}$ and those in $\{p_3, p_4\}$ means that the user can not compare them (e.g. because they are entirely different things). Consequently, $p_1 \oplus p_2$ would be deemed incomparable to $p_3 \oplus p_4$ (i.e. $p_1 \oplus p_2 ? p_3 \oplus p_4$), rather than roughly equivalent.

These assumptions can be formalised in a more general definition of the ordering relation $\prec$. Let $\mathbb{B}(p)$, $p \in \mathbb{B}$, be the subset of $\mathbb{B}$ that contains the BPQs of the same order of magnitude as p, i.e. $\mathbb{B}(p) = \{p_i \mid p_i \in \mathbb{B}, p_i \sim p\}$. Then, the constituent BPQs of an OMP P_1 that are within the same order of magnitude as a given BPQ p, are less than or equal to those of an OMP P_2 if $\forall p_i \in \mathbb{B}(p)$:

$$(f_{P_1}(p_i) + \sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_1}(p_j)) \leq (f_{P_1}(p_i) + \sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_1}(p_j)) \tag{5.6}$$

This is denoted by $P_1 \preccurlyeq_p P_2$. The constituent BPQs of an OMP P_1 that are within the same order of magnitude as that of a given BPQ p, are less than but not equal to those of an OMP P_2 if $P_1 \preccurlyeq_p P_2 \wedge \neg(P_2 \preccurlyeq_p P_1)$. This is denoted by $P_1 \prec_p P_2$.

More generally, an OMP P_1 is less than an OMP P_2 if, for each distinct order of magnitude, either P_1 is less than P_2 for the BPQs within this order of magnitude, or there are BPQs at a higher order of magnitude for which P_1 is less than P_2:

$$P_1 \prec P_2 \leftarrow \forall p_a \in \mathbb{B}, (P_1 \prec_{p_a} P_2) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_1 \prec_{p_b} P_2) \tag{5.7}$$

Two OMPs are said to be equal ($P_1 = P_2$) if P_1 and P_2 are a combination of the same collection of BPQs. Alternatively, two OMPs P_1 and P_2 are incomparable with one another ($P_1 ? P_2$) if

$$\begin{aligned} P_1 ? P_2 \leftarrow & P_1 \neq P_2 \\ & \forall p_a \in \mathbb{B}, (P_1 \prec_{p_a} P_2) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_1 \prec_{p_b} P_2) \\ & \forall p_a \in \mathbb{B}, (P_2 \prec_{p_a} P_1) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_2 \prec_{p_b} P_1) \end{aligned} \tag{5.8}$$

Theorem 5.5.

$$P_1 \prec P_2 \rightarrow \forall p_a \in \mathbb{B}, (P_1 \prec_{p_a} P_2) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_1 \prec_{p_b} P_2)$$

Proof: See appendix A, page 287.

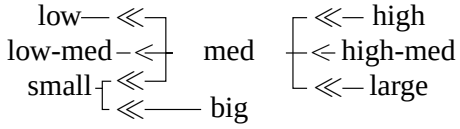


Figure 5.3: Sample OM preference scale

The definition of $\prec$ results in a partial ordering of OMPs. Figure 5.3 shows a sample set of BPQs and ordering relations between them. Based on it, the following comparisons can be made between OMPs:

$$\text{high-med} \oplus \text{med} \prec \text{high} \tag{5.9}$$

$$\text{high-med} \oplus \text{med} \succ \text{med} \oplus \text{low-med} \tag{5.10}$$

$$\text{high} \oplus \text{large?big} \oplus \text{small} \tag{5.11}$$

Relation (5.9) is true because high is an order of magnitude greater than high-med and med. Relation (5.10) is justified by $(\text{high-med} > \text{med})$ and $(\text{med} > \text{low-med})$. The OMPs in (5.11) are incomparable because there is no path between big and high or between big and large.

5.3 Efficient comparison of OMPs

The theory above is sufficient to compare order of magnitude preferences, albeit inefficient. Given no total ordering, the BPQs form a network of symbolic values related by the $<$ and $\ll$. In the worst case scenario, all the BPQs of two OMPs must be pairwise compared, leading to a complexity of $O(q^q)$ where q is the number of BPQs defined. This section introduces a much more efficient approach.

5.3.1 Comparing BPQs

An order of preference scale definition formalises a partial ordering of BPQs. The preference values that need to be compared with one another are combinations of these BPQs. However, before any comparison of preference values is possible, comparisons between BPQs need to be discussed.

Definition 5.6. A *positive path* between two BPQs x and y and with respect to an ordering relation O, denoted path+(O,x,y), is an ordered set of BPQs $\{q_1,q_2,\ldots,q_{n-1},q_n\}$ such that $(x,q_1) \in O$, $(q_1,q_2) \in O$,..., $(q_{n-1},q_n) \in O$ and $(q_n,y) \in O$. A *negative path* between two BPQs x and y and with respect to an ordering relation O, denoted path-(O,x,y), is an ordered set of BPQs $\{q_n,q_{n-1},\ldots,q_2,q_1\}$ such that $(y,q_n) \in O$, $(q_n,q_{n-1}) \in O$,..., $(q_2,q_1) \in O$ and $(q_1,x) \in O$. The empty set $\{\}$ is both a positive

and a negative path between any BPQ and itself with respect to any ordering relation.

Theorem 5.7. $\text{path}+(O,x,y) = \text{path}-(O,y,x)$

Proof: See appendix A, page 287.

Note that the partial orders defined by $O_<$ and $O_\ll$ form a directed acyclic graph (DAG) over the BPQs in $\mathbb{B}$. The problem of finding a path between two nodes in a DAG is a well understood problem. However, this operation is inadequate for the purpose of comparing two OMPs. Because OMPs are combinations of BPQs, algorithms are needed to efficiently determine which of the BPQs in two sets are related to one another and in what order. As mentioned earlier, establishing whether a path exists between each pair of BPQs would be very inefficient. Therefore, this subsection introduces the basic concepts of scheme to annotate each BPQ with a label that will enable a more efficient comparison between OMPs.

Obviously, the orderings in a order of magnitude preference scale should not be contradictory, i.e. inconsistent. An ordering is inconsistent if there are two different BPQs such that a positive and a negative path exist from one BPQ to the other. Indeed, the positive path indicates that the first BPQ is greater than the second whereas the negative path implies the reverse. More formally,

Definition 5.8. An order of magnitude preference scale $\langle \mathbb{B}, O_<, O_\ll \rangle$ is inconsistent if

- $\neg(\exists q_1, q_2 \in \mathbb{B}, (q_1 \neq q_2) \wedge \text{path}+(O_<, q_1, q_2) \wedge \text{path}-(O_<, q_1, q_2))$, and
- $\neg(\exists q_1, q_2 \in \mathbb{B}, (q_1 \neq q_2) \wedge \text{path}+(O_\ll, q_1, q_2) \wedge \text{path}-(O_\ll, q_1, q_2))$.

An order of magnitude preference scale is consistent if it is not inconsistent.

In what follows, it is implicitly assumed that all order of preference magnitude scales and orderings are consistent. In practical applications, consistency of the order of magnitude preference scale can be verified by representing it as directed graph in which the nodes denote BPQs and the arcs correspond to ordering relations between the BPQs. If is easy to prove that the order of magnitude preference scale is consistent if and only if the directed graph does not contain cycles.

These concepts can be illustrated by means of the ordering presented in figure 5.4. This figure contains a number of BPQs in squares (e.g. extra sml, small, medium and large and extra lg) and a set of ordering relations depicted by the arrows (e.g. (extra

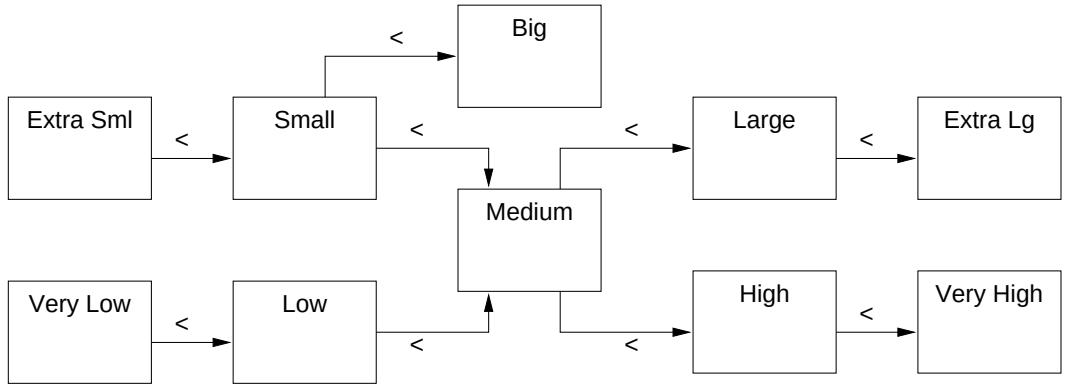


Figure 5.4: A simple set of ordering relations of BPQs

sml,small), (small,medium), (medium,large) and (large,extra lg)). In this particular order of preference scale definition, a positive path path+(O,extra sml,extra lg) equals {small,medium,large}. If the order of magnitude preference scale presented in figure 5.4 were to contain a BPQ q such that (medium,q)$\in O$ and (q,small) $\in O$, then that order of preference magnitude scale would be inconsistent.

Based on the above definition, a BPQ q_x is larger than a BPQ q_y with respect to an ordering relation, if a positive path path$+(O,q_y,q_x)$ and hence a negative path path$-(O,q_x,q_y)$ exists. Comparing BPQs by determining paths between them is quite cumbersome. The approach taken herein is to store some information with respect to each BPQ's position in the scale in advance. This requires the introduction of the notions of cross-over quantity, distance and strand.

Definition 5.9. A BPQ q is a *cross-over quantity* with respect to an ordering relation O, denoted cross-over(O,q) if there are at least two BPQs x and y, with $x \neq y$, for which $[(x,q) \in O \wedge (y,q) \in O] \vee [(q,x) \in O \wedge (q,y) \in O]$.

Definition 5.10. Given two BPQs x and y and an ordering relation O,

- a value n, with $n \in \mathbb{N}^+$ (the set of positive integers), is a distance between q_x and q_y with respect to O if there exists a path+(O,q_x,q_y) involving n elements,
- a value $-n$, with $n \in \mathbb{N}^+$, is a distance between two BPQs x and y with respect to an ordering relation O if there exists a path-(O,q_x,q_y) involving n elements,
- A value $m \in \mathbb{N}$ (the set of integers) is the maximum distance between x and y with respect to O, denoted $d_{\max}(q_x,q_y,O)$, if there is no other distance n between q_x

and q_y with respect to O such that $\mid m \mid < \mid n \mid$.

Definition 5.11. Given two BPQs q_x and q_y and an ordering relation O such that $(q_x, q_y) \in O$, the q_y-strand of q_x with respect to O, denoted strand$+(O, q_x, q_y)$ is the set of all BPQs q for which a positive path path$+(O, q_y, q)$ exists and strand$-(O, q_y, q_x)$ is the set of all BPQs q for which a negative path path$-(O, q_x, q)$ exists.

Hereafter, the predicate strand(O, q_x, q_y) denotes strand$+(O, q_x, q_y)$ or strand$-(O, q_x, q_y)$, whichever is a non-empty set.

Definition 5.12. Given two BPQs q_x and q_y and an ordering relation O such that $(q_x, q_y) \in O$ or $(q_y, q_x) \in O$, the *extended* q_y*-strand* of q_x with respect to O, denoted strand$\circ(O, q_x, q_y)$ is the set of all BPQs q such that $\neg(q \in \text{strand}(O, q_x, q_y))$ and that either q is a member of a strand of a cross-over point in strand(O, q_x, q_y) or, recursively, q is a member of a strand of a cross-over point in strand$\circ(O, q_x, q_y)$.

In the example of figure 5.4, small and medium are cross-over quantities. Cross-over quantity medium has four strands {large, extra lg}, {high, very high}, {extra sml, small} and {very low, low}. The BPQ big is not an element of any strand of medium, but it is an element of strand$\circ$(O,medium,small). The latter indicates that big is connected to medium via the small strand.

For each cross-over quantity q_c, there exist multiple strands of BPQs greater than q_c or there are multiple strands of BPQs smaller than q_c. Generally speaking, BPQs in different strands on the same side of a cross-over point q_c, i.e. either both strands consist of BPQs greater than q_c or both strands consist of BPQs smaller than q_c, can not be compared with one another. BPQs on different strands that are on different sides of a cross-over quantity q_c, i.e. one is smaller and the other is greater than q_c, can be compared with one another. As a result of this, the following relations between strands can be defined, for a given $\mathbb{B}$ and O:

$$\text{strand}(O, x, y) = \text{strand}(O, x, y) \tag{5.12}$$

$$\forall x, y, z \in \mathbb{B}, (x, y) \in O, (y, z) \in O \rightarrow \text{strand}-(O, y, x) \prec \text{strand}+(O, y, z) \tag{5.13}$$

$$\forall x, y, z \in \mathbb{B}, (x, y) \in O, (x, z) \in O \rightarrow \text{strand}+(O, x, y) ? \text{strand}+(O, x, z) \tag{5.14}$$

$$\forall x, y, z \in \mathbb{B}, (y, x) \in O, (z, x) \in O \rightarrow \text{strand}-(O, x, y) ? \text{strand}-(O, x, z) \tag{5.15}$$

For example, high and large in figure 5.3 are BPQs that can not be compared with one another because they members of different strands with respect to cross-over quantity med. For this reason, all BPQs are annotated with the strands (with respect to each cross-over quantity) they part of. Given a set of BPQs $\mathbb{B}$ and a set of ordering relations over these BPQs O, the strand assignments of the BPQs $q \in \mathbb{B}$, denoted $S_{(\mathbb{B},O)}(q)$, is assigned as follows:

1. For each $q \in \mathbb{B}$

$$\begin{aligned} S_{(\mathbb{B},O)}(q) =& \{\text{strand}(O,c,x) \mid \text{cross-over}(O,c) \wedge q \in \text{strand}(O,c,x)\} \cup \\ & \{\text{strand} \circ (O,c,x) \mid \text{cross-over}(O,c) \wedge q \in \text{strand} \circ (O,c,x)\} \end{aligned}$$

2. Select a random $q_c \in \mathbb{B}$ such that $\exists q_1, q_2 \in \mathbb{B}, \text{strand}(O,q_1,q_2) \in S_{(\mathbb{B},O)}(q_c)$

3. For each $q \in \mathbb{B}$

$$\begin{aligned} S_{(\mathbb{B},O)}(q) =& S_{(\mathbb{B},O)} \cup \{\text{strand}(O,q_c,x) \mid q \in \text{strand}(O,q_c,x)\} \cup \\ & \{\text{strand} \circ (O,c,x) \mid q \in \text{strand} \circ (O,q_c,x)\} \end{aligned}$$

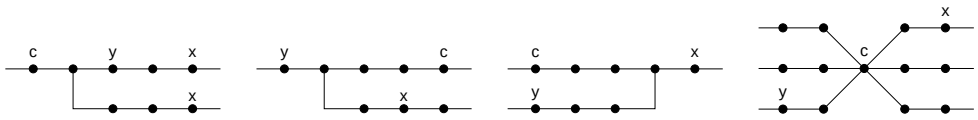
4. Repeat steps 2 and 3 until $\{q_c \in \mathbb{B} \mid S_{(\mathbb{B},O)}(q_c) = \emptyset\} = \emptyset$.

The strand assignments of two BPQs may provide insufficient information for a comparison of the BPQs with respect to a set of ordering relations. Therefore, the strand assignments need to be augmented with the maximum distance from the respective cross-over quantities. For this, the concept of the label of a BPQ is introduced.

Definition 5.13. Given a BPQ $q \in \mathbb{B}$ and a set of ordering relations over these BPQs O, the *label* of q with respect to $\mathbb{B}$ and O is defined by:

$$\begin{aligned} L_{(\mathbb{B},O)}(q) = & \{\langle \text{strand}(O,c,s), d, 1\rangle \mid \text{strand}(O,c,s) \in S_{(\mathbb{B},O)}(q), d = d_{\max}(O,c,q)\} \cup \\ & \{\langle \text{strand} \circ (O,c,s), d_x, 1\rangle \mid \langle \text{strand} - (O,c,s), d_x, 1\rangle \in L_{(\mathbb{B},O)}(q_x), \text{path} + (O,q_x,q\} \cup \\ & \{\langle \text{strand} \circ (O,c,s), d_x, 1\rangle \mid \langle \text{strand} + (O,c,s), d_x, 1\rangle \in L_{(\mathbb{B},O)}(q_x), \text{path} - (O,q_x,q\} \end{aligned}$$

The label of a BPQ is a set of triplets representing strand, distance and an integer. The integers will be ignored in the remainder of this subsection, because they are



(a) case a (b) case b (c) case c (d) case d

Figure 5.5: Comparing strand distance pairs

all equal to 1 (but they will become important in the next subsection). As such, the combination of strand s with distance d can be represented as a pair (s,d).

Definition 5.14. A BPQ $q \in \mathbb{B}$, such that $\langle s,d,1\rangle \in L_{(\mathbb{B},O)}(q)$, is said to be a member of a strand distance pair (s,d), denoted $q \in (s,d)$.

As explained earlier, each strand represents a specific subordering in the order of preference magnitude scale and each distance is an ordinal description of the specific position in that subordering. The strand distance pairs of the labels of a BPQ uniquely identify the location of that BPQ in the scale.

Theorem 5.15. The combination of labels of a BPQ is unique to that BPQ

Proof: See appendix A, page 287.

As a result of this theorem, the labels can be used to order BPQs with respect to one another. To show this property, the notion of ordering strand distance pairs needs to be introduced.

Definition 5.16. Given a set of BPQs $\mathbb{B}$ and an ordering O over these BPQs, and given two strand distance pairs (s_1,d_1) and (s_2,d_2) such that

$$\forall q_1,q_2 \in \mathbb{B}, \langle s_1,d_1,1\rangle \in L_{(\mathbb{B},O)}(q_1) \wedge \langle s_2,d_2,1\rangle \in L_{(\mathbb{B},O)}(q_2) \rightarrow \text{path}+(O,q_2,q_1)$$

(s_1,d_1) is said to be *ranked higher, with respect to ordering O,* than (s_2,d_2). This feature is denoted $(s_1,d_1) \succ_O (s_2,d_2)$. A strand distance pair (s_1,d_1) is said to be *ranked higher than or equal to* a strand distance pair (s_2,d_2), with respect to ordering O, if $(s_1,d_1) \succ_O (s_2,d_2)$ or $(s_1,d_1) = (s_2,d_2)$. This is denoted as $(s_1,d_1) \succcurlyeq_O (s_2,d_2)$.

Figure 5.5 illustrates definition 5.16 with a graphical representation of the four different cases where two BPQs x and y have the respective strand distance pairs (s_1,d_1)

and (s_2, d_2) such that $(s_1, d_1) \succ_O (s_2, d_2)$. These illustrated cases are:

$$s_1 = \text{strand}(O, c, s) \wedge s_2 = \text{strand}(O, c, s) \wedge d_1 > d_2, \text{ or} \quad \text{(case a)}$$

$$s_1 = \text{strand} \circ (O, c, s) \wedge s_2 = \text{strand}(O, c, s) \wedge d_1 < 0 \wedge d_2 < d_1, \text{ or} \quad \text{(case b)}$$

$$s_1 = \text{strand}(O, c, s) \wedge s_2 = \text{strand} \circ (O, c, s) \wedge d_2 > 0 \wedge d_2 < d_1, \text{ or} \quad \text{(case c)}$$

$$s_1 \succ s_2. \quad \text{(case d)}$$

The dots denote BPQs and the lines denote orderings between BPQs. When two dots are connected by a line, this implies that the leftmost BPQ is to be ranked lower than the rightmost BPQ. Note that, because the order of magnitude preference scale is assumed to be consistent, this representation allows for all possible cases to be represented. The BPQ denoted as c represents the cross-over quantity of the strand or the extended strand in the definition.

Theorem 5.17. $[(s_1, d_1) \succ_O (s_2, d_2)] \wedge [(s_2, d_2) \succ_O (s_3, d_3)] \rightarrow [(s_1, d_1) \succ_O (s_3, d_3)]$

Proof: This theorem is a direct consequence of definition 5.16.

Theorem 5.18. If $(s_1, d_1) \succ_O (s_2, d_2)$, then for each pair of BPQs $q_1, q_2 \in \mathbb{B}$ such that $q_1 \in (s_1, d_1)$ and $q_2 \in (s_2, d_2)$, O defines a positive path from q_2 to q_1.

Proof: This theorem is a direct consequence of definition 5.16.

By means of theorems 5.17 and 5.18, BPQs can be ordered by means of the strand distance pairs in their labels. This ordering, formalised in definition 5.19 will be shown to be equivalent to the ordering described by the path relations between BPQs, in theorem 5.21.

Definition 5.19. Given a set of BPQs, $\mathbb{B}$, and an ordering O over the BPQs, a BPQ q_1 *is ranked at least as high, with respect to ordering O,* as a BPQ q_2 (denoted $q_1 \succcurlyeq_O q_2$) if:

$$\forall \langle s_2, d_2, 1 \rangle \in L_{(\mathbb{B},O)}(q_2), [\exists \langle s_1, d_1, 1 \rangle \in L_{(\mathbb{B},O)}(q_1),$$
$$((s_1, d_1) \succ_O (s_2, d_2)) \vee ((s_1 = s_2 \neq \text{strand} \circ (O, c, q)) \wedge (d_1 = d_2))]$$

Theorem 5.20. $\forall q_1, q_2, q_3 \in \mathbb{B}, (q_1 \succcurlyeq_O q_2) \wedge (q_2 \succcurlyeq_O q_3) \rightarrow (q_1 \succcurlyeq_O q_3)$

Proof: See appendix A, page 287.

Now, sufficient concepts have been introduced to explain how the existence of a positive path between two BPQs can be computed solely on the basis of the respective labels between these BPQs. A positive path between two BPQs q_1 and q_2 (i.e. path$+$ (O,q_1,q_2)), and hence a negative path path$-(O,q_2,q_1)$ exists if and only if q_1 is ranked higher, with respect to the ordering (O), than BPQ q_2. This is formalised in the following theorem.

Theorem 5.21. $\forall q_1,q_2 \in \mathbb{B}, \text{path}+(O,q_2,q_1) \leftrightarrow q_1 \succcurlyeq_O q_2$

Proof: See appendix A, page 288.

As mentioned earlier, the BPQs are taken from an order of magnitude preference scale $\langle \mathbb{B}, O_<, O_{\ll} \rangle$. As such, two sets of ordering relations $O_<$ and $O_{\ll}$ are defined over these BPQs and, hence, two labels $L_{(\mathbb{B},O_<)}(q)$ and $L_{(\mathbb{B},O_{\ll})}(q)$ can be computed for each BPQ q.

As mentioned earlier, the relations $O_<$ and $O_{\ll}$ order BPQs at a different level of granularity. That is, the relation $O_{\ll}$ defines a coarse ordering of BPQs at different orders of magnitude whereas the relation $O_<$ establishes an order amongst different BPQs within the same order of magnitude. It can be shown that all BPQs within the same order of magnitude have the same label:

Theorem 5.22. Given two BPQs $q_1,q_2 \in \mathbb{B}$ such that $q_1 \sim q_2$, $L_{(\mathbb{B},O_{\ll})}(q_1) = L_{(\mathbb{B},O_{\ll})}(q_2)$.

Proof: See appendix A, page 290.

As a consequence, the following theorems applies:

Theorem 5.23. In each order of magnitude preference scale $\langle \mathbb{B}, O_<, O_{\ll} \rangle$, two BPQs within the same order of magnitude must have the same $O_{\ll}$ labels:

$$\forall q,q' \in \mathbb{B}, (q,q') \in O_< \rightarrow L_{(\mathbb{B},O_{\ll})}(q) = L_{(\mathbb{B},O_{\ll})}(q')$$

Proof: Follows from (5.5) and theorem 5.22.

Similarly, BPQs with a different label with respect to $O_{\ll}$ can not normally be ordered with respect to $O_<$ because $O_<$ relates BPQs of a higher/lower preference, not those of a higher/lower preference magnitude, to one another. Therefore, comparing two BPQs by means of the labels of BPQs can be implemented by first comparing

$L_{(\mathbb{B},O_{\ll})}$ for both BPQs. If one is not of a higher preference magnitude than the other, then the labels $L_{(\mathbb{B},O_{<})}$ must be compared. The next subsection shows how these principles generalise to comparing order-of-magnitude preferences that consist of multiple BPQs.

5.3.2 Combining BPQs

Order-of-magnitude preferences (OMPs) are combinations (by $\oplus$) of BPQs defined on a given OMP scale. OMPs can be compared by extending the concepts introduced in section 5.3.1 from individual BPQs to combinations of multiple BPQs. In 5.3.2.1, the extension of these concepts is introduced. Then, section 5.3.2.2 presents an algorithm to perform the actual comparison. The use of this algorithm is demonstrated by means of an illustrative example in 5.3.2.3.

5.3.2.1 Foundations

Similar to the comparison of BPQs, OMPs can be computed by means of their labels. The labels of OMPs are computed by adding the numbers of unique strand distance combinations. This idea is formalised in definition 5.24.

Definition 5.24. Given two OMPs P_1 and P_2 with labels $L_{(\mathbb{B},O)}(P_1) = \{\ldots, \langle s_i, d_i, x_i\rangle, \ldots\}$ and $L_{(\mathbb{B},O)}(P_2) = \{\ldots, \langle s_j, d_j, x_j\rangle, \ldots\}$ respectively, the label of the OMP $P_1 \oplus P_2$ equals:

$$\begin{aligned} L_{(\mathbb{B},O)}(P_1 \oplus P_2) = &\{\langle s,d,(x_i+x_j)\rangle \mid \langle s,d,x_i\rangle \in L_{(\mathbb{B},O)}(P_1) \wedge \langle s,d,x_j\rangle \in L_{(\mathbb{B},O)}(P_2)\} \\ &\cup \{\langle s,d,x_i\rangle \mid \langle s,d,x_i\rangle \in L_{(\mathbb{B},O)}(P_1) \wedge \langle s,d,x_j\rangle \notin L_{(\mathbb{B},O)}(P_2)\} \\ &\cup \{\langle s,d,x_j\rangle \mid \langle s,d,x_i\rangle \notin L_{(\mathbb{B},O)}(P_1) \wedge \langle s,d,x_j\rangle \in L_{(\mathbb{B},O)}(P_2)\} \end{aligned}$$

Theorem 5.25. The combination of labels of an OMP is unique to that OMP.

Proof: See appendix A, page 290.

Theorem 5.26. Given an OMP P and an ordering of BPQs O,

$$\forall \langle s,d,x\rangle \in L_{(\mathbb{B},O)}(P), x = \sum_{p,\langle s,d,1\rangle \in L_{(\mathbb{B},O)}(p)} f_P(p)$$

Proof: See appendix A, page 290.

As with BPQs, two labels are computed for an order of magnitude preference P, namely $L_{(\mathbb{B},O_{\ll})}(P)$ and $L_{(\mathbb{B},O_{<})}(P)$. However, both labels will contain strand (s), distance (d), BPQ (x) triplets in which the different combinations of strand and distance stem from different BPQs.

In this respect, the algorithm, presented formally in 5.3.2.2, to decide the ordering between two orders of preference magnitude via their respective pairs of labels can be generalised (a more formal discussion follows). Because distinctions at higher orders of magnitude are more significant than distinctions at lower orders of magnitude, the triplets with the highest strand distance combinations in the $O_{\ll}$ labels are compared first. When these are equivalent, the comparison proceeds at a lower granularity by comparing the triplets of the same BPQs in the $O_{<}$ label. When both are equal, the comparison continues at the next (lower) orders of magnitude, and hence, this sequence of two comparisons is repeated for the triplets of the next highest strand distance combinations in the $O_{\ll}$.

The triplets with "the highest strand distance combinations" are the ones for which no other triplets exist with a strand distance combination that is ranked higher. More formally:

Definition 5.27. Given an OMP P, the set of triplets of $L \in L_{(\mathbb{B},O)}(P)$ with the highest strand distance pairs (denoted $\text{maxSD}(L)$) is

$$\text{maxSD}(L_{(\mathbb{B},O)}) = \{\langle s,d,x\rangle \,|\langle s,d,x\rangle \in L_{(\mathbb{B},O)}, \\ \neg(\exists\langle s',d',x'\rangle \in L_{(\mathbb{B},O)}, ((s',d') \neq (s,d)) \wedge ((s',d') \succ_O (s,d)))\}$$

As noted in 5.3.1, ordering $O_{<}$ can not normally distinguish between BPQs that are related via $O_{\ll}$. Therefore, it is necessary to determine what subset of $L_{(\mathbb{B},O_{<})}(P)$ provides a more detailed ordering information for two OMPs for a given subset of $L_{(\mathbb{B},O_{\ll})}(P)$. Once an OMP scale and the corresponding $O_{\ll}$ and $O_{<}$ labels have been defined, tuples in the $O_{\ll}$ labels that correspond to a single BPQ $q \in \mathbb{B}$ are linked to the $O_{<}$ label of q. From this, the specific $O_{<}$ labels can be computed according to the following:

Definition 5.28. Given an OMP scale $\langle \mathbb{B}, O_{<}, O_{\ll}\rangle$ and an OMP P defined over this scale, a triplet $\langle s_i,d_i,x_i\rangle \in L_{(\mathbb{B},O_{<})}(P)$ is said to be *specific* to a triplet $\langle s_j,d_j,x_j\rangle \in$

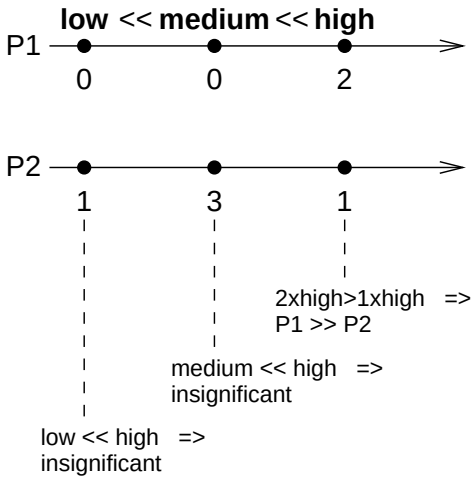


Figure 5.6: Comparing two OMPs with BPQs from an $O_{\ll}$ ordering

$L_{(\mathbb{B},O_{\ll})}(P)$ (denoted $\langle s_j,d_j,x_j\rangle \rightarrow \langle s_i,d_i,x_i\rangle$) if $\exists! q \in \mathbb{B}, \langle s_i,d_i,x_i\rangle \in L_{(\mathbb{B},O_<)}(q) \wedge \langle s_j,d_j,x_j\rangle \in L_{(\mathbb{B},O_{\ll})}(q)$.

The *specific* $O_<$ *label* of a subset S of the $O_{\ll}$ label of P (denoted $L_{S\rightarrow(\mathbb{B},O_<)}(P)$) is defined as:

$$\{\langle s_i,d_i,x_i\rangle \mid \langle s_i,d_i,x_i\rangle \in L_{(\mathbb{B},O_<)}(P) \wedge (\exists \langle s_j,d_j,x_j\rangle \in S, \langle s_j,d_j,x_j\rangle \rightarrow \langle s_i,d_i,x_i\rangle)\}$$

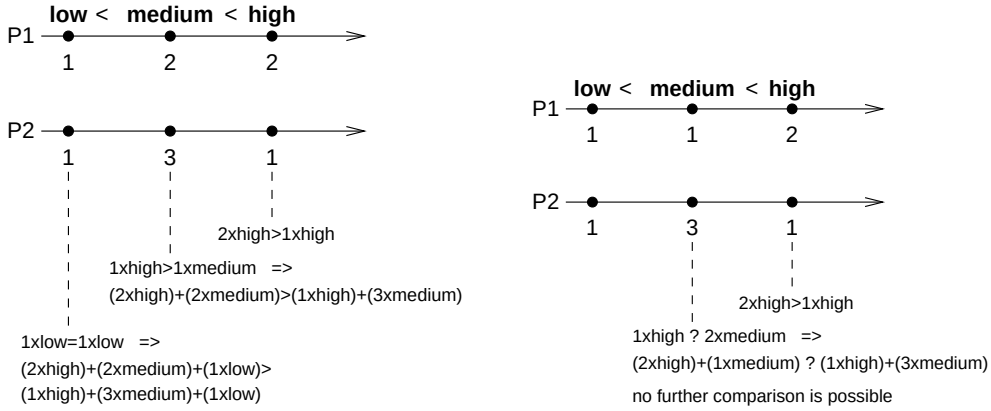
Because of (5.7), the $L_{(\mathbb{B},O_{\ll})}$ and $L_{(\mathbb{B},O_<)}$ labels are compared in different ways. For the $L_{(\mathbb{B},O_{\ll})}$ labels, distinctions at higher orders of magnitude dominate the result and make distinctions at lower orders of magnitude irrelevant. An example will illustrate this point.

Figure 5.6 shows a pair of OMPs. Each OMP is the combination of BPQs "low", "medium" and "high" that are ordered by $O_{\ll}$: P1 $= 2 \times$ high (meaning P1 $=$ high $\oplus$ high) and P2 $= 2 \times$ high $\oplus 3 \times$ medium $\oplus 1 \times$ low. Because of (5.7), $3 \times$ medium $\oplus 1 \times$ low = medium $\oplus$ medium $\oplus$ medium $\oplus$ low $\ll$ high. Therefore, P1 $\succ$ P2, as shown in the figure.

Definition 5.29. Given two OMPs P_1 and P_2 and two subsets of their labels $L_1 \subset L_{(\mathbb{B},O_{\ll})}(P_1)$ and $L_2 \subset L_{(\mathbb{B},O_{\ll})}(P_2)$, such that

$$\forall \langle s_1,d_1,x_1\rangle \in L_1, \exists \langle s_2,d_2,x_2\rangle \in L_2, (s_2,d_2) \succ_{O_{\ll}} (s_1,d_1)$$

then L_2 is said to be of *higher order of magnitude preference* than L_1. This feature is



(a) greater than (b) incomparable

Figure 5.7: Comparing two OMPs with BPQs from an $O_<$ ordering

denoted $L_2 \succ\succ L_1$.

Theorem 5.30.

$$\text{maxDS}(L_{(\mathbb{B},O_\ll)}(P_1)) \prec\prec \text{maxDS}(L_{(\mathbb{B},O_\ll)}(P_2)) \leftrightarrow L_{(\mathbb{B},O_\ll)}(P_1) \prec\prec L_{(\mathbb{B},O_\ll)}(P_2) \tag{5.16}$$

Proof: See appendix A, page 291.

Theorem 5.31. Given to OMPs P_1 and P_2 such that $L_{(\mathbb{B},O_\ll)}(P_1) \prec\prec L_{(\mathbb{B},O_\ll)}(P_2)$, $P_1 \prec P_2$.

Proof: See appendix A, page 291.

Theorem 5.32. Given two OMPs P_1 and P_2 such that $\forall p_1 \in P_1, f_{P_1}(p_1) > 0, \exists p_2 \in P_2, f_{P_2}(p_2) > 0, p_1 \ll p_2, L_{(\mathbb{B},O_\ll)}(P_1)$ and $L_{(\mathbb{B},O_\ll)}(P_2)$ can each be partitioned into two sets L_{i1} and L_{i2} such that $L_{11} \prec\prec L_{21}$, $L_{L_{12}\rightarrow(\mathbb{B},O_<)}(P_1) = \emptyset$ and $L_{L_{22}\rightarrow(\mathbb{B},O_<)}(P_2) = \emptyset$.

Proof: See appendix A, page 291.

When comparing two OMPs that consist of BPQs of the same order of magnitude, the BPQs that are are ranked lower with respect to $O_<$ and that are equivalent with respect to $O_\ll$ can not be ignored. As an example, figure 5.7 shows two pairs of OMPs that consist of BPQs within the same order of magnitude. Both OMPs are a combination of BPQs “low”, “medium” and “high” that are ordered by $O_<$ (that is, the

BPQs are within the same order of magnitude). The OMPs in figure 5.7(a) are P1 $=$ $2\times\text{high}\oplus 2\times\text{medium}\oplus 1\times\text{low}$ and P2 $=2\times\text{high}\oplus 3\times\text{medium}\oplus 1\times\text{low}$. When comparing the number of "highs", P1>P2 because 2 highs are preferred over 1 high. When comparing the number of mediums, 3 mediums might be preferred of 2 (and hence P2>P1), but the extra medium of P2 can be compensated by the extra high of P1. As both OMPs have an equal number of "lows", P1 must be deemed of higher preference than P2. The OMPs in figure 5.7(b) are P1 $=2\times\text{high}\oplus 1\times\text{medium}\oplus 1\times\text{low}$ and P2 $=2\times\text{high}\oplus 3\times\text{medium}\oplus 1\times\text{low}$. Here, a similar situation occurs. However, a decision for this case could only be made if 1 high could be compared with 2 mediums. Such information is not available, and therefore P1 is deemed incomparable to P2.

Definition 5.33. Given two OMPs P_1 and P_2 and two subsets of their labels $L_1\subset L_{(\mathbb{B},O_<)}(P_1)$ and $L_2\subset L_{(\mathbb{B},O_<)}(P_2)$, such that

$$\forall\langle s,d,x_1\rangle\in L_1,\quad \sum_{\substack{\langle s_i,d_i,x_i\rangle\in L_1,[((s_i,d_i)\succ_{O_<}(s,d))\vee\\ (s_i=s\neq\text{strand}\circ(O_<,q,c)\wedge d_i=d)]}} x_i \leq \sum_{\substack{\langle s_i,d_i,x_i\rangle\in L_2,[((s_i,d_i)\succ_{O_<}(s,d))\vee\\ (s_i=s\neq\text{strand}\circ(O_<,q,c)\wedge d_i=d)]}} x_i$$

then, the label L_2 is said to be *ranked higher or at an equal level as* than label L_1. This feature is denoted by $L_2\succcurlyeq L_1$.

It is shown by theorems 5.34 and 5.35 that this ranking of labels of two preferences is equivalent to ordering preferences within an order of magnitude.

Theorem 5.34. Given two OMPs $P_1=p_{11}\oplus\ldots\oplus p_{1n_1}$ and $P_2=p_{21}\oplus\ldots\oplus p_{2n_2}$ such that $p\sim p_{11}\sim\ldots\sim p_{1n_1}\sim p_{21}\sim\ldots\sim p_{2n_2}$,

$$P_1\preccurlyeq_p P_2\leftrightarrow L_{(\mathbb{B},O_<)}(P_1)\preccurlyeq L_{(\mathbb{B},O_<)}(P_2)$$

Proof: See appendix A, page 292.

Theorem 5.35. Given two OMPs $P_1=p_{11}\oplus\ldots\oplus p_{1n_1}$ and $P_2=p_{21}\oplus\ldots\oplus p_{2n_2}$ such that no pair of BPQs p_{ij} and p_{kl} can be taken from $\{p_{11},\ldots,p_{1n_1},p_{21},\ldots,p_{2n_2}\}$ such that $p_{ij}\ll p_{kl}$,

$$P_1\preccurlyeq_p P_2\leftrightarrow L_{(\mathbb{B},O_<)}(P_1)\preccurlyeq L_{(\mathbb{B},O_<)}(P_2)$$

Proof: See appendix A, page 294.

5.3.2.2 Algorithm

Having established concise notations for the important concepts to be involved, a formal procedure to compare two OMPs P_1 and P_2 that are defined by a combination of BPQs on an OMP scale $\langle \mathbb{B}, O_<, O_\ll \rangle$ can be described, as given in algorithm 5.1 (which calls for algorithms 5.2, 5.3 and 5.4).

COMPARE-OMP$(\mathbb{B}, O_\ll, O_<, P_1, P_2)$ explores the $O_\ll$ labels of P_1 and P_2 via two "open" sets O_1 and O_2 that initially contain the entire contents of the $O_\ll$ labels. The main **while** -loop of this algorithm iterates through different sets of orders of magnitude. That is, at the beginning of each iteration, the tuples with the highest strand distance pairs are grouped in two sets $O_{(1,\max)}$ and $O_{(2,\max)}$, and at the end, the contents of $O_{(1,\max)}$ and $O_{(2,\max)}$ are removed from O_1 and O_2. In this way, all highest strands distance pairs are explored in parallel. The tuples in $O_{(1,\max)}$ and $O_{(2,\max)}$ are both grouped in the following four different types of set:

- Tuples from $O_{(i,\max)}$ with strand distance pairs that are higher than any strand distance pair in $O_{(j,\max)}$, with $i \neq j$, are grouped in set type $H_{(i,\gg)}$.
- Tuples from $O_{(i,\max)}$ with strand distance pairs that are smaller than a strand distance pair in $O_{(j,\max)}$, with $i \neq j$, are grouped in set type $H_{(i,\ll)}$.
- Tuples with strand distance pairs that appear in both $O_{(1,\max)}$ and $O_{(2,\max)}$ are grouped in a set $H_\sim$.
- The remaining tuples are those tuples that can not be compared to tuples from the $O_\ll$ label of the other OMP, and they are respectively stored in sets $H_{(1,\text{rest})}$ and $H_{(2,\text{rest})}$.

The tuples in these four types of set are compared as follows:

- The sets $H_{(1,\gg)}$ and $H_{(2,\gg)}$ are analysed by calling

$$\text{COMPARE-OM}(\mathbb{B}, O_\ll, O_<, P_1, P_2, H_{(1,\gg)}, H_{(2,\gg)}).$$

If both sets are empty, no result can be committed to. If one is empty and the other is not, then smaller than (if $H_{(1,\gg)} = \emptyset$) or greater-than (if $H_{(2,\gg)} = \emptyset$) is returned. If both are non-empty, the two OMPs are incomparable because neither is smaller than or equal to the other.

- The tuples in the sets $H_{(1,\ll)}$ and $H_{(2,\ll)}$ are on the other end of the comparisons that yielded $H_{(1,\gg)}$ and $H_{(2,\gg)}$, and hence, they are ignored.

- For all the tuples in $H_{\sim}$, the $O_{<}$ labels are compared by calling

$$\text{COMPARE-WM}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2, H_{\sim}, H_{(1,\ll)}, H_{(2,\ll)}).$$

  This procedure performs the comparison of the dependent $O_{<}$ labels for each tuple from $H_{\sim}$ individually. Each tuple for which this comparison yields a smaller-than result, is added to $H_{(2,\gg)}$. Each tuple for which this comparison yields a greater-than result, is added to $H_{(1,\gg)}$. After this step, $H_{(1,\gg)}$ and $H_{(2,\gg)}$ contain tuples with strand distance pairs that describe an order of magnitude at which the component BPQs of one OMP are found to be greater than or smaller than those of the other OMP. Strand distance pairs lower than these need not be considered any more because comparisons BPQs on strand distance pairs at lower orders of magnitude can not change the outcome of the overall comparison.

- The tuples in $H_{(1,\text{rest})}$ and $H_{(2,\text{rest})}$ are analysed by means of the procedure

$$\text{COMPARE-REST}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2, H_1, H_2).$$

  If for a given tuple $\langle \text{strand}(O_{\ll}, q, q'), d, x \rangle$ in the label of one OMP, there is a tuple in the label of the other OMP that contains an extended strand strand $\circ$ $(O_{\ll}, q, q')$, then, by definition 5.12, there exists another cross-over quantity from which the ordering between the corresponding BPQs can be made more precisely. Otherwise, the OMP for which $H_{(i,\text{rest})}$ is not empty should be deemed greater than the other. If both are not empty, then the only possible result is incomparable.

	equal-to	smaller-than	greater-than	incomparable
equal-to	equal-to	smaller-than	greater-than	incomparable
smaller-than	smaller-than	smaller-than	incomparable	incomparable
greater-than	greater-than	incomparable	greater-than	incomparable
incomparable	incomparable	incomparable	incomparable	incomparable

Table 5.1: Combining partial comparisons

Algorithm 5.1: COMPARE-OMP($\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2$)

$L_{(1,O_<)} \leftarrow L_{(\mathbb{B},O_<)}(P_1); L_{(1,O_\ll)} \leftarrow L_{(\mathbb{B},O_\ll)}(P_1);$
$L_{(2,O_<)} \leftarrow L_{(\mathbb{B},O_<)}(P_2); L_{(2,O_\ll)} \leftarrow L_{(\mathbb{B},O_\ll)}(P_2);$
result $\leftarrow$ equal-to;
$O_1 \leftarrow L_{(1,O_\ll)}; O_2 \leftarrow L_{(2,O_\ll)};$
while $(H_{(1,O_\ll)} \neq \emptyset \wedge H_{(2,O_\ll)} \neq \emptyset) \vee$ (result $\neq$ equal-to)
do
$O_{(1,\max)} \leftarrow \max\mathrm{SD}(O_1); O_{(2,\max)} \leftarrow \max\mathrm{SD}(O_2);$
$H_{(1,\gg)} \leftarrow \{\langle s_1,d_1,x_1\rangle \in O_{(1,\max)} \mid \nexists \langle s_2,d_2,x_2\rangle \in O_{(2,\max)}, (s_1,d_1) \preccurlyeq_{O_\ll} (s_2,d_2)\};$
$H_{(2,\gg)} \leftarrow \{\langle s_2,d_2,x_2\rangle \in O_{(2,\max)} \mid \nexists \langle s_1,d_1,x_1\rangle \in O_{(1,\max)}, (s_2,d_2) \preccurlyeq_{O_\ll} (s_1,d_1)\};$
$H_\sim \leftarrow \{\langle s,d,x_1\rangle \in O_{(1,\max)} \mid \exists \langle s,d,x_2\rangle \in O_{(1,\max)}\};$
$H_{(1,\mathrm{rest})} \leftarrow \begin{cases} O_{(1,\max)} - H_{(1,\gg)} - H_\sim - \\ \{\langle s_1,d_1,x_1\rangle \in O_{(1,\max)} \mid \exists \langle s_2,d_2,x_2\rangle \in O_{(2,\max)}, (s_1,d_1) \prec_{O_\ll} (s_2,d_2)\}; \end{cases}$
$H_{(2,\mathrm{rest})} \leftarrow \begin{cases} O_{(2,\max)} - H_{(2,\gg)} - H_\sim - \\ \{\langle s_1,d_1,x_1\rangle \in O_{(1,\max)} \mid \exists \langle s_2,d_2,x_2\rangle \in O_{(2,\max)}, (s_1,d_1) \prec_{O_\ll} (s_2,d_2)\}; \end{cases}$
om-result $\leftarrow$ COMPARE-OM($\mathbb{B}, O_\ll, O_<, P_1, P_2, H_{(1,\gg)}, H_{(2,\gg)}$);
wm-result $\leftarrow$ COMPARE-WM($\mathbb{B}, O_\ll, O_<, P_1, P_2, H_\sim, H_{(1,\gg)}, H_{(2,\gg)}$);
rest-result $\leftarrow$ COMPARE-REST($\mathbb{B}, O_\ll, O_<, P_1, P_2, H_{(1,\mathrm{rest})}, H_{(2,\mathrm{rest})}$);
result $\leftarrow$ result $\wedge$ om-result $\wedge$ wm-result $\wedge$ rest-result;
if result = greater-than
then
for each $\langle s,d,x\rangle \in H_{(1,\gg)}$
do
$O_1 \leftarrow O_1 - \{\langle s_1,d_1,x_1\rangle \in O_1 \mid (s_1,d_1) \prec_{O_\ll} (s,d)\};$
$O_2 \leftarrow O_2 - \{\langle s_2,d_2,x_2\rangle \in O_2 \mid (s_2,d_2) \prec_{O_\ll} (s,d)\};$
if result = smaller-than
then
for each $\langle s,d,x\rangle \in H_{(2,\gg)}$
do
$O_1 \leftarrow O_1 - \{\langle s_1,d_1,x_1\rangle \in O_1 \mid (s_1,d_1) \prec_{O_\ll} (s,d)\};$
$O_2 \leftarrow O_2 - \{\langle s_2,d_2,x_2\rangle \in O_2 \mid (s_2,d_2) \prec_{O_\ll} (s,d)\};$
$O_1 \leftarrow O_1 - O_{(1,\max)}; O_2 \leftarrow O_2 - O_{(2,\max)};$
return (result).

Throughout algorithm 5.1, the results of partial comparisons based on subsets of constituent BPQs of the OMPs are combined with one another. Table 5.1 summarises the way in which outcomes of partial comparisons can be combined with one another. That is, in table 5.1, the top row and the leftmost column each list the set of four possible outcomes of a (partial) comparison and the other cells each indicate the combination of the outcomes of the corresponding row and column.

Finally, whenever for some strand distance pair (s,d) in the $O_{\ll}$ labels, the BPQs of one OMP are found to be greater than the BPQs of the other, then all strand distance pairs that are smaller than (s,d) are no longer relevant to the comparison. Therefore, at the end of each iteration, those strand distance pairs are removed from O_1 and O_2 together with the currently analysed subsets of O_1 and O_2, namely $O_{(1,\max)}$ and $O_{(2,\max)}$.

Procedure 5.2: COMPARE-OM($\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2, H_1, H_2$)

---

if $(H_1 \neq \emptyset)$
- **then**
  - **if** $(H_2 \neq \emptyset)$
    - **then** result $\leftarrow$ incomparable;
    - **else** result $\leftarrow$ result $\wedge$ greater-than;
- **else**
  - **if** $(H_2 \neq \emptyset)$
    - **then** result $\leftarrow$ result $\wedge$ smaller-than;
    - **else** result $\leftarrow$ result $\wedge$ equal-to;

Theorem 5.36.

$$P_1 = P_2 \leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{equal-to}$$
$$P_1 \prec P_2 \leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{smaller-than}$$
$$P_1 \succ P_2 \leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{greater-than}$$
$$P_1 ? P_2 \leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{incomparable}$$

Proof: See appendix A, page 294

Procedure 5.3: COMPARE-WM($\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2, H_{\sim}, H_{(1,\gg)}, H_{(2,\gg)}$)

result $\leftarrow$ equal-to;
for each $\langle s,d,x\rangle \in H_{\sim}$
do {
$L_1 \leftarrow L_{\{\langle s,d,x\rangle\}\rightarrow(\mathbb{B},O_{<})}(P_1)$;
$L_2 \leftarrow L_{\{\langle s,d,x\rangle\}\rightarrow(\mathbb{B},O_{<})}(P_2)$;
if $L_1 \prec L_2$
then {result $\leftarrow$ result $\wedge$ smaller-than; $H_{(2,\gg)} \leftarrow H_{(2,\gg)} \cup \{\langle s,d,x\rangle\}$;
else if $L_1 \succ L_2$
then {result $\leftarrow$ result $\wedge$ greater-than; $H_{(1,\gg)} \leftarrow H_{(1,\gg)} \cup \{\langle s,d,x\rangle\}$;
else if $(L_1 \neq L_2)$
then return (incomparable);
return (result).

5.3.2.3 Example

Algorithm 5.1 can be illustrated by means of the sample OMP scale given in figure 5.8. The partial ordering defined therein can be formalised as follows:

$$\begin{aligned}
\text{very low} &\ll \text{low} & \text{small} &\ll \text{medium}\\
\text{low} &\ll \text{medium} & \text{medium} &\ll \text{large}\\
\text{low medium} &< \text{medium} & \text{large} &\ll \text{extra large}\\
\text{medium} &< \text{high medium} & \text{small} &\ll \text{below average}\\
\text{medium} &\ll \text{high} & \text{below average} &< \text{average}\\
\text{high} &\ll \text{very high} & \text{average} &< \text{above average}\\
\text{extra small} &\ll \text{small} & \text{above average} &\ll \text{big}
\end{aligned}$$

Figure 5.8 does not make it clear as to how the individual ordering relations $O_{<}$ and $O_{\ll}$ work, but this is reflected in figure 5.9. With respect to $O_{\ll}$ no distinction can be made between the different kinds of medium (low medium, medium and high medium) nor between the different kinds of average (below average, average and above average). This ordering relation also specifies two obvious cross-over quantities: medium and small. With respect to $O_{<}$, separate orderings must be considered for the "medium" BPQs, the "average" BPQs and all other BPQs individually. In order to enable the computation of the corresponding $O_{<}$ labels, a cross-over quantity has to be defined

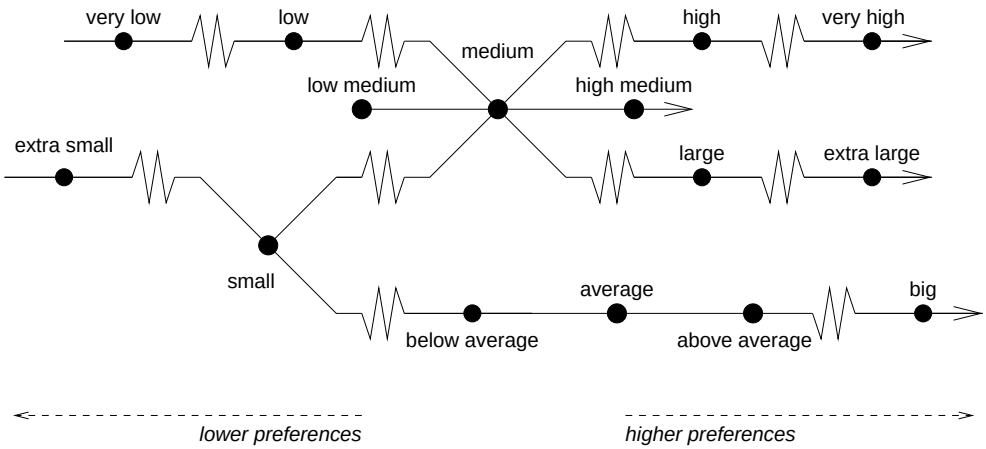
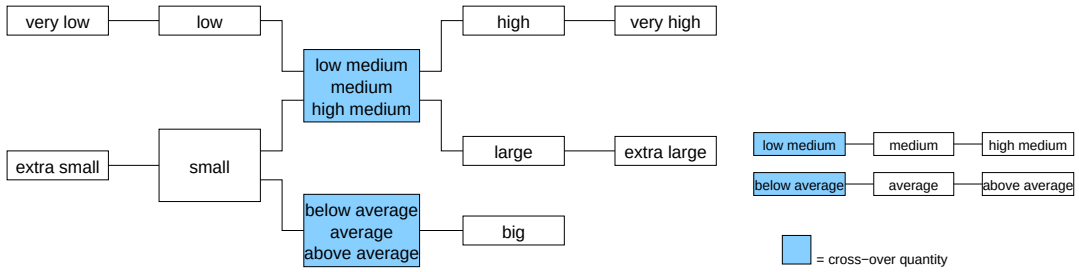


Figure 5.8: A sample OMP scale defining a partial ordering of BPQs



(a) $O_{\ll}$ ordering (b) $O_{<}$ ordering

Figure 5.9: Example BPQ orderings defined by an OMP scale

Procedure 5.4: COMPARE-REST($\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2, H_1, H_2$)

$r_{12} \leftarrow \forall\langle \text{strand}(O_{\ll}, q, q'), d_1, x_1\rangle \in H_1, (\exists \text{strand} \circ (O_{\ll}, q, q'), d_2, x_2\rangle \in L_{(\mathbb{B}, O_{\ll})}(P_2))$;
$r_{21} \leftarrow \forall\langle \text{strand}(O_{\ll}, q, q'), d_2, x_2\rangle \in H_2, (\exists \text{strand} \circ (O_{\ll}, q, q'), d_1, x_1\rangle \in L_{(\mathbb{B}, O_{\ll})}(P_1))$;
if $r_{12} \wedge r_{21}$
  then return (equal-to);
if $r_{12} \wedge \neg r_{21}$
  then return (greater-than);
if $\neg r_{12} \wedge r_{21}$
  then return (smaller-than);
if $\neg r_{12} \wedge \neg r_{21}$
  then return (incomparable);

within each of these orderings: one medium, one average and all other BPQs are cross-over BPQs with respect to. Isolated cross-over quantities will not be considered herein since they do not affect the result. In addition, it is assumed that, with respect to $O_{<}$, low medium and below average are arbitrarily appointed to act as cross-over quantities.

Suppose that it is required to compare the following two preferences defined over the aforementioned OMP scale:

$$P_1 = \text{very high} \oplus \text{extra large} \oplus \text{medium} \oplus \text{low medium} \oplus \text{low} \oplus \text{extra small}$$
$$P_2 = \text{very high} \oplus \text{extra large} \oplus \text{high medium} \oplus \text{medium} \oplus \text{very low}$$

The labels of these OMPs with respect to $O_{\ll}$ are:

$$\begin{aligned} L_{(\mathbb{B}, O_{\ll})}(P_1) = \{&\langle \text{strand}(O_{\ll}, \text{medium}, \text{high}), 2, 1\rangle, \\ &\langle \text{strand}(O_{\ll}, \text{medium}, \text{large}), 2, 1\rangle, \\ &\langle \text{strand}(O_{\ll}, \text{medium}, \text{medium}), 0, 2\rangle, \\ &\langle \text{strand}(O_{\ll}, \text{medium}, \text{small}), -1, 1\rangle, \\ &\langle \text{strand}(O_{\ll}, \text{small}, \text{medium}), 3, 2\rangle, \\ &\langle \text{strand}(O_{\ll}, \text{small}, \text{medium}), 1, 2\rangle, \\ &\langle \text{strand}(O_{\ll}, \text{small}, \text{small}), 0, 1\rangle\} \end{aligned}$$

$$
\begin{aligned}
L_{(\mathbb{B},O_{\ll})}(P_2) = \{&\langle \text{strand}(O_{\ll}, \text{medium}, \text{high}), 2, 1\rangle,\\
&\langle \text{strand}(O_{\ll}, \text{medium}, \text{large}), 2, 1\rangle,\\
&\langle \text{strand}(O_{\ll}, \text{medium}, \text{medium}), 0, 2\rangle,\\
&\langle \text{strand}(O_{\ll}, \text{medium}, \text{low}), -2, 1\rangle,\\
&\langle \text{strand}(O_{\ll}, \text{small}, \text{medium}), 3, 2\rangle,\\
&\langle \text{strand}(O_{\ll}, \text{small}, \text{medium}), 1, 2\rangle,\\
&\langle \text{strand} \circ (O_{\ll}, \text{small}, \text{medium}), 1, 2\rangle\}
\end{aligned}
$$

The labels computed with respect to $O_<$ are mostly irrelevant because they involve strands with only one BPQ. However, low medium, medium and high medium have related to one another by $O_<$. The labels relevant to $\text{strand}(O_{\ll}, \text{medium}, \text{medium})$ and distance 0 are:

$$
\begin{aligned}
&L_{\{\langle \text{strand}(O_{\ll},\text{medium},\text{medium}),0,2\rangle\} \rightarrow (\mathbb{B},O_<)}(P_1) =\\
&\{\langle \text{strand}(O_<, \text{low medium}, \text{medium}), 1, 1\rangle, \langle \text{strand}(O_<, \text{low medium}, \text{low medium}), 0, 1\rangle\}\\
&L_{\{\langle \text{strand}(O_{\ll},\text{medium},\text{medium}),0,2\rangle\} \rightarrow (\mathbb{B},O_<)}(P_2) =\\
&\{\langle \text{strand}(O_<, \text{low medium}, \text{medium}), 2, 1\rangle, \langle \text{strand}(O_<, \text{low medium}, \text{medium}), 1, 1\rangle\}
\end{aligned}
$$

To illustrate these labels in a more comprehensible manner, figure 5.10 shows the labels of both preferences in relation to one another, and how algorithm 5.1 is used to compare both preferences.

First, for each cross-over quantity and each of the positive strands with respect to $O_{\ll}$, the highest strand distance pairs are explored, involving (strand($O_{\ll}$,medium,high),2), (strand($O_{\ll}$,medium,large),2) and (strand($O_{\ll}$,small,medium),3). As shown in figure 5.10, no difference can be detected for the highest strand distance pairs. Next, the second highest strand distance pairs are explored, involving (strand($O_{\ll}$,medium,medium),0) and (strand($O_{\ll}$,small,medium),1). Again, there is no difference with respect to $O_{\ll}$ but a within magnitude comparison can be made for $\langle \text{strand}(O_{\ll}, \text{medium}, \text{medium}), 0, 2\rangle$. In figure 5.10, this computation is shown in the shaded rectangle. This computation has been illustrated earlier and results in:

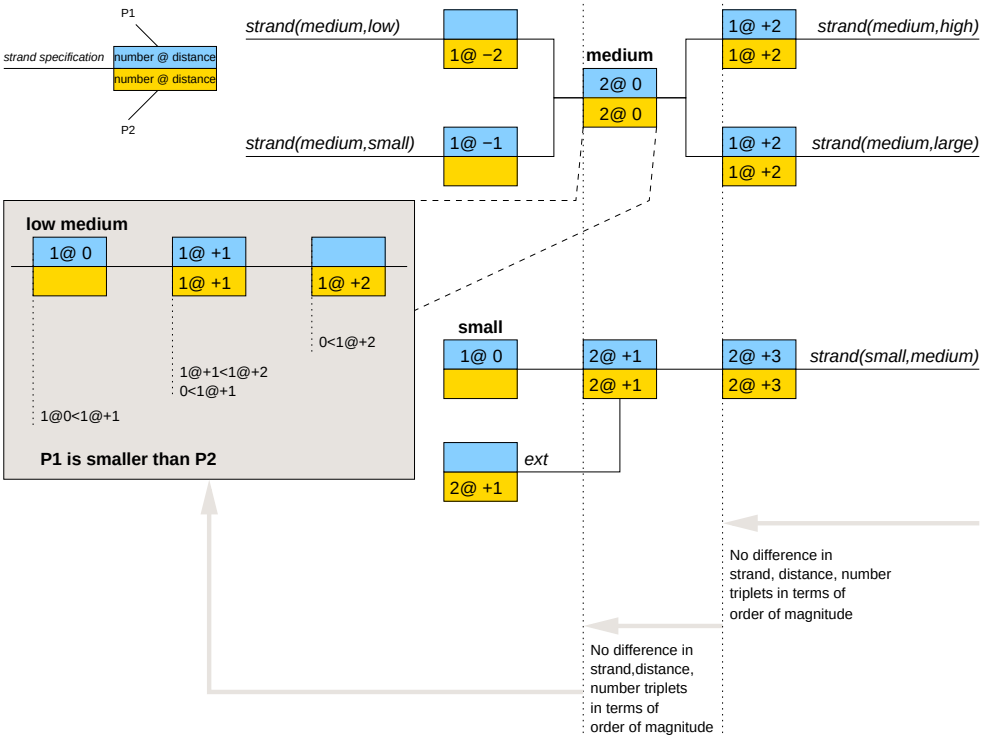


Figure 5.10: Compare two OMPs

$$\langle \text{strand}(O_<, \text{low medium}, \text{medium}), 1, 1\rangle \oplus \langle \text{strand}(O_<, \text{low medium}, \text{low medium}), 0, 1\rangle \prec \langle \text{strand}(O_<, \text{low medium}, \text{medium}), 2, 1\rangle \oplus \langle \text{strand}(O_<, \text{low medium}, \text{medium}), 1, 1\rangle$$

Since there are no other distinctions at this order of magnitude, the computation can end. By definition, any further differences at lower orders of magnitude can not offset the difference at the current order of magnitude. The algorithm therefore concludes that P_1 is smaller than P_2 (or $P_1 \prec P_2$).

5.4 Discussion

Much of this chapter has been devoted to the development of an algorithm for combining and comparing partially ordered preference valuations. The partial ordering of these preference valuations has two important consequences: (1) the comparison algorithm is potentially inefficient and (2) the set of available combination operators is naturally restricted. The efficiency issue has been addressed by the algorithm presented

in the previous section, which is discussed section 5.4.1. Then, section 5.4.2 discusses the restrictions imposed on the combination operators.

5.4.1 Efficiency

Comparing OMPs by means of the algorithm presented in the last section provides significant efficiency gains over searching for paths between the constituent elementary BPQs involved in the OMPs. Assume that, on average, there are q BPQ units per OMP, c cross-over quantities or other BPQs for which strand assignments have been made, and s positive and negative strands per cross-over quantity. An ordinary backtracking search through paths between two orders of magnitude preference would have a worst case time complexity of $O(2^q \times c^s)$.

Because algorithm 5.1 potentially explores all distinct strand distance pairs that can be identified in an OMP scale, the worst case complexity is proportional to the total number of distinct strand distance pairs. It is known that the number distinct strands equals $c \times s$ and it is assumed that d denotes the number of distinct distances per strand. Therefore, the worst case time complexity of algorithm 5.1 is $O(c \times s \times d)$. The particular value of d depends on the configuration of the OMP scale, but it is proportional to the number of distinct BPQs in the OMP scale and inversely proportional to the number of cross-over quantities.

As opposed to a naive algorithm, where the complexity of a comparison depends on the number of BPQs in the OMPs, the complexity of algorithm 5.1 presented herein depends solely on the configuration of the OMP scale. But, generally speaking, as the comparison becomes non-trivial, i.e. as q becomes sufficiently high, it can be concluded that $2^q \times c^s \gg c \times s \times d$.

5.4.2 Semantics

The preference calculus presented in this chapter employs only a single combination operator: the preference calculus equivalent of the addition operator as it is normally defined for the real number line $\mathbb{R}$. The way in which the preference scale is defined affects the types of operation that can be meaningfully implemented. The theory of *scales of measurements* [162, 163] is the most influential in this respect. A summary of the different types of scale and the operations they allow is summarised in tables 5.2

Scale	Description	Example
Nominal	unordered classification	gender
Ordinal	ordered measurements	restaurant ratings
Interval	ordinal + well-defined value differences	temperature
Ratio	interval + rational zero point	distance

Table 5.2: Scales of measurement

Scale	Permissible operations
Nominal	counting
Ordinal	ranking
Interval	ranking, addition and subtraction
Ratio	all arithmetic operations

Table 5.3: Permissible operations in scales of measurement

and 5.3.

By itself, the ordering of coarse preference values defines at least an ordinal scale, albeit one with only a partial ranking. The preference calculus defined herein employs ordering operators that have a well-defined semantics that provides information on the relative distance between preferences. As such, the set of OMPs defined by means of an order of magnitude preference scale form at least an interval scale. However, because there is no total ordering of values within the underlying domain of the OMP calculus, the distance of each preference valuation to the neutral preference 0 (that is $p \oplus 0 = p$) can not be determined. Therefore, addition of preferences is feasible, but multiplication is not.

To make this point clearer, consider three OMPs P_1, P_2 and P_c such that:

$$P_1 \succ P_c \quad \text{and} \quad P_2 \succ P_c$$

To compute the product of P_1 and P_2, a fourth OMP P_0 which corresponds to the absolute zero value would need to be known as well as the distance between the original OMPs and P_0. Such distance information is not available, however. As figure 5.11 shows, information on the distance d_c between P_0 and P_c, the distance d_1 between P_c and P_1 and the distance d_2 between P_c and P_2 would enable the projection of all OMPs to a totally ordered set. However, if it were required that such distance information be

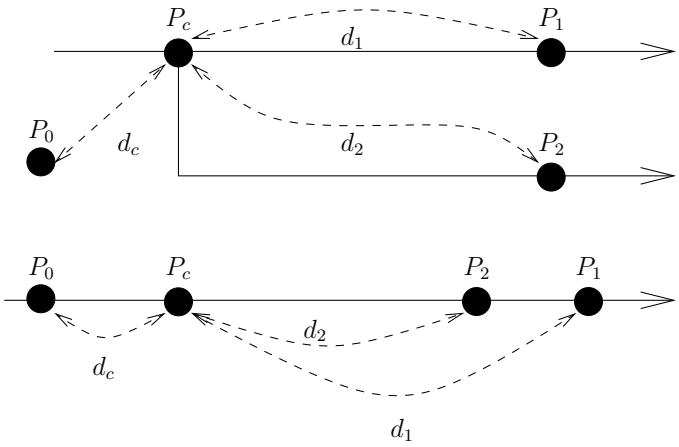


Figure 5.11: OMP calculus and multiplication

available, then the preference calculus would suffer the same disadvantages of a utility calculus as discussed in section 5.2. Therefore, an operation equivalent to multiplication is left out of the preference calculus.

5.5 Summary

In this chapter, a preference calculus has been introduced to express user's preferences for the assumptions that a scenario model will be based on. The preference calculus follows the principles of order of magnitude reasoning. It aims at combining basic preferences on a measurement scale, whilst the preferences themselves are defined with only minimal information to realistically accommodate for limited knowledge sources about such preferences. This has been achieved by building preference from BPQs that have been ordered using relations that have a simple but well-defined semantics. The majority of this chapter has been dedicated to the development of an algorithm that can efficiently compare combined OMPs, as it is important that the preference calculus does not unnecessarily complicate the DPCSP algorithms.

Chapter 6

Solution Techniques for Dynamic Order-of-Magnitude Preference Constraint Satisfaction

As illustrated in figure 6.1, the compositional modelling framework presented in this dissertation relies on solution techniques for DPCSPs. The solution techniques called for to tackle this kind of constraint satisfaction problem require an approach capable of dealing with both dynamicity and preferences. Unfortunately, little exists in the literature in this regard, especially if preferences must be combined in a strictly monotonic manner [118].

More specifically, the problem is particularly complicated because *strictly monotonic* preference combination operators may be used. For example, the preference calculus presented in chapter 5 employs such an operator. Strict monotonicity rules out the use of an *idempotent* preference combination operator ($P \oplus P = P$) as explained in [157]. This prevents the use many existing techniques that depend on idempotency, but it makes the DPCSP more expressive for the problem at hand.

This chapter discusses a number of solution techniques for the dynamic preference constraint satisfaction problems (DPCSPs) introduced earlier. Section 6.1 formally summarises the specification of a DPCSP and lists the assumptions underlying the preference calculus. Note that it is not necessary that the OMP calculus given in chapter 5 be employed, though the OMP calculus is the one employed in this thesis. Section 6.2 presents the most basic search algorithm for solving a DPCSP. Three (po-

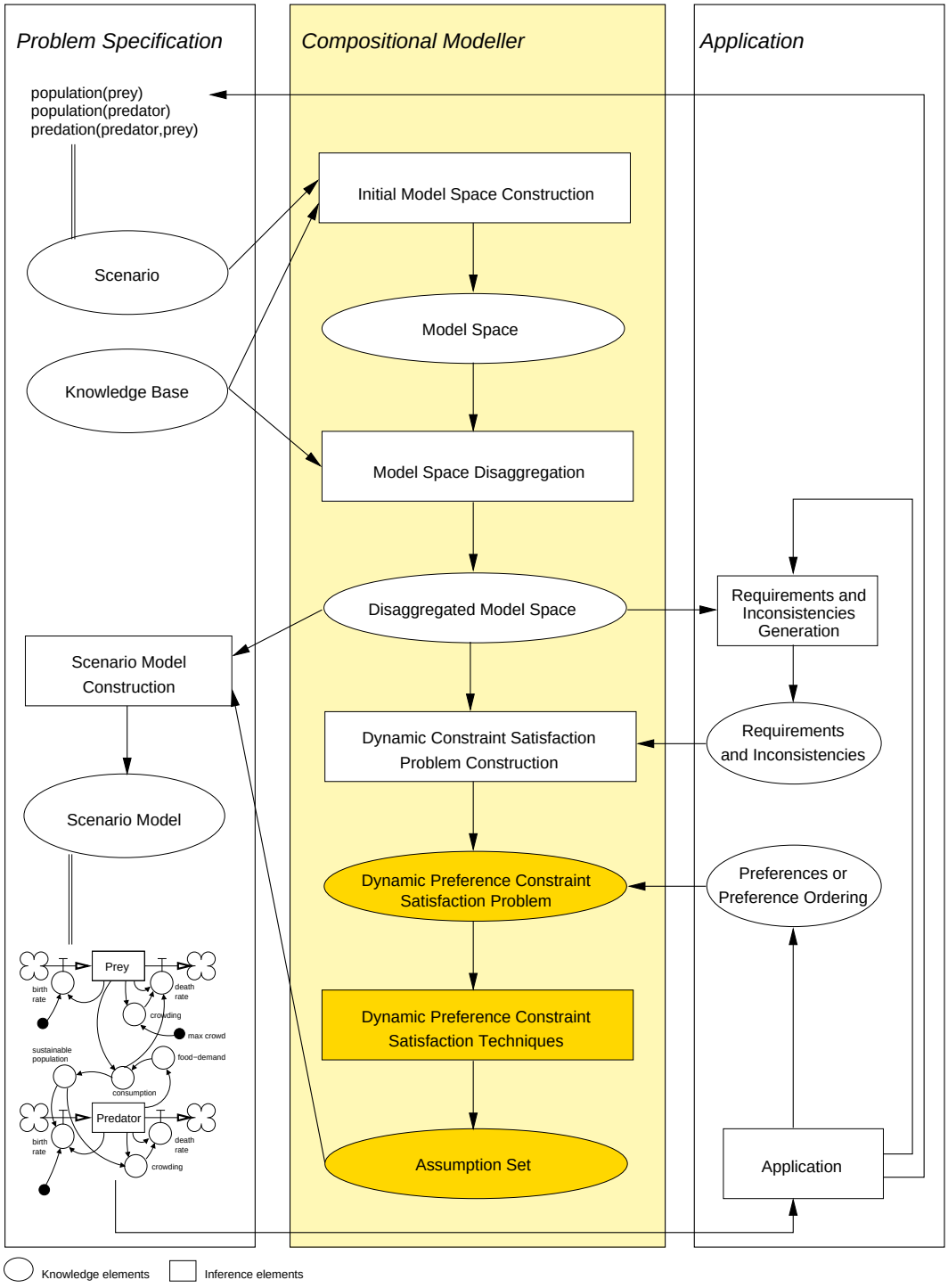


Figure 6.1: The role of DPCSP solution techniques in compositional modelling

tential) improvements on this algorithm are discussed in section 6.3. Finally, section 6.5 summarises this chapter.

6.1 Background

Before presenting the solution algorithms, a summary of the problem specification is shown first. Section 6.1.1 formalises the notion of a DPCSP and section 6.1.2 lists the assumption underlying the preference calculus.

6.1.1 Dynamic preference constraint satisfaction problems

To remind the reader, a CSP is specified by means of

- a set of attributes $\mathbf{X} = \{x_1, \ldots, x_n\}$,
- a set of domains $\mathbf{D} = \{D_1, \ldots, D_n\}$ containing a domain $D_i = \{d_{i1}, \ldots, d_{in_i}\}$ for each attributes,
- a bijection $D : \mathbf{X} \mapsto \mathbf{D}$, which maps each attribute $x \in \mathbf{X}$ to a domain $D(x) \in \mathbf{D}$, and
- a set of compatibility constraints $\mathbf{C} = \{c_{Y_1}, \ldots, c_{Y_v}\}$, where each $Y_k \subset \mathbf{X}$, such that c_{Y_k} defines a function $c_k : D_i \times \ldots \times D_j \rightarrow \{\top, \bot\}$.

A DCSP, as defined in [123], is an extension of a CSP in which attributes can be active and inactive. An attribute x_i is said to be active, denoted active(x_i), if and only if it is assigned a value from its domain. That is,

$$\forall x_i \in \mathbf{X}, (\text{active}(x_i) \leftrightarrow \exists d_{ij} \in D_i, x_i : d_{ij})$$

The activity of attributes is governed by a set of activity constraints $\mathbf{A} = \{\ldots, a_{Y,x_k}, \ldots\}$ that are defined via rules establishing conditions, in terms of attribute value assignments, under which other attributes become active. More generally, each activity constraint $a_l \in \mathbf{A}$ corresponds to a function:

$$\begin{aligned} a_{Y,x_k} :& D_i \times \ldots \times D_j \times \{\text{active}(x_k), \neg\text{active}(x_k)\} \rightarrow \{\top, \bot\} \\ & \text{with } Y = \{x_i, \ldots, x_j\} \end{aligned}$$

The aim of a DCSP is to find a set of assignments $\{x_i : d_{i,k_i}, \ldots, x_j : d_{j,k_j}\}$ such that it is consistent with the compatibility and activity constraints:

$$\forall c_{\{x_p,\ldots,x_q\}} \in \mathbf{C}, \{x_p,\ldots,x_q\} \subset \{x_i,\ldots,x_j\} \rightarrow c_{\{x_p,\ldots,x_q\}}(d_{p,k_p},\ldots,d_{q,k_q}) = \top \quad (6.1)$$

$$\forall x_r \in \mathbf{X}, \forall a_{\{x_p,\ldots,x_q\},x_r} \in \mathbf{A},$$

$$x_r \in \{x_i,\ldots,x_j\} \rightarrow a_{\{x_p,\ldots,x_q\},x_r}(d_{p,k_p},\ldots,d_{q,k_q}, \text{active}(x_r)) = \top \quad (6.2)$$

$$x_r \notin \{x_i,\ldots,x_j\} \rightarrow a_{\{x_p,\ldots,x_q\},x_r}(d_{p,k_p},\ldots,d_{q,k_q}, \neg\text{active}(x_r)) = \top$$

A dynamic preference constraint satisfaction problem (DPCSP) extends a DCSP with a preference valuation $P(x_i : d_{ij}) \in \mathbb{P}$ for each assignment $x_i : d_{ij}$, where $\mathbb{P}$ the domain preference valuations. The preference of a (partial) solution $\{x_i : d_{ik_i}, \ldots, x_j : d_{jk_j}\}$ is computed as

$$P(x_i : d_{ik_i}, \ldots, x_j : d_{jk_j}) = P(x_i : d_{ik_i}) \oplus \ldots \oplus P(x_j : d_{jk_j}) \quad (6.3)$$

where $\oplus$ is a commutative, associative closed binary operation on $\mathbb{P}$. The preference values in $\mathbb{P}$ are partially ordered by $\prec$. The aim of a DPCSP is to find a solution with a maximum preference valuation. In other words, a solution of a DPCSP is every set of assignments S for which (6.1) and (6.2) hold, such that no other set of assignments S' exist for which (6.1) and (6.2) hold and for which $P(S) \not\prec P(S')$.

A number of applications are constraint satisfaction problems of this type. Within this work, examples of this form of DPCSP will be given on compositional modelling, configuration and planning. Table 6.1 shows an overview of how these problems map to a DPCSP.

6.1.2 Assumptions about the preference calculus

Without losing generality, the preferences taken from a domain $\mathbb{P}$ and generated by the preference calculus are assumed to be partially ordered. That is, for each pair of preferences $P_1, P_2 \in \mathbb{P}$ on of the following is true:

- $P_1 \prec P_2$: P_1 is smaller than P_2, or
- $P_1 \succ P_2$: P_1 is greater than P_2, or
- $P_1 = P_2$: P_1 is equal to P_2, or

Application	Compositional Modelling	Configuration	Planning
Attributes	Assumptions	Components	Activities at time instance
Domains	Assumption classes	Component options	Enabled activity at given state
Compatibility constraints	Inconsistent parts of models and requirements	Inconsistent combinations and requirements	Inconsistent states and requirements
Activity constraints	Source participants and structural conditions	Component prerequisites	Prerequisite states
Preferences	Utility contribution	Utility contribution or cost	Reward, resource requirement, time requirement

Table 6.1: Applications of dynamic preference constraint satisfaction

- $P_1?P_2$: P_1 is incomparable with P_2.

The solution techniques below are applicable for any calculus that employs combination operator $\oplus$ such that

$$\neg(P_1 \prec P_2) \rightarrow \neg(P_1 \oplus P \prec P_2 \oplus P) \tag{6.4}$$

In other words, it is assumed that the combination operator is monotonic (note that max and min fall in this category). This operator is allowed, but not required, to be strictly monotonic.

For the sake of clarity, a preference is presumed to be a *utility* in this work, though it does not have to be in general. Thus, combining preferences with one another does not decrease the overall preference:

$$\neg(P_1 \oplus P_2 \prec P_1) \quad \text{and} \quad \neg(P_1 \oplus P_2 \prec P_2)$$

It is important to note that the algorithms described below can be applied cases where preferences express *costs* or where costs and utilities are combined. Consider a function f that reverses the ordering of preference and combinations of preferences. That is, f is a function $f : \mathbb{P} \mapsto \mathbb{P}$ with the following properties:

$$\forall P_1, P_2 \in \mathbb{P}, f(P_1) \prec f(P_2) \leftarrow P_1 \succ P_2$$
$$\forall P_1, P_2 \in \mathbb{P}, f(P_1) = f(P_2) \leftarrow P_1 = P_2$$
$$\forall P_1, P_2 \in \mathbb{P}, f(P_1 \oplus P_2) = f(P_1) \oplus f(P_2)$$

If such a function exists, a DPCSP solution with a minimal cost $P(x_i : d_{ik_i}, \ldots, x_j : d_{jk_j})$ can be easily found by applying the algorithms below to find a solution with a maximal preference $f(P(x_i : d_{ik_i}, \ldots, x_j : d_{jk_j}))$. It should be fairly easy to find the required function f. For example, if $\mathbb{P}$ is the set of real numbers $\mathbb{R}$, then the function $f : \mathbb{R} \mapsto \mathbb{R} : r \rightarrow f(r) = -r$ has the required properties. For the OMP calculus presented in chapter 5, it can be shown that a function which reverses the ordering of the BPQs in the preference scale, i.e. $f(p_1) \ll f(p_2) \rightarrow p_1 \gg p_2$ and $f(p_1) < f(p_2) \rightarrow p_1 > p_2$, has the required properties.

6.2 The basic algorithm

This section describes an algorithm to solve a DPCSP of the above description. In essence, this algorithm is a variation on the A* algorithm. Section 6.2.1 contains a

brief overview of the A* algorithm. The basic DPCSP solution algorithm is presented in section 6.2.2 and its use is illustrated with an example in section 6.2.3.

6.2.1 Algorithm A*

The A* algorithm, introduced in [72], implements a solution strategy to path finding problems that require a shortest path-to-goal solution. The approach involves constructing a search tree in which the leaf nodes represent goal states, or solutions to the problem at hand, and the other nodes correspond to intermediate states, or partial solutions. Each arc in the search tree may have a value associated with it. This value typically describes the cost of moving from one state to the next on a path towards a goal state. The total cost associated with a goal state equals the sum of the costs associated with the arcs between the nodes on the path from the root node of the search tree to the node that corresponds to the goal state.

Algorithm 6.1: A* ALGORITHM(INITIALSTATE(), SUCCESSORS(), ISGOAL(), ())

$O \leftarrow$ createOrderedQueue();
$n \leftarrow$ INITIALSTATE();
$\hat{f}(n) \leftarrow g(n) + \hat{h}(n)$;
enqueue$(O, n, \hat{f}(n))$;
while $O \neq \emptyset$
do
  $n \leftarrow$ dequeue(O);
  if ISGOAL(n)
    then return (n)
    else
      $N \leftarrow$ SUCCESSORS(n);
      $\hat{f}(n_{\text{child}}) \leftarrow g(n_{\text{child}}) + \hat{h}(n_{\text{child}})$;
      for each $n_{\text{child}}, (n_{\text{child}} \in N)$
        do enqueue$(O, n_{\text{child}}, \hat{f}(n_{\text{child}}))$;

Algorithm 6.1 presents the pseudocode of the generic A* algorithm, based on its description in [147]. This algorithm maintains a number nodes n in a *priority queue* O, called the *open set*, in ascending order of $\hat{f}(n) = g(n) + \hat{h}(n)$, where

- $g(n)$ is the sum of the cost associated with the arcs between the nodes on the path from the root node to n, and

- $\hat{h}(n)$ is a heuristic estimate of the sum of the cost associated with the arcs between nodes on the shortest path from n to a goal state.

At each iteration, algorithm A* removes the node n with the lowest value of $\hat{f}(n)$ from O. If n is not a goal state (determined by the domain dependent function ISGOAL(n)), all possible successor states are computed by means of the domain dependent function SUCCESSORS() and enqueued in O. The algorithm terminates if a goal state is found or if O becomes empty. Obviously, in the latter case, there are no solutions to the problem at hand.

A shortest path finding algorithm is said to be *admissible* when it is guaranteed to find the goal state with the lowest cost, provided a goal state exists. It has been shown in [72, 73, 139] that a sufficient condition for the A* algorithm to be admissible is:

$$\hat{h}(n) \leq h(n)$$

where $h(n)$ is the smallest sum of costs associated with the arcs between nodes on a path from n to a goal state. In other words, algorithm A* is admissible if the heuristic estimate $\hat{h})(n)$ of the cost associated with a path from n to a goal state is a lower bound on the actual cost associated with such a path,

Of course, not all heuristics that are employed by A* algorithms are equally good. A heuristic h_1 is said to be *more informed* than a heuristic h_2 if:

$$\forall n, h_1(n) \geq h_2(n)$$

It has been shown in [72, 73, 139] that if an A* algorithm employs a more informed heuristic, it will find an optimal solution by creating a smaller or equal number of nodes than a less informed heuristic. Thus, more informed heuristics lead to more efficient A* algorithms, provided the time and space complexity involved in computing the heuristic does not offset the efficiency gain. In general, the development of A* algorithms requires finding a trade-off between informedness and computational complexity of the employed heuristics. In section 6.3, three more informed heuristics are developed and their usefulness will be discussed in chapter 7.

6.2.2 An A* algorithm for DPCSPs

When applying the approach discussed in section 6.2.1 to solve a DPCSP, the state of an node n corresponds to a partial solution $S(n) = \{x_i : d_{il_i}, \ldots, x_j : d\,jl_j\}$. Moving from

one node n_1 to one of its children n_2 involves adding an attribute value assignment to $S(n_1)$. Instead of a cost, each arc has a preference associated with it. If n_1 is a parent node of n_2, such that $S(n_2) = S(n_1) \cup \{x_k : d_{kl_k}\}$, then the preferences value $P(x_k : d_{kl_k})$ corresponds to the arc from n_1 to n_2. The combination of all the preferences on a path from the root to a node n is the preference of the partial solution $S(n)$. This preference value will be called the committed preference of that node n, and it is denoted $CP(n)$. The aim of using the A* algorithm is to find a node n that corresponds to a solution of the DPCSP and maximises $CP(n)$. To that end, the algorithm may employ a heuristic such as the potential preference.

Definition 6.1. The *committed preference* $CP(n)$ of a node n is the combination of the preferences of the attribute values assignments in $S(n)$:

$$CP(n) = \oplus_{x:d \in S(n)} P(x : d)$$

The *potential preference* $PP(n)$ of a node n is the highest preference that a solution S of the DPCSP, such that $S(n) \subseteq S$. That is,

$$PP(n) = [\oplus_{x:d \in S(n)} P(x : d)] \oplus \left[\max_{d_i \in D(x_i), \ldots, d_j \in D(x_j), S(n) \cup, \{x_i : d_i, \ldots, x_j : d_j\} \mathbf{C}, \mathbf{A} \nvdash \bot} (P(x_i : d_i) \oplus \ldots \oplus P(x_j : d_j)) \right]$$

where $\{x_i, \ldots, x_j\} = X_{nd}(n)$.

The potential preference could be used as heuristic of an A* algorithm that is able to find the preferred solution to a DPCSP, as is shown in theorem 6.2.

Theorem 6.2. Any A* algorithm that is applied to a DPCSP and stores the nodes n it creates in the set O in descending order of a heuristic $\widehat{PP}(n)$, such that $\widehat{PP}(n) \succcurlyeq PP(n)$, is admissible and it is guaranteed to find a node n which maximises $CP(n)$.

Proof: See appendix A, page 297.

However, calculating potential preference is computationally illogical as it requires finding a solution for the DPCSP in the first place. Therefore, an estimate of the potential preference that is easier to determine may be useful. This work employs upper bounds on the potential preference. The definitions of these upper bounds use a partition $X_a(n)$,$X_d(n)$,$X_{nd}(n)$ of the set of all attributes $\mathbf{X}$, where

- $X_a(n)$ is the set of active assigned attributes, i.e.

$$X_a(n) = \{x \in \mathbf{X} \mid \exists d \in D(x), x : d \in S(n)\}$$

- $X_d(n)$ is the set of inactive unassigned attributes that can not be activated under $S(n)$ and the set of activity constraints **A**, i.e.

$$X_d(n) = \{x \in \mathbf{X} \mid S(n) \cup \{\text{active}(x)\}, \mathbb{A} \vdash \bot\}$$

- $X_{nd}(n)$ is the set of unassigned attributes (active or inactive) that may still be activated under $S(n)$ and the set of activity constraints **A**, i.e.

$$X_{nd}(n) = \{x \in \mathbf{X} \mid [\nexists d \in D(x), x : d \in S(n)] \wedge [S(n) \cup \{\text{active}(x)\}, \mathbb{A} \nvdash \bot]\}$$

The basic algorithm orders the open nodes in descending order of $\widehat{PP}_{\text{basic}}(n)$, where

$$\widehat{PP}_{\text{basic}}(n) = CP(n) \oplus [\oplus_{x \in X_{nd}(n)} \max_{d \in D(x)} P(x : d)] \tag{6.5}$$

Given a node n, which corresponds to a partial assignment $S(n)$, $\widehat{PP}_{\text{basic}}(n)$ is the combination of the preferences associated with the assignments of all assigned attributes, i.e. all $x \in X_a(n)$, and the highest preferences associated with the unassigned attributes that can still be activated, i.e. all $x \in X_{nd}$. Theorem 6.3 shows that $\widehat{PP}_{\text{basic}}(n)$ computes an upper bound on $PP(n)$.

Theorem 6.3. $\forall n, \widehat{PP}_{\text{basic}}(n) \succcurlyeq PP(n)$

Proof: See appendix A, page 297.

It follows from theorems 6.2 and 6.3 that an A* algorithm that searches for a solution to a DPCSP and that orders the open nodes in descending order of $\widehat{PP}_{\text{basic}}(n)$ is admissible and, hence, it is guaranteed to find a node n containing a solution to the DPCSP with a maximal preference $CP(n)$. The basic algorithm, which consists of procedures 6.2, 6.3 and 6.4, forms such an A* algorithm.

In procedure 6.2, nodes are sorted in the priority queue O such that they remain in descending order of $\widehat{PP}(n)$, and such that nodes with an equal $\widehat{PP}(n)$ remain in descending order of CP.

At each iteration, the node n with the highest $\widehat{PP}$ is dequeued from O. If, however, O is empty, the algorithm terminates and there is not solution to the DPCSP. Otherwise, child nodes are created for n and enqueued in O. In most cases, this involves taken the next unassigned active attribute from the set of unassigned active attributes for that node $X_u(n)$ and exploring all possible assignments of that attribute. Procedure 6.3

Algorithm 6.2: SOLVE$(\mathbf{X},\mathbf{D},\mathbf{C},\mathbf{A},P)$

$X_a(n) \leftarrow \{x_i \mid \{\}, A \vdash \text{active}(x_i)\}$;
$n \leftarrow \text{createNode}(\text{nil}, X_a(n))$;
$O \leftarrow \text{createOrderedQueue}()$; $CP(n) \leftarrow 0$;
$PP(n) \leftarrow \oplus_{x \in X} \max_{d \in D(x)} P(x:d)$;
PROCESSATTRIBUTE$(\text{first}(X_a(n)), X_a(n), n, C, A, P, O)$;
while $O \neq \emptyset$
do
- $n \leftarrow \text{dequeue}(O)$;
- **if** $X_u(n) \neq \emptyset$
  - **then**
    - $x \leftarrow \text{first}(X_u(n))$;
    - PROCESSATTRIBUTE(x, n, C, A, P, O);
  - **else**
    - $X_u(n) \leftarrow \{x_i \mid \text{solution}(n), \mathbf{A} \vdash \text{active}(x_i)\} - X_a(n)$;
    - **if** $X_u(n) = \emptyset$
      - **then**
        - $n_{\text{next}} \leftarrow \text{first}(O)$;
        - **if** $CP(n) \not\prec PP(n_{\text{first}})$
          - **then return** $(S(n))$;
          - **else**
            - $PP(n) \leftarrow CP(n)$;
            - $\text{enqueue}(O, n, CP(n), PP(n))$;
      - **else**
        - $x \leftarrow \text{first}(X_a(n))$;
        - PROCESSATTRIBUTE(x, n, C, A, P, O);

contains the pseudo-code for this procedure. If there are no remaining unassigned active attributes ($X_u(n) = \emptyset$), the activity constraints $\mathbf{A}$ are fired and $X_u(n)$ is computed as:

$$\{x_i \mid \text{solution}(n), \mathbf{A} \vdash \text{active}(x_i)\} - X_a(n)$$

Procedure 6.3, PROCESSATTRIBUTE$(x, n_{\text{parent}}, \mathbf{C}, \mathbf{A}, P, O)$, takes an unassigned active attribute x, the most recently dequeued node n_{parent}, the set of compatibility constraints $\mathbf{C}$, the set of activity constraints $\mathbf{A}$, the preference assignments P and the open set O. This algorithm considers each domain value $d \in D(x)$ and it checks whether $S(n_{\text{parent}}) \cup \{x:d\}$ is consistent with the compatibility constraints $\mathbf{C}$. If this is the case, a new child node n_{child} is created with $S(n_{\text{child}}) = S(n_{\text{parent}}) \cup \{x:d\}$ and $CP(n_{\text{child}})$ is computed. Then, procedure 6.4 is called where $\widehat{PP}(n_{\text{child}})$ is computed and the new node n_{child} is enqueued in O.

In section 6.3, procedures 6.3 and 6.4 will be replaced by ones that perform more computations per node, but employ more informed heuristics $\widehat{PP}(n)$ and hence explore

Procedure 6.3: PROCESSATTRIBUTE$(x, n_{\text{parent}}, \mathbf{C}, \mathbf{A}, P, O)$

for $d \in D(x)$
do
  if $S(n_{\text{parent}}) \cup \{x:d\}, \mathbf{C} \nvdash \bot$
  then
    $n_{\text{child}} \leftarrow$ new node;
    $S(n_{\text{child}}) \leftarrow S(n_{\text{parent}}) \cup \{x:d\}$;
    $X_d \leftarrow \text{deactivated}(S(n_{\text{child}}), X(n_{\text{parent}}))$;
    $X_{nd}(n_{\text{child}}) \leftarrow X_{nd}(n_{\text{parent}}) - \{x\} - X_d$;
    $X_a(n_{\text{child}}) \leftarrow X_a(n_{\text{parent}}) \cup \{x\}$;
    $X_u(n_{\text{child}}) \leftarrow X_u(n_{\text{parent}}) - \{x\}$;
    $CP(n_{\text{child}}) \leftarrow CP(n_{\text{parent}}) \oplus P(x:d)$;
    COMPUTEPOTENTIAL$(x, n_{\text{child}}, n_{\text{parent}}, P, O)$;

Procedure 6.4: COMPUTEPOTENTIAL$(x, n_{\text{child}}, n_{\text{parent}}, P, O)$

$PP(n_{\text{child}}) \leftarrow CP(n_{\text{child}}) \oplus (\oplus_{x \in X_{nd}(n)} \max_{d \in D(x)} P(x:d))$;
enqueue$(O, n_{\text{child}}, PP(n_{\text{child}}), CP(n_{\text{child}}))$;

fewer nodes.

6.2.3 An illustrative example

The algorithm described above can be illustrated by a simple compositional modelling example. The attributes, domains and the compatibility and activity constraints are generated with the techniques presented in chapter 4. Using the OMP calculus, preferences can be introduced and assigned to the various configuration options. Suppose that the following, rather simple, order of preference calculus is employed:

$$\text{low} \ll \text{medium} \ll \text{high}$$

Consider the problem of modelling the spread of disease in the population of an eco-system presented in table 6.2. The compositional modelling problem is restricted to 6 features: relevance of the population-growth phenomenon (attribute pop-growth), the model of disease (attribute disease), the model of population growth (attribute

Attribute	Domain	Preferences
pop-growth	{yes,no}	P(pop-growth:yes)=high,
		P(pop-growth:no)=low.
disease	{statistic,organism,particle}	P(disease:statistic)=high,
		P(disease:organism)=medium,
		P(disease:particle)=medium.
growth-mod	{const,exp,log}	P(growth-mod:const)=0,
		P(growth-mod:exp)=0,
		P(growth-mod:log)=0.
contam	{yes,no}	P(contam:yes)=high,
		P(contam:no)=0.
org-death	{yes,no}	P(org-death:yes)=high,
		P(org-death:no)=0.
contam-mod	{air,cont,fluids}	P(contam-mod:air)=0,
		P(contam-mod:cont)=0,
		P(contam-mod:fluids)=high.

Table 6.2: Sample dynamic order of preference magnitude CSP

growth-mod), relevance of contamination phenomenon (attribute contam), relevance of death of the (disease) organism (attribute org-death) and the model of contamination (attribute contam-mod). A possible set of preferences for the various modelling choices is assigned in table 6.2.

Of course, not every combination of features is possible. There are hard restrictions imposed the knowledge of model construction that the organism death phenomenon can not be combined with any contamination model or with a population growth model. The deduction of such inconsistencies has been explained in more detail in sections 4.1.2.2 and 4.2.

$$\text{contam:yes} \wedge \text{org-death:yes} \rightarrow \bot$$
$$\text{pop-growth:yes} \wedge \text{org-death:yes} \rightarrow \bot$$

Also, certain features will depend on the choices made with respect to other features. In this example, the population growth and contamination models can only

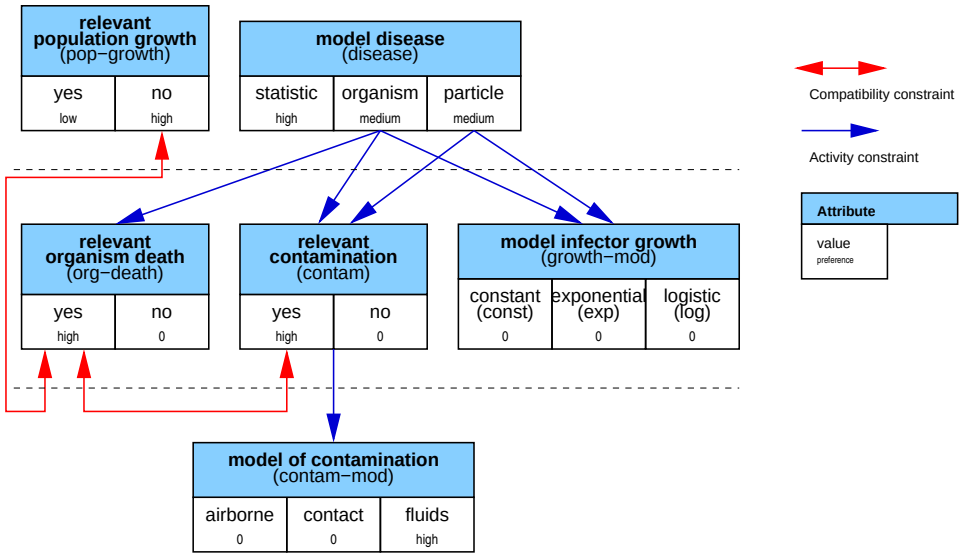


Figure 6.2: DPCSP of a compositional modelling problem

be used in a scenario ‘model that describes the disease as an organism or a particle. Furthermore, only if the behaviour of the disease is represented as an organism can its death be modelled. Finally, the contamination models required the contamination phenomenon to be relevant. This knowledge is formalised by the following activity constraints.

$$\begin{aligned}
\text{active}(\text{pop-growth}) &\leftarrow \top \\
\text{active}(\text{disease}) &\leftarrow \top \\
\text{active}(\text{growth-mod}) &\leftarrow \text{disease:organism} \vee \text{disease:particle} \\
\text{active}(\text{contam}) &\leftarrow \text{disease:organism} \vee \text{disease:particle} \\
\text{active}(\text{org-death}) &\leftarrow \text{disease:organism} \\
\text{active}(\text{contam-mod}) &\leftarrow \text{contam:yes}
\end{aligned}$$

For easy reference, the specification of this DPCSP has been summarised in figure 6.2.

Figures 6.3, 6.4 and 6.5 depict how the proposed algorithm constructs and proceeds through the search space. For simplicity, the attributes are assumed to be instantiated in the following order: disease, pop-growth, org-death, contam, growth-mod, contam-mod. Note that the search algorithm itself does not depend on a fixed ordering of

variables. Within the figures, the combined preferences are summarised to a concatenation of the first characters of the constituent basic preference quantities (e.g. lmh denotes low $\oplus$ medium $\oplus$ high).

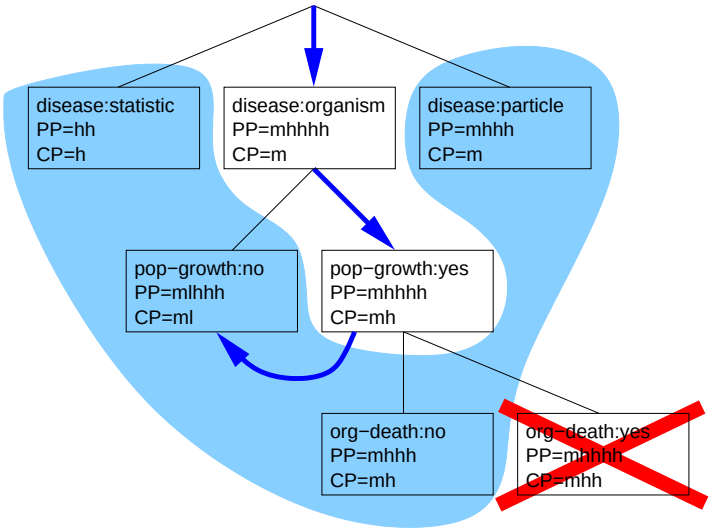


Figure 6.3: Search space for the sample DPCSP

Initially, disease is assigned a value. The initial potential preferences are computed as follows:

$$
\begin{aligned}
PP(\text{disease:statistic}) =& P(\text{disease:statistic}) \oplus \max_{d_i \in D_{\text{pop-growth}}} P(\text{pop-growth} : d_i) \\
=& \text{high} \oplus \text{high} \\
PP(\text{disease:organism}) =& P(\text{disease:organism}) \oplus \max_{d_i \in D_{\text{pop-growth}}} P(\text{pop-growth} : d_i) \\
& \oplus \max_{d_i \in D_{\text{growth-mod}}} P(\text{growth-mod} : d_i) \oplus \max_{d_i \in D_{\text{contam}}} P(\text{contam} : d_i) \\
& \oplus \max_{d_i \in D_{\text{org-death}}} P(\text{org-death} : d_i) \oplus \max_{d_i \in D_{\text{contam-mod}}} P(\text{contam-mod} : d_i) \\
=& \text{medium} \oplus \text{high} \oplus \text{high} \oplus \text{high} \oplus \text{high} \\
PP(\text{disease:particle}) =& P(\text{disease:particle}) \oplus \max_{d_i \in D_{\text{pop-growth}}} P(\text{pop-growth} : d_i) \\
& \oplus \max_{d_i \in D_{\text{growth-mod}}} P(\text{growth-mod} : d_i) \oplus \max_{d_i \in D_{\text{contam}}} P(\text{contam} : d_i) \\
& \oplus \max_{d_i \in D_{\text{contam-mod}}} P(\text{contam-mod} : d_i) \\
=& \text{medium} \oplus \text{high} \oplus \text{high} \oplus \text{high}
\end{aligned}
$$

When disease is assigned organism, all other attributes can still be activated and, therefore, they might still be assigned a preferred value from their domain. In the case of disease:statistic, no more assignments will activate org-death, contam, growth-mod and contam-mod. The potential preferences computed in this way are assigned to the respective nodes in the search space. These nodes are then put in a *priority queue*, called the *open set*, where they are ordered with respect to their potential preference. In addition, nodes with the same potential preferences are further ordered with respect to their committed preference.

During each iteration, and for as long as no solution is found or nodes remain in the priority queue, the first (and hence preferred) node is dequeued and nodes are added to the open set for each possible assignment of the next attribute. As shown in figure 6.3, the algorithm first dequeues the node that is exploring $\{$disease:organism$\}$ and then $\{$disease:organism, pop-growth:yes$\}$. The consecutive node that is dequeued is not a child of $\{$disease:organism, pop-growth:yes$\}$, but of $\{$disease:organism, pop-growth:no$\}$, because the potential preference computed for pop-growth:yes can not be realised due to the compatibility constraint pop-growth:yes $\wedge$ org-death:yes $\rightarrow \bot$.

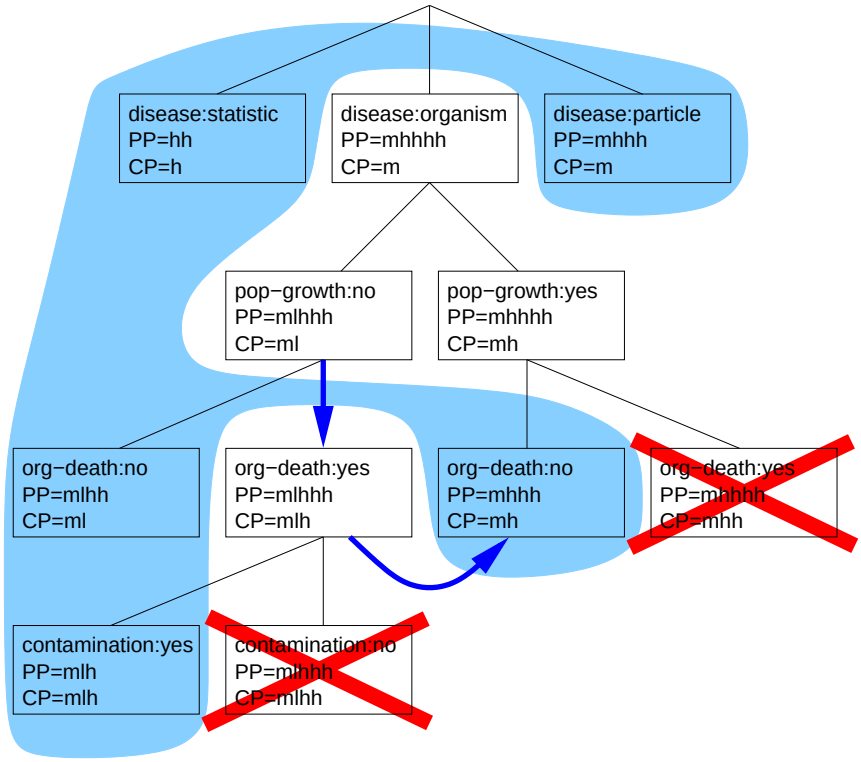


Figure 6.4: Search space for the sample DPCSP (cont'd)

As figure 6.4 shows, the next explored node is {disease:organism, pop-growth:no, org-death:yes}. From this node, the algorithm jumps back to exploring children of {disease:organism, pop-growth:yes} because of the compatibility constraint org-death:yes $\wedge$ contam:yes $\rightarrow \perp$.

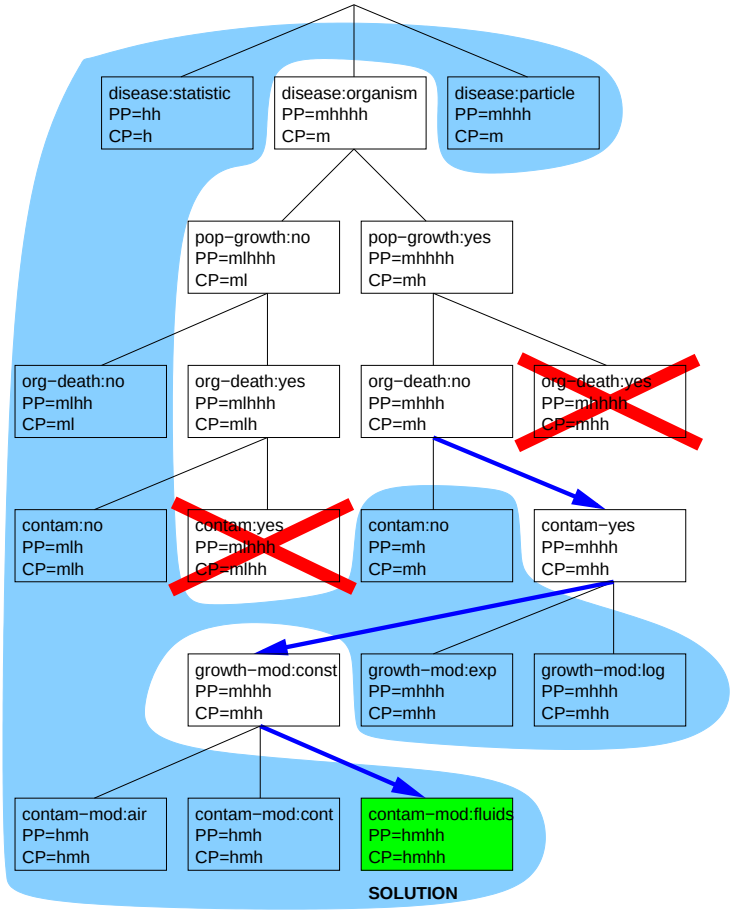


Figure 6.5: Search space for the sample DPCSP (cont'd)

The remainder of the search path to the first solution is presented in figure 6.5. Eventually, the algorithm arrives at a node containing {disease:organism, pop-growth:yes, org-death:no, contam:yes, growth-mod:const, contam-mod:fluids}, which has a preference equal to high $\oplus$ medium $\oplus$ high $\oplus$ high. Five other solutions exists that have the same preference. These are listed in table 6.3.

It is clear that the particular sequence of node-exploring depends on the order in which attributes are considered. For illustration purposes, a particularly inefficient ordering of attributes is considered in the remainder of this chapter, more specifically:

{disease:organism, pop-growth:yes, org-death:no, contam:yes, growth-mod:exp, contam-mod:fluids}
{disease:organism, pop-growth:yes, org-death:no, contam:yes, growth-mod:log, contam-mod:fluids}
{disease:particle, pop-growth:yes, org-death:no, contam:yes, growth-mod:const, contam-mod:fluids}
{disease:particle, pop-growth:yes, org-death:no, contam:yes, growth-mod:exp, contam-mod:fluids}
{disease:particle, pop-growth:yes, org-death:no, contam:yes, growth-mod:log, contam-mod:fluids}

Table 6.3: Alternative solutions to the sample DPCSP

pop-growth, disease, growth-mod, contam, org-death, contam-mod. The search space that is explored with this ordering is shown in figure 6.6.

6.3 Improving informedness

As with backtracking based approaches towards finding a solution for this CSP, the basic algorithm suffers from *thrashing*, the problem of repeatedly searching through the same subtree when changing a value higher up in the tree. The heuristic causes its own variation on thrashing as it may repeatedly make the same optimistic assumptions, and hence mistakes, with respect to the $\widehat{PP}(n)$ value of a node n. To alleviate this problem, several improvements have been developed. These approaches aim at computing a more informed heuristic $\widehat{PP}(n)$.

6.3.1 Forward checking based improvement

Forward checking is an approach that evaluates potential assignments of unassigned attributes with respect to the current partial solution. It is used in conventional hard CSPs to check whether unassigned attributes can be assigned consistently, thereby pruning branches in the search tree that can not lead to a consistent assignment earlier.

The FC algorithm consists of

- procedure 6.2: the SOLVE$(\mathbf{X}, \mathbf{D}, \mathbf{C}, \mathbf{A}, P)$ procedure of the basic algorithm,
- procedure 6.3: the PROCESSATTRIBUTE$(x, n_{\text{parent}}, \mathbf{C}, \mathbf{A}, P, O)$ of the basic algorithm, and
- procedure 6.5: a version of COMPUTEPOTENTIAL$(x, n_{\text{child}}, n_{\text{parent}}, O, O)$ specific to the FC based improvement.

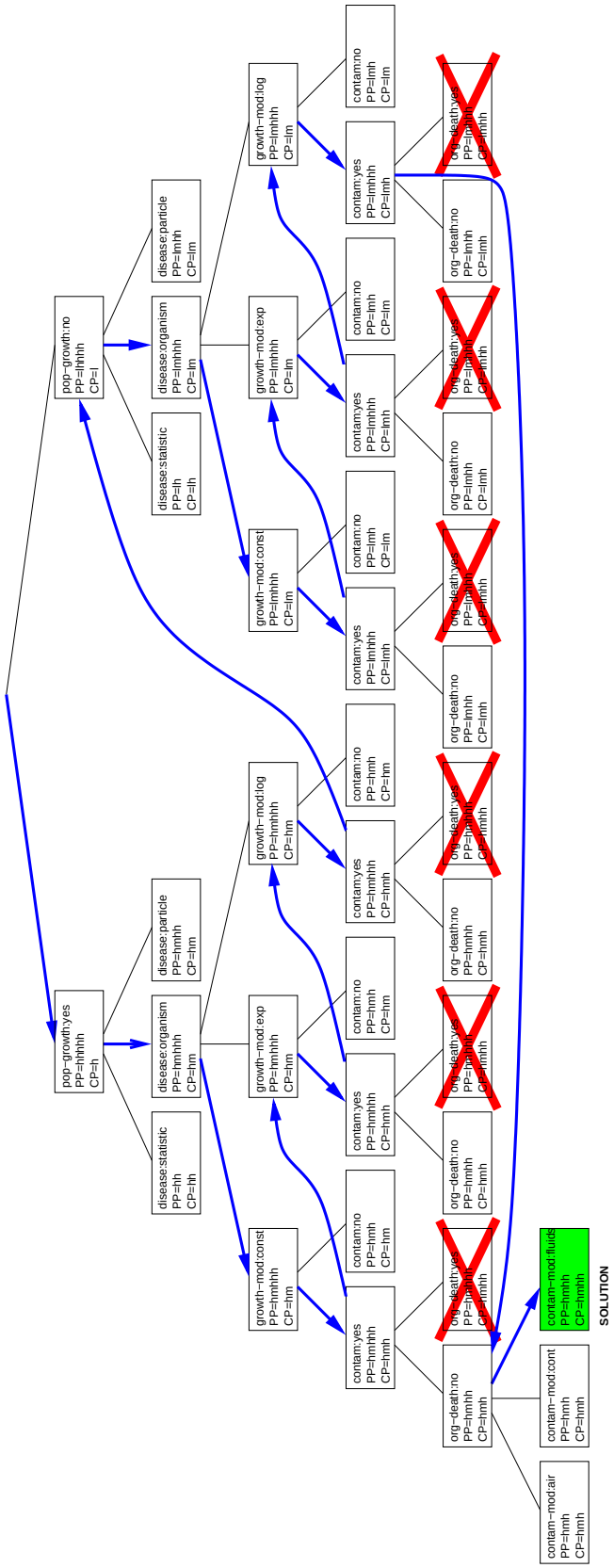


Figure 6.6: Search space for the sample DPCSP

When computing $\widehat{PP}(n_{\text{child}})$, the latter procedure takes the highest preference of the assignments that are consistent with the current partial solution $S(n_{\text{child}})$ (instead of assuming that all assignments may be consistent). That is,

$$\widehat{PP}_{\text{fc}}(n) = CP(n) \oplus [\oplus_{x \in X_{nd}(n)} \max_{d \in D(x), S(n) \cup \{x:d\}, \mathbf{C} \nvdash \bot} P(x:d)] \tag{6.6}$$

If an unassigned attribute is known to be active, and there is no assignment for it that is consistent with the current partial solution $S(n_{\text{child}})$, then the current assignment is deemed inconsistent and it is therefore not queued in the open set O. If no consistent assignments exist for an attribute that is not known to be active, a consistent assignment is still possible provided that the attribute is not activated.

Note that this approach is affected by the arity of the compatibility constraints. The arity of a compatibility constraint equals the number of attributes involved in the constraint. In other words, a compatibility constraints c_Y is said to be k-ary, if Y contains k attributes. Obviously, in order to be able to determine whether an attribute-value assignment $x:d$, with $x \in Y$ satisfies a k-ary compatibility constraint c_Y, the $k-1$ other attributes must already have been assigned. Hence, as k increases, this approach become significantly less efficient. It is possible to translate non-binary compatibility constraints into binary ones, however, as is discussed in section 6.4.1.

Procedure 6.5: COMPUTEPOTENTIAL$(x, n_{\text{child}}, n_{\text{parent}}, P, O)$

$PP(n_{\text{child}}) \leftarrow CP(n_{\text{child}})$;
for $y \in X_u(n_{\text{child}})$
do {
  $D_y \leftarrow \{v \in D(y) \mid S(n) \cup \{x:d\} \cup \{y:v\}, \mathbf{C} \nvdash \bot\}$;
  if $D_y \neq \emptyset$
  then {
    $v_{\max} \leftarrow \max_{v \in D_y} P(y:v)$;
    $PP(n_{\text{child}}) \leftarrow PP(n_{\text{child}}) \oplus v_{\max}$;
  }
  else $PP(n_{\text{child}}) \leftarrow$ nil
}
if $PP(n_{\text{child}}) \neq$ nil
then {
  for $y \in [X_{nd}(n_{\text{child}}) - X_u(n_{\text{child}}]$
  do {
    $D_y \leftarrow \{v \in D(y) \mid S(n) \cup \{x:d\} \cup \{y:v\}, \mathbf{C} \nvdash \bot\}$;
    $v_{\max} \leftarrow \max_{v \in D_y} P(y:v)$;
    $PP(n_{\text{child}}) \leftarrow PP(n_{\text{child}}) \oplus v_{\max}$;
  }
  enqueue$(O, n_c, PP(n_{\text{child}}), CP(n_{\text{child}}))$;
}

Theorem 6.4. $\forall n, \widehat{PP}_{\text{basic}}(n) \succcurlyeq \widehat{PP}_{\text{fc}}(n) \succcurlyeq PP(n)$

Proof: See appendix A, page 297.

Theorem 6.4 states that $\widehat{PP}_{\text{fc}}(n)$ is an upper bound on $PP(n)$. According to theorems 6.2 and 6.3, this implies that an A* algorithm based on the $\widehat{PP}_{\text{fc}}(n)$, i.e. the FC based improvement on the basic algorithm, is an A* algorithm as well. Theorem 6.4 also states that $\widehat{PP}_{\text{fc}}(n)$ is a lower bound on $\widehat{PP}_{\text{basic}}(n)$, and hence, it is a more informed heuristic. The latter implies that the FC based improvement will not explore more nodes to reach a solution than the basic algorithm.

When applying this algorithm to the compositional modelling problem discussed earlier, the explored proportion of the search space is considerably reduced. Figure 6.7 shows what nodes are opened by this improved algorithm for attribute ordering pop-growth, disease, growth-mod, contam, org-death, contam-mod. Compared to figure 6.6, the reduction in the search tree is obvious. The most important changes in the sequence of node exploration occur at the instantiations of pop-growth and contam. At these specific nodes, the compatibility constraint between pop-growth and org-death and that between contam and org-death affect the computation of potential preference *PP*. Note that no assignment for these pairs of attributes will contribute a preference of high $\oplus$ high to the overall preference. The basic algorithm discussed earlier had to attempt the instantiation of both attributes before it could detect this feature of the problem.

6.3.2 Maintaining preference boundaries based improvement

FC only discovers when the highest preference of an unassigned attribute can not be realised, if the (compatibility or activity) constraints that cause this involve just assigned attributes other than the unassigned attribute. No form of forward checking can discover suboptimal assignments when multiple attributes involved the offending constraints are unassigned. However, the search algorithm could learn from the previous overestimates of preference. The Maintaining Preference Boundaries (MPB) based improvement sets out for this, discovering upper bounds $b(\{x_i, \ldots, x_j\})$ on the combined preference of a consistent assignment of the attributes $x_i, \ldots, x_j$:

$$\left[\max_{d_i \in D_i, \ldots, d_j \in D_j, \{x_i:d_i, \ldots, x_j:d_j\} \nvdash \bot} P(x_i : d_i) \oplus \ldots \oplus P(x_j : d_j)\right] \preccurlyeq b(\{x_i, \ldots, x_j\}) \qquad (6.7)$$

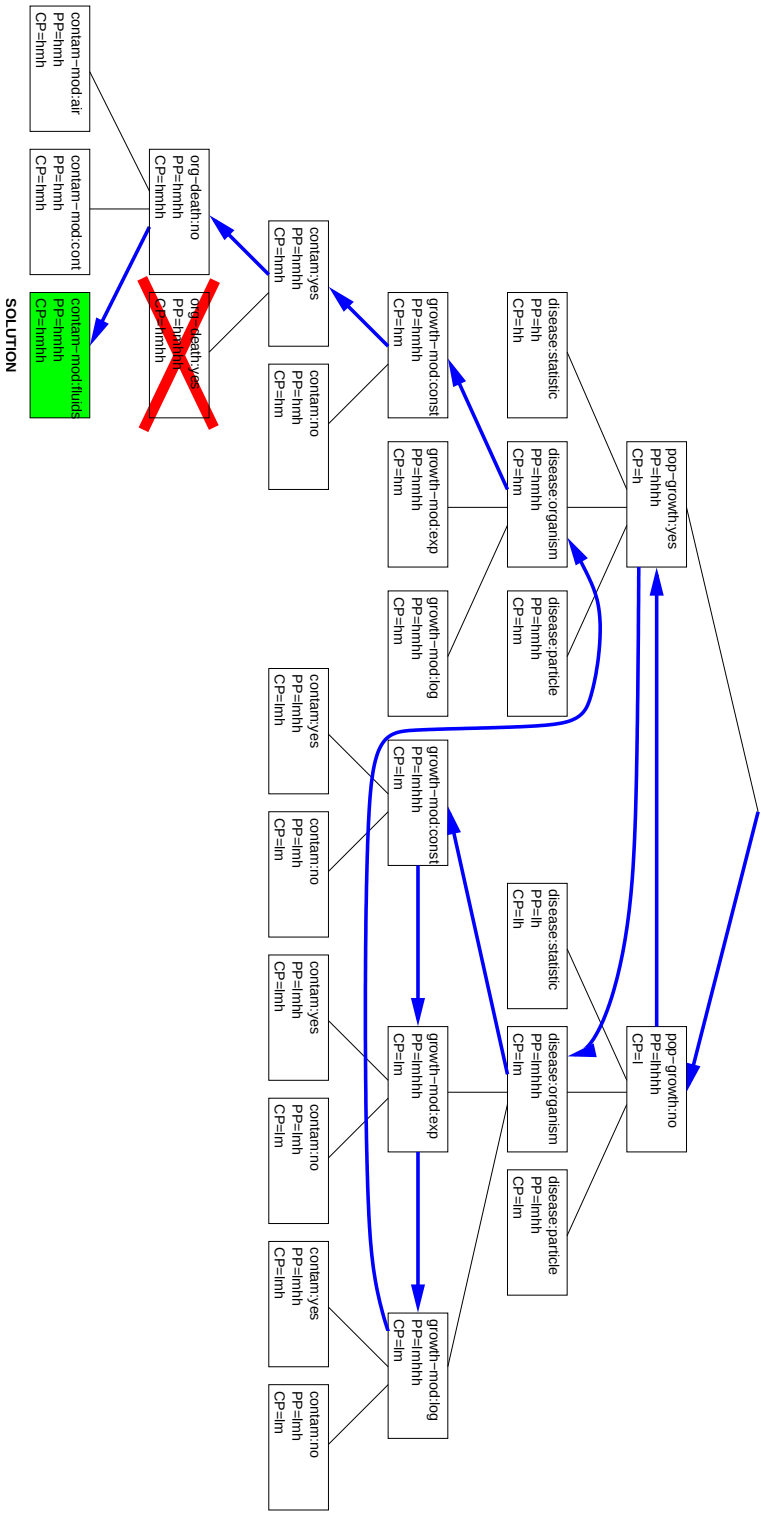


Figure 6.7: Search space under FC based improvement

Upper bounds on attainable preference levels are due to compatibility constraints, activity constraints or a combination of both. Compatibility constraints may impose an upper bound on the attainable preference if they make the combination of assignments with relatively high preferences inconsistent. Activity constraints may impose an upper bound on the attainable preference if an assignment of a relatively lower preference is required to activate an attribute that has a domain containing some highly preferred values. Preference boundaries due to activity constraints can be difficult to detect because they may be caused by multiple interacting activity constraints. Therefore, this work focuses on preferences boundaries due to compatibility constraints only.

The MPB algorithm consists of:

- procedure 6.2: the SOLVE$(\mathbf{X},\mathbf{D},\mathbf{C},\mathbf{A},P)$ procedure of the basic algorithm,
- procedure 6.6: a version of PROCESSATTRIBUTE$(x,n_{\text{parent}},\mathbf{C},\mathbf{A},P,O)$ specific to the MPB based improvement, and
- procedure 6.8: a version of COMPUTEPOTENTIAL$(x,n_{\text{child}},n_{\text{parent}},P,O)$ specific to the MPB based improvement.

Procedure 6.6 extends procedure 6.3 by taking note of the compatibility constraints that cause certain values to be removed from consideration. For each of these constraint violations, a label, i.e. a set of nogood environments in the ATMS sense [34], the preference of the abandoned attribute value assignment and the attribute value assignment itself is stored in a structure called mpbc (meaning MPB constraint). Once all potential values have been considered, the MPB constraints are transformed into boundaries by procedure 6.7.

In the first phase, procedure 6.7 takes the preference of each of the inconsistent attribute value assignments that were noted as part of the MPB constraints provided that they are higher than the preference of a valid assignment. This results in a queue of preferences that might have been achieved if certain compatibility constraints had not be violated. Then, in the second phase, procedure 6.7 identifies the smallest sets of attributes involved in these constraints and computes an upper bound for this set of attributes. A small example may explain how this works more clearly.

Assume a CSP (or a part of a CSP) consisting of three attributes x_1,x_2,x_3, each with a domain $\{a,b,c\}$ such that $P(x_i:a)=$ high, $P(x_i:b)=$ medium and $P(x_i:c)=$ low.

Procedure 6.6: PROCESSATTRIBUTE$(x, n_{\text{parent}}, C, A, P, O)$

MPBC $\leftarrow \{\}$;
$P_{\text{best}} \leftarrow -\infty$;
for $d \in D(x)$
do
  $\S_\perp \leftarrow \{S \mid S \subset S(n_{\text{parent}}), S \cup \{x:d\} \vdash \perp\}$;
  if $S = \emptyset$
  then
    $n_{\text{child}} \leftarrow$ new node;
    $S(n_{\text{child}}) \leftarrow S(n_{\text{parent}}) \cup \{x:d\}$;
    $X_d \leftarrow \text{deactivated}(S(n_{\text{child}}), X(n_{\text{parent}}))$;
    $X_{nd}(n_{\text{child}}) \leftarrow X_{nd}(n_{\text{parent}}) - \{x\} - X_d$;
    $X_a(n_{\text{child}}) \leftarrow X_a(n_{\text{parent}}) \cup \{x\}$; $X_u(n_{\text{child}}) \leftarrow X_u(n_{\text{parent}}) - \{x\}$;
    $CP(n_{\text{child}}) \leftarrow CP(n_{\text{parent}}) \oplus P(x:d)$;
    COMPUTEPOTENTIAL$(x, n_{\text{child}}, n_{\text{parent}}, P, O)$;
    $P_{\text{best}} \leftarrow \max(P_{\text{best}}, P(x:d))$;
  else
    mpbc $\leftarrow$ new mpb-constraint;
    $L(\text{mpbc}) \leftarrow S_\perp$;
    $P(\text{mpbc}) \leftarrow P(x:d)$;
    $a(\text{mpbc}) \leftarrow x:d$;
    MPBC $\leftarrow$ MPBC $\cup \{\text{mpbc}\}$;
PROCESSMPBCS(MPBC, $n_{\text{parent}}, P_{\text{best}}, \mathbf{C}, \mathbf{A}, P$);

(Note that this is the same preference scale as used in the ongoing example.) The constraints are $x_1 : a \wedge x_3 : b \rightarrow \perp$ and $x_2 : a \wedge x_3 : a \rightarrow \perp$. Figure 6.8 presents the search space generated for this CSP after an instantiation of x_1, x_2 and x_3 has been attempted. At that point, two inconsistent assignments $x_3 : a$ and $x_3 : b$ are encountered.

First, the constraints that prevent the instantiation $x_3 : a$, where $P(x_3 : a) = \text{high}$ are considered. Only one constraint, $x_2 : a \wedge x_3 : a \rightarrow \perp$ causes this. Therefore, the assignment that causes $x_3 : a$ to be inconsistent is $x_2 : a$.

Then, the constraints that prevent the instantiation $x_3 : b$ are considered. In this case, the constraint for assignments with a preference higher than $P(x_3 : b)$ must be considered as well. If these constraints were not active, $x_3 : b$ would be a possible assignment not worthy of consideration. Thus, the constraints being considered in this case are $x_2 : a \wedge x_3 : a \rightarrow \perp$ and $x_1 : a \wedge x_3 : b \rightarrow \perp$. In general, this operation may produce two or more sets of constraints, dependent upon the number of compatibility constraints in the DPCSP and the number of attribute value assignments with higher

Procedure 6.7: PROCESSMPBCS($MPBC, n_{\text{child}}, P_{\text{best}}, \mathbf{C}, \mathbf{A}, P$)

$M \leftarrow$ createOrderedQueue();
for each $\text{mpbc}_i \in \text{MPBC}$
do { **if** $P(\text{mpbc}_i) \succ P_{\text{best}}$ **then** enqueue$(M, P(\text{mpbc}_i), P(\text{mpbc}_i))$;
while $\neg$empty(M)
do {
$\text{mpbc}_i \leftarrow$ dequeue(M);
$L \leftarrow$ crossProduct$(\{L(\text{mpbc}) \mid \text{mpbc} \in \text{MPBC}, \neg[P(\text{mpbc}) \prec P(\text{mpbc}_i)]\})$;
for each $E \in L$
do {
$X_E \leftarrow$ attributeSet(E);
$P_{\text{current}} \leftarrow \oplus_{a_i \in S(n_{\text{child}}) \cap E} P(a_i) \oplus P(x:d)$;
$P_{\max} \leftarrow \max(P_{\text{current}}, P \oplus \text{bestAlternative}(E, \mathbf{C}, \mathbf{A}, P))$;
addBoundary$(X_E, P_{\max})$;
}
}

preferences than the current attribute value assignment. By taking the minimal cross product of these sets, the smallest sets of assignments responsible for the whole range of inconsistencies becomes available. In this example, $x_1 : a, x_2 : a$ is the cause.

The bestAlternative function seeks the best potential alternative values for the attributes in the offending constraint(s) such that the assignment becomes consistent. For this computation, each attribute in each offending constraint[1], other than the attribute currently being assigned, the alternative domain values are considered. Two cases are possible:

- There may only be alternative values with a lower preference. In this case, the highest consistent alternative assignment is searched for each attribute and the alternative that reduces the overall preference the least is maintained. This situation occurs in the example as illustrated in figure 6.9. Computing a consistent alternative assignment for x_2 with $x_3 : a$ is easy as: $x_2 : b$ is the next best assignment. This results in an upper bound $b(\{x_2, x_3\}) = \text{high} \oplus \text{medium}$. Computing consistent alternative assignments for x_1 and x_2 with $x_3 : a$ or $x_3 : b$ involves looking at the effect of changing individual assignments for x_1 and x_2, i.e. the assignments $\{x_1 : b, x_2 : a, x_3 : b\}$ and $\{x_1 : a, x_2 : b, x_3 : b\}$. The prefer-

[1] Note that, similar to FC, the efficiency of this approach is affected by the arity of the compatibility constraints.

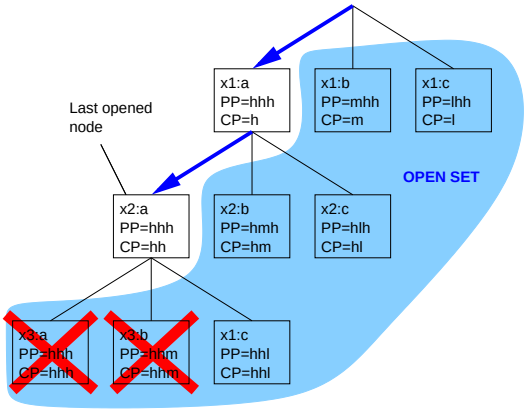
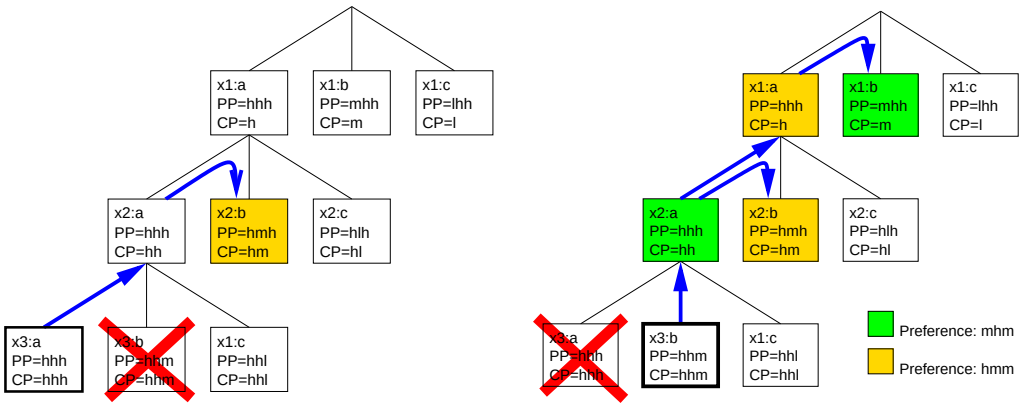


Figure 6.8: Maintaining preference boundaries (MPB)



(a) Alternative for $x_3 : a$ (b) Alternative for $x_3 : a, x_3 : b$

Figure 6.9: Computing the best alternative assignment

ence of both assignments is the same and yields an upper bound $b(\{x_1, x_2, x_3\}) =$ high $\oplus$ medium $\oplus$ medium.

- If there is an alternative and consistent assignment with a higher preference for one or more of the attributes, then such an assignment is obviously retained.

Let B denote the set of all stored boundaries. The addBoundary function adds a boundary from B. Initially, boundaries are assumed to be added for all singleton attribute sets equal to the highest preference that can be achieved for an attribute value assignment. That is,

$$b(\{x_i\}) = \max_{d_{ij} \in D_i} P(x_i : d_{ij}), \quad \forall x_i \in X$$

Ideally, addBoundary would only add a boundary if the boundary adds to the information already stored. This means that a boundary $b(X) = b_X$ should only be added if,

$$\forall b(X_1),\ldots,b(X_n) \in B, X_1 \cup \ldots \cup X_n = X \rightarrow b(X_1) \oplus \ldots \oplus b(X_n) > b(X) \tag{6.8}$$

Procedure 6.8: COMPUTEPOTENTIAL$(x, n_{\text{child}}, n_{\text{parent}}, P, O, X, X_a, X_d)$

$(X_b, PP_b) \leftarrow \text{getBoundary}(X - X_a - X_d)$;
$PP(n_{\text{child}}) \leftarrow CP(n_{\text{child}}) \oplus PP_b \oplus (\oplus_{x \in X_{nd}(n) - X_b} \max_{d \in D(x)} P(x:d))$;
$\text{enqueue}(O, n_{\text{child}}, PP(n_{\text{child}}), CP(n_{\text{child}}))$;

Procedure 6.8 employs the stored boundaries to compute a more informed heuristic. Such a boundary consists of a set of attributes X_b and a preference value $PP_b = b(X_b)$. The preference values returned by getBoundary(X) should be:

$$\min_{b(X_1),\ldots,b(X_n) \in B, X_1 \cup \ldots \cup X_n = X} b(X_1) \oplus \ldots \oplus b(X_n) \tag{6.9}$$

The problem with this approach is that, when considering a boundary with respect to a set of attributes X, all partitions of X need to be considered and the associated boundary combinations must be computed. To avoid this, a simpler approach is taken by only considering the most significant boundaries.

Definition 6.5. Given three attribute sets X, X_1 and X_2 such that $X_1 \subset X$ and $X_2 \subset X$, $b(X_1)$ is said to be more significant than a boundary $b(X_2)$ if $X_2 \subset X_1$. The set of most significant boundaries of a set of attributes X, equals

$$B(X) = \{b(X_i) \mid \neg(\exists b(X_j) \in B, X_i \subset X_j \subset X)\}$$

A boundary $b(X)$ is now added when it is smaller than the most significant boundaries, each combined with the domain boundaries for the attributes not covered in the respective most significant boundaries, or

$$\forall b(X_b) \in B(X), \oplus_{x \in X - X_b} b(\{x\}) \oplus b(X_b) \succ b(X) \tag{6.10}$$

Similarly, getBoundary for a set of attributes X returns:

$$\min_{b(X_b)\in B} \oplus_{x\in X-X_b} b(\{x\}) \oplus b(X_b) \tag{6.11}$$

The boundary $b(X_b)$ found in this way is then employed to compute the MPB version of the $\widehat{PP}(n)$ heuristic:

$$\begin{gathered}\widehat{PP}_{\text{mpb}}(n) = CP(n) \oplus b(X_b) \oplus (\oplus_{x\in X_{nd}(n)-X_b} \max_{d\in D(x)} P(x:d)) \\ \text{where } b(X_b) \preccurlyeq \oplus_{x\in X_b} \max_{d\in D(x)} P(x:d)\end{gathered} \tag{6.12}$$

Theorem 6.6. $\forall n, \widehat{PP}_{\text{basic}}(n) \succcurlyeq \widehat{PP}_{\text{mpb}}(n) \succcurlyeq PP(n)$

Proof: See appendix A, page 298.

Theorem 6.6 states that $\widehat{PP}_{\text{mpb}}(n)$ is an upper bound on $PP(n)$. According to theorems 6.2 and 6.3, this implies that an A* algorithm based on the $\widehat{PP}_{\text{mpb}}(n)$, i.e. the MPB based improvement on the basic algorithm, is an A* algorithm as well. Theorem 6.6 also states that $\widehat{PP}_{\text{mpb}}(n)$ is a lower bound on $\widehat{PP}_{\text{basic}}(n)$, and hence, it is a more informed heuristic. The latter implies that the MPB based improvement will not explore more nodes to reach a solution than the basic algorithm.

Figure 6.10 shows the search space resulting from the application of MPB based improvement to the ongoing compositional modelling problem. Initially, there are no savings in terms of covered search space. However, as soon as the first constraints are encountered, savings compared to figure 6.6 occur. In particular, when the algorithm opens the node representing {pop-growth:no}, more accurate estimates $\widehat{PP}$ are possible. As opposed to the search space shown in figure 6.6, $PP(\{\text{pop-growth:no}, \text{disease:organism}\}) = \text{low} \oplus \text{medium} \oplus \text{high} \oplus \text{high}$. The basic algorithm had a higher PP for that node and needed to open 6 more nodes before it could safely establish that the PP estimate needed to be reduced.

6.3.3 Maintaining preference boundaries with forward checking

Although MPB based algorithm is capable of learning upper bounds imposed by some constraints, on the preference of assignments of certain attributes, no early detection of restrictive compatibility constraints is carried out. Both these features can be combined, however, by integrating maintaining preference boundaries and forward check-

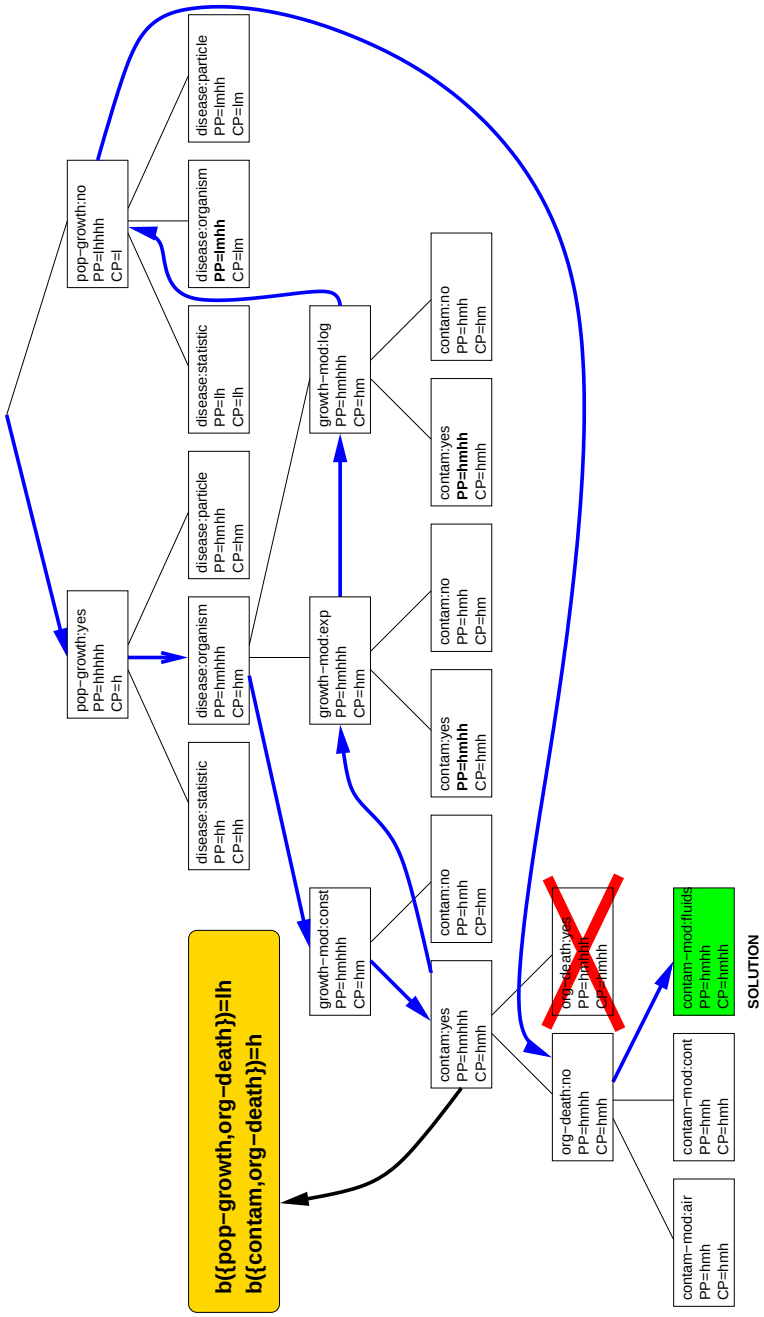


Figure 6.10: Search space for the sample DPCSP (MPB approach)

ing. This approach will be called *maintaining preference boundaries with forward checking* (MPB-FC).

Procedure 6.9: PROCESSATTRIBUTE$(x, n_{\text{parent}}, \mathbf{C}, \mathbf{A}, P, O)$

MPBC $\leftarrow \{\}$;
$P_{\text{best}} \leftarrow -\infty$;
for $d \in D(x)$
do { **if** $S(n_{\text{parent}}) \cup \{x:d\}, \mathbf{C} \nvdash \bot$
then { $n_{\text{child}} \leftarrow$ new node;
$S(n_{\text{child}}) \leftarrow S(n_{\text{parent}}) \cup \{x:d\}$;
$X_d \leftarrow \text{deactivated}(S(n_{\text{child}}), X(n_{\text{parent}}))$;
$X_{nd}(n_{\text{child}}) \leftarrow X_{nd}(n_{\text{parent}}) - \{x\} - X_d$;
$X_a(n_{\text{child}}) \leftarrow X_a(n_{\text{parent}}) \cup \{x\}$; $X_u(n_{\text{child}}) \leftarrow X_u(n_{\text{parent}}) - \{x\}$;
$CP(n_{\text{child}}) \leftarrow CP(n_{\text{parent}}) \oplus P(x:d)$;
COMPUTEPOTENTIAL$(x, n_{\text{child}}, n_{\text{parent}}, P, O)$; }
for (AMPBC $\in$ MPBC)
do processMPBCs$(X_b(\text{AMPBC}), n_{\text{child}}, P_{\text{best}}(\text{AMPBC}), \mathbf{C}, \mathbf{A}, P)$;

The pseudo-code for the resulting hybrid between MPB and FC is given in:

- procedure 6.2: the SOLVE$(\mathbf{X}, \mathbf{D}, \mathbf{C}, \mathbf{A}, P)$ procedure of the basic algorithm,
- procedure 6.7: the PROCESSMPBCS$(MPBC, n_{\text{child}}, P_{\text{best}}, \mathbf{C}, \mathbf{A}, P)$ procedure of the MPB based improvement,
- procedure 6.9: a version of PROCESSATTRIBUTE$(x, n_{\text{parent}}, \mathbf{C}, \mathbf{A}, P, O)$ specific to the MPB-FC based improvement, and
- procedure 6.10: a version of COMPUTEPOTENTIAL$(x, n_{\text{child}}, n_{\text{parent}}, P, O)$ specific to the MPB-FC based improvement.

The processAttribute algorithm is similar to that of MPB. However, the MPB constraints are generated by the computePotential algorithm. As in the FC based improvement, computePotential searches the domains of all attributes that are not impossible to activate for the value that is not inconsistent with the current assignment and that has the highest preference. If there are values with higher preferences that are inconsistent with the current assignment, MPB constraints are generated as explained in 6.3.2.

Procedure 6.10: COMPUTEPOTENTIAL$(x, n_{\text{child}}, n_{\text{parent}}, P, O)$

$PP_b^{\text{fc}} \leftarrow$ nil;
$(X_b, PP_b) \leftarrow$ getBoundary$(X - X_a - X_d)$;
$PP(n_{\text{child}}) \leftarrow CP(n_{\text{parent}})$;
MPBC $\leftarrow \{\}$;
for each $y \in X_u(n_{\text{child}})$
do {
  AMPBC $\leftarrow$ FORWARDCHECK$(y, n_{\text{child}}, \text{MPBC})$;
  if $P_{\text{best}}(\text{AMPBC}) \neq -\infty$
  then {
    if $y \in X_b$
    then $PP_b^{\text{fc}} \leftarrow PP_b^{\text{fc}} \oplus P_{\text{best}}(\text{AMPBC})$;
    else $PP(n_{\text{child}}) \leftarrow PP(n_{\text{child}}) \oplus P_{\text{best}}(\text{AMPBC})$;
  }
  else return (MPBC);
}
for each $y \in [X_{nd}(n_{\text{child}}) - X_u(n_{\text{child}}]$
do {
  AMPBC $\leftarrow$ FORWARDCHECK$(y, n_{\text{child}}, \text{MPBC})$;
  if $P_{\text{best}}(\text{AMPBC}) \neq -\infty$
  then {
    if $y \in X_b$
    then $PP_b^{\text{fc}} \leftarrow PP_b^{\text{fc}} \oplus P_{\text{best}}(\text{AMPBC})$;
    else $PP(n_{\text{child}}) \leftarrow PP(n_{\text{child}}) \oplus P_{\text{best}}(\text{AMPBC})$;
  }
}
$PP(n_{\text{child}}) \leftarrow PP(n_{\text{child}}) \oplus \min(PP_b, PP_b^{\text{fc}})$;
enqueue$(O, n_{\text{child}}, PP(n_{\text{child}}), CP(n_{\text{child}}))$;
return (MPBC);

These MPB constraints are not grouped in a single set but in attribute specific sets, which are denoted AMPBC in the algorithm. For each of these, the best available preference of a consistent value is maintained ($P_{\text{best}}(\text{AMPBC})$). When computePotential terminates, processAttribute calls processMPBCs (see section 6.3.2) for each attribute specific set of MPB constraints and the associated best (potentially) realised preference for that attribute.

Similar to the previous algorithms, the MPB-FC based improvement is an A* algorithm in which the nodes n in the priority queue O are stored in descending order of $\widehat{PP}_{\text{mpb-fc}}(n)$, where

$$\begin{aligned}\widehat{PP}_{\text{mpb-fc}}(n) = CP(n) \oplus \min(b(X_b), \oplus_{x \in X_b} \max_{d \in D(x)} P(x:d)) \\ \oplus (\oplus_{x \in X_{nd}(n) - X_b} \max_{d \in D(x)} P(x:d))\end{aligned} \tag{6.13}$$

Theorem 6.7. $\forall n, [\widehat{PP}_{\text{fc}}(n) \succcurlyeq \widehat{PP}_{\text{mpb-fc}}(n)] \wedge [\widehat{PP}_{\text{mpb}}(n) \succcurlyeq \widehat{PP}_{\text{mpb-fc}}(n)] \wedge [\widehat{PP}_{\text{mpb-fc}}(n) \succcurlyeq$

Procedure 6.11: FORWARDCHECK(y, n_{child}, MPBC)

AMPBC $\leftarrow$ new attribute-mpbc-set;
$X_b(textAMPBC) \leftarrow \{\}$;
$P_{\text{best}}(\text{AMPBC}) \leftarrow -\infty$;
MPBC $\leftarrow$ MPBC $\cup \{\text{AMPBC}\}$;
for each $v \in D(y)$
do
  $S_\perp \leftarrow \{X_i \mid X_i \subset S(n_{\text{child}}), X_i \cup \{x : d\} \vdash \perp\}$;
  if $S_\perp = \emptyset$
    then $P_{\text{best}}(\text{AMPBC}) \leftarrow \max(P_{\text{best}}(\text{AMPBC}), P(y : v))$;
    else
      mpbc $\leftarrow$ new mpb-constraint;
      $X_b(\text{AMPBC}) \leftarrow X_b(\text{AMPBC}) \cup \{\text{mpbc}\}$;
      $L(\text{mpbc}) \leftarrow S_\perp$;
      $P(\text{mpbc}) \leftarrow P(y : v)$;
      $a(\text{mpbc}) \leftarrow y : v$;
return $(AMPBC)$;

$PP(n)]$

Proof: See appendix A, page 298.

Theorem 6.7 states that $\widehat{PP}_{\text{mpb-fc}}(n)$ is an upper bound on $PP(n)$. According to theorems 6.2 and 6.3, this implies that an A* algorithm based on the $\widehat{PP}_{\text{mpb-fc}}(n)$, i.e. the MPB-FC based improvement on the basic algorithm, is an A* algorithm as well. Theorem 6.6 also states that $\widehat{PP}_{\text{mpb-fc}}(n)$ is a lower bound on $\widehat{PP}_{\text{mpb}}(n)$ and $\widehat{PP}_{\text{fc}}$, and hence, it is a more informed heuristic.

In the worst case, MPB-FC will visit the same number of nodes as FC. In fact, in the ongoing example, the same search space depicted in figure 6.7 is reproduced by MPB-FC. However, as the CSP involves more attributes, the result of learning upper bounds will become more noticeable.

It is difficult to illustrate this potential of MPB-FC with an example that is small enough to remain tractable for the basic algorithm but is sufficiently complex to have any benefits to MPB-FC. In order to illustrate the potential advantages of MPB-FC, the ongoing DPCSP is extended at this stage. Assume for example that a number of additional attributes, say one representing the model of organism growth (attribute org-growth) and one describing the model for the incubation of the disease (attribute incub), are added to the problem. For illustration purposes, these attributes are acti-

vated by the disease:organism assignment and will be instantiated after the growth-mod attribute.

Figure 6.11 shows the search space that is explored by forward checking for the extended DPCSP. The search space within the grayed rectangle is repeated three times because the upper boundary that exists over attributes contam and org-death is not learned. Figure 6.12 shows the search space produced under MPB-FC. The advantage of the latter approach over forward checking is that this algorithm does not iterate the same subtree of assignments to org-growth, incub and contam since it learns the upper boundary over contam and org-death.

The advantage of MPB-FC over MPB lies in the early detection of upper bounds. Note that MPB needed to instantiate org-death before it was able to discover $b(\{\text{contam}, \text{org-death}\})$. Obviously, in this case, the difference this makes is minimal. However, if org-growth and incub were to be instantiated after contam but before org-death, the difference would be far more significant.

6.4 Efficiency

This section discusses a number of factors affecting the efficiency of the algorithms presented in this chapter. First, section 6.4.1 returns to the problem of non-binary compatibility constraints in DPCSPs, and it suggests a way of translating these into binary ones. Then, section 6.4.2 relates the efficiency of the algorithms of this chapter to that of other approaches, which might have been explored.

6.4.1 The arity of compatibility constraints

The examples in this chapter and the experiments in chapter 7 are DPCSPs with binary compatibility constraints. Binary constraints are constraints involving two attributes (i.e. constraints of the form $c_{\{x_i,x_j\}}$). This issue is important as certain algorithms can only be effective when the compatibility constraints involve few attributes. The potential benefits of forward checking, for instance, become insignificant as the compatibility constraints restrict the assignments of three or more attributes.

The DPCSPs generated from compositional modelling problems do not necessarily meet this assumption. However, a simple extension of the approach introduced

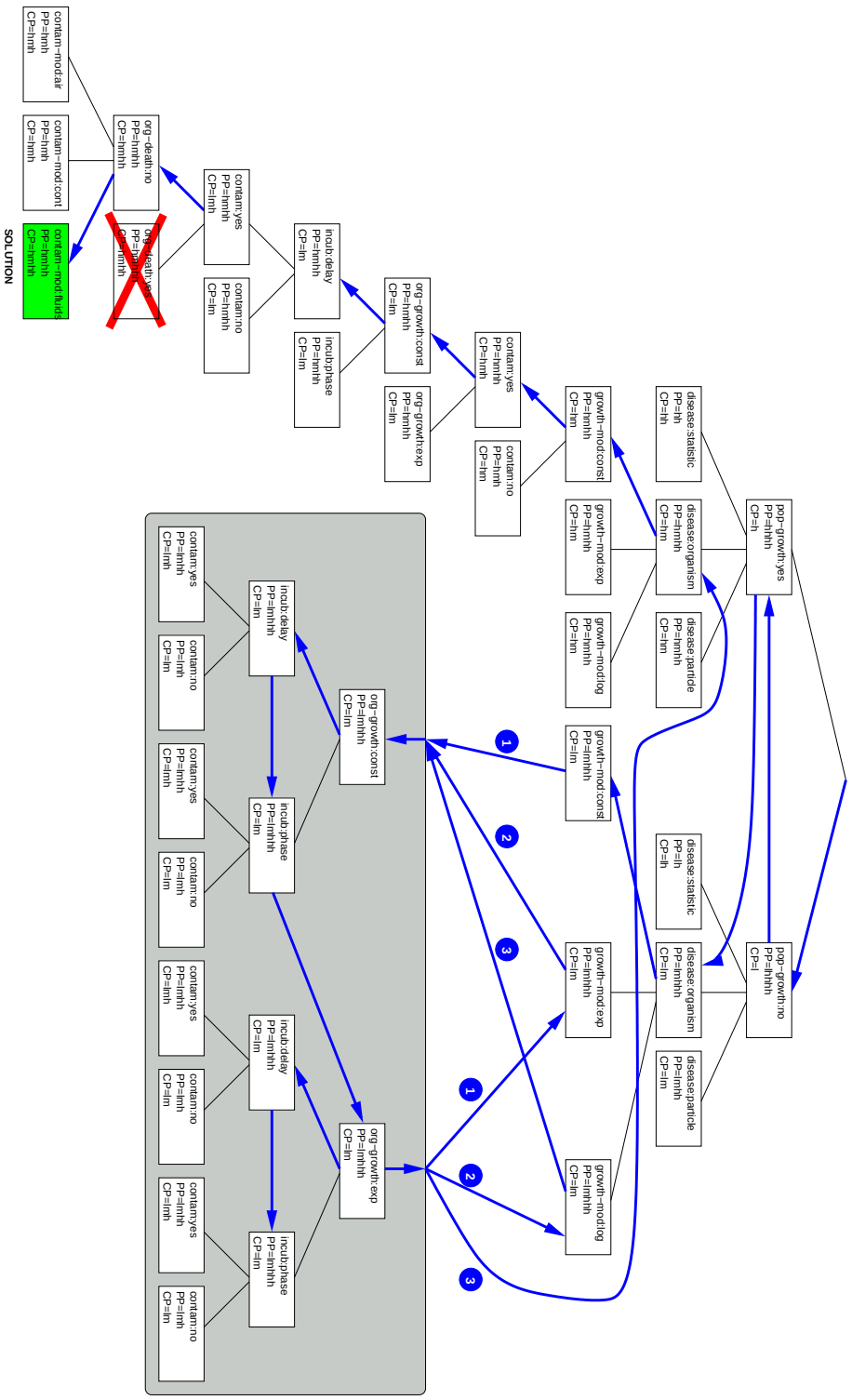


Figure 6.11: Forward checking applied to the extended DPCSP

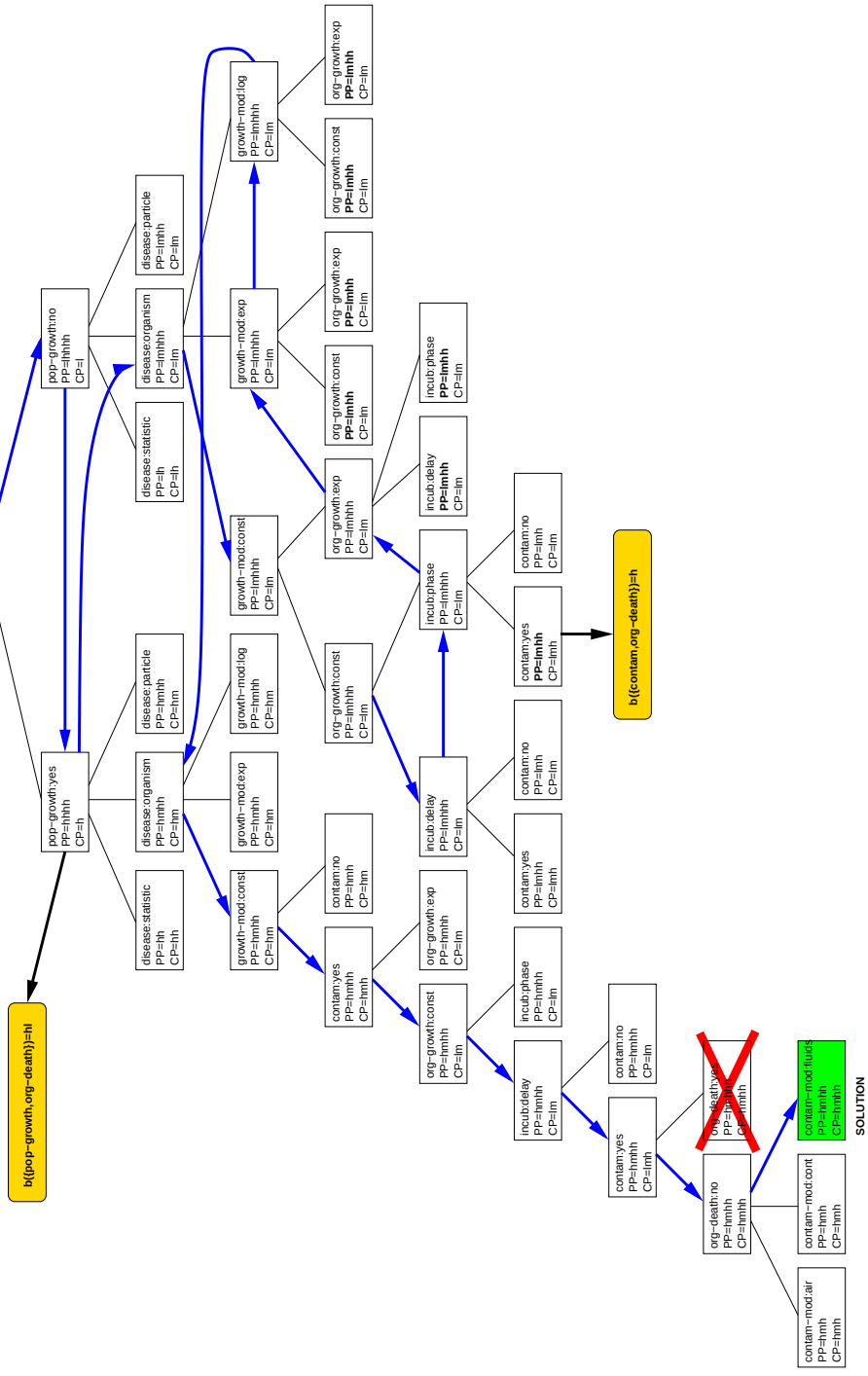


Figure 6.12: maintaining preference boundaries by forward checking applied to the DPCSP

in [40] can translate DPCSPs with non-binary compatibility constraints into DPCSPs with binary constraints. The procedure described in [40] works as follows. Each k-ary compatibility constraints $c_{\{x_i,\ldots,x_j\}}$ (i.e. each constraint involving three or more attributes: $|\{x_i,\ldots,x_j\}| \geq 3$) is replaced by:

- An attribute x_c
- A domain $D(x_c)$ that contains a value $d_{x_i:d_{ik_i},\ldots,x_j:d_{jk_j}}$ for each distinct partial assignment $\{x_i : d_{ik_i},\ldots,x_j : d_{jk_j}\}$ that satisfies the compatibility constraint (i.e. $c_{\{x_i,\ldots,x_j\}}(d_{ik_i},\ldots,d_{jk_j}) = \top$).
- A set of binary compatibility constraints $c_{\{x_i,x_c\}},\ldots,c_{\{x_j,x_c\}}$ that link the domain values of $D(x_c)$, which correspond to sets of assignments of x_i, ..., and x_j, to the associated domain values of $D(x_i)$, ..., and $D(x_j)$. That is, each constraint $c_{\{x_l,x_c\}}$ defines a function:

$$c_{\{x_l,x_c\}} : D(x_l) \times D(x_c) \mapsto \{\top,\bot\} : c_{\{x_l,x_c\}}(d_{\ldots,x_l:d,\ldots},d') = \begin{cases} \top & \text{if } d = d' \\ \bot & \text{if } d \neq d' \end{cases}$$

Obviously, in the setting of a *dynamic* CSP, compatibility constraints c_Y only influence the solution if all of the attributes in the constraint (i.e. all attributes in Y), are active. Constraint c_Y is irrelevant whenever one of the attributes in Y is not active. Hence, the attribute that corresponds to the constraint, x_c, must not be active whenever one of the attributes in Y is prohibited from being active. Constraint c_Y restricts attribute-value assignments whenever all the attributes in Y must be active. Hence, x_c has to be active whenever all the attributes in Y is prohibited from being inactive.

This implies that two sets of activity constraints have to be added to the DCSP, for each of the aforementioned k-ary compatibility constraints $c_{\{x_i,\ldots,x_j\}}$ and its corresponding attribute x_c:

- The first set of activity constraints prohibits the activation of the new attribute x_c whenever the active state of one of the attributes in the compatibility constraint (i.e. one of $\{x_i,\ldots,x_j\}$) is inconsistent. Formally, for each activity constraint $a_{\{x_p,\ldots,x_q\},x_l}$, with $x_l \in \{x_i,\ldots,x_j\}$, an activity constraint

$$a_{\{x_p,\ldots,x_q\},x_c} : D(x_p) \times \ldots \times D(x_q) \times \{\text{active}(x_c), \neg\text{active}(x_c)\} \mapsto \{\top,\bot\}$$

is created, such that

$$a_{\{x_p,\ldots,x_q\},x_c}(d_{pk_p},\ldots,d_{qk_q},\text{active}(x_c)) = \begin{cases} \bot & \text{if } a_{\{x_p,\ldots,x_q\},x_l}(d_{pk_p},\ldots,d_{qk_q},\text{active}(x_l)) = \bot \\ \top & \text{otherwise} \end{cases}$$

- The second set of activity constraints prohibits the inactive state of x_c whenever all the attributes in compatibility constraint (i.e. all of $\{x_i,\ldots,x_j\}$) are active. Formally, for each set of activity constraints $\{a_{\{x_p,\ldots,x_q\},x_i},\ldots,a_{\{x_v,\ldots,x_w\},x_j}\}$, an activity constraint

  $$a_{\{x_p,\ldots,x_q\}\cup\ldots\cup\{x_v,\ldots,x_w\},x_c} : D(x_p)\times\ldots\times D(x_q)\times\{\text{active}(x_c),\neg\text{active}(x_c)\}\mapsto\{\top,\bot\}$$

  is created, such that

  $$a_{\{x_p,\ldots,x_q\}\cup\ldots\cup\{x_v,\ldots,x_w\},x_c}(d_{pk_p},\ldots,d_{qk_q},\ldots,d_{vk_v},\ldots,d_{wk_w},\neg\text{active}(x_c)) = \begin{cases} \bot & \text{if } (a_{\{x_p,\ldots,x_q\},x_i}(d_{pk_p},\ldots,d_{qk_q},\neg\text{active}(x_i)) = \bot)\wedge \\ & \quad\vdots \\ & \quad\wedge(a_{\{x_v,\ldots,x_w\},x_j}(d_{vk_v},\ldots,d_{wk_w},\neg\text{active}(x_j)) = \bot) \\ \top & \text{otherwise} \end{cases}$$

As discussed in section 6.1.1, a DPCSP extends a DCSP by associating domain values with preference valuations. Preference valuations are not attached to compatibility constraints, though. Because these preferences are not directly related to compatibility constraints, the translation of k-ary constraints into binary constraints does not affect the existing preference assignments. That is, the existing preference assignments remain, and no preference valuations are assigned to the values of the domain of the attributes introduced by the translation.

Note that the activity constraints are also not guaranteed to be binary. However, the arity of activity constraints is not expected to significantly affect the relative performance of the different solution techniques presented in this chapter. The main reason for this is that the activity constraints are not resolved through constraint satisfaction but by means of constraint propagation. An attribute is not activated (or assigned) by any of the algorithms until the emerging solution allows activation. Therefore, the moment of activation of an attribute only depends on the restrictions imposed on it, not

on the form of the constraints that are employed to express these restrictions. Also, none of the algorithms employ techniques that depend on the arity of the activity constraints, such as FC or MPB (these approaches are only affected by the arity of the compatibility constraints).

The experiments presented in [4] suggest that substantial efficiency gains can be obtained in certain solution algorithms (FC in particular) by translating a non-binary CSP into a binary one. It is not clear, however, that these results extend to DCSPs and DPCSPs. In particular, in the case of MPB, translating non-binary constraints into binary ones may decrease performance. Because the constraints over k attributes (with $k > 2$) are broken up into binary constraints involving a new attribute that does not contribute to the overall preferences, less preference boundaries are likely to be detected. Generally speaking, however, more research is needed in order to reach conclusive results on this issue.

6.4.2 Alternative solution techniques

In order to discuss the efficiency of the DPCSP solution algorithms in relation to solution algorithms devised for other types of CSP, it is important to understand the characteristics of the DPCSP. As explained above, a solution to a DPCSP is a set of attribute value assignments that satisfies the activity and compatibility constraints and optimises the combination of the preferences of attribute value assignments.

Conventional approaches to such problems (which are commonly used in good existing solution algorithms) employ a *branch and bound* (B&B) strategy [81, 169]. B&B algorithms require (1) a mechanism to divide the solution space into regions and (2) an algorithm to compute a upper bound on the quality (i.e. lowest cost or highest preference) of all solutions in a region by solving a subproblem. As soon as one feasible, but not necessarily optimal, solution is found for the problem as a whole, a lower bound (i.e. minimal preference or maximal cost) on the quality of the optimal solution is established. All solutions in regions with an upper bound that is lower than the current lower bound can be removed from consideration.

When applying B&B to CSPs, a region in the solution space is typically defined by the set of attribute-value assignments: i.e. the region is the set of all solutions that are supersets of the specified set of attribute-value assignments. The aim of the CSP is to

find a solution of maximal preference or minimal cost, where the combined preference is defined as the minimum of the constituent preferences or where the combined cost is the maximum of the constituent costs. As such, the combined preference and cost of a solution to part of the CSP (i.e. a subset of the attributes and the corresponding domains and constraints) are respectively an upper bound on the preference and a lower bound on the cost of the overall solution. As such, upper bounds of the quality of solutions within a certain region are easily computed for partial assignments of the attributes. This principle is widely used in existing approaches.

In valued CSPs [157] and semiring-based CSPs [11], this approach is used to extend local consistency techniques to minimise the degree to which constraints are violated. Examples of CSPs that are instances of both valued and semiring-based CSPs are the conventional hard CSP, the possibilistic CSP [156], the fuzzy CSP [43, 128] and the partial CSP [61, 62]. In general, the concepts underlying local consistency, i.e. k-consistency and strong k-consistency of a CSP, are defined as follows.

Definition 6.8. If a CSP is defined such that, for every consistent set S of domain-value assignments to $k-1$ attributes with a quality valuation v, there exists a consistent set S', with $S \subset S'$, of domain-value assignments to k attributes such that S' has a quality valuation of v, then that CSP is called *k-consistent*. A CSP is called strongly k-consistent if it is j-consistent for all values $j \leq k$.

An algorithm that enforces strong k-consistency can find a solution, of some quality valuation, to a CSP with constraints restricting the assignments of up to k attributes, without backtracking. This idea could be successfully applied to DPCSPs by using the potential preference to evaluate the quality of partial solutions. However, as discussed before, computing the potential preference of a partial solution is as complex as solving the DPCSP itself, and hence it is not practical as a part of a solution algorithm.

In dynamic flexible CSPs [118], B&B is used to develop a flexible version of the local repair approach to constraint satisfaction. Flexible local repair (FLR) attempts to assign subsequent attributes such that consistency and a certain minimal quality valuation (initially, the maximal quality valuation) is retained. If this fails, FLR tries to repair the current partial solution by reassigning those attributes whose assignment is causing the failure to retain consistency or the current minimal quality valuation. This procedure may involve lowering the current minimal quality valuation to be upheld. Note that FLR is crucially dependent upon a mechanism to reassign attributes in a

way that retains a minimal quality valuation. In its current version, this implies that FLR requires an idempotent operator $\oplus$. An interesting piece of future research would be to investigate whether evaluation functions exist that allow FLR to be extended to DPCSPs.

B&B can also be employed in the context of depth first search (DFS). In itself, DFS leads to the so-called British Museum search, which requires exploring all possible solution as well, and retaining the best ones. B&B can improve the the efficiency of this approach. As soon as a solution node n_s is found, it can be used to prune the paths that can not lead to improvements over the existing solution. A path can not lead to an improvement over the existing solution if it ends with a node n whose potential preference estimate $\widehat{PP}(n)$ is lower that $CP(n_s)$.

It can be easily shown, however, that if a solution node n_s can be found, then an A* algorithm will not explore paths that end in a node n, such that $\widehat{PP}(n) \prec CP(n_s)$ any further. Indeed, it follows from definition 6.1 and theorems 6.3, 6.4, 6.6 and 6.7 that for any node n', for which $S(n') \subseteq S(n_s)$, $\widehat{PP}(n') \succcurlyeq PP(n') \succcurlyeq CP(n_s)$. Consequently, any node n' will be stored ahead of n in the priority queue O, and when n_s is reached, the priority queue need not be explored any further. Therefore, the A* algorithm is guaranteed to require no more nodes and consistency checks to find the optimal solution then a DFS with B&B [72]. Because an A* algorithm has higher memory requirements than DFS, however, it might be useful in certain cases to combine A* with DFS. But such a hybrid approach is beyond the scope of this dissertation.

Overall, the efficiency of an A* algorithm depends on the informedness of its heuristic and computational complexity of calculating the heuristic. Four approaches have been discussed in this chapter. Because all of these approaches make a different trade-off between informedness and computational complexity of the heuristic, a theoretical comparison of the efficiency of the four A* algorithms is not possible. Chapter 7 will present some experimental results on this issue.

Finally, it must be pointed out that the development of solution algorithms for DPCSPs are an ongoing research issue. Still, further improvements to the existing algorithms, as discussed in sections 9.2.1 and 9.2.2, can be explored and radically different approaches, such as the genetic algorithms mentioned in section 9.2.3, can be tried.

6.5 Summary

This chapter has presented solution techniques for dynamic preference constraint satisfaction problems (DPCSP). Four algorithms for solving the DPCSP have been introduced and illustrated these by means of a sample configuration problem that utilises the aforementioned preference calculus. Because the preference calculus' combination operator is not idempotent, local optimisation techniques such as local repair are not feasible. The basic algorithm is a variation on the A* algorithm and employs an informed estimate of the preference (called potential preference) that can be reached to guide the search. In order to avoid local optima, the potential preference is guaranteed to be an upper boundary on the realisable preference that monotonically decreases as additional attributes are assigned. The three additional algorithms are elaborations on the first that trade additional processing for a better estimate for potential preference. To that end, one approach uses forward checking whereas another (called maintaining preference boundaries) attempts to learn from failing to realise potential preference due to compatibility constraints. The final algorithm is the combination of both forward checking and maintaining preference boundaries.

Chapter 7

Experimental Evaluation of Constraint Satisfaction Techniques

In this chapter, the performance of the solution techniques introduced in the last chapter is analysed by running experiments with batches of different configurations of randomly generated DPCSPs. The aim of this study is to examine how the different algorithms compare in terms of time and space requirements and what the benefits are of the features specific to DPCSPs, i.e. activity constraints and order of magnitude preferences. To focus on the use of DPCSP for model construction, a detailed and complete analysis of the algorithms and DPCSPs in general is left out of the scope of this work. Instead, this chapter will concentrate on the most significant features of DPCSP and evaluate how these affect the performance on the solution algorithms. It is hoped that this information will guide future work in the development of new solution strategies for DPCSPs and help avoid less promising avenues of research.

The remainder of this chapter is organised as follows. Section 7.1 presents the experimental setting, section 7.2 describes the results obtained through these experiments and section 7.3 summarises the chapter.

7.1 Experimental setting

As explained earlier, a DPCSP is a novel type of CSP. It is therefore difficult to compare the corresponding solution algorithms with existing approaches. Instead, a small comparative empirical study was done, which focussed on the four algorithms that

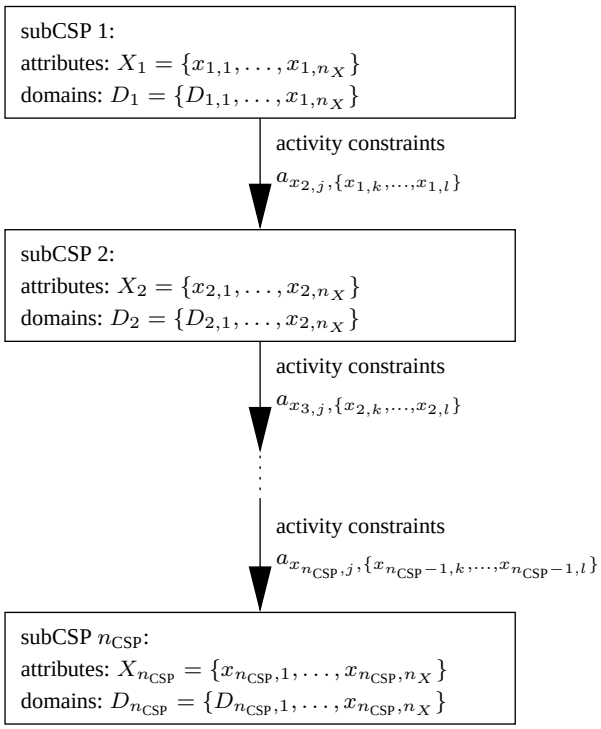


Figure 7.1: Problem instance structure

have been presented in chapter 6.

The subject of this study consists of randomly generated problem instances. The advantage of this approach is that the conclusions drawn from such a study are not biased by the chosen compositional modelling problem domains. When creating random DPCSPs, experimental design choices are made such that the family from which problem instances are sampled can be described in terms of certain parameters. This section discusses the way in which these DPCSP instances were generated and how they relate to DPCSPs representing possible compositional modelling problems.

7.1.1 The template of the random DPCSP instances

The randomly generated problem instances all conform to the structure presented in figure 7.1. That is, each DPCSP instance consists of a number of subCSPs, where each subCSP contains a set of attributes and a domain for each attribute. The subCSPs are totally ordered in the sense that activity constraints describe how partial assignments of the attributes of one subCSP activate attributes in the next subCSP. More formally,

activity constraints $a_{x_{i+1,j},\{x_{i,k},\ldots,x_{i,l}\}}$ govern the way in which a set of n_a attributes in subCSP i $\{x_{i,k},\ldots,x_{i,l}\}$ activate an attribute $x_{i+1,j}$ in subCSP $i+1$. The attributes of the first subCSP (i.e. attribute set X_1 and domain set D_1) are all active.

The generation of problem instances is governed by the following parameters:

- n_{CSP} is the number of subCSPs:

$$\mathbf{X} = \{X_1, X_2, \ldots, X_{n_{\text{CSP}}}\}$$
$$\mathbf{D} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_{n_{\text{CSP}}}\}$$

- n_X is the number of attributes per subCSP:

$$X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,n_X}\}$$
$$\mathbf{D}_i = \{D_{i,1}, D_{i,2}, \ldots, D_{i,n_X}\}$$

- n_D is the number of values per attribute domain:

$$D_{i,j} = \{d_{i,j,1}, d_{i,j,2}, \ldots, d_{i,j,n_D}\}$$

- n_a is the number of attributes in each activity constraint. As such, each activity constraint $a_{x_{i+1,j},\{x_{i,k},\ldots,x_{i,l}\}}$ is a mapping

$$a_{x_{i+1,j},\{x_{i,k},\ldots,x_{i,l}\}} : D_{i,k} \times \ldots \times D_{i,l} \mapsto \{\text{active}(x_{i+1,j}), \top\}$$

  where the cardinality of $\{x_{i,k},\ldots,x_{i,l}\}$ is n_a. Note that, if the activity constraint maps an assignment $x_{i,k} : d_{i,k,m_k}, \ldots, x_{i,l} : d_{i,l,m_l}$ onto $\text{active}(x_{i+1,j})$, then $x_{i+1,j}$ is active under that assignment. Otherwise, the assignment has no effect on $x_{i+1,j}$, but another activity constraint may still activate $x_{i+1,j}$.

- p_{aX} is the probability that for a given attribute $x_{i+1,j} \in X_{i+1}$ and a set containing n_a attributes $x_{i,k} \in X_i, \ldots, x_{i,l} \in X_i$, an activity constraint $a_{x_{i+1,j},\{x_{i,k},\ldots,x_{i,l}\}}$ exists. p_{aX} will be referred to as the *density* of the activity constraints.

- p_{aD} is the probability that for a given activity constraint $a_{x_{i+1,j},\{x_{i,k},\ldots,x_{i,l}\}}$, an assignment to $x_{i,k},\ldots,x_{i,l}$ activates $x_{i+1,j}$. That is,

$$P(a_{x_{i+1,j},\{x_{i,k},\ldots,x_{i,l}\}}(x_{i,k} : d_k, \ldots, x_{i,l} : d_l) = \text{active}(x_{i+1,j})) = p_{aD}$$

  where $d_k \in D_{i,k}, \ldots, d_l \in D_{i,l}$. p_{aD} will be referred to as the *tightness* of the activity constraints.

- n_c is the number of attributes involved in each compatibility constraint. Each compatibility constraint is a mapping:

$$c_{\{x_{i,j},\ldots,x_{i,k}\}} : D_{i,j} \times \ldots \times D_{i,k} \mapsto \{\top, \bot\}$$

  where the cardinality of $\{x_{i,j},\ldots,x_{i,k}\}$ is n_c.

- p_{cX} is the probability that for a given set of n_c attributes $\{x_{i,j},\ldots,x_{i,k}\}$, a compatibility constraint $c_{\{x_{i,j},\ldots,x_{i,k}\}}$ exists. p_{cX} will be referred to as the *density* of the compatibility constraints.

- p_{cD} is the probability that in a given compatibility constraint $c_{\{x_{i,j},\ldots,x_{i,k}\}}$, an assignment $x_{i,j} : d_{i,j,m_j},\ldots,x_{i,k,m_k}$ is deemed inconsistent. That is,

$$P(c_{\{x_{i,j},\ldots,x_{i,k}\}}(x_{i,j} : d_{i,j,m_j},\ldots,x_{i,k,m_k}) = \bot) = p_{cD}$$

  p_{cD} will be referred to as the *tightness* of the compatibility constraints.

- $\mathbb{P}$ is the set of OMPs ascribed to attribute-value assignment.

- $p_{\mathbb{P}}$ is the probability that an attribute-value assignment has a corresponding OMP.

7.1.2 Random DPCSPs and real-world compositional modelling

The structure of the random DPCSP instances is motivated by the need to specify the DPCSP configuration in terms of a limited set of parameters, and by the requirement that they reflect the DPCSPs which may be derived from compositional modelling problems. Because these two considerations have, to some extent, opposite requirements, a trade-off has been made between them. Overall, for any DPCSP instance that can be constructed from a compositional modelling problem, an equivalent DPCSP instance can be randomly generated with the following exceptions:

- The cardinality of the domains (n_D) and the number of attributes in the constraints (n_a and n_c) of the randomly generated DPCSPs are fixed.

- The subCSPs in the randomly generated DPCSPs contain an equal number of attributes.

- Each activity constraint $a_{x_{i+1,j},X'}$ specifies the conditions for activation of an attribute $x_{i+1,j}$ in terms of assignments of attributes $x_{i,k} \in X' \subseteq X_i$. That is, each activity constraint specifies a partial assignment of attributes in subCSP i that activate an attribute in subCSP $i+1$. In general, the DPCSP instances generated from a compositional modelling problem contain activity constraints of the form $a_{x_{i+1,j},X'}$ where $X' \subseteq (X_1 \cup \ldots \cup X_i)$. However, the tightness of the activity constraints will vary with the distribution of the numbers of attributes $x' \in X'$ taken from each set $X_1, \ldots, X_i$. Since all the algorithms employ the information contained in the activity constraints in the same way, this source of variability in the random DPCSP instances has been avoided. In any case, a set of activity constraints of the form $a_{x_{i+1,j},X'}$ where $X' \subseteq (X_1 \cup \ldots \cup X_i)$, can be reformulated into a set of activity constraints $a_{x_{i+1,j},X'}$ where $X' \subseteq X_i$, by introducing additional attributes and activity constraints.

7.1.3 Performance measurement

To evaluate the performance of the algorithms on different configurations of DPCSP, the time and space requirements must be measured during the experimental runs. One common approach of measuring these involves monitoring the runtime and memory usage of the algorithms. However, it is technically difficult to do this consistently and to isolate the performance of the algorithm from outside influences. Both runtime and memory usage may be affected by the implementation of the algorithms, the design of the programming language, external processes, and processes supporting the algorithm (e.g. garbage collection and loading and unloading classes). Because of the implementational problems involved, the alternative approach of recording performance data during the experimental runs was chosen.

To measure the *time requirements* of the algorithms, the total number of consistency checks to find the first solution was counted. This heuristic is often used in evaluating the time requirements of CSP solution algorithms because these algorithms essentially explore the search space by checking the consistency of partial solutions. The main difference between algorithms lies in the manner in which they organise the consistency checks. For example, a forward checking (FC) based algorithm will perform more consistency checks per explored node, but it will usually visit fewer nodes.

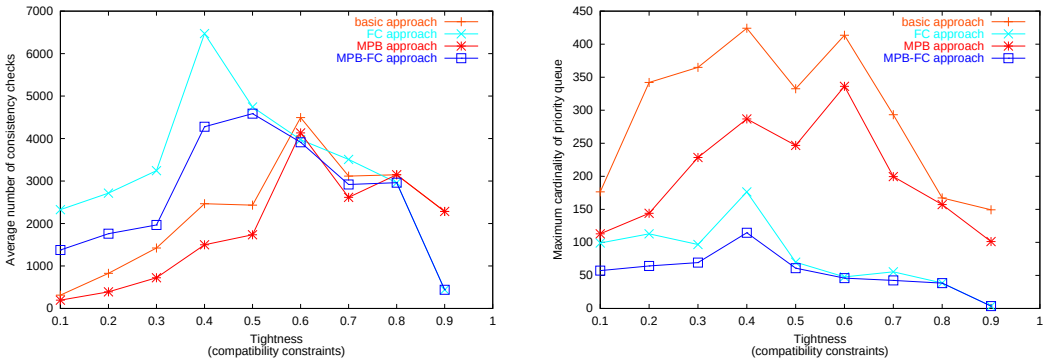
The maintaining preference boundaries (MPB) based algorithms also perform preference boundary lookups in addition to extra consistency checks per node (to determine a preference boundary). However, the cost of these preference boundary lookups is negligible compared to the cost of consistency checks. The cost of these types of operation respectively depends on the number of preference boundaries and number of constraints. The former is proportional to the number of attribute combinations available, whereas the latter is proportional to the number of attribute-value assignments over an arbitrary number of values (because they are derived from the label of the ALTMS nogood node). Furthermore, preference boundaries are established in response to one or more compatibility constraints that prevent the assignment of preferred domain values to certain attributes. Consequently, the number of preference boundaries is very small compared to the number of constraints.

Another factor that the proposed measurement of time requirements potentially ignores is the application of activity constraints when the solution algorithm runs out of active unassigned attributes. However, this operation is only performed whenever a number of attributes has been assigned. Thus, it is also negligible with respect to the total number of consistency checks.

To measure the *space requirements* of the algorithms, the maximum number of nodes in the priority queue was maintained. As with time requirements, the space requirements of the algorithms will be proportional to the maximum size of the priority queue. In addition to the priority queue, the MPB based algorithms also require storage space for the discovered preference boundaries. These are again proportional to the number of attribute combinations whereas the size of priority queue becomes proportional to the size of the solution space for the most complex problems. Therefore, the space requirements for the preference boundaries are very small compared to the space requirements imposed by the priority queue and they can safely be ignored.

7.2 The results

The solution techniques presented in the previous chapter are evaluated by generating batches of random DPCSPs with different values for the parameters n_{CSP}, n_X, n_D, n_a, p_{aX}, p_{aD}, n_c, p_{cX}, p_{cD}, $\mathbb{P}$ and $p_{\mathbb{P}}$. By analysing how the performance of the solution techniques evolve as certain parameter values change, some conclusions can be drawn



(a) Time requirements

(b) Space requirements

Figure 7.2: The effect of compatibility constraint tightness under density 0.25 [$n_{\text{CSP}} = 1$, $n_X = 10$, $n_D = 4$, $n_c = 2$, $p_{cX} = 0.25$, p_{cD} is variable, $|\mathbb{P}| = 5$, $p_p = 1$]

on the suitability of the different techniques in different problem settings.

7.2.1 The effect of DPCSP configuration

Two issues are looked at here: the constrainedness of compatibility constraints and that of activity constraints.

7.2.1.1 Constrainedness imposed by compatibility constraints

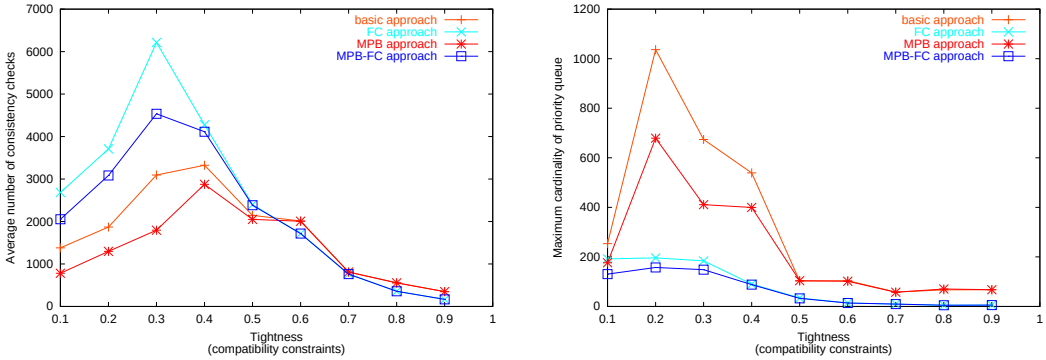
In the first set of experiments, the effect of the tightness and density of compatibility constraints is studied. To this end, the performance of the solution techniques for batches of 50 random DPCSPs under different parameter settings is studied. To avoid the potential confusion between the effect of compatibility constraints and the effect of activity constraints, only a single subCSP is used, i.e. $n_{\text{CSP}} = 1$, and the parameter values for n_a, p_{aX} and p_{aD} are irrelevant since all attributes are active anyway.

Overall, the evolution of time and space requirements in function of compatibility constraint tightness appears to follow the similar pattern for all four algorithms, peaking between the tightness values 0.4 and 0.6. The most significant differences are exhibited between the forward checking based approaches, i.e. FC and MPB-FC based improvements, and the others, i.e. the basic algorithm and the MPB based improvement.

In terms of space requirements (as measured by the maximum number of nodes in the priority queue, averaged over 50 random DPCSPs), the forward checking based approaches significantly outperform other two. The number of nodes in the priority queue is proportional to the number of nodes that are visited (provided $n_D \geq 2$) and to the number of child nodes created per visit. That is, each visit of a node involves removing it from the priority queue and adding its child nodes to the priority queue (up to n_D nodes, if all of the n_D domain values of the next unassigned active attributes are consistent with the current partial solution). Hence, if $n_D \geq 2$, node visits will tend to increase the size of the priority queue. Thanks to the relative informedness of the heuristic employed in the different algorithms, FC visits fewer or an equal number of nodes than the basic algorithm and MPB-FC visits fewer or an equal number of nodes than MPB. In addition, the forward checking based approaches may generate less child nodes because some assignments of the next unassigned active attribute may be inconsistent with all the assignments in the domain of another unassigned active attribute. Note that the lower space requirements of the forward checking approaches provide an important advantage in BFS approaches. Of course, this may be expected intuitively..

The comparison of time requirements is less clear-cut, but the partitioning into forward checking and other approaches can remain. For relatively low values of p_{cD}, the forward checking approaches require significantly more consistency checks than the others and only for very high values of p_{cD} do the forward checking approaches provide a real benefit.

Forward checking approaches are normally used as a weak form of local consistency checking in order to fail early the paths that lead to an inconsistency, generated by a depth-first search (DFS) strategy fail early. In DFS, every time a path leading to inconsistency is detected early by forward checking, a reduction is achieved on the total number of nodes visited. The benefit of forward checking in the proposed BFS approach depends on the realisation of improvements of informedness. Here, the early discovery of paths leading up to inconsistency is only beneficial if the corresponding improvement of the $\widehat{PP}(n)$ heuristic causes the node n to be put so far back in the priority queue that it is unlikely to be reconsidered. In addition, forward checking requires a significantly higher amount of consistency checking per node. Therefore, for the forward checking approaches to provide a real benefit, sufficiently fewer nodes must



(a) Time requirements

(b) Space requirements

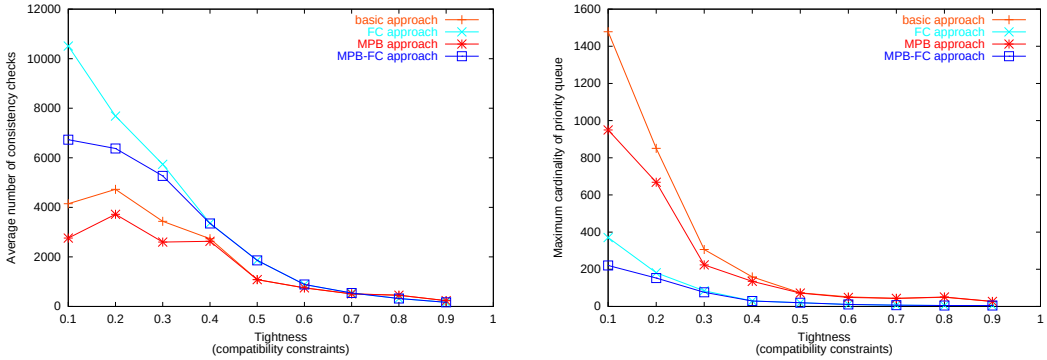
Figure 7.3: The effect of compatibility constraint tightness under density 0.50 [$n_{\text{CSP}} = 1$, $n_X = 10$, $n_D = 4$, $n_c = 2$, $p_{cX} = 0.50$, p_{cD} is variable, $|\mathbb{P}| = 5$, $p_p = 1$]

be visited to offset the additional computation effort required per node.

The results summarised in figure 7.2 indicate that the improvements realised by forward checking based approaches within a DFS strategy do not necessarily translate to BFS strategies. Although forward checking tends to visit fewer nodes and to have lower space requirements than non forward checking based approaches, it generally requires more computational effort (except in highly constrained problems).

The results also indicate that for lower values of compatibility constraint tightness ($p_{cD} < 0.5$) the MPB based approaches have an important efficiency gain over the other two approaches. As with the FC based improvement, the MPB based algorithms can reduce the time and space requirements by increasing the informedness of the $\widehat{PP}(n)$ heuristic. However, the benefit of a more informed $\widehat{PP}(n)$ heuristic is only realised for nodes n whose corresponding partial solution $S(n)$ is not inconsistent with the compatibility constraints. Because the chances of finding such nodes decrease as the tightness of compatibility constraints increases, the benefits of these more informed heuristics are only realised for lower values of p_{cD}.

The results for different values of density of compatibility constraints follow the same pattern shown in figure 7.2. This is illustrated in figures 7.3 and 7.4, which plot time and space requirements against compatibility constraint tightness for values of compatibility constraint density of respectively $p_{cX} = 0.5$ and $p_{cX} = 0.75$. For higher values of p_{cX}, the peak values of time requirements correspond to lower values of



(a) Time requirements

(b) Space requirements

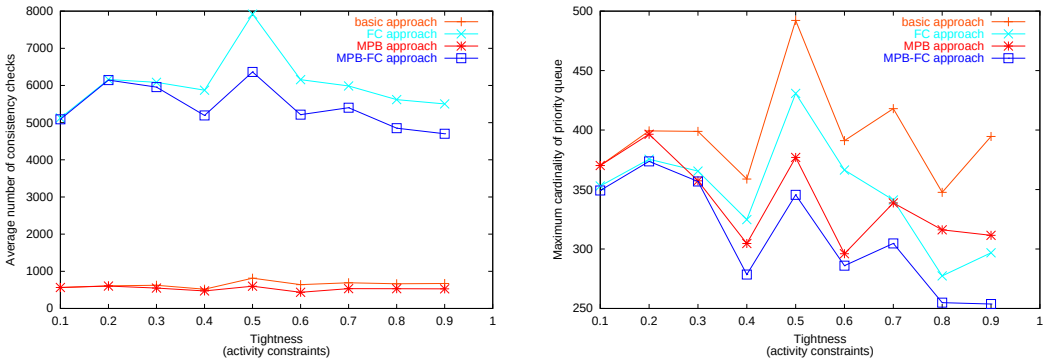
Figure 7.4: The effect of compatibility constraint tightness under density 0.75 [$n_{\mathrm{CSP}} = 1$, $n_X = 10$, $n_D = 4$, $n_c = 2$, $p_{cX} = 0.75$, p_{cD} is variable, $|\mathbb{P}| = 5$, $p_p = 1$]

compatibility constraint tightness. This observation is commonly made in the present experiments as the time and space requirements are affected by the overall constrainedness of the CSP and the constrainedness is proportional to both tightness and density of the constraints.

7.2.1.2 Constrainedness imposed by activity constraints

The second set of experiments aims to investigate the effect of activity constraints on the time and space requirements over finding a DPCSP solution. The main difference between the processing of activity constraints and that of compatibility constraints lies in the fact that the former is resolved through constraint propagation whereas the latter is resolved through actual constraint satisfaction. Indeed, each algorithm presented in chapter 6 fires the implications representing activity constraints whenever the algorithm runs out of active unassigned attributes. However, in order to find sets of attribute-value assignments that are consistent with the compatibility constraints, these sets, which are representing partial solutions, must be generated and tested.

In general, constraint propagation is more efficient than constraint satisfaction. The effect of applying activity constraints upon the time and space, required in solving a given problem, is limited to the way in which they reconfigure in response to attribute-value assignments. Low values of activity constraint tightness and density bring down the likelihood that an attribute and the compatibility constraints it is involved in, are



(a) Time requirements

(b) Space requirements

Figure 7.5: The effect of activity constraint tightness [$n_{\text{CSP}} = 2$, $n_X = 4$, $n_D = 4$, $n_a = 2$, $p_{aX} = 0.25$, p_{aD} is variable, $n_c = 2$, $p_{cX} = 0.25$, $p_{cD} = 0.25$, $|\mathbb{P}| = 5$, $p_p = 1$]

activated. This reduces the size of search space and the constrainedness of the problem as well as the size of the solution space. Generally speaking, the former makes the problem less complex, whereas the latter makes the problem more complex. It is therefore difficult to predict the general impact of tightness and density of activity constraints.

Figure 7.5 plots the time and space requirements against p_{aD}, as measured whilst solving batches of 50 random DPCSPs generated with the following parameters: $n_{\text{CSP}} = 2$, $n_X = 4$, $n_D = 4$, $n_a = 2$, $p_{aX} = 0.25$, $n_c = 2$, $p_{cX} = 0.25$, $p_{cD} = 0.25$, $|\mathbb{P}| = 5$, $p_p = 1$. From these results, no significant relation can be established between the tightness of the activity constraints and the complexity of the problem. This observation is consistent with the two opposite effects of changing the constrainedness imposed by the activity constraints upon the problem complexity, which were discussed earlier. The current results indicate that the impact of the former upon the latter seems to be relatively small, although this result might vary when far greater sample sets were used (which itself remains a piece of future work).

The results do suggest a relation between the tightness of the activity constraints and the benefits of the MPB approaches. More specifically, the differences in time and space requirements between the basic and the MPB approaches and those between the FC and the MPB-FC approaches increase in proportion to the tightness of the activity constraints (up to a maximum at around $p_{aD} = 0.5$). By definition, the likelihood

that certain sets of attributes are activated increases as p_{aD} becomes greater. As a result, the amount of thrashing due to overly optimistic estimates of the preference that may be caused by being able to activate additional attributes also increases with p_{aD}. Because the MPB approaches are well suited to improve preference estimates for potential assignments of sets of not yet activated attributes, their positive effects on time and space requirements become more pronounced with larger values of p_{aD}.

7.2.2 Advantages of specifying a problem as a DPCSP

Again, two issues are examined here: the utility of activity constraints and that of order-of-magnitude preferences.

7.2.2.1 The utility of activity constraints

Each DPCSP can be specified as a non-dynamic preference constraint satisfaction problem (PCSP). A PCSP can be defined as a DPCSP where all attributes are already active without the involvement of activity constraints. In other words, a PCSP is specified by:

- A set of n attributes $\mathbf{X} = \{x_1, \ldots, x_n\}$,
- A set of n domains $\mathbf{D} = \{D_1, \ldots, D_n\}$, one for each attribute,
- A set of compatibility constraints $\mathbf{C}$ where each $c_{\{x_i,\ldots,x_j\}}$ is a mapping:

$$c_{\{x_i,\ldots,x_j\}} : D_i \times \ldots \times D_j \mapsto \{\top, \bot\}$$

- A preference assignment $P(x_i : d_{ij})$ for each attribute value assignment.

Each DPCSP can be translated into a PCSP: The notion of inactivity of an attribute can be represented by means of a value in the domain of that attribute and the activity constraints can be translated into compatibility constraints. There are, however, significant advantages in using activity constraints as they are substantially easier to resolve than the corresponding compatibility constraints.

To demonstrate this, consider the random DPCSPs defined in section 7.1.1. These DPCSPs can be translated into a non-dynamic preference CSPs as follows:

- Each domain $D_{i,j}$ is replaced by $D'_{i,j} = D_{i,j} \cup \{\neg\text{active}(x_{i,j})$,

- Let $a_{x_{i+1,j}}$ be the function that maps each assignment of the attributes of a subCSP i into a predicate whether $x_{i+1,j}$ is active or not. More formally, $a_{x_{i+1,j}}$ is a function:

$$a_{x_{i+1,j}} : D_{i,1} \times \ldots \times D_{i,n_X} \mapsto \{\text{active}(x_{i+1,j}), \neg\text{active}(x_{i+1,j})\}$$

such that:

$$a_{x_{i+1,j}}(d_{i,1,k_1}, \ldots, d_{i,n_X,k_{n_X}}) = \begin{cases} & \text{if } \exists\{d_{i,g,k_g}, \ldots, d_{i,h,k_h}\} \subseteq \{d_{i,1,k_1}, \ldots, d_{i,n_X,k_{n_X}}\}, \\ \top & \quad \exists a_{x_{i+1,j},\{x_{i,g},\ldots,x_{i,h}\}}, \\ & \quad a_{x_{i+1,j},\{x_{i,g},\ldots,x_{i,h}\}}(d_{i,g,k_g}, \ldots, d_{i,h,k_h}) = \text{active}(x_{i+1,j}) \\ \bot & \text{otherwise} \end{cases}$$

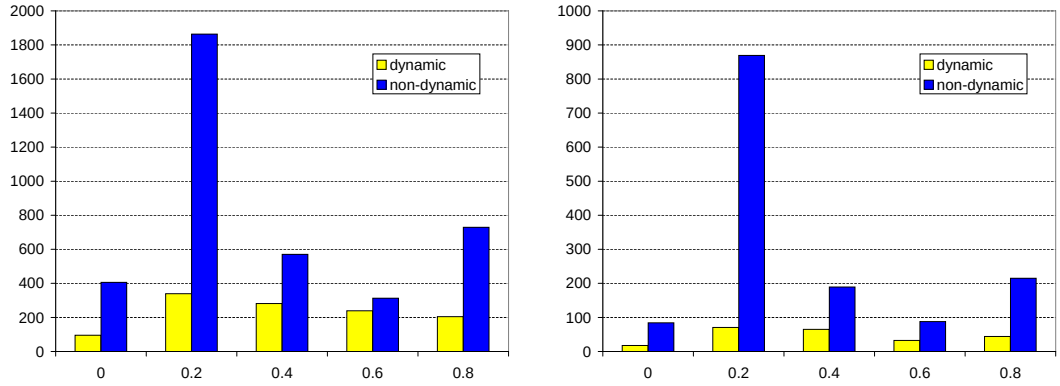
Given such a function, each set of activity constraints $\{a_{x_{i+1,j},X_{ij}} \mid X_{ij} \subseteq X_i\}$ is replaced by a compatibility constraint $c_{X_i \cup \{x_{i+1,j}\}}$:

$$c_{X_i \cup \{x_{i+1,j}\}} : D'_{i,1} \times \ldots \times D'_{i,n_X} \times D'_{i+1,j} \mapsto \{\top, \bot\}$$

such that:

$$c_{X_i \cup \{x_{i+1,j}\}}(d_{i,1,k_1}, \ldots, d_{i,n_X,k_{n_X}}, d_{i+1,j,l}) = \begin{cases} \bot & \text{if } a_{x_{i+1,j}}(d_{i,1,k_1}, \ldots, d_{i,n_X,k_{n_X}}) = \\ & \quad \text{active}(x_{i+1,j}) \wedge d_{i+1,j,l} = \neg\text{active}(x_{i+1,j}), \\ \bot & \text{if } a_{x_{i+1,j}}(d_{i,1,k_1}, \ldots, d_{i,n_X,k_{n_X}}) = \\ & \quad \neg\text{active}(x_{i+1,j}) \wedge d_{i+1,j,l} \neq \neg\text{active}(x_{i+1,j}), \\ \top & \text{otherwise} \end{cases}$$

In general, it is much more efficient to solve a DPCSP than it is to solve the equivalent PCSP. The activity constraints of a DPCSP are resolved through a simple constraint propagation procedure whereas the equivalent compatibility constraints in a PCSP are resolved through constraint satisfaction. More specifically, when an activity constraint is checked, it may activate an attribute or have no effect at all. When compatibility constraints are used instead of activity ones, the attributes are assigned in a random order. Checking a compatibility constraint may result in an inconsistency, in which case the exploration of the set of assignments leading up to the inconsistency



(a) Time requirements

(b) Space requirements

Figure 7.6: Dynamic vs. non-dynamic preference CSPs [$n_{\text{CSP}} = 2$, $n_X = 4$, $n_D = 4$, $n_a = 2$, $p_{aX} = 0.25$, p_{aD} is variable, $n_c = 2$, $p_{cX} = 0.25$, $p_{cD} = 0.25$, $|\mathbb{P}| = 5$, $p_p = 1$]

would be useless. As such, solving a PCSP is expected to involve a significant number of additional consistency checks compared to the equivalent DPCSP.

To empirically validate this, the performance of a set of randomly generated DPCSPs is compared against a set of equivalent PCSPs. The equivalence of the PCSP with the randomly generated DPCSP is guaranteed by the translation procedure presented earlier in this section. Figure 7.6 compares the performance of the algorithms on batches of 50 random DPCSPs and their equivalent PCSPs for different values of activity constraint tightness $p_{aD} \in \{0, 0.2, 0.4, 0.6, 0.8\}$. The other parameters employed for random DPCSP generation are $n_{\text{CSP}} = 2$, $n_X = 4$, $n_D = 4$, $n_a = 2$, $p_{aX} = 0.25$, $n_c = 2$, $p_{cX} = 0.25$, $p_{cD} = 0.25$, $| \mathbb{P} | = 5$, and $p_p = 1$.

Figures 7.6(a) and 7.6(b) respectively plot the average number of consistency checks and the maximum number of nodes in the priority queue for the batches of 50 DPCSPs and PCSPs with different p_{aD} values. As seen before, the performance of the DPCSP solution algorithms is not significantly affected by different values of $p_{aD} > 0$ because the activity constraints are resolved through constraint propagation rather than constraint satisfaction. In the PCSPs, these activity constraints are expressed by means of equivalent compatibility constraints, which are resolved through constraint satisfaction. This has two consequences. Firstly, the BFS algorithm is clearly far more efficient when activity constraints are used (i.e. in the DPCSPs), in terms of both time and

space complexity. Secondly, when solving the equivalent PCSPs, the time and space requirements vary significantly with different values of p_{aD}. The latter phenomenon results from the translation of activity constraints into compatibility constraints: the values of p_{aD} in the DPCSP affect the compatibility constraint tightness in the equivalent PCSPs, which in turn causes the fluctuations in time and space complexity of the equivalent PCSPs.

7.2.2.2 The utility of OMPs

The essential benefit of using the OMP calculus introduced in chapter 5 is that it provides a convenient way of expressing, combining and comparing imprecise and partially ordered preference valuations. However, the use of such partially ordered preferences rather than a totally ordered calculus may also reduce the time and space complexity required by the search algorithms to find a solution.

Indeed, if a calculus of totally ordered preference valuations were to be used to express partially ordered preferences in a DPCSP, then a total ordering would be artificially imposed over the space of consistent attribute value assignments. The totally ordered calculus would unnecessarily order certain sets of attribute-value assignments that are not comparable with one another. Due to these spurious orderings, the space of most preferred and consistent attribute-value assignments (i.e. the solution space for the problem) would be greatly reduced under the total ordering of preferences. Obviously, it is harder to find an individual optimal solution when a smaller proportion of the search space contains optimal solutions (and when the total size of the search space remains equal). Therefore, the search algorithms are expected to have higher time and space requirements if a total ordering of preferences is imposed.

To test this hypothesis, experiments were performed with 50 batches of random DPCSPs. The DPCSPs in each batch contain the same set of attributes, domains, activity constraints and compatibility constraints and they employ an equal number of distinct OMPs that were assigned to the attribute-value assignments. The OMPs in the different DPCSPs in the same batch were ordered differently with respect to one another.

To emulate the effect of partially ordering the OMPs, each DPCSP uses the same number of OMPs o, partitioned into a different number of scales s. Each scale defines a total ordering over a subset of the OMPs, unrelated to the OMPs that belong to

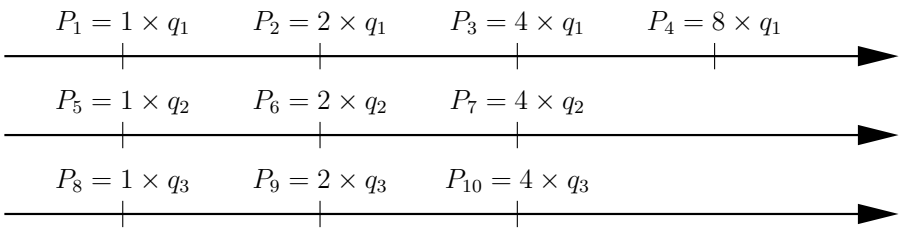
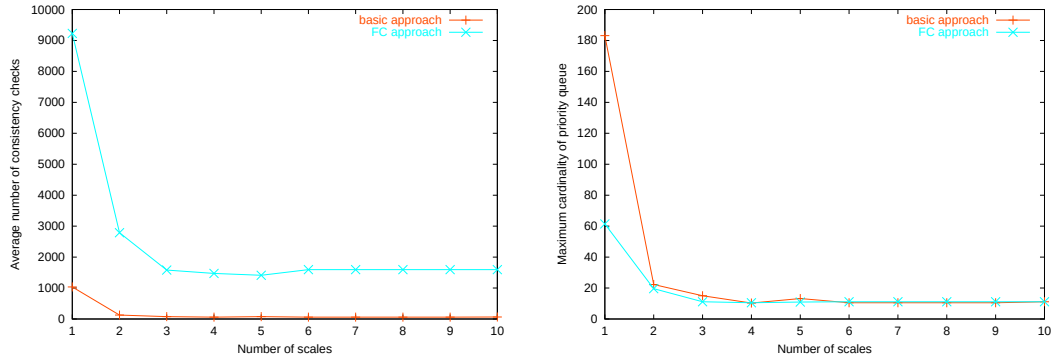


Figure 7.7: Sample random OMP scales



(a) Time requirements

(b) Space requirements

Figure 7.8: The effect of ordering preferences with orders of magnitude $n_{\text{CSP}} = 1$, $n_X = 8$, $n_D = 4$, $n_a = 2$, $p_{aX} = 0.25$, $p_{aD} = 0.25$, $n_c = 2$, $p_{cX} = 0.25$, $p_{cD} = 0.25$, $|\mathbb{P}| = 20$, $p_p = 1$

different scales. To ensure that the OMPs within a scale are totally ordered, they are defined as combinations of the same BPQ. That is, the i^{th} OMP in a scale is defined as $2^{(i-1)} \times q$, where q is the BPQ employed to define the OMPs in the scale. As such, all combinations of OMPs within the same scale are comparable with one another. Importantly, as the number of scales s increases, the set of OMPs becomes more partially ordered. An example may clarify this idea. If, for instance, $o = 10$ and $s = 3$, then the 10 OMPs (denoted $P_1, \ldots, P_{10}$) based on three BPQs (q_1, q_2, q_3) shown in figure 7.7 are created.

Figure 7.8 plots the time and space requirements of batches of 50 random DPCSPs against the number of scales in the experiment. The DPCSPs were generated with the following parameters: $n_{\text{CSP}} = 2$, $n_X = 8$, $n_D = 4$, $n_a = 2$, $p_{aX} = 0.25$, $p_{aD} = 0.25$, $n_c = 2$, $p_{cX} = 0.25$, $p_{cD} = 0.25$, $| \mathbb{P} | = 20$, $p_p = 1$. The number of scales s varied

from 1 to 10. The result, shown in figure 7.8 clearly confirm the earlier hypothesis: both time and space requirements drop very significantly as s increases from 1 to 3 (and a partial ordering of preferences of solutions is introduced. For values of s greater than 3, time and space requirements remain more or less constant, suggesting that a critical number of optimal consistent solutions has been reached. When the tightness and density of the constraints in the DPCSPs change, this critical value of s is likely to vary as well (but investigating this remains a piece of future work).

7.3 Summary

This chapter has analysed the performance of the solution algorithms of chapter 6 in solving batches of randomly generated DPCSPs. In the worst case, DPCSPs are very hard problems because most conventional approaches to improve efficiency are not applicable and, currently, few DPCSP specific approaches have been developed. The factors that primarily affect the time and space complexity in finding a single DPCSP solution have been identified in this chapter as the constrainedness (i.e. tightness and density) imposed by the compatibility constraints and the size of space of consistent solutions with equal or incomparable preference. The constrainedness imposed by the activity constraints has no significant effect on the average time and space requirements. In addition, this chapter has shown that there are considerable efficiency benefits to the use of activity constraints and a calculus of partially ordered preference in solving DPCSPs.

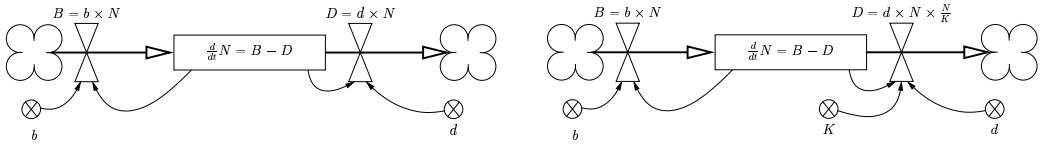
Chapter 8

Applying the Compositional Modeller

This chapter aims to demonstrate the applicability of the compositional modelling techniques presented in this dissertation to ecological model construction problems. To that end, knowledge bases for two problem domains have been constructed. First, the application of compositional ecological modelling to traditional population dynamics problems is considered. The corresponding knowledge base and its use are discussed in detail in section 8.1. Then, an application of compositional ecological modelling to the significantly larger domain of Mediterranean vegetation growth is presented. Given the enormous size of this domain, a significant and complex part of the domain knowledge has been selected for automation. The knowledge base that has been developed and the types of model that can be constructed with it are discussed in section 8.2.

8.1 Putting it all together: An application to population dynamics

The previous chapters could be read in relative isolation. This section presents an overview of the use of the compositional modeller in a particular problem domain. It employs some basic but historically important models taken from population dynamics, it shows how these are translated into a knowledge base of model fragments, and it discusses how a scenario model can be derived from this knowledge base, a given scenario and a set of preferences. Although the knowledge base that forms the subject of this section is small, it is capable of generating a scenario model for any feasible



(a) Exponential growth (b) Logistic growth

Figure 8.1: Population growth models

scenario of populations related via predation or competition relations.

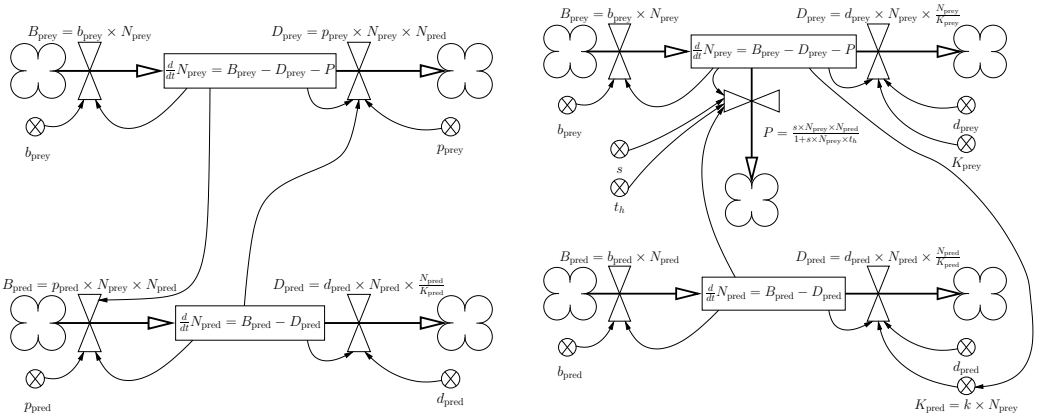
8.1.1 Population dynamics

Before showing how the compositional modeller automatically constructs mathematical simulation models from an ecological scenario, some of the most basic ecological models are introduced. These models will then be translated into sets of model fragments in the next subsection. The subsequent subsections will then demonstrate how this knowledge base can be employed to construct ecological models based on given scenarios.

Note that some of the models presented herein are somewhat simplistic. However, they are easy to understand and combine with one another to form complex models. More realistic system dynamics models, such as the ModMed *n*-species model to be discussed in section 8.2, work upon the same principles, but using equations that are more sophisticated and better justified. Employing the latter model directly to illustrate how the compositional modeller functions would unnecessarily complicate the discussions.

Models for three different phenomena are discussed in this subsection: population growth, predation and competition. Mathematical models of population growth are probably the first mathematical models of ecological systems in existence. Despite their age, these models still form an important building block in ecological models today. Two population growth models will be used here: exponential growth and logistic growth.

Exponential growth [112] introduced the principle that the size of a population of a species may increase according to a geometric series (i.e. $a + ar + ar^2 + \ldots + ar^n + \ldots$).



(a) Lotka-Volterra predation (b) Holling predation

Figure 8.2: Predation models

As such, the change in size in a population can be represented as:

$$\frac{d}{dt}N(t) = B(t) - D(t)$$
$$B(t) = b \times N(t)$$
$$D(t) = d \times N(t)$$

where $N(t)$, $B(t)$ and $D(t)$ respectively denote the population size, the number of births and the number of deaths at each time instance, b denotes the rate of births per time unit, and d denotes the rate of deaths per time unit. A system dynamics representation of this model is shown in figure 8.1(a).

Logistic growth [178] is based on the idea that population growth may depend upon population density. Population density is described in terms of a hypothesised maximum limit on the population size K, sometimes called the carrying capacity. Many variations on the logistic growth model exist, but for the sake of simplicity, the following model is employed herein:

$$B(t) = b \times N(t)$$
$$D(t) = d \times N(t) \times \frac{N(t)}{K}$$

A system dynamic representation of this model is shown in figure 8.1(a).

Predation models describe the mutual dependency of the sizes of two populations when one population feeds on the other in order to survive. The two most basic models

of predation are the Lotka-Volterra predation model [110, 179] and the predation model based on Holling's disc equation [79]. The *Lotka-Volterra predation model* employs the exponential growth model for the birth rate of the prey population and the death rate of the predator population. Both the death rate of the prey and the birth rate of the predators are deemed to be proportionate to their respective population sizes. The formal model specification, also shown in figure 8.2(a), is:

$$
\begin{aligned}
\frac{d}{dt}N_{\text{prey}}(t) &= B_{\text{prey}}(t) - D_{\text{prey}}(t) \\
B_{\text{prey}}(t) &= b_{\text{prey}} \times N_{\text{prey}}(t) \\
D_{\text{prey}}(t) &= p_{\text{prey}} \times N_{\text{prey}}(t) \times N_{\text{pred}}(t) \\
\frac{d}{dt}N_{\text{pred}}(t) &= B_{\text{pred}}(t) - D_{\text{pred}}(t) \\
B_{\text{pred}}(t) &= p_{\text{pred}} \times N_{\text{prey}}(t) \times \frac{N_{\text{pred}}(t)}{K_{\text{pred}}} \\
D_{\text{pred}}(t) &= d_{\text{pred}} \times N_{\text{pred}}(t)
\end{aligned}
$$

where p_{prey} and p_{pred} are factors respectively representing the number of prey deaths per prey per predator and predator births per predator per prey. Obviously, this model is a gross abstraction of reality. For example, reproduction of predators increases proportionally to available prey without taking into account the number of prey required by the predators to survive. Also, the overall death rate of prey is proportional to the number of predators, and hence, predation is considered the only cause of death for prey in this case. Nevertheless, the Lotka-Volterra model is employed in educational and small experimental settings where simplicity of the model is more important than its realism, as it is the case here to demonstrate the theories developed in this thesis in a managed manner.

The *Holling predation* model is designed to take into account the relative effort required to hunt and consume prey by the predators. Furthermore, as shown in figure 8.2(b), predation is seen as separate cause of death for prey, independent of death by natural causes. The available prey in the ecosystem to be modelled does not immediately affect birth or death rates of the predator population, but instead, it affects the capacity of the predator population that the ecosystem can sustain. As such, the Holling model requires that predator population growth is described by the logistic

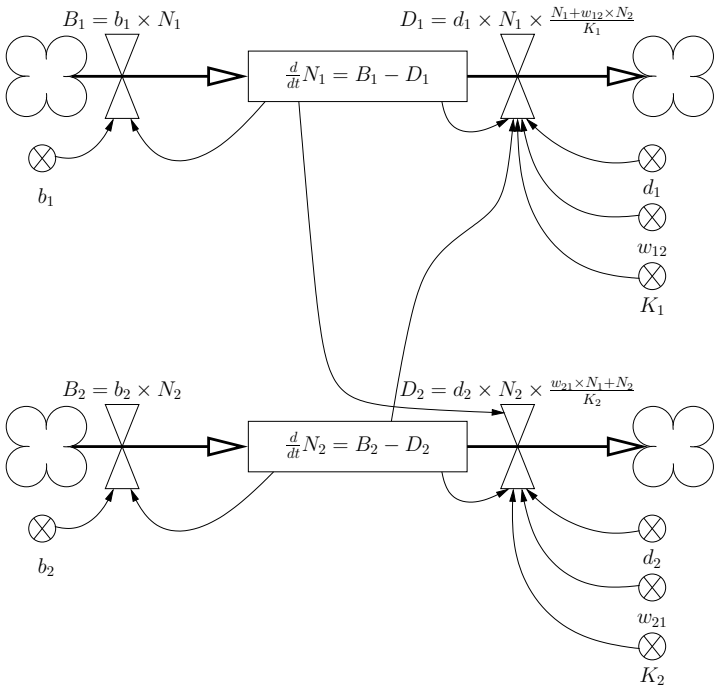


Figure 8.3: A species competition model

model. The formal model specification, also shown in figure 8.2(b), is:

$$\begin{aligned}
\frac{d}{dt}N_{\text{prey}}(t) &= B_{\text{prey}}(t) - D_{\text{prey}}(t) - P(t) \\
B_{\text{prey}}(t) &= b_{\text{prey}} \times N_{\text{prey}}(t) \\
D_{\text{prey}}(t) &= d_{\text{prey}} \times N_{\text{prey}}(t) \times \frac{N_{\text{prey}}(t)}{K_{\text{prey}}} \\
P(t) &= \frac{s \times N_{\text{prey}} \times N_{\text{pred}}}{1 + s \times N_{\text{prey}}(t) \times t_h} \\
\frac{d}{dt}N_{\text{pred}}(t) &= B_{\text{pred}}(t) - D_{\text{pred}}(t) \\
B_{\text{pred}}(t) &= b_{\text{pred}} \times N_{\text{pred}}(t) \\
D_{\text{pred}}(t) &= d_{\text{pred}} \times N_{\text{pred}}(t) \times \frac{N_{\text{pred}}(t)}{K_{\text{pred}}} \\
K_{\text{pred}}(t) &= k \times N_{\text{prey}}(t)
\end{aligned}$$

where s denotes the search rate or the proportional hunting area a predator can search per time unit spent on searching, and t_h denotes the amount of time required to handle a single prey (e.g. eating, digesting, etc.).

A *competition model* describes the evolution of respective sizes of n populations

that must compete over the similar resources. Similar to the Holling model, the competition model employed herein builds upon the logistic model. The formal model specification, as shown in figure 8.3, is:

$$\begin{aligned}
\frac{d}{dt}N_1(t) &= B_1(t) - D_1(t) \\
B_1(t) &= b_1 \times N_1(t) \\
D_1(t) &= d_1 \times N_1(t) \times (1 - \frac{N_1(t) + w_{12} \times N_2(t)}{K_1}) \\
\frac{d}{dt}N_2(t) &= B_2(t) - D_2(t) \\
B_2(t) &= b_2 \times N_2(t) \\
D_2(t) &= d_2 \times N_2(t) \times (1 - \frac{w_{21} \times N_1(t) + N_2(t)}{K_2})
\end{aligned}$$

where w_{12} and w_{21} are weights, representing the relative dominations of the competing species in consuming scarce resources.

8.1.2 Constructing the knowledge base

This subsection illustrates how a set of model fragments can be constructed based on the descriptions of ecological models of the previous subsection. The challenge of this task lies in the fact that model fragments must encompass a sufficiently general and reusable component part of the ecological models. In instances of models found in literature on ecological modelling, the boundaries of the recurring component parts are hidden, and it is therefore up to the knowledge engineer to identify them.

First, a hierarchy of entity types is set up. The system dynamics models shown above contain only three types of participant: variables, stocks and flows. Here, stocks and flows are a special type of variable with a predetermined meaning. That is, a flow f into a stock s corresponds to the equation $\frac{d}{dt}s = C^+(f)$ and a flow f out of a stock s denotes $\frac{d}{dt}s = C^-(f)$. Hence, stocks and flows are defined as subclasses of the participant class variable:

```
(defEntity variable)
(defEntity stock
  :subclass-of (variable))
(defEntity flow
  :subclass-of (variable))
```

The sample properties defined in section 3.3.3, which describe the condition under which a variable is endogenous or exogenous, are employed in this knowledge base. In order to keep this section relatively self-contained, the definitions are repeated here:

```
(defproperty endogenous-1
  :source-participants ((?m :type variable))
  :structural-conditions ((== ?m *))
  :property (endogenous ?m))

(defproperty endogenous-2
  :source-participants ((?m :type variable))
  :structural-conditions ((d/dt ?m *))
  :property (endogenous ?m))

(defproperty exogenous
  :source-participants ((?m :type variable))
  :structural-conditions ((not (endogenous ?m)))
  :property (exogenous ?m))
```

Once the above declarations are in place, the knowledge base of model fragments can be defined. The first model fragment describes the population growth phenomenon. Note that all of the aforementioned growth, predation and competition models contain a stock representing population size and two flows, one flow of births into the stock and another flow of deaths out of the stock. This common feature of models on population dynamics is contained in a single model fragment.

```
(defModelFragment growth
  :source-participants
    ((?population :type population))
  :assumptions
    ((relevant growth ?population))
  :target-participants
    ((?size :type stock :name size)
     (?birth-flow :type flow :name births)
     (?death-flow :type flow :name deaths))
  :postconditions
    ((flow ?birth-flow source ?size)
     (flow ?death-flow ?size sink)
     (size-of ?size ?population))
  :purpose-required
    ((endogenous ?birth-flow)
     (endogenous ?death-flow)))
```

Importantly, other model fragments must be able to be hooked up to this model fragment. In other words, this model fragment is only one piece of a puzzle and the other model fragments may correspond to other pieces of that puzzle that must connect to it. The information useful to support the interconnection of model fragments is contained in the participants (source-participants and target-participants) and relations (structural conditions and postcondition) of the model fragments. For the population growth model, other model fragments contain equations that provide a description of the behaviour of births and deaths over time, in terms of population size and other variables. Naturally, a population growth model containing just one stock and the birth and death flows is inadequate (because the simulator would require data on the number of births and deaths at each time instance). Therefore, a purpose-required constraint is added that demands that both flows are endogenous variables if this model fragment is applied. This guarantees that this model fragment will not be instantiated unless other model fragments provide equations for the birth and death flows.

The following two model fragments represent exponential and logistic population growth. Their source-participants and structural conditions essentially contain the same partial model as produced by the growth model fragment. The reason for a separate model fragment to produce the stock and flows is that other model fragments (e.g. Lotka-Volterra predation, as shown below) may attach their own growth model to the stock and flows. Both model fragments have a different model assumption, describing the growth model being used. The target-participants and postconditions correspond to the models shown in the previous subsection.

```
(defModelFragment exponential
  :source-participants
    ((?population :type population)
     (?size :type stock)
     (?birth-flow :type flow)
     (?death-flow :type flow))
  :structural-conditions
    ((flow ?birth-flow source ?population)
     (flow ?death-flow ?population sink)
     (size-of ?size ?population))
  :assumptions
    ((model ?size exponential))
  :target-participants
    ((?birth-rate :type variable :name birth-rate)
     (?death-rate :type variable :name death-rate))
```

```
  :postconditions
    ((== ?birth-flow (* ?birth-rate ?size))
     (== ?birth-flow (* ?death-rate ?size))))

(defModelFragment logistic
  :source-participants
    ((?population :type population)
     (?size :type stock)
     (?birth-flow :type flow)
     (?death-flow :type flow))
  :structural-conditions
    ((flow ?birth-flow source ?size)
     (flow ?death-flow ?size sink)
     (size-of ?size ?population))
  :assumptions
    ((model ?size logistic))
  :target-participants
    ((?birth-rate :type variable :name birth-rate)
     (?death-rate :type variable :name death-rate)
     (?density :type variable :name total-population)
     (?capacity :type variable :name capacity))
  :postconditions
    ((== ?birth-flow (* ?birth-rate ?size))
     (== ?death-flow (* ?death-rate ?size ?density))
     (== ?density (C+ (/ ?size ?capacity)))
     (population-density-of ?density ?population)
     (population-capacity-of ?capacity ?population)))
```

In addition to the exponential and logistic growth models, a third model fragment is added to describe population growth. As mentioned earlier, Lotka-Volterra predation implicitly contains its own models for births and deaths. Therefore, models depicting Lotka-Volterra predation between two population p_1 and p_2 can not be combined with equations describing population growth of p_1 and p_2 (e.g. the equations formalising the exponential or logistic theory). In this case, no population growth model must be instantiated. However, following the definition of model assumptions, for each instance of `?size` a model assumption must be selected and deemed true. Hence, the model fragment below is added to leave open the option of other equations for model growth introduced elsewhere.

```
(defModelFragment other-growth
  :source-participants
    ((?population :type population)
     (?size :type variable)
```

```
    (?birth-flow :type flow)
    (?death-flow :type flow))
  :structural-conditions
    ((flow ?birth-flow source ?population)
     (flow ?death-flow ?population sink))
  :assumptions
    ((model ?population other)))
```

Apart from population growth, two other phenomena are considered by the knowledge base: predation and competition. Predation and competition relations between species are represented by predicates over the populations: e.g. `(predation grizzly-bear salmon)` and `(competition grizzly-bear brown-bear)`. However the existence of a phenomenon does not necessarily mean that it must be included in the model. It would make little sense to model predation and competition without modelling the size of the populations, because models of these phenomena relate population sizes to one another. Therefore, the incorporation of the predation phenomenon is made dependent upon the existence of variables representing population size. Note that the previous model fragments used a stock rather than an ordinary variable to represent population size. Since a stock is defined as a type of variable, that is a stock is a variable, the more general type variable will still match any instance of stock. In addition, human expert modellers may prefer to leave a phenomenon out of the model. To leave this choice open, the following two model fragments construct a participant representing the phenomena of predation and competition and make it dependent upon a relevance assumption.

```
(defModelFragment predation-phenomenon
  :source-participants
    ((?predator :type population)
     (?prey :type population)
     (?predator-size :type variable)
     (?prey-size :type variable))
  :structural-conditions
    ((predation ?predator ?prey)
     (size-of ?predator-size ?predator)
     (size-of ?prey-size ?prey))
  :assumptions
    ((relevant predation ?predator ?prey))
  :target-participant
    ((?predation-phenomenon :type phenomenon :name predation-phenomenon))
  :postconditions
    ((predation-phenomenon ?predation-phenomenon ?predator ?prey)))
```

```
(defModelFragment competition-phenomenon
  :source-participants
    ((?population1 :type population)
     (?population2 :type population)
     (?size1 :type variable)
     (?size2 :type variable))
  :structural-conditions
    ((competition ?population1 ?population2)
     (size-of ?size1 ?population1)
     (size-of ?size2 ?population2))
  :assumptions
    ((relevant competition ?population1 ?population2))
  :target-participant
    ((?competition-phenomenon :type phenomenon :name competition-phenomenon))
  :postconditions
    ((competition-phenomenon ?competition-phenomenon ?population1 ?population2)))
```

The creation of a predation phenomenon requires that a new property, `has-model`, is satisfied. This property is defined as follows:

```
(defproperty has-model
  :source-participants ((?p :type phenomenon))
  :structural-conditions ((is-model-of * ?p))
  :property (has-model ?p))
```

This property indicates the situation where a model has been selected for a phenomenon. It is required if different types of model for the same phenomenon require a different configuration in the rest of model. This is the case for the two predation models, Lotka-Volterra and Holling. The next two model fragments will contain these predation models and will demonstrate the usefulness of the `has-model` property. Both of these model fragments require that the participant representing the predation phenomenon be instantiated for the two populations between which a predation relation exists. They also contain different sets of requirements on the participants and relations between them that exist in the emerging model. The Holling predation model, for example, is dependent upon a participant (`?capacity`) representing the carrying capacity of the predator population. Both model fragments introduce a number of new participants and equations, adding new variables and influences to the model. As with the growth models, these correspond to the equations shown in the previous subsection.

```
(defModelFragment Lotka-Volterra
  :source-participants
```

```
    ((?predation-phenomenon :type phenomenon)
     (?predator :type population)
     (?predator-size :type stock :name population-size)
     (?predator-birth-flow :type flow :name births)
     (?predator-death-flow :type flow :name deaths)
     (?prey :type population)
     (?prey-size :type stock :name population-size)
     (?prey-birth-flow :type flow :name births)
     (?prey-death-flow :type flow :name deaths))
  :structural-conditions
    ((predation-phenomenon ?predation-phenomenon ?predator ?prey)
     (flow ?predator-birth-flow source ?predator-size)
     (flow ?predator-death-flow ?predator-size sink)
     (size-of ?predator-size ?predator)
     (flow ?prey-birth-flow source ?prey-size)
     (flow ?prey-death-flow ?prey-size sink)
     (size-of ?prey-size ?prey))
  :assumptions
    ((model predation lotka-volterra))
  :target-participants
    ((?prey-birth-rate :type variable :name birth-rate)
     (?predator-factor :type variable :name predator-factor)
     (?prey-factor :type variable :name prey-factor)
     (?predator-death-rate :type variable :name death-rate))
  :postconditions
    ((== ?prey-birth-flow (* ?prey-birth-rate ?prey-size))
     (== ?predator-birth-flow (* ?predator-factor ?prey-size ?predator-size))
     (== ?prey-death-flow (* ?prey-factor ?prey-size ?predator-size))
     (== ?predator-death-flow (* ?predator-death-rate ?predator-size))
     (is-model-of lotka-volterra ?predation-phenomenon)))

(defModelFragment Holling
  :source-participants
    ((?predation-phenomenon :type phenomenon)
     (?predator :type population)
     (?predator-size :type stock :name population-size)
     (?capacity :type variable)
     (?prey :type population)
     (?prey-size :type stock :name population-size))
  :structural-conditions
    ((predation-phenomenon ?predation-phenomenon ?predator ?prey)
     (size-of ?predator-size ?predator)
     (size-of ?prey-size ?prey)
     (capacity-of ?capacity ?predator))
  :assumptions
    ((model ?predation-phenomenon holling))
  :target-participants
```

```
   ((?search-rate :type variable :name search-rate)
    (?handling-time :type variable :name handling-time)
    (?prey-requirement :type variable :name prey-requirement)
    (?predation :type flow :name predation))
 :postconditions
   ((flow ?predation ?prey-size sink)
    (== ?predation (/ (* ?search-rate ?prey-size ?predator-size)
                      (+ 1 (* ?search-rate ?prey-size ?handling-time))))
    (== ?capacity (C+ (* ?prey-requirement ?prey)))
    (is-model-of holling ?predation-phenomenon)))
```

Clearly, the Holling model fragment will not be instantiated if there is no variable representing the carrying capacity of the predator species. However, without the `has-model` property, it is still possible that the predation phenomenon is deemed relevant and that the Holling model assumption is selected, even if no variable describing the carrying capacity of the predator species is available in the emerging model. Indeed, neither of the two assumptions require a carrying capacity and hence, they can be instantiated without such a variable. Of course, in this case, the Holling model fragment will not be applied, but the user might expect it to be. Because the `(has-model ?predation-phenomenon)` property is only satisfied if either the Lotka-Volterra or the Holling model fragment is applied, this property ensures that the inclusion of a predation model assumption in a consistent set of assumptions leads to the generation of a set of equations describing the committed predation model.

More specifically, because the creation of `?predation-phenomenon` comes with the purpose-required property `(has-model ?predation-phenomenon)`, it generates the requirement that either the Lotka-Volterra or the Holling model is added to the emerging model. If the Holling model would be selected, but there is no variable that corresponds to `?capacity`, the consequents of the Holling predation model fragment will not be instantiated (including the `(is-model-of holling ?predation-phenomenon)` relation), and hence, the purpose-required property would not be satisfied.

The selection of a set of assumptions that support the required properties is resolved through *abduction*. That is, the knowledge base contains a number of ways in which the property `(has-model ?predation-phenomenon)` can be satisfied. Let $A_1, \ldots, A_n$ be the sets of assumptions that satisfy this property. If the property `(has-model ?predation-phenomenon)` must be satisfied, then the inference mechanism, to be illustrated later, will add constraints to the system such that $A_1, \ldots, A_n$ must be deemed

true:

$$\frac{\begin{array}{rl} & \texttt{(has-model ?predation-phenomenon)} \\ A_1 \rightarrow & \texttt{(has-model ?predation-phenomenon)} \\ \vdots & \vdots \\ A_n \rightarrow & \texttt{(has-model ?predation-phenomenon)} \end{array}}{A_1 \vee \ldots \vee A_n}$$

In the present case, the required property generates an inconsistency:

$$A \wedge \neg\texttt{(has-model ?predation-phenomenon)} \rightarrow \bot$$

where A is the set of assumptions that causes the model fragment with the purpose-required property `(has-model ?predation-phenomenon)` to be applied. Following the approach introduced in section 4.1.2.2.2, a new inconsistency can be derived:

$$\frac{\begin{array}{rl} & A \wedge \neg\texttt{(has-model ?predation-phenomenon)} \rightarrow \bot \\ A_1 \rightarrow & \texttt{(has-model ?predation-phenomenon)} \\ \vdots & \vdots \\ A_n \rightarrow & \texttt{(has-model ?predation-phenomenon)} \end{array}}{A \wedge \neg(A_1 \vee \ldots \vee A_n) \rightarrow \bot}$$

As explained in section 4.2, the inconsistency $A \wedge \neg(A_1 \vee \ldots \vee A_n) \rightarrow \bot$ is translated to a compatibility constraint. The DPCSP solution techiques presented in chapter 6 guarantee that all solutions will satisfy the compatibility constraints and hence, that no set of assumptions will be selected such that $A \wedge \neg(A_1 \vee \ldots \vee A_n)$.

Using this line of reasoning, the knowledge engineer only needs to provide the following:

- The model fragment that specifies that the predation phenomenon is relevant must contain the purpose required property `(has-model ?predation-phenomenon)`.
- Each of the model fragments that contain such a model and hence, should satisfy the purpose-required property, must include a postcondition, e.g. `(is-model-of holling ?predation-phenomenon)`, which itself satisfies the aforementioned property.

The conditions under which the property is satisfied need not concern the knowledge engineer. Previous work in compositional modelling, such as [45] required that assumption classes contained activity conditions which explicitly stated under which

conditions the assumptions become part of the problem. Thanks to the use of an assumption-literal based truth maintenance system, the compositional modeller presented herein generates these activity conditions automatically.

Note that in the model fragment representing the Holling predation model, the equation assigning the carrying capacity of the predator `?capacity` employs the compositional addition operator. This allows the model fragment to be used in models with multiple prey species for the same predator. Each of the prey species adds to the total carrying capacity for the predator species. When multiple predator species hunt the same prey species, the same model fragment adds for each predator a `?predation` flow, from the stock denoting prey population size to a sink.

The final model fragment describes competition. Generalising the competition model of the previous subsection to n species, the model for the death flow of species i would be:

$$D_i = d_i \times N_i \times \frac{w_{i1} \times N_1 + \ldots + w_{i(i-1)} \times N_{i-1} + N_i + w_{i(i+1)} \times N_{i+1} + \ldots + w_{in} \times N_n}{K_i}$$

Such an equation is not easily specified in a format that can be instantiated. However, the equation can be easily rewritten as:

$$D_i = d_i \times N_i \times \delta_i$$
$$\delta_i = \frac{w_{i1} \times N_1}{K_i} + \ldots + \frac{w_{i(i-1)} \times N_{i-1}}{K_i} + \frac{N_i}{K_i} + \frac{w_{i(i+1)} \times N_{i+1}}{K_i} + \ldots + \frac{w_{in} \times N_n}{K_i}$$

where δ_i denotes the population density. The righthand side of the equation describing δ_i can be decomposed into composable constituent terms, as follows:

$$D_i = d_i \times N_i \times \delta_i$$
$$\delta_i = C^+(\frac{N_i}{K_i}) \tag{8.1}$$
$$\delta_i = C^+(\frac{w_{ij} \times N_j}{K_i}) \quad j = 1, \ldots, i-1, i+1, \ldots, n \tag{8.2}$$

These equations can easily be described by means of model fragments, as is illustrated in the `competition` model fragment show below. Note that the logistic growth model introduced a participant `?density`, which corresponds to δ_i, and a relation equivalent to 8.1. The model fragment for competition adds the relations that correspond to 8.2.

```
(defModelFragment competition
```

```
:source-participants
  ((?competition-phenomenon :type phenomenon)
   (?population-1 :type population)
   (?size-1 :type stock)
   (?density-1 :type variable)
   (?capacity-1 :type variable)
   (?population-2 :type population)
   (?size-2 :type stock)
   (?density-2 :type variable)
   (?capacity-2 :type variable))
:structural-conditions
  ((competition-phenomenon ?competition-phenomenon ?population-1 ?population-2)
   (density-of ?density-1 ?size-1)
   (capacity-of ?capacity-1 ?size-1)
   (density-of ?density-2 ?size-2)
   (capacity-of ?capacity-2 ?size-2))
:assumptions
  ((relevant competition ?population-1 ?population-2))
:target-participants
  ((?weight-12 :type variable :name weight)
   (?weight-21 :type variable :name weight))
:postconditions
  ((== ?density-1 (C+ (/ (* ?weight-12 ?size-2) ?capacity-1)))
   (== ?density-2 (C+ (/ (* ?weight-21 ?size-1) ?capacity-2)))))
```

With this small set of model fragments, a very large set of ecological models can be composed. Basically, models for any configuration of populations related to one another via predation and competition relations can be generated by means of this set of model fragments. Of course, readers whose interests lie primarily in ecological modelling may come up with a number of improvements to this knowledge base. For example, predators that live on the same set of prey species are effectively competing with one another and hence, the competition model may apply to them. When two predators share a subset of prey species, but not all of them, some model for partial competition may be required. Be that as it may, such extensions can be added *ad infinitum* to the knowledge base, and they are not required for illustration purposes.

8.1.3 Knowledge base + scenario = model space

A model space is constructed when the knowledge base is instantiated with respect to a given scenario. Consider for example the scenario depicted in figure 8.4. This scenario

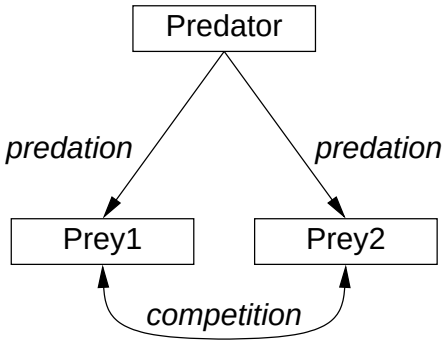


Figure 8.4: The 1 predator and 2 competing prey scenario

describes a predator population that preys on two other populations, prey1 and prey2, whilst the two prey populations compete with one another.

A graphical representation of the model space for this scenario is shown in figure 8.5. As discussed in section 4.1, the model space is a hypergraph with hyperarcs connecting sets of nodes, representing participants, relations and assumptions, to one another. Each of the hyperarcs is annotated with a reference to the model fragment that contains the particular inference shown. Given the obvious restrictions on the available space, the detailed participant and relation instances have been replaced by descriptions of contents of the model fragment antecedents and consequents.

In essence, this model space contains the following knowledge:

- From each of the three populations in the scenario, a set of population growth models (i.e. exponential, logistic and other) is derived. This inference is dependent upon a relevance assumption of the population growth phenomenon, and a model assumption that corresponds to one of the three population growth models.

- From both predation relations (i.e. `(predation predator prey1)` and `(predation predator prey2)`), and the populations related by them, a set of predation models (i.e. Lotka-Volterra predation and Holling predation) is derived. This inference is dependent upon a relevance assumption of the predation phenomenon, and a model assumption that corresponds to one of the two predation models.

- From the competition relation (i.e. `(competition prey1 prey2)`), and the populations related by them, a competition model is derived. Because there

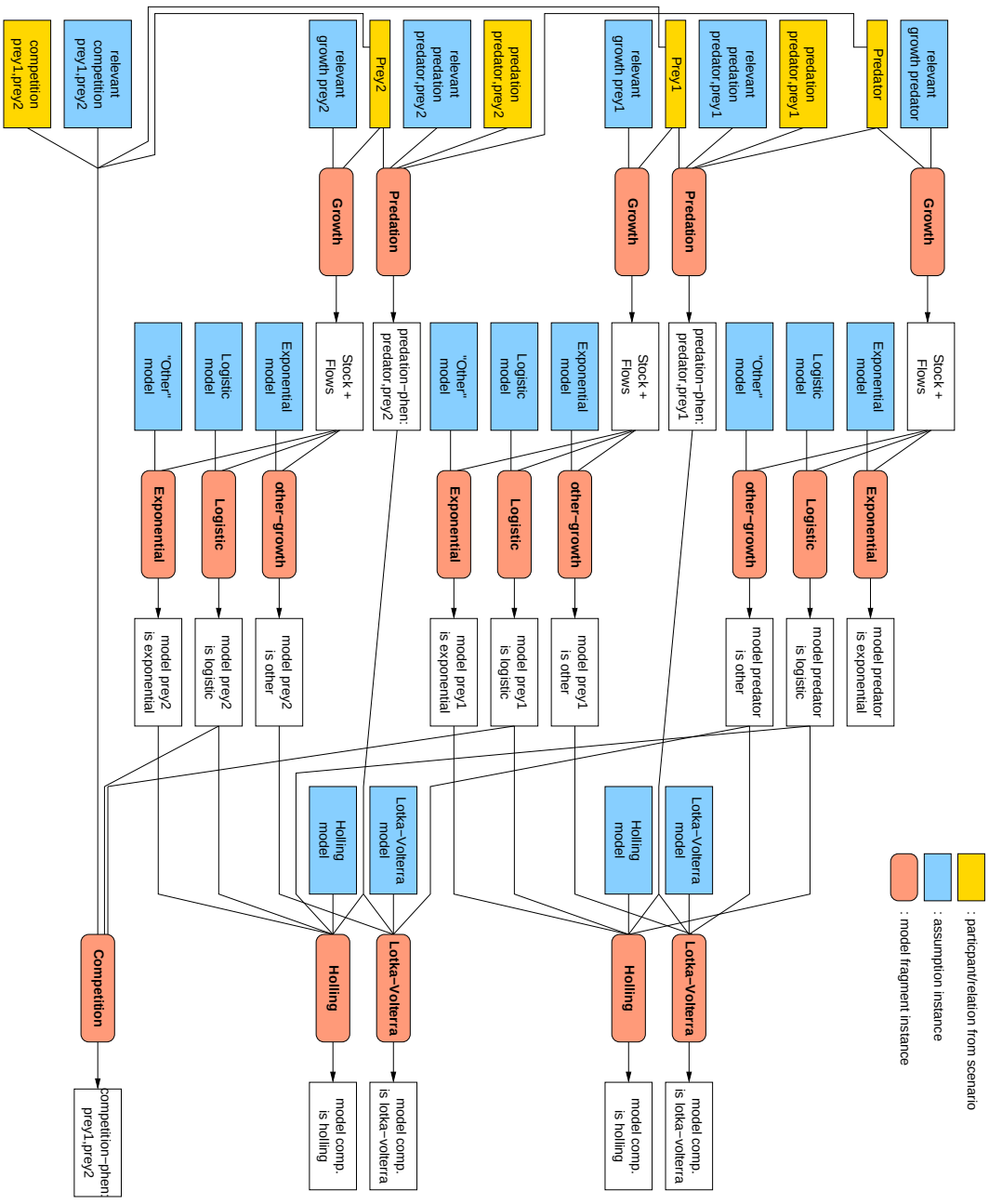


Figure 8.5: Model space for the 1 predator and 2 competing prey scenario

is only one competition model, the inference of the competition model is only dependent upon a relevance assumption that corresponds to the competition phenomenon.

In addition to the hypergraph of figure 8.5, the model space also contains a number of constraints on the conjunctions of assumptions that are consistent. In the mathematical modelling domain, this source of inconsistency arises from the fact that non-composable equations that assign a different formula to the same variable can not be part of the same model. Hence, the conjunctions of assumptions from which such an inconsistency can arise must be deemed inconsistent. Also, the consequents of different assumptions with the same subject are inconsistent with one another. Such assumptions will become DPCSP values in the domain of a single attribute and hence, the corresponding inconsistencies can be ignored. In the present problem, the combination of lotka-volterra predation with exponential or logistic growth also leads to such inconsistencies. This entails the following nogoods to be added to the hypergraph:

$$\texttt{(model size-1 exponential)} \wedge \texttt{(model predation-phenomenon-1 lotka-volterra)} \rightarrow \bot$$
$$\texttt{(model size-1 logistic)} \wedge \texttt{(model predation-phenomenon-1 lotka-volterra)} \rightarrow \bot$$
$$\texttt{(model size-2 exponential)} \wedge \texttt{(model predation-phenomenon-2 lotka-volterra)} \rightarrow \bot$$
$$\texttt{(model size-2 logistic)} \wedge \texttt{(model predation-phenomenon-2 lotka-volterra)} \rightarrow \bot$$
$$\texttt{(model size-3 exponential)} \wedge \texttt{(model predation-phenomenon-1 lotka-volterra)} \rightarrow \bot$$
$$\texttt{(model size-3 logistic)} \wedge \texttt{(model predation-phenomenon-1 lotka-volterra)} \rightarrow \bot$$
$$\texttt{(model size-3 exponential)} \wedge \texttt{(model predation-phenomenon-2 lotka-volterra)} \rightarrow \bot$$
$$\texttt{(model size-3 logistic)} \wedge \texttt{(model predation-phenomenon-2 lotka-volterra)} \rightarrow \bot$$

Other inconsistencies may stem from the two purpose-required properties. For example, by instantiating the abductive reasoning process described in section 8.1.2, page 229 to the predation relation between `predator` and `prey1`, the following inconsistent conjunction of assumptions can be derived from the purpose-required property `(has-model ?preference-phenomenon)`.

$$\texttt{(relevant growth predator)} \wedge \texttt{(model size-3 exponential)} \wedge$$
$$\texttt{(relevant growth prey1)} \wedge \texttt{(model-size-1 exponential)} \wedge$$
$$\texttt{(relevant predation predator prey1)} \wedge \texttt{(model predaton-phenomenon-1 holling)} \rightarrow \bot$$

As explained earlier, under this combination of assumptions, the Holling model fragment is not applied because there is no variable describing the carrying capacity of

Attribute	Meaning
x_1	`(relevant growth prey1)`
x_2	`(relevant growth prey2)`
x_3	`(relevant growth predator)`
x_4	`(relevant predation predator prey1)`
x_5	`(relevant predation predator prey2)`
x_6	`(relevant competition prey1 prey2)`
x_7	`(model size-1 *)`
x_8	`(model size-2 *)`
x_9	`(model size-3 *)`
x_{10}	`(model predation-phenomenon-1 *)`
x_{11}	`(model predation-phenomenon-2 *)`

Table 8.1: The DCSP for the 1 predator and 2 competing prey scenario: attributes and their meaning

the predator population. Since no predation model exists in this case, even though the predation phenomenon is deemed relevant under this set of assumptions, it is marked as an inconsistency.

8.1.4 The dynamic preference constraint satisfaction problem

From the generated model space a DCSP is derived. A set of preferences are then assigned to the assumptions corresponding to the attribute-value assignments, thus turning the DCSP into a DPCSP. The first step in constructing the DCSP involves generating an attribute for each set of assumptions with the same subject. The attributes that can be generated for the assumptions in the model space of figure 8.5 are summarised in table 8.1.

The assumptions from which the attributes were generated form domains of values. The resulting domains of the aforementioned attributes are summarised in table 8.2.

As discussed in section 4.2, the activity constraints in the DCSP are implications that activate an attribute, under the conditions that instantiate the subject of the assumptions that correspond to that attribute, in the emerging model. Since each participant and relation has a label in the model space, a minimal set of assumptions under which it becomes part of the emerging model is available. When a participant or a relation is the subject of an assumption, this label explicitly describes the sets of assumptions under which the attribute that corresponds to that subject should be activated. By trans-

Domain	Content	Meaning
D_1	$\{d_{1,y}, d_{1,n}\}$	{population,none}
D_2	$\{d_{2,y}, d_{2,n}\}$	{population,none}
D_3	$\{d_{3,y}, d_{3,n}\}$	{population,none}
D_4	$\{d_{4,y}, d_{4,n}\}$	{(population,population),none}
D_5	$\{d_{5,y}, d_{5,n}\}$	{(population,population),none}
D_6	$\{d_{6,y}, d_{6,n}\}$	{(population,population),none}
D_7	$\{d_{7,l}, d_{7,e}, d_{7,o}\}$	{logistic,exponential,other}
D_8	$\{d_{8,l}, d_{8,e}, d_{8,o}\}$	{logistic,exponential,other}
D_9	$\{d_{9,l}, d_{9,e}, d_{9,o}\}$	{logistic,exponential,other}
D_{10}	$\{d_{10,h}, d_{10,lv}\}$	{Holling,Lotka-Volterra}
D_{11}	$\{d_{11,h}, d_{11,lv}\}$	{Holling,Lotka-Volterra}

Table 8.2: The DCSP for the 1 predator and 2 competing prey scenario: domains and their contents and meaning

lating the label of a subject into sets of attribute-value assignments, the antecedents of the activity constraints are constructed.

In the present example problem, the relevance assumptions (attributes $x_1, \ldots, x_6$) take their subjects from the scenario, and hence, they are always active. The attributes related to the model assumptions for population growth are active if the corresponding assumptions denoting relevance of population growth, are true. That is,

$$x_1 : d_{1,y} \rightarrow \text{active}(x_7)$$
$$x_2 : d_{2,y} \rightarrow \text{active}(x_8)$$
$$x_3 : d_{3,y} \rightarrow \text{active}(x_9)$$

The attributes related to the assumptions about the predation models are active if the corresponding assumptions denoting relevance of predation, and the assumptions describing relevance of population growth, are true for the populations involved in the predation relation. That is,

$$x_1 : d_{1,y} \wedge x_3 : d_{3,y} \wedge x_4 : d_{4,y} \rightarrow \text{active}(x_{10})$$
$$x_2 : d_{2,y} \wedge x_3 : d_{3,y} \wedge x_5 : d_{5,y} \rightarrow \text{active}(x_{11})$$

Figure 8.6 shows a graphical representation of these activity constraints.

The compatibility constraints correspond directly to the inconsistencies in the nogood node. These inconsistencies have been discussed in the previous section and are depicted graphically in figure 8.6.

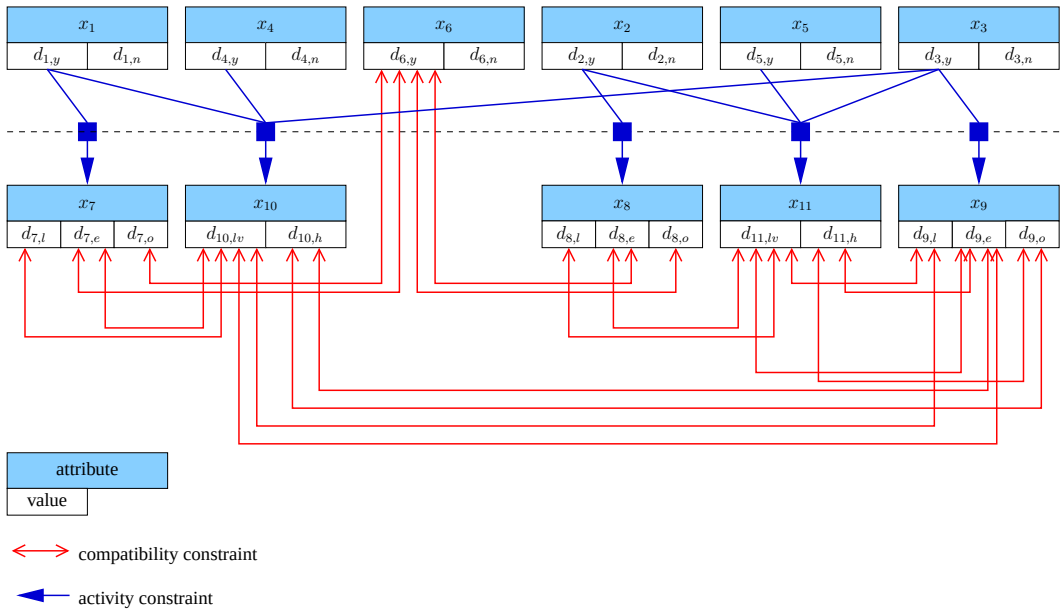


Figure 8.6: DCSP derived from the models space reflecting the 1 predator and 2 competing prey scenario

Once the DCSP is constructed, preferences may be attached to attribute-value assignments. Assume that preferences will only be assigned to the standard population modelling choices, i.e. exponential growth, logistic growth, lotka-volterra predation and holling predation, and to the relevance of competition (because only one type model has been implemented for this phenomenon). For example, the following BPQs could be employed:

$$p_{\text{exponential}} < p_{\text{logistic}}$$

$$p_{\text{lotka-volterra}} < p_{\text{holling}}$$

$$p_{\text{competition}}$$

The logistic and Holling models are preferred over the exponential and Lotka-Volterra models because the former are generally regarded as being more accurate. Note, however, that the preferences have been ordered in such a way that those corresponding to different phenomena are not related to one another. The justification for this is that, even though the models are structurally connected (there are restrictions over which models can combined with one another), models of different phenomena inherently describe behaviours that can not be compared with one another. The preference as-

Attribute	Preference assignments
$x_1, \ldots, x_5$	no preference assignments
x_6	$P(x_6 : d_{6,y}) = p_{\text{competition}}$
x_7	$P(x_7 : d_{7,l}) = p_{\text{logistic}}, P(x_7 : d_{7,e}) = p_{\text{exponential}}$
x_8	$P(x_8 : d_{8,l}) = p_{\text{logistic}}, P(x_8 : d_{8,e}) = p_{\text{exponential}}$
x_9	$P(x_9 : d_{9,l}) = p_{\text{logistic}}, P(x_9 : d_{9,e}) = p_{\text{exponential}}$
x_{10}	$P(x_{10} : d_{10,h} = p_{\text{holling}}, P(x_{10} : d_{10,lv}) = p_{\text{lotka-volterra}}$
x_{11}	$P(x_{11} : d_{11,h} = p_{\text{holling}}, P(x_{11} : d_{11,lv}) = p_{\text{lotka-volterra}}$

Table 8.3: Preference assignments for the 1 predator and 2 competing prey problem

signments for attribute value assignments are summarised in table 8.3.

Solving this DPCSP is simple. First, the attributes $x_1, \ldots, x_6$ are activated. Each of these attributes is assigned $x_i : d_{i,y}$ because that assignment maximises the potential preference. Then, the attributes $x_7, \ldots, x_{11}$ are activated. Here, attributes $x_7, \ldots, x_9$ are assigned $x_i : d_{i,l}$ because the logistic growth model has the highest preference. Finally, x_{10} and x_{11} are assigned $x_{10} : d_{10,h}$ and $x_{11} : d_{11,h}$ because the holling models have the highest preference and they are not inconsistent with the logistic model committed earlier.

8.1.5 Sample scenario model

In the previous subsection, a solution was computed that corresponds to the following set of assumptions:

```
(relevant growth prey1)
(relevant growth prey2)
(relevant growth predator)
(relevant competition prey1 prey2)
(relevant predation predator prey1)
(relevant predation predator prey2)
(model size-1 logistic)
(model size-2 logistic)
(model size-3 logistic)
(model predation-phenomenon-1 holling)
(model predation-phenomenon-2 holling)
```

Figure 8.7 shows how a scenario model can be deduced from this set of assumptions by exploiting the model space. In this figure, the blue nodes correspond to the aforementioned assumptions and the yellow nodes correspond to those nodes that are deemed true.

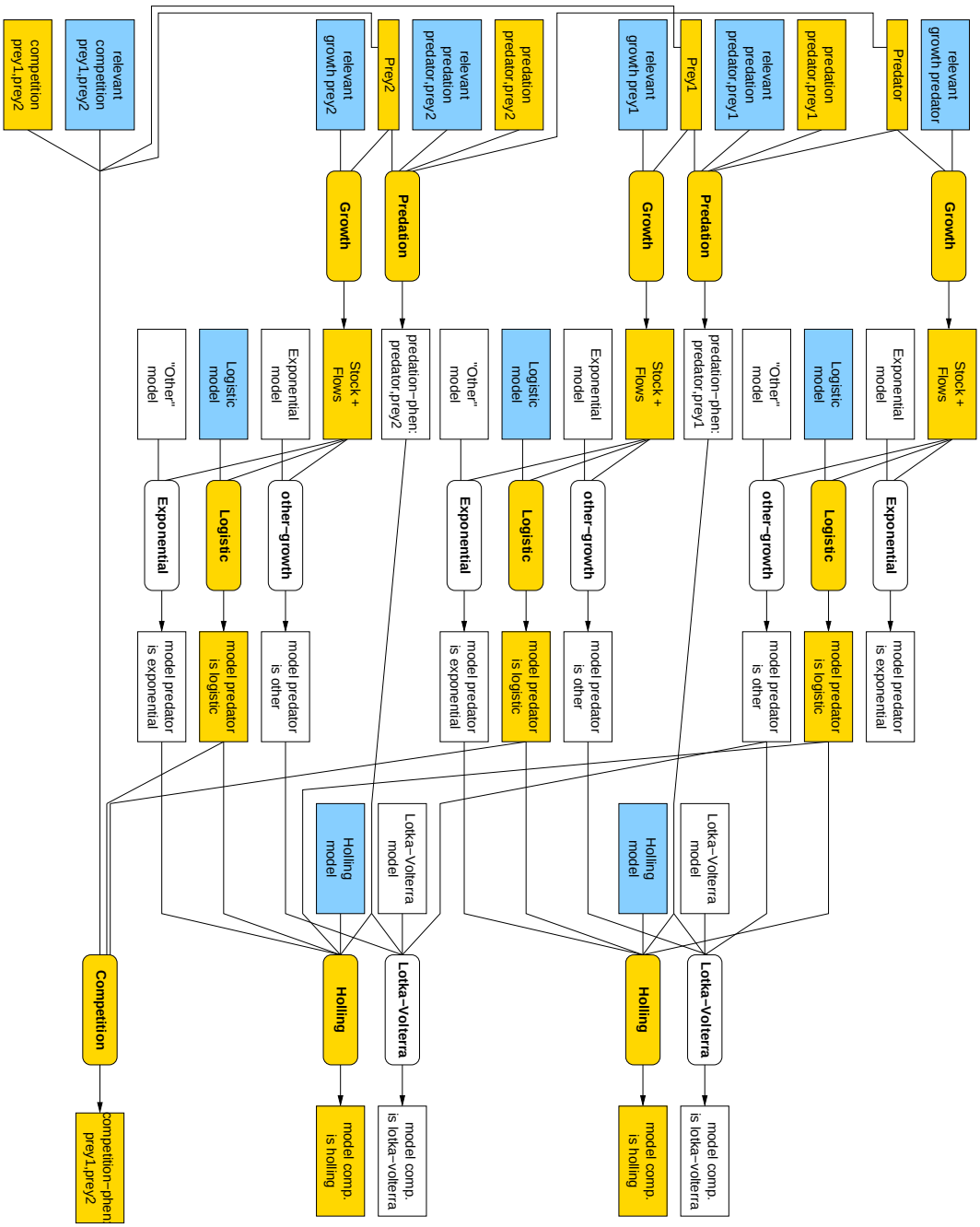


Figure 8.7: Deducing a scenario model from the model space, given a set of assumptions

When combining the participants and relations in the resulting scenario model, the model given in figure 8.8 can be drawn. This model corresponds to the model that an ecologist would draw if the logistic growth and Holling predation models were selected as appropriate for the task at hand.

8.2 The ModMed *n*-species model

The last section has shown the success of applying the ideas of this thesis to a systematic demonstrating case. The question is now whether the work presented in the thesis scales up to be able to deal with larger, real-world problems. An existing model that is relevant to applied up-to-date research in ecological modelling is used to show the scale-up-ability of the present work. A knowledge base is constructed based on the information contained in a model, called the ModMed *n*-species model, and dialogue with some of the experts that have developed it. This knowledge base has then been validated by two sets of tests. On the one hand, reproductions of instances of the original model created by the compositional modeller have shown that it contains the necessary knowledge. On the other hand, tests have been carried out to determine that infeasible variations of this model are not constructed.

Given the size of the ModMed *n*-species knowledge base, it is not feasible to discuss it here with the same amount of detail as the population dynamics knowledge base of section 8.1 (a complete listing of the knowledge base is provided in appendix C[1]). Therefore, this section will highlight a number of important issues in the creation and use of the knowledge base. First, background information about the knowledge base of this project will be discussed. Then, its significant features will be presented. And finally, an overview of the types of model that can be generated will be given.

8.2.1 Background

The term ModMed comes from the name of a large EU-funded project studying Mediterranean ecosystems. It is producing a number of general models that, when combined, describe the Mediterranean ecosystem at the level of the landscape (e.g. distribution

[1]The development of the knowledge base of appendix C required approximately two man-weeks of work: to study the ModMed model, to identify appropriate component models and to translate them into model fragments.

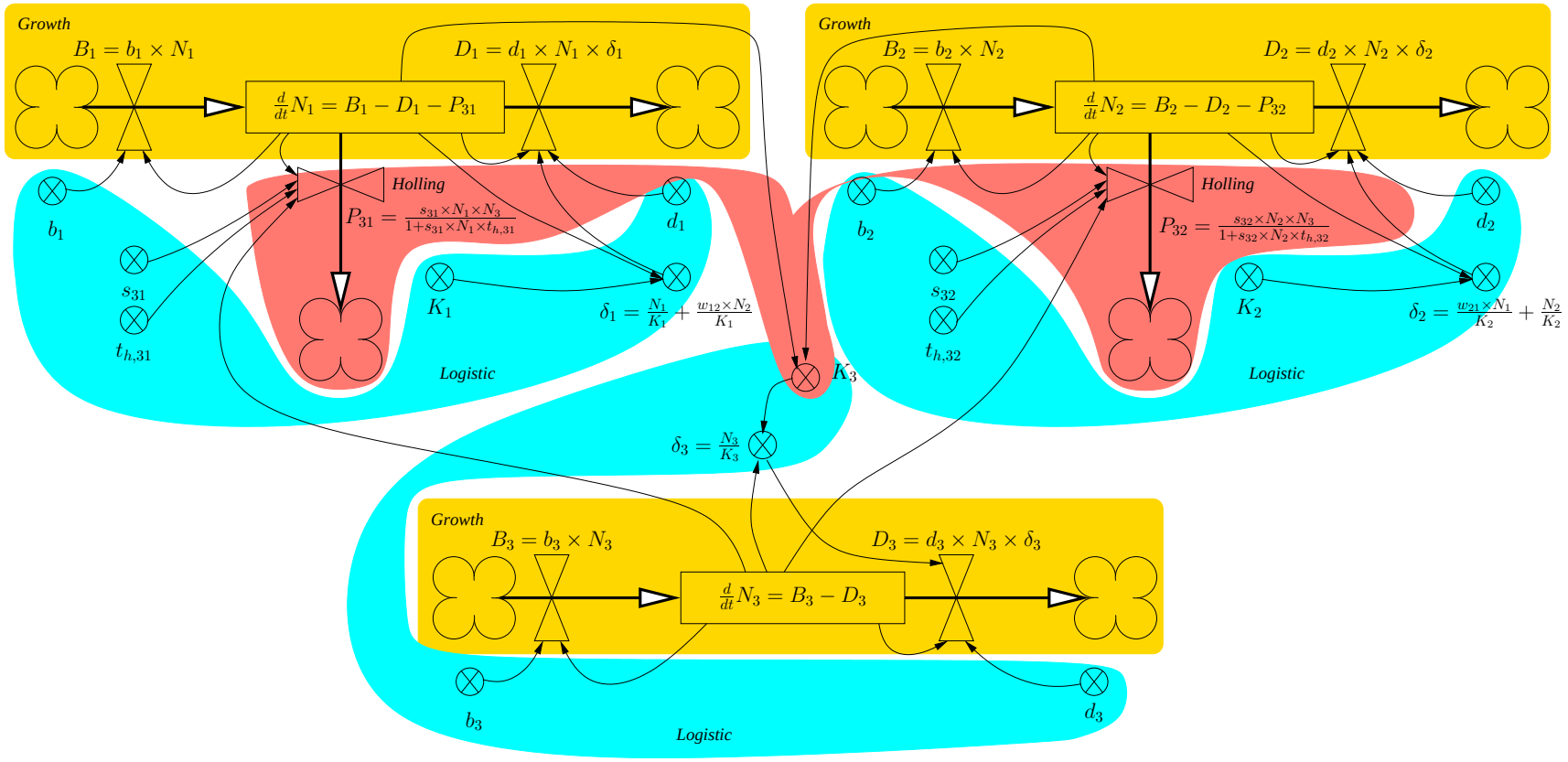


Figure 8.8: Sample scenario model for the 1 predator and 2 competing prey scenario

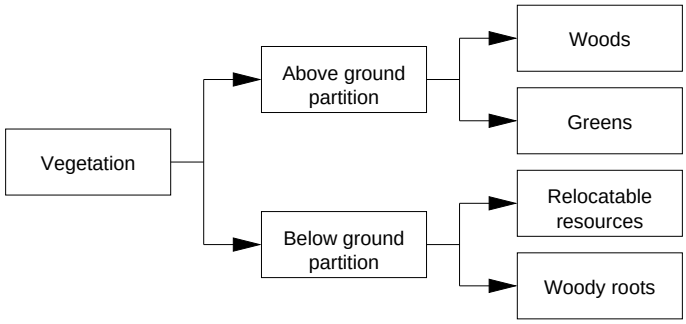


Figure 8.9: Functional decomposition of vegetation

of fire, vegetation and grazing animals), the communities (e.g. the *n*-species model) and the individuals (e.g. water uptake by leaves, responses to grazing damage). The landscape model contains a spatial grid. For each cell in this grid, a community model is created. This could be a simple Markov network, a collection of individuals or the *n*-species model discussed herein. From the landscape and individuals level models, the *n*-species model takes parameter values (e.g. availability of grazing animals from the landscape model, average water uptake by a tree from an individuals model).

8.2.2 Modelling issues

Rather than explaining the content and role of each individual model fragment, this section discusses a number of important modelling issues that had to be addressed in the construction of the knowledge base. The most important features of the modelling knowledge in this domain are presented and, for each of them, an approach is introduced to formalise them with the compositional modelling techniques of this thesis.

8.2.2.1 Component hierarchy

Central to the *n*-species model is the representation of vegetation. In the original model, each species is divided according to the functional components of the individuals: woods (stems and branches), greens (leaves and green twigs), relocatable resources (energy stored in the roots that can be utilised for growth of greens) and woody roots. This is illustrated by the component hierarchy in figure 8.9.

In this example, pairs of model assumptions are used in the knowledge base for the vegetation: one denoting that the behaviour of a vegetation component is described in

terms of its constituent components and the other describing that the vegetation component is not partitioned. For instance, (model ?veg ground-level-partitioning) states that ?veg is partitioned into an above ground and a below ground vegetation component, and (model ?veg aggregate) denotes the case that the behaviour of ?veg is not described in terms ?veg itself rather than its constituent components. The (model ?veg ground-level-partitioning) assumption is employed in a model fragment as follows:

```
(defModelFragment vegetation->above-and-below-ground-partitions
  :source-participants
    ((?veg :type vegetation))
  :assumptions
    ((model ?veg ground-level-partitioning))
  :target-participants
    ((?above-ground-veg :type vegetation-live-component)
     (?below-ground-veg :type vegetation-live-component))
  :postconditions
    ((partitioning ?veg ?above-ground-veg ?below-ground-veg)
     (above-ground-vegetation ?above-ground-veg)
     (below-ground-vegetation ?below-ground-veg)
     (part-of ?above-ground-veg ?veg)
     (part-of ?below-ground-veg ?veg)))
```

If the assumption model(?veg, ground-level-partitioning) is deemed be true, two vegetation components ?above-ground-veg and ?below-ground-veg are added to the emerging model. The growth and death of each of these can be modelled via generic or component-specific models. The components themselves can also be further decomposed in the same way as is described here for vegetation decomposition. The model(?veg,aggregate) assumption denotes that a vegetation component is not decomposed any further. The following model fragment illustrates how this assumption is utilised within the setting of modelling population growth:

```
(defModelFragment growth-phenomenon
  :source-participants
    ((?vegetation :type vegetation)
     (?component :type vegetation-live-component))
  :structural-conditions
    ((part-of ?component ?vegetation))
  :assumptions
    ((relevant growth ?vegetation)
     (model ?component aggregated))
```

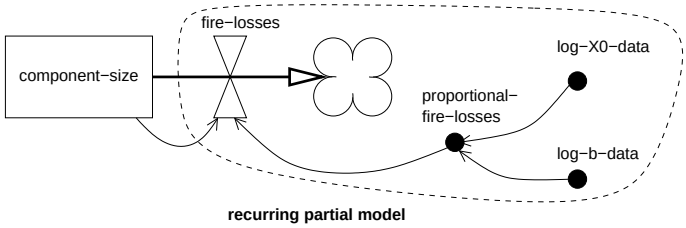


Figure 8.10: Recurring partial model of fire losses

```
:target-participants
  ((?stock :type stock))
:postconditions
  ((size-of ?stock ?component)))
```

It is interesting to note that the functional decomposition approach discussed here can also be applied to granularity selection of components in conventional engineering applications of compositional modelling [45]. It is slightly more general than most existing approaches because the possible component decompositions need not be expressed by means of a single hierarchy. For example, by employing a third assumption model(`?veg`, yet-another-decomposition), alternative ways of functional decomposition can be added to the knowledge base. Also, decompositions of different types of component may yield the same or similar components. As such, functional decomposition can be implemented by means of a component network.

8.2.2.2 Reuse of equivalent partial models

An important motivation for compositional modelling is that it enables reuse of partial model fragments. This is achieved by choosing the scope of the model fragments such that it is small enough that it contains a regularly recurring pattern, whilst it is large enough to be significant as a knowledge entity. The partial model used to describe fire losses in the *n*-species model is one such regularly recurring pattern. More specifically, the partial model shown in figure 8.10 recurs for all potential fuel types (woods, greens, dead wood and litter).

At a higher level, reuse of such knowledge entities may occur within a model if there are recurring subsets of objects and relations in the scenario. Ecological modelling theory typically develops models that can describe interactions for the different species with the same models using different sets of parameters. Hence, the same set

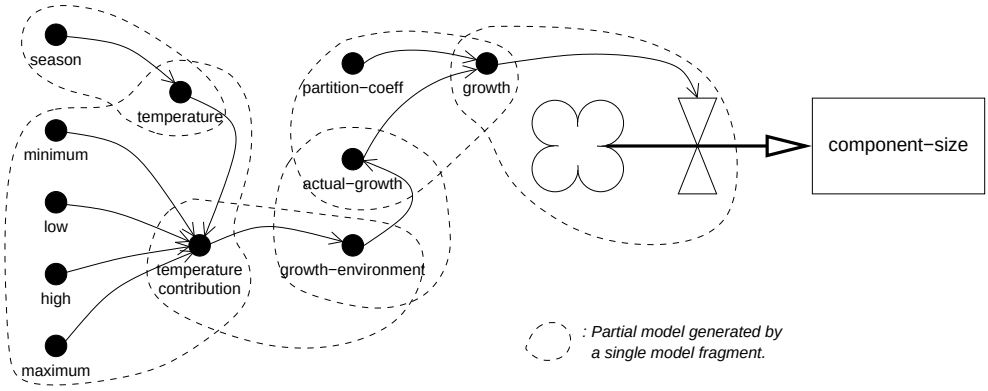


Figure 8.11: Modelling incremental influences

of model fragments can be reused for each of the species mentioned in the scenario. For example, each species in the eco-system interacts with the fire incidents, the water supply, the climate and the season in a way specific to the corresponding species. In fact, each species contributes to the total supply of potential fuel, thus affecting the overall fire hazard, and each species will regenerate differently to damage caused by fire. Each species consumes a portion of the available water supply and each species reacts differently to climate parameters such as average temperature, and drought and frost occurrences. As explained, the differences between responses of various species are described by means of parameters rather than changes in the structure of the model.

8.2.2.3 Incremental elaboration

A regularly recurring feature of ecological system dynamics models is a sequence of variables linked via influences. For example, in figure 8.11, the growth of a vegetation component depends on the total growth and on a partition coefficient. The total growth depends on environmental factors, which in turn depends on the suitability of the temperature. The latter, further down the chain, depends on species-specific temperature ranges and on the actual temperature. Moreover, the actual temperature may vary from season to season.

Such sequences of influences can be incorporated in the knowledge base by sets of dependent model fragments. In the present example, one model fragment introduces the variables describing the growth of greens. A second model fragment introduces the total growth for the species and adds an equation that derives greens growth from the

total growth. The third and fourth model fragments add the remaining two influences to the chain. The advantage of this approach is that the chain of influences can be cut at any point, depending on the needs of the user (e.g. to suit the available data that may serve as input to the simulation model).

Another example of this approach, and of reusing recurring model fragments, is the implementation of seasonal variations. Each parameter that may be modelled as a seasonally changing parameter is represented as a "potentially seasonal parameter" (`pot-seasonal-parameter`). The model fragment is activated for each of such parameters, provided that seasons are incorporated in the model. For each "potentially seasonal parameter", an individual decision is allowed to be made on whether to represent it as a seasonally changing parameter or as a constant. To achieve the former, the following model fragment is used, incorporating a data table containing the parameter value for each season. To achieve the latter, another model fragment needs to be employed.

```
(defModelFragment seasonally-changing-parameters
  :source-participants
    ((?parameter :type pot-seasonal-parameter)
     (?season :type variable)
     (?season-number :type integer))
  :structural-conditions
    ((season ?season)
     (season-number ?season-number))
  :assumptions
    ((model ?parameter seasonal-table))
  :target-participants
    ((?data-table :type (array (1 ?season-number) variable) :name table))
  :postconditions
    ((== ?parameter (?data-table ?season))))
```

Incidentally, this section has highlighted the limitations of flat model fragment structure, which translate scenario level objects directly into scenario model constructs. From a knowledge engineering perspective, using a flat model fragment structure would require that the constraints between model fragments be made explicit, which inhibits maintenance.

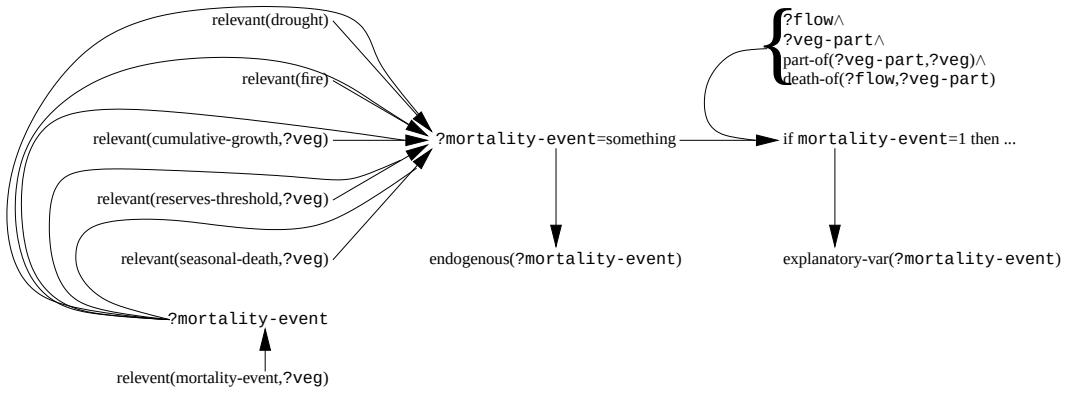


Figure 8.12: Deduction of eradication phenomena

8.2.2.4 Composition of multiple phenomena

Very often, ecological phenomena can not be considered in isolation from each other. Take, for example, the eradication events discussed earlier. A mortality event variable should only be included once per species of vegetation, because it affects all component vegetation and every phenomenon that may trigger a mortality event should set this variable. Therefore, the mortality event variable is created under the assumption of a relevance phenomenon associated with the vegetation: relevant(mortality-event,?veg).

Figure 8.12 shows how the mortality event variable, the equations that set its value and the equations it is used in are created, and what assumptions they each depend on. Note that this does not guarantee that all sensible models are created. The creation of the mortality event variable does not necessarily guarantee that appropriate equations exist where it is assigned and used (because the corresponding assumptions are not always made).

The present compositional modeller has included a tool to resolve this issue in an elegant fashion. In particular, the model fragment that creates the mortality event variable is set to contain a purpose-required property:

```
(defModelFragment mortality-switch
  :source-participants
    ((?vegetation :type vegetation))
  :assumptions
    ((relevant death ?vegetation))
  :purpose-required
    ((endogenous ?mortality-event)
```

```
    (explanatory ?mortality-event))
  :target-participants
    ((?mortality-event :type event-switch :name mortality))
  :postconditions
    ((mortality-event-of ?mortality-event ?vegetation)
     (== ?mortality-event (C-else 0))))
```

This renders any conjunction of assumptions from which an instantiation of this model fragment can be derived (and hence `?mortality-event`) but which does not make the `?mortality-event` instance endogenous and explanatory (i.e. the variable is used in an equation) illegal.

In conventional engineering applications, this situation rarely occurs because system's behaviour stems directly from the components and the connections between the components. This is not the case in ecological systems, where the relation between system structure and system behaviour is less clear. To incorporate such structural knowledge of system's behaviour, conventional compositional modellers would have to consider all potentially illegal (or insensible) combinations of assumptions within the knowledge base. This would obviously complicate the maintenance of the knowledge base and make the model fragments less legible. The work presented in [45] is the only exception to this as it employs constraint satisfaction techniques to validate an emerging model. However, this approach requires that the relevant constraint be specified explicitly rather than be deduced automatically from a model space and predefined properties. As with other conventional approaches, this will complicate the maintenance of the knowledge base, as the meaning of the constraints is less clear than property combinations.

8.2.3 Model construction

Overall, the knowledge base consists of 4 property definitions and over 60 model fragments. Given a scenario that consists of a single species, a dynamic constraint satisfaction problem is constructed that consists of 44 attributes. The overall tightness and density of the compatibility constraints of the problem, however, is very low. To illustrate this, figure 8.13 shows part of this DPCSP (more specifically, the set of attributes whose activity is dependent upon the model assumption that decomposes the vegetation into an above-ground and below-ground partition).

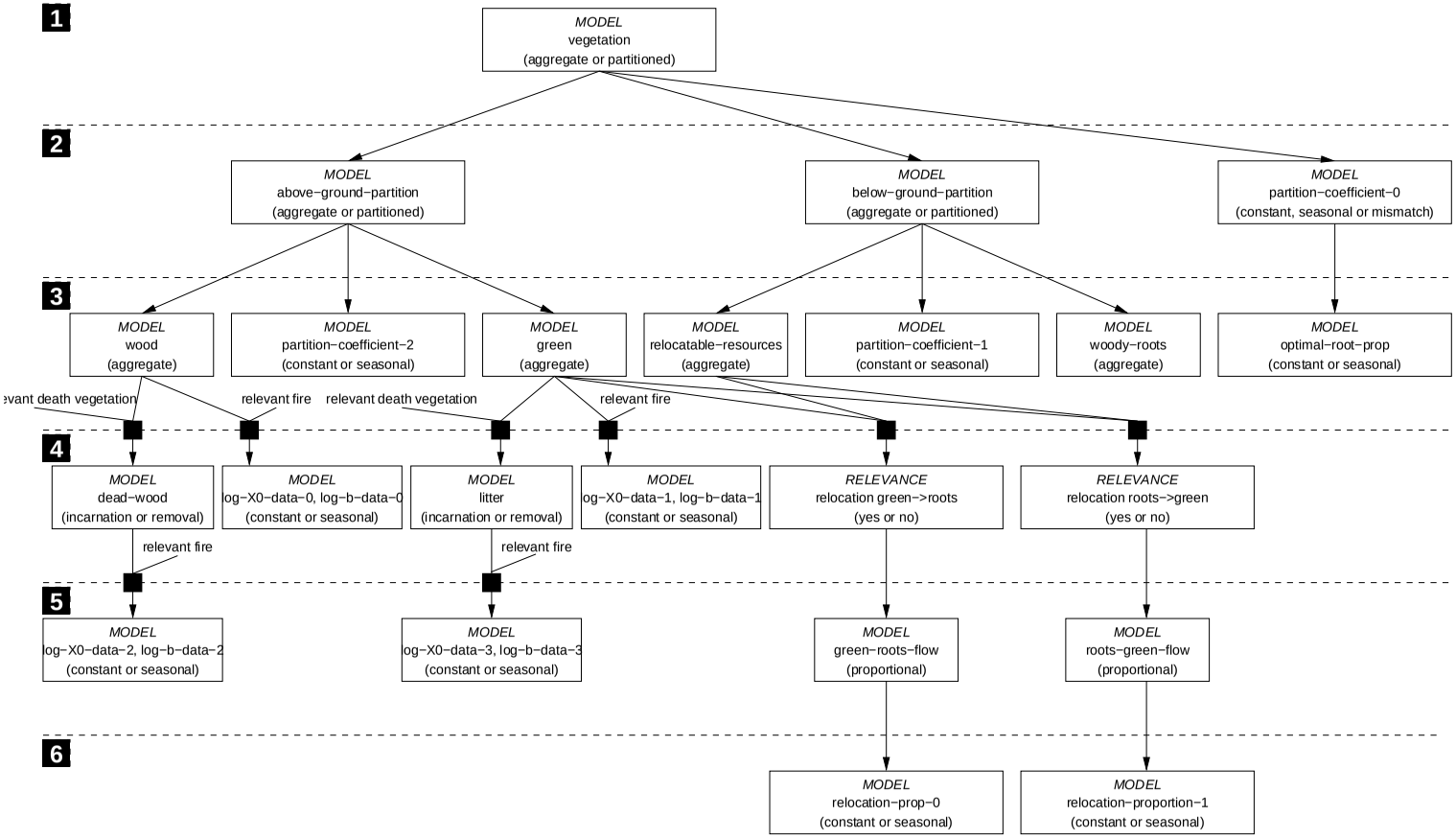


Figure 8.13: Partial DCSP for the *n*-species model

For ease of understanding, the attributes in figure 8.13 are grouped in levels, such that the attributes of one level are activated by assignments to attributes in the previous levels. In this part of the DPCSP, there are only activity constraints.

By choosing a set of preferences, different models can be generated. The choice of preferences may also affect the efficiency of the search, but that is a topic that is discussed in the previous chapter. By assigning the highest preferences to the relevance and model assumptions that correspond to the most detailed models (e.g. prefer the partitioned vegetation model over the aggregate one), the resulting model is shown in figures 8.14 and 8.15. This model matches the original ModMed model closely.

In these figures, the sets of stocks, flows and variables grouped in the same yellow area are generated by a single model fragment. The diamonds correspond to variables that are transferred between figures 8.14 and 8.15. For ease of reading the sample scenario model, coloured squares that indicate important submodels of the n-species model have been added.

In this scenario model, the vegetation is decomposed the wood, green, relocatable energy resource and woody root components. This selection is made through the aforementioned attributes corresponding to model assumptions in the first 2 levels. At this granularity, a number of additional phenomena are possible. For instance,

- *Dead wood*: Although not part of the total vegetation population, dead wood is an important consideration in a forest that experiences regular fires as it adds to total available fuel. Here, the model assumption to represent dead-wood as a new incarnation has been selected (level 3 of the DCSP). The alternative would be to represent it by means of a flow into a sink.

- *Littering*: Similar to dead wood, dead leaves and twigs (called litter) add to the total fuel content. The model for littering, however, depends on the inclusion of the frost-effect phenomenon, which in turn, depends on the inclusion of minimal temperature in the model. As with dead-wood, the model assumption to represent litter as a new incarnation has been selected (level 3 of the DCSP).

- *Resource relocation*: When total available greens and relocatable resources are represented as separate variables, relocation of resources between greens and roots can be modelled. Resource relocation is a poorly understood phenomenon in ecology and hence, the inclusion of such a model for resource relocation may

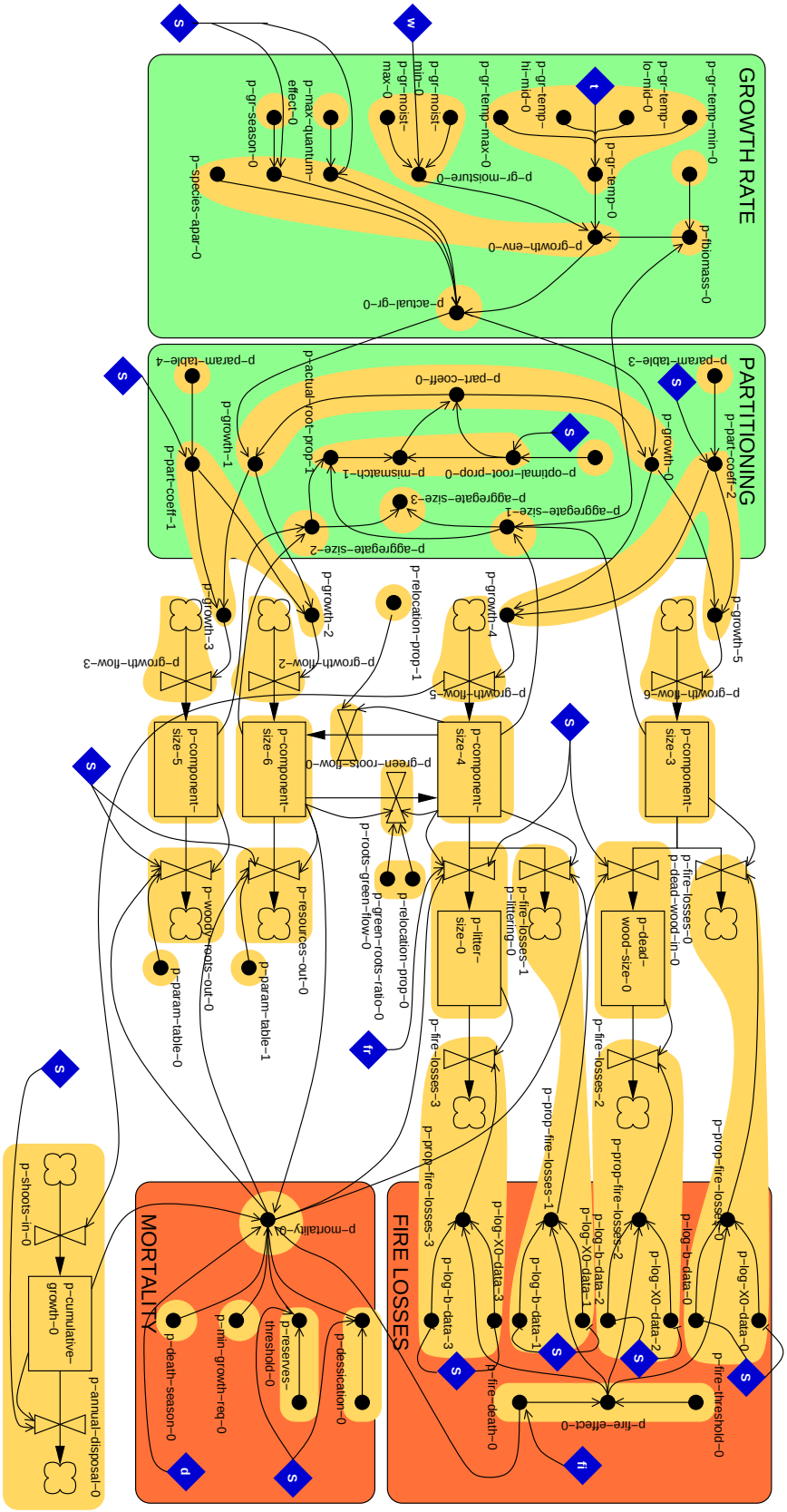


Figure 8.14: Sample partial instance of the n-species model

be deemed debatable by expert ecologists. Therefore, both resource flows (from greens to relocatable resources and from relocatable resources to greens) are considered as two specific phenomena that the user may prefer to include or exclude. Two attributes corresponding to relevance assumptions at level 4 of the DCSP enable this choice.

On the left hand side of figure 8.14 are the variables and influences that affect the total growth and those that describe how this growth is divided (partitioned) between the four functional components. Note that the knowledge base given in appendix C will always construct a submodel that deduces participant `p-actual-gr` representing the total actual growth of the vegetation in a time instance. Depending on the partitioning choices made earlier, the value of this variable is then decomposed by exploiting the so-called partitioning coefficients.

Different approaches can be employed to compute the values of the partitioning coefficients and hence, attributes corresponding to the model assumption at levels 3 and 4, allow for this choice to be made. An additional model assumption, named "mismatch", is available for the partitioning coefficient associated with the {above-ground, below-ground} partition. This assumption corresponds to a model that is specific to this partition: it describes a redistribution of growth energy between the roots and above ground partition if there is a mismatch between the actual and the optimal distribution of energy between both component parts of the vegetation.

The models used to describe the total growth and the actual growth may differ, depending on the inclusion or exclusion of various factors such as seasonality, average temperature, water supply and others as shown in figure 8.15. However, this part of model is not affected by the choice of partitioning and hence, none of the modelling choices available in this part of the model can be found in figure 8.13

On the right hand side of figure 8.14 are the variables and influences that affect losses due to fire and mortality events. Fire losses affect woods, greens, dead wood and litter and each of these is modelled in exactly the same way, although the parameters are specific to the fuel type. Mortality events are situations where the entire species is eradicated. It is described by a boolean variable expressing whether the conditions are correct for the eradication of the species. The model in 8.14 includes a number of phenomena that may trigger an eradication event, more specifically: drought (which depends on a water supply model), extreme fires, seasonal death, insufficient cumula-

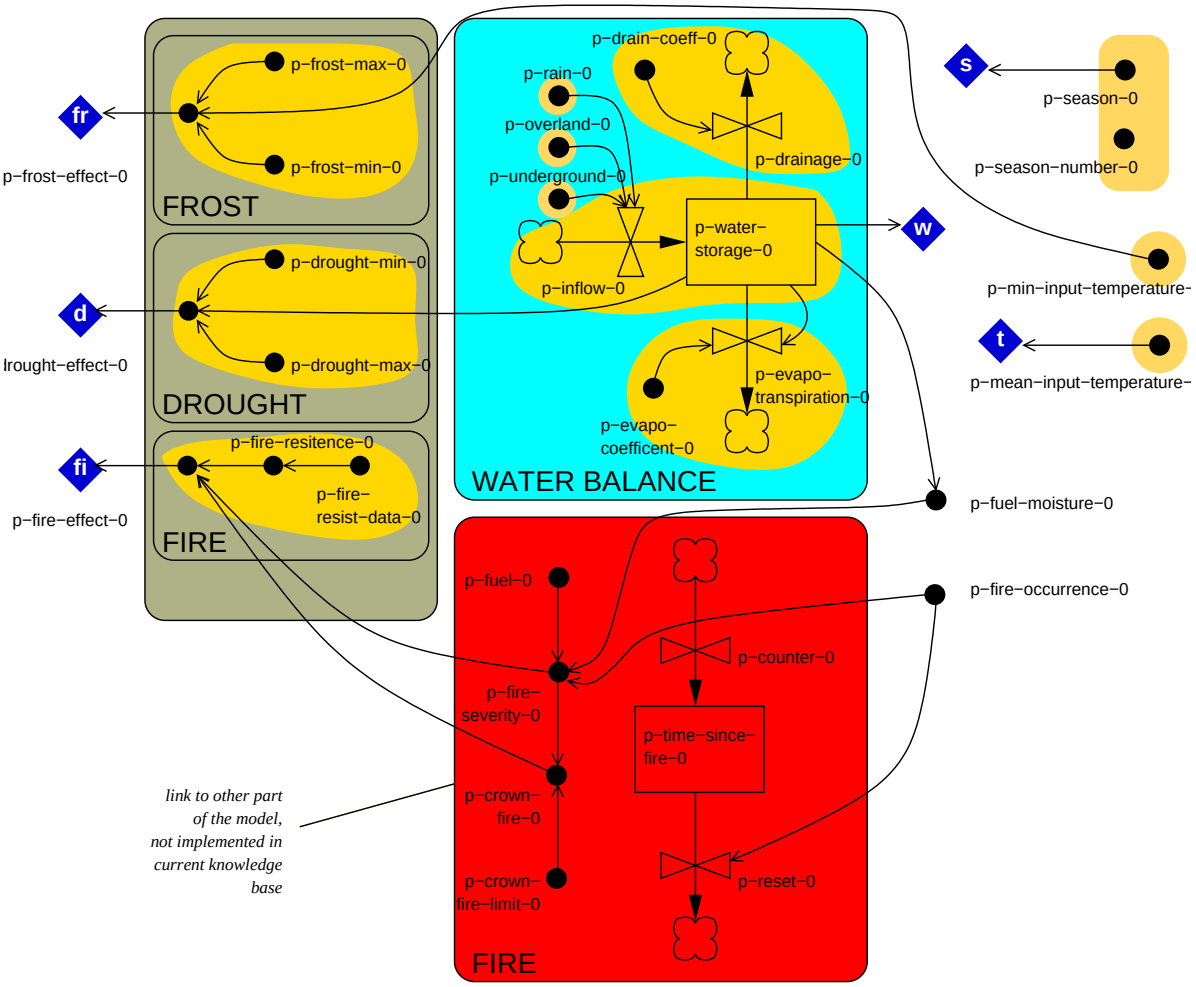


Figure 8.15: Environmental factors in the *n*-species model

tive growth and insufficient reserves.

Of particular relevance to the ongoing example of the partial DCSP of figure 8.13 are the models of fire losses. As discussed in section 8.2.2.2, fire losses are described by a single model fragment that is applicable to vegetation components that are deemed burnable (wood, green, dead-wood and litter are denoted as burnable in the knowledge base). Although only one component model is employed to describe fire losses, the parameters (whose existence depends upon the existence of the component vegetation in question and upon the relevance of the fire phenomenon) can be modelled in different ways (as seasonally varying parameters or as constants). Hence, levels 4 and 5 of the DCSP of figure 8.13 contain attributes that correspond to these model assumptions.

8.3 Summary

This chapter has presented two sample applications of the compositional modeller in more detail. Section 8.1 has presented how a number of ecological models can be translated into a coherent knowledge base that can be employed by the compositional modeller. It was then shown how this knowledge base can be translated into a model space and from there into a dynamic constraint satisfaction problem. Finally, an ecological model was composed from the model fragments in the knowledge base, which is consistent with the ecological modelling theory presented earlier. Section 8.2 has presented a larger application, i.e. the construction and use of a knowledge base, based on a real-world ecological model, the ModMed *n*-species model. Given the size of this application, a complete presentation was not possible. Instead, important modelling issues were highlighted, and the model construction process was discussed by means of a small, but representative portion of the model space.

Chapter 9

Conclusions and Future Work

This chapter concludes the dissertation. In section 9.1, a summary of the thesis is presented and in section 9.2, important pieces of future research, which would be beneficial to complement this work, are outlined.

9.1 Summary of the thesis

This thesis is primarily concerned with compositional modelling of ecological systems, via exploiting the modelling task as representing and solving a constraint satisfaction problem. Based on a critical survey of existing compositional modelling techniques and an analysis of the requirement of automated ecological model construction, the thesis has shown that existing approaches were restrictive in several areas:

- *Abductive reasoning*: Modelling choices in compositional ecological modelling are not only constrained by inconsistencies amongst them, but also by the properties that they must satisfy. Many existing compositional modellers do not provide the truth maintenance or other techniques to support the non-monotonic reasoning involved in finding a set of modelling choices that is both consistent and meeting the imposed criteria.

- *Disaggregation*: Many ecological models describe the behaviour of populations. Populations can be disaggregated into component-populations along multiple dimensions, such as age, spatial location and gender. None of the existing compositional modelling techniques supports this form of granularity selection.

- *Reasoning about subjective preferences*: In the ecological modelling domain, model adequacy is very difficult to assess as it is still poorly understood. As a result, ecologists do not generally agree on what model most appropriately describes a given scenario. These disagreements can be traced back to different opinions guiding the decisions made during the model construction process. Because compositional modellers explicitly represent these modelling decisions in the form of assumptions, the corresponding opinions can be formalised by means of preferences for the assumptions. Existing compositional modellers, however, can only handle hard constraints over the modelling decisions and their inference mechanisms are therefore not equipped to handle such preferences.

- *Interactions between model fragments*: Many compositional modellers employ model fragments that translate scenario level participants and relations directly into participants and relations of the scenario model. This is reflected in the terminology adopted by CML [13], where the source-participants are called participants and the target-participants are either quantities or attributes of the scenario model. In ecological modelling, certain modelling choices for one set of processes may introduce new sets of modelling choices for another set of processes. This can be more easily expressed if more complex interactions between model fragments are allowed.

- *Representation formalisms*: Many compositional modellers make specific assumptions about the representational formalism that is employed and their respective model fragment selection algorithms are based on that formalism. Obviously, different formalisms are used in ecology, and these should be accommodated.

This thesis has made initial attempts to address most of these outstanding issues, and the results have been very positively received by the research community [94, 96, 95, 99, 97]

9.1.1 Knowledge representation formalism

In order to adequately represent ecological modelling knowledge, a new representation formalism was developed in this dissertation. This representation formalism enables

the introduction of the following features to compositional modelling, which should make this approach better suited for the ecological modelling domain:

- *Incremental model specification*: The knowledge representation formalism allows partial models to be specified incrementally in the knowledge base of the automated modeller. As opposed to conventional approaches that impose a flat model fragment structure, the approach presented herein allows the model fragments to be organised in more complex structures. In particular, instances of participants and relations required by the antecedents of certain model fragments can be generated by other model fragments. This is useful as, for example, ecological models often contain sequences of influences, where the result of one influence becomes the input of another. The incremental approach allows such sequences to be broken down in component parts.

- *Formal property definitions*: Ecologists are often interested in models that satisfy certain properties. Therefore, in this thesis, an approach has been developed to formally define such properties and how they are satisfied by a model. The search for an adequate scenario model can then be constrained by such properties. That is, the model composition algorithm can be manipulated to produce scenario models that possess (or that do not possess) certain properties. The representation formalism introduced in this work also allows properties to be specified as required conditions of model fragments. This information is used to further constrain the model construction algorithms, by preventing it from generating scenario models composed of model fragments for which the required properties are not satisfied.

- *Disaggregation*: In this thesis, the way in which an aggregate model can be transformed into disaggregate ones is formally described by means of disaggregation fragments. As disaggregation fragments express the way in which a model can be transformed, rather than the way in which a model is constructed, they are less likely to be affected by updates of the knowledge on model construction. This use of model transformations also implies that multiple disaggregation fragments can be combined with one another. Hence, only individual types of disaggregation and not the combinations of them need to be specified in the knowledge

base, providing a more maintainable knowledge representation than the equivalent set of specifically disaggregated model fragments.

9.1.2 Compositional modelling as a DCSP

The combination of a given scenario description, a set of required properties and a knowledge base constitutes a compositional modelling problem - constructing a model of the scenario, which meets all the given properties, by the use of the knowledge base. Conventional approaches to compositional modelling employ a wide variety of purpose built inference mechanisms and search algorithms to find solution that meet the requirements. However, none of these techniques is equipped to deal with the interdependencies between the model fragments (largely due to the complex structures by which the model fragments are organised) and the requirement of finding scenario models that meet certain predefined properties.

This thesis has introduced a novel approach to solving compositional modelling problems by automatically translating them into DCSPs. This approach has two key benefits. Firstly, it enables the use of efficient existing algorithms to solve DCSPs. Secondly, it accommodates more easily for future extensions to this work, as many extensions may be incorporated into the highly generic DCSP framework, minimising the need for new inference mechanisms to be developed.

In order to construct the DCSP, the given compositional modelling problem is first translated into a model space, which is then translated into a DCSP. The first translation procedure consists of four phases. A hypergraph is built first that contains all participant and relation instances, which can be derived from the scenario description and from the assumption instances referred to in the employed model fragments. This hypergraph, i.e. the model space, is stored using an ALTMS. It is then extended by means of the disaggregation fragments. This involves copying the parts of the model space, specifying the partial model to be disaggregated and applying the model transformations dictated by the disaggregation fragments. Next, the property definitions are instantiated in the model space. That is, for all sets of participant and relation instances that satisfy the property definition, a new node representing the property is created and justified by the aforementioned participant and relation instances. Finally, inconsistent relations are detected in the knowledge base and registered as nogoods. The negations

of required properties are also added as nogoods to the ALTMS.

All the information needed to generate the DCSP is now contained in the model space. Each set of assumptions that have the same subject form the domain of a DCSP attribute. The activity constraints that activate a particular attribute are derived from the ALTMS label of the subject associated with that attribute. Finally, the standard compatibility constraints are derived from the nogood node. A solution to the DCSP that is created in this way corresponds to a consistent set of assumptions that satisfy all of the required properties, resulting in a suitable system model.

9.1.3 Order-of-magnitude preferences

The users of the scenario models generated by a compositional modeller may have different sets of preferences over the available model design decisions, in addition to hard requirements. However, the knowledge about such preferences is typically incomplete and imprecise. To cope with this, a calculus of partially ordered preferences, rooted in order of magnitude reasoning has been developed. It aims at combining basic preferences on a measurement scale, whilst the preferences themselves are defined with only minimal information to realistically accommodate for the limited information available. This has been achieved by combining basic preference quantities (BPQs) that have been ordered using relations that have a simple but well-defined semantics, into order of magnitude preferences (OMPs).

Generally speaking, a naive algorithm to compare partially ordered OMPs would be very inefficient because each pair of constituent BPQs may be related via one or more ordering relations. Therefore, a novel algorithm has been devised in this thesis, which is capable of comparing preferences much more efficiently. This algorithm employs a purpose built representation for describing labels that describe the relative position of each BPQ in the partial ordering. This representation allows all pairs of BPQs that have an ordering imposed between them, to be compared with one another. Because all OMPs are combinations of BPQs, the BPQ labels are combined into a label as well. This latter type of label specifies the relative orderings of OMPs and the algorithm built exploits information contained within such labels to efficiently order two OMPs.

9.1.4 Solution algorithms for the DPCSP

A dynamic preference constraint satisfaction problem (DPCSP) has been defined for the first time in this work as a DCSP in which the domain values are assigned preference valuations. A solution to the DPCSP is consistent with the constraints of the DCSP and maximises the combined preferences of the assigned values. Because no appropriate solution algorithm existed previously, four algorithms have been devised here, to solve such a DPCSP. These algorithms are all best-first search algorithms (or A* algorithms to be exact) that employ different heuristics. The following four heuristics have been employed:

- The basic approach employs a heuristic that computes an upper boundary on the preference of any solution containing the current partial solution. It is assumed that the assignment with the highest preference can be made for each unassigned attribute which is or may be activated.

- A forward checking algorithm improves on the basic approach by exploring all unassigned attributes for each partial solution. In this way, future inconsistencies can be detected early and a more accurate estimate of the upper boundary can be achieved. This algorithm may therefore help avoid unnecessary exploration of parts of the search tree that will lead to less preferred solutions.

- A third approach, called maintaining preference boundaries, learns more accurate upper boundaries as the search progresses. Each time an inconsistency is encountered which restricts the maximum combined preference valuation that can be realised, the algorithm updates a database of upper boundaries. This information helps the algorithm to avoid being misled by repeated overestimates due to incomplete information.

- The final algorithm integrates forward checking with the method of maintaining preference boundaries, exploiting the benefits of both.

Each of these algorithms trades off the informedness of the heuristic with the complexity of computing it. The relative efficiency of them has been evaluated by measuring their relative performance on batches of randomly generated DPCSPs.

Overall, the integration of the four main contributions of this work, i.e. the knowledge representation formalism, the compositional modelling to DCSP translation algorithm, the order of magnitude preference calculus and the DPCSP solution algorithms, offers a compositional modeller suitable for tackling ecological model construction problems. In order to support this claim, the techniques introduced in this dissertation have been applied to two important ecological modelling problems: predator-prey population dynamics and vegetation modelling, and they proved to be successful.

9.2 Future Work

The present research opens up much space for further investigation in both constraint satisfaction and compositional modelling. In particular, the DPCSPs have been found to be a particularly hard to resolve, further work on more efficient DPCSP solution techniques would be desirable.

One strategy of improving the efficiency of the search algorithms in a conventional CSP involves adapting the problem specification by means of graph theory or constraint propagation. The aim of such approaches is to exploit the structure of the CSP such that backtracking or other conventional search algorithms require less work. An investigation into how such techniques can be adapted to suit the needs of a DPCSP (see [169, 119] for more details on such approaches) is an interesting research topic, but one that is beyond the scope of this dissertation. In sections 9.2.1 and 9.2.2, two approaches of particular relevance to DPCSPs for compositional modelling are introduced instead.

Another strategy involves the development of alternative search techniques. Given the absence of useful local search techniques to improve the efficiency of DPCSP solution algorithms, genetic algorithms may provide a promising option. Therefore, in section 9.2.3, an approach is outlined to apply genetic algorithms to solve DPCSP.

In addition, some important issues in compositional modelling remain. The primary outstanding issue in compositional ecological modelling is development of future applications. This would require techniques for elicitation or automatic generation of suitable preferences and required properties for the space of available models, as discussed in section 9.2.4. In addition, it would also be useful to investigate the potential generalisation of the application domain of compositional modelling, which

is discussed in sections 9.2.5 and 9.2.6.

9.2.1 Independent subproblems in a DPCSP

Ideally, a problem can be decomposed into independent subproblems. Consider a hard CSP:

$$\begin{aligned}\mathbf{X} &= \{x_1,\ldots,x_n\}\\ \mathbf{D} &= \{D_1,\ldots,D_n\}\\ &\quad \text{where } D_i = \{d_{i1},\ldots,d_{ij_i}\}\\ \mathbf{C} &= \{c_{Y_1},\ldots,c_{Y_m}\}\\ &\quad \text{where } Y_i \subset \mathbf{X}, \text{ and}\\ &\quad c_{\{x_i,\ldots,x_j\}} : D_i \times \ldots \times D_j \rightarrow \{\top,\bot\}\end{aligned}$$

Assume that there are m values in each domain and that d_C is the relation of all pairs of attributes whose consistent assignments are constrained by one another via a compatibility constraint. Clearly, d_C is the relation defined over $\mathbf{X}\times\mathbf{X}$ such that $(x_i,x_j)\in d_C$, if there exists a constraint $c_Y \in \mathbf{C}$ over $x_i \in Y$ and $x_j \in Y$. If a partition $X_1,\ldots,X_k$ of $\mathbf{X}$ can be found, such that:

$$\forall x_i \in X_v, \forall x_j \in X_w, (x_i,x_j) \notin d_C, \quad \text{with } v \neq w \tag{9.1}$$

it would be better to solve the k smaller problems that, respectively, only involve the attributes in one of $X_1,\ldots,X_k$, rather than solving the problem as a whole [38]. In the original problem, the total search space contains m^n nodes. If this problem can be decomposed into k subCSPs with, say, $\frac{n}{k}$ attributes each, then the overall size of the search space is reduced to $km^{\frac{n}{k}}$, which is significantly smaller than m^n [169].

As explained earlier, a DCSP is a hard CSP with a set of activity constraints. For the purposes of this section, a set of activity constraints $\mathbf{A}$ are defined as:

$$\begin{aligned}\mathbf{A} &= \{a_{Y_1,x_v},\ldots,a_{Y_m,x_w}\}\\ &\quad \text{where } Y_i \subset \mathbf{X}, \text{ and}\\ &\quad a_{\{x_i,\ldots,x_j\},x_l} : D_i \times \ldots \times D_j \times \{\text{active}(x_l), \neg\text{active}(x_l)\} \rightarrow \{\top,\bot\}\end{aligned}$$

Let d_A be a relation defined over $\mathbf{X}\times\mathbf{X}$ containing all pairs (x_i,x_j) of attributes, where

the assignment of x_i may affect whether or not x_j is activated. That is:

$$d_A = \{(x_i, x_j) \mid x_i, x_j \in \mathbf{X}, ((\exists a_{Y,x_j} \in \mathbf{A}, x_i \in Y) \vee (\exists a_{Y,x_k} \in \mathbf{A}, x_i \in Y, (x_k, x_j) \in d_A))\}$$

In a DCSP, the activity constraints can be safely ignored when considering the decomposition of the DCSP into smaller subproblems because the activity constraints are triggered in isolation from the compatibility constraints when there are no more attributes to instantiate. In a DPCSP, however, the instantiation of attributes is affected by the preference assignments of attributes that can still be activated. Two attributes $x_i, x_j \in \mathbf{X}$ can not be assigned in isolation from each other when they may activate the same attribute or two attributes related via compatibility constraints, because this will otherwise affect the assignments chosen for x_i and x_j via the estimate of PP. Therefore, for any partition $X_1, \ldots, X_k$ of $\mathbf{X}$ such that (9.1) holds and that,

$$\forall x_i \in X_v, \forall x_j \in X_w, \nexists x_k, x_l \in \mathbf{X}, (x_i, x_k) \in d_A \wedge (x_j, x_l) \in d_A \wedge ((x_k, x_l) \in d_C \vee (x_k = x_l)) \quad \text{with } v \neq w \tag{9.2}$$

it is possible to solve the k smaller problems rather than the whole problem.

Figure 9.1 (with the representation borrowed from [169]) illustrates the reduction in the search space generated by a DCSP or DPCSP. Instead of generating a search space where all combinations of attribute value assignments are considered (denoted by the gray triangle), several search spaces are generated (denoted by the coloured triangles) that can explored in isolation (or in parallel). Each of the independent search spaces may activate a new set of sub search spaces that can be solved independently and from which the algorithm only needs to backtrack to the partial search space that activated the attributes.

Note that a generalisation of this form of problem decomposition is implicitly performed by approaches that employ local repair techniques [120, 177]. For example, as constraints are violated to some extent in a DFCSP, the flexible local repair algorithm (FLR) [118] attempts to repair the degree of constraint satisfaction by reassigning only those attributes that were involved in the constraints in the first place. As such, FLR would consider assignments to attributes involved in independent subproblems in isolation. Of course, as explained in section 6.4, FLR and similar techniques

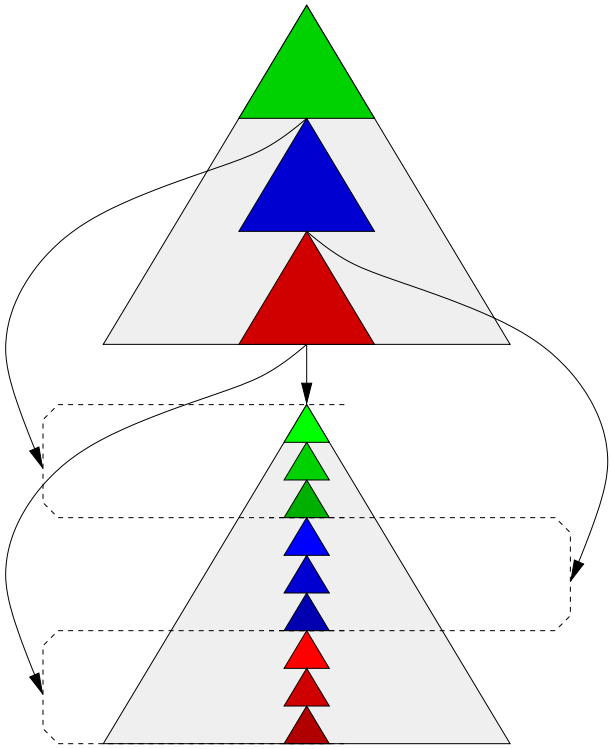


Figure 9.1: Decomposition of subCSPs

are not applicable to DPCSPs in general, because of the less restrictive presumptions of its preference calculus. Therefore, the explicit problem decomposition algorithms discussed in this section would be required in order to benefit from the existence of independent subproblems.

In a compositional modelling problem, this form of decomposition may be applicable to all but the most trivial problems. Scenarios will usually consist of multiple processes and components. At the highest level, these will be described by modelling constructs that are mostly all related to one another (e.g. stocks and flows in a system dynamics model). At more detailed levels, the model description will focus on specific component parts of the model. Such descriptions tend to be totally unrelated to other detailed features of the model and hence involve modelling assumptions that are unrelated to those other parts. Indeed, if this were not the case, the model construction task would be too difficult for human experts to perform too.

9.2.2 Problem decomposition and solution reuse

The method employed for the generation of DPCSPs in compositional modelling (see chapter 4) also allows for the development of novel approaches to improve the efficiency of the solution algorithms. In particular, DPCSPs representing compositional modelling problems may contain multiple instances of the same partial DPCSP. Whenever a scenario consists of multiple subsets of participants that are related in the same way, the same partial model space and hence, the same partial DPCSP may be constructed, depending on the way the preferences are assigned. For example, a scenario containing multiple predator and prey species would be translated into a model space that contains the same sets of growth models for the predators and prey species and the same sets of predator-prey models for each pair of predator and prey.

In general, a given scenario $\langle P, R\rangle$ contains a recurring partial scenario $\langle P', R'\rangle$, where $P' = \{p_1, \ldots, p_v\}$ if, for more than one subset of participants $P_1 = \{p_{11}, \ldots, p_{1v}\} \subset P$,..., and $P_n = \{p_{n1}, \ldots, p_{nv}\} \subset P$ and for more than one subset of relations $R_1 \subset R$, ..., and $R_n \subset R$, there exist n substitutions $\sigma_1, \ldots, \sigma_n$ such that:

$$\sigma_1 = \{p_1/p_{11}, \ldots, p_v/p_{1v}\}$$
$$\vdots$$
$$\sigma_n = \{p_1/p_{n1}, \ldots, p_v/p_{nv}\}$$

and that $\sigma_1 R' = R_1$, ..., and $\sigma_n R' = R_n$.

In figure 9.2 a scenario is shown which contains multiple sets of participant instances of the same type (i.e. $\{a_1, b_1, c_1, d_1\}$, ..., and $\{a_n, b_n, c_n, d_n\}$) that have the same relations between them. The algorithm GENERATEMODELSPACE() (see page 95) will generate equivalent partial model spaces as indicated in figure 9.2. These are,

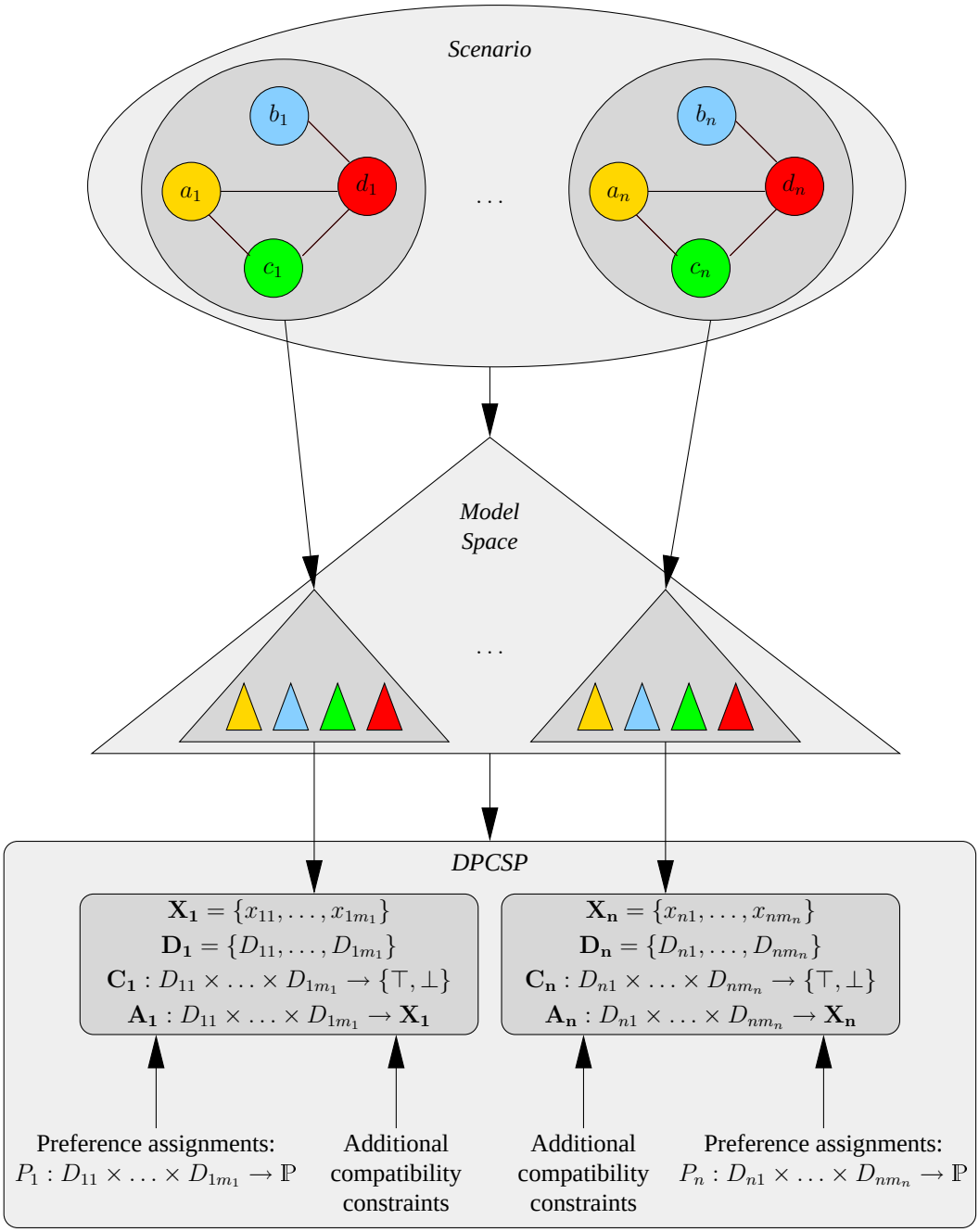


Figure 9.2: Scenario based decomposition

in turn, translated into a set of partial DPCSPs:

$$
\begin{array}{lll}
\text{DPCSP}_1 = \langle \mathbf{X_1}, \mathbf{D_1}, \mathbf{C_1}, \mathbf{A_1}, \mathbf{P_1} \rangle : & & \text{DPCSP}_n = \langle \mathbf{X_n}, \mathbf{D_n}, \mathbf{C_n}, \mathbf{A_n}, \mathbf{P_n} \rangle : \\
\mathbf{X_1} = \{x_{11}, \ldots, x_{1m}\} & & \mathbf{X_n} = \{x_{n1}, \ldots, x_{nm}\} \\
\mathbf{D_1} = \{D_{11}, \ldots, D_{1m}\} & & \mathbf{D_n} = \{D_{n1}, \ldots, D_{nm}\} \\
\mathbf{C_1} : D_{11} \times \ldots \times D_{1m} \rightarrow \{\top, \bot\} & \cdots & \mathbf{C_n} : D_{n1} \times \ldots \times D_{nm} \rightarrow \{\top, \bot\} \\
\mathbf{A_1} : D_{11} \times \ldots \times D_{1m} \rightarrow \mathbf{X_1} & & \mathbf{A_n} : D_{n1} \times \ldots \times D_{nm} \rightarrow \mathbf{X_n} \\
\mathbf{P_1} : D_{11} \times \ldots \times D_{1m} \rightarrow \mathbb{P} & & \mathbf{P_n} : D_{11} \times \ldots \times D_{1m} \rightarrow \mathbb{P} \\
\text{with } D_{1i} = \{d_{1i1}, \ldots, d_{1ij_{1i}}\} & & \text{with } D_{ni} = \{d_{ni1}, \ldots, d_{nij_{ni}}\}
\end{array}
$$

for which n substitutions exist:

$$
\begin{array}{c}
\sigma_1 = \{x_{11}/x_1, \ldots, x_{1m}/x_m, d_{111}/d_{11} \ldots, d_{11j_{11}}/d_{1j_1}, \ldots, d_{1m1}/d_{m1}, \ldots, d_{1mj_{1m}}/d_{mj_m}\} \\
\vdots \\
\sigma_n = \{x_{n1}/x_1, \ldots, x_{nm}/x_m, d_{n11}/d_{11} \ldots, d_{n1j_{11}}/d_{1j_1}, \ldots, d_{nm1}/d_{m1}, \ldots, d_{nmj_{1m}}/d_{mj_m}\}
\end{array}
$$

such that $\sigma_1\mathbf{X_1} = \ldots = \sigma_n\mathbf{X_n} = \mathbf{X}'$, $\sigma_1\mathbf{D_1} = \ldots = \sigma_n\mathbf{D_n} = \mathbf{D}'$, $\sigma_1\mathbf{C_1} = \ldots = \sigma_n\mathbf{C_n} = \mathbf{C}'$, and $\sigma_1\mathbf{A_1} = \ldots = \sigma_n\mathbf{A_n} = \mathbf{A}'$.

Typically, there will be a number of additional constraints over the attributes in $\mathbf{X_1}$, …, and $\mathbf{X_n}$, relating them to other attributes in the original, decomposed DPCSP and these are not necessarily equivalent. Also, it is not necessarily the case that $\sigma_1\mathbf{P_1} = \ldots = \sigma_n\mathbf{P_n} = \mathbf{P}'$.

However, in many cases, there should be no need for different preference assignments or different sets additional constraints. Advanced modelling tools such as Simile provide constructs that enable the same partial model to be instantiated several times, for different instances of the same type of component or process in the ecological system [130, 129]. The use of this feature in Simile corresponds the case of identical preference assignments and additional constraints for DPCSPs $\text{DPCSP}_1, \ldots, \text{DPCSP}_n$. In this case, only the DPCSP $\langle \mathbf{X}', \mathbf{D}', \mathbf{C}', \mathbf{A}', \mathbf{P}' \rangle$ needs to be solved and the result can be translated back via the substitutions $\sigma_1, \ldots, \sigma_n$. A future research challenge involves finding a similar approach for cases where there are minor differences between the preference assignments or the additional constraints.

9.2.3 Using genetic algorithms as alternative search methods

In [170], it has been demonstrated that genetic algorithms (GAs) are a suitable approach for solving valued constraint satisfaction problems, which are relatively loosely constrained and where a near optimal solution may suffice. Strict optimality of the solution is not always absolutely vital for many compositional modelling problems in ecology (and indeed in many other domains such as configuration). However, the use of GAs to solve a DPCSP introduces some new challenges to research in developing suitable GAs themselves.

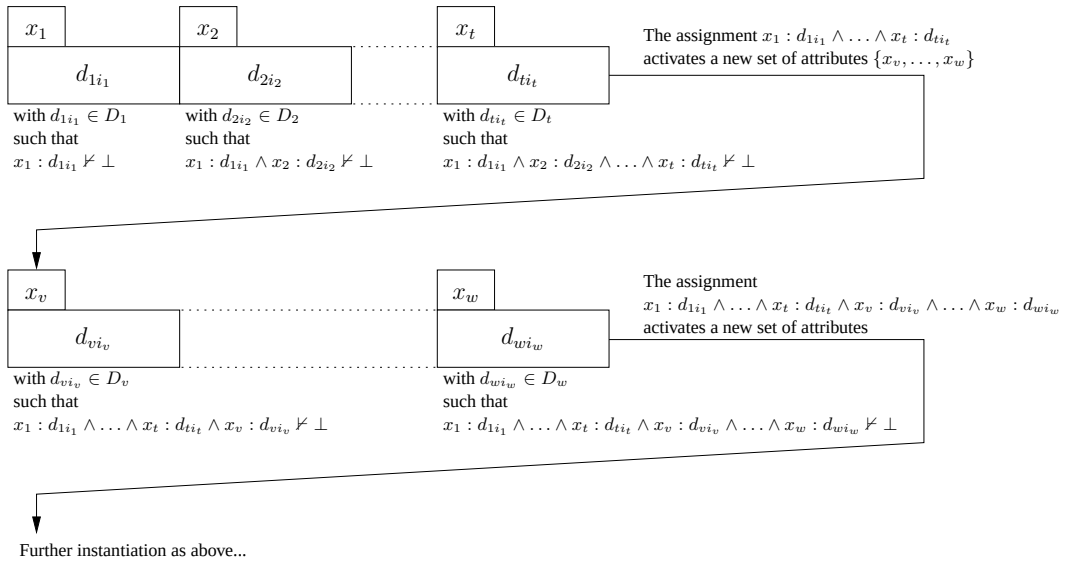


Figure 9.3: Chromosome encoding of a DPCSP

The existing approach for encoding valued (non-dynamic) CSPs as a GA, as presented in [170], can be adapted to accommodate the needs of DPCSPs. Figure 9.3 shows how the chromosomes for a GA solving a DPCSP can be encoded and constructed. Let $\{x_1, x_2, \ldots, x_t\}$ be the attributes that are initially active. The attributes are assigned in a random order, but each new assignment has to be consistent with the previous assignments. As suggested in [170], forward checking may be used to reduce the domains from which values are taken in order to avoid some of the potential backtracking in this process. Once the attributes $x_1, x_2, \ldots, x_t$ have been assigned, the activity constraints are applied and a new set of attributes $\{x_v, \ldots, x_w\}$ is activated. These attributes are consistently assigned in the same way as $\{x_1, x_2, \ldots, x_t\}$ were. This process of successive attribute assignments and attribute activations is repeated until

no more attributes can be activated.

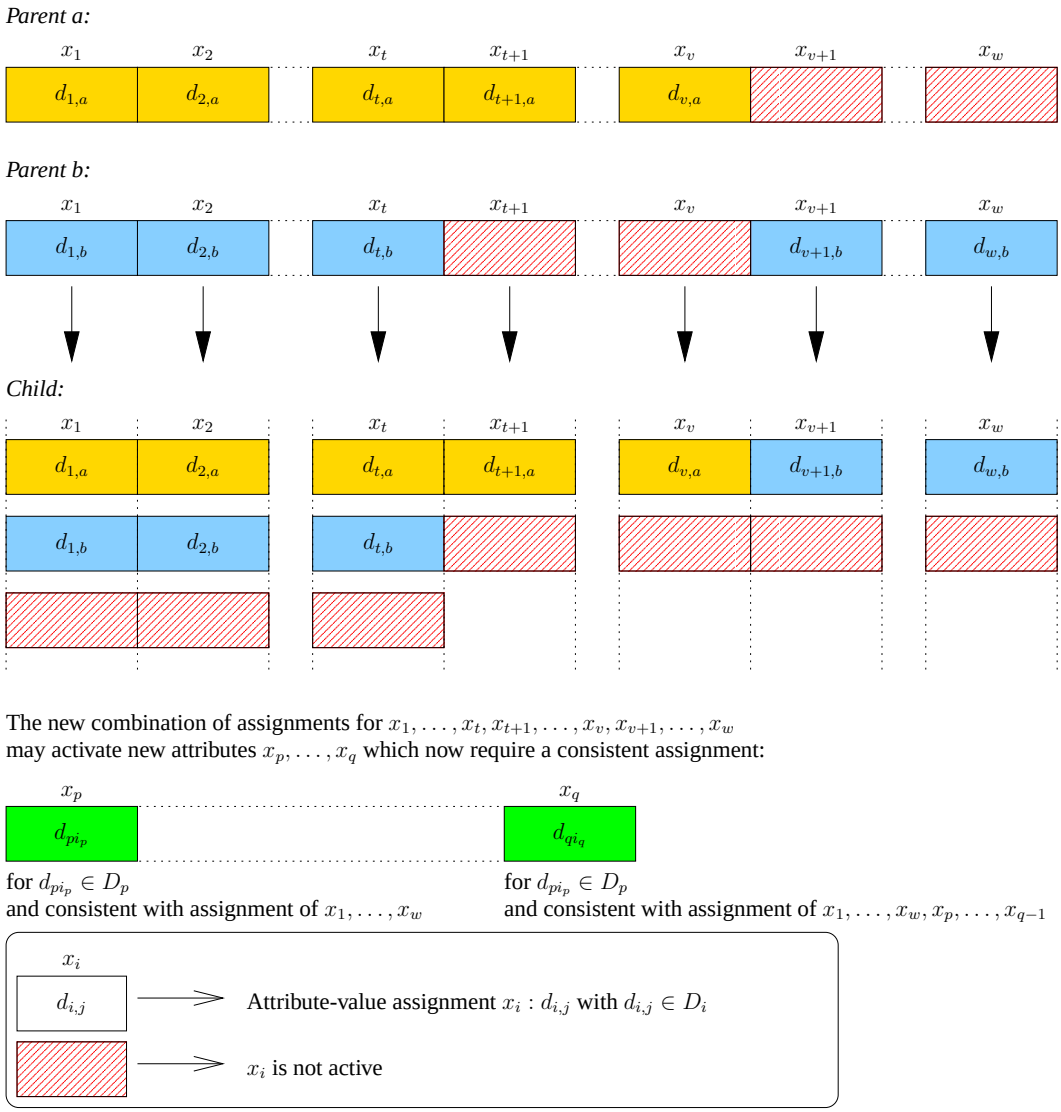


Figure 9.4: Crossover of DPCSP solutions

Every time a population of chromosomes has been generated, a set of chromosome pairs are selected for reproduction. GAs employ a number of techniques for this, some of which are not compatible with the symbolic calculus suggested in this thesis. For example, *roulette wheel selection* [78] and *stochastic universal sampling* [6] both take a random sample of chromosomes from the population by means of a procedure whereby the probability of a chromosome being selected is proportional to its fitness. A number of variations of these methods, such as *sigma scaling* [122] and *Boltsmann selection* [64, 142] bias propositionality to fitness in order to improve the performance of the

GA. However, as explained in 5.2.2, only ordinal information is available with respect to the order of magnitude preferences and hence, no scale information is available on the fitness with which to compute the required probabilities.

Other selection methods exist that only require ordering information. In *rank selection* [5], the chromosomes are totally ordered in terms of fitness. Then, a probability for selection is computed for each chromosome proportionate to its rank. Based on these probabilities, a random sample is taken from the population. Note that if the preferences are only partially ordered as they are in the present work, any imposed total ordering puts some chromosomes at an unfair advantage, whether incomparable chromosomes are given equal rank or not.

Perhaps a fairer approach is tournament selection. In *tournament selection* [65], a tournament is simulated between the chromosomes. Pairs are taken at random from the population according to a uniform distribution. Based on the value of a random number, either the higher or the lower of the two is selected. Of course, the odds of this experiment are in favour of the higher preference and these odds may be changed in order to tune the speed at which the population converges.

Figure 9.4 shows how crossover can be applied to a pair of chromosomes. In essence, the same approach is followed in [170]. That is, given a set of two or more (parent) chromosomes, genes are taken at random from one or the other chromosome, as long as this does not lead to inconsistency. Similar to the construction of the initial population of chromosomes, this procedure is first applied for attributes that are initially active. Then, the activity constraints are applied and the procedure is repeated for the genes that correspond to the newly activated attributes. It is possible that new attributes are activated that were not active in either of the parents. For these attributes, new random values will have to be assigned. In order to reduce the risk that new attributes are introduced to the problem, the crossover procedure could be applied to sets of more than two parents.

Finally, after the crossover phase, mutation may be applied by randomly changing the assignments in some of the genes to an alternative but consistent value.

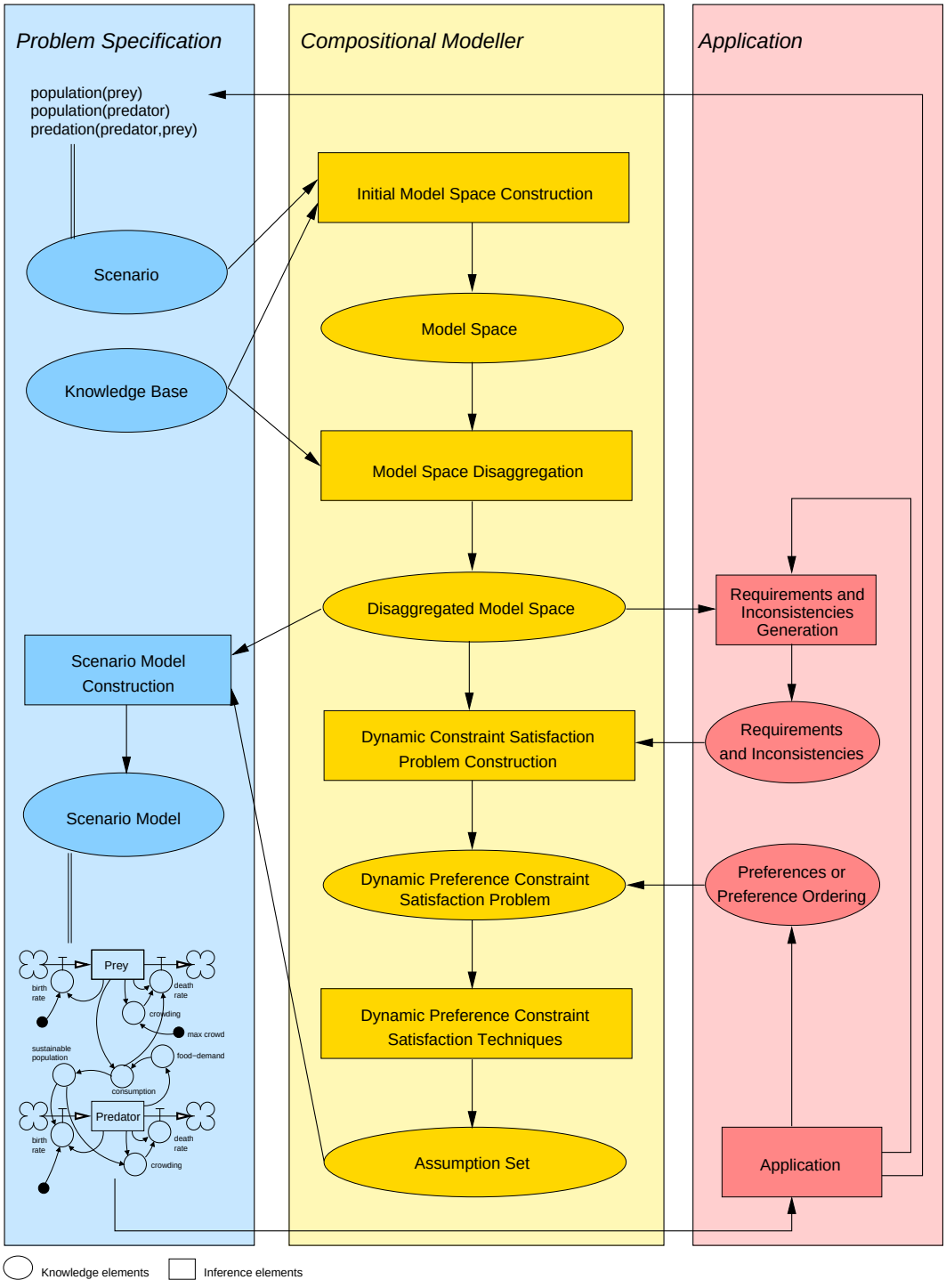


Figure 9.5: Architecture of the compositional ecological modeller

9.2.4 Compositional ecological modelling applications

Given the compositional modelling and constraint satisfaction techniques devised in this dissertation, their combined use gives rise to a general framework for a compositional ecological modeller, as depicted in figure 9.5. However, one important component part must still be developed, in order to complete this architecture: a compositional ecological modelling application module. Such a module would provide two important functions to the overall framework:

- *Preference and property elicitation*: As shown in figure 9.5, an application of the compositional modeller that requires ecological models as its input would need to provide a set of preferences and properties to guide model composition. In this work, it was assumed that the preferences and properties were provided by an external source. To that end, convenient formalisms have been developed to specify them, thus providing an interface for future applications. However, the automated generation of preferences and properties poses a number of new challenges. In particular, the types of knowledge required for preference and property elicitation and the algorithms suitable for handling the corresponding inference must be developed. Interesting initial work for this is nearing completion. For example, in [22], ontologies are developed to aid in the automated selection of ecological models based on the available data. An avenue of future research would be to integrate the use of such approaches to support the present work.

- *Adaptive compositional ecological modelling*: In certain types of application, the generated models may need to invoke a revision of the required properties and the elicited preferences. For example, in a model-based diagnostic application [33], the behaviour extrapolated from a generated model may match the observed behaviour only partially. In figure 9.5, this feature is represented by a feedback loop from the scenario model to the preferences and properties supporting the scenario model. For future applications, the need for such a feedback loop must be investigated and, if necessary, the mechanisms required for its implementation must be devised. With respect to the latter, the future research should address two issues: a) how the properties and preferences responsible for the undesirable features of the scenario model may be determined, and b) what

DPCSP techniques can be developed to repair an existing solution, such that it satisfies the constraints imposed by the new properties and that it maximises the new set of preferences.

The range of potential applications of the compositional ecological modeller presented herein can be quite extensive. The domain model-based reasoning, in which compositional modelling plays a very prominent role, provides a particularly useful source of inspiration. The following tasks are identified as potentially suitable application domains for the compositional ecological modeller:

- *Model-based monitoring* aims at establishing the current state of a system under investigation by determining which model most appropriately describes current behaviour of the system. Although model-based monitoring is primarily applied to physical systems, monitoring is an important task in ecology. For example, in the ongoing climate debate, scientists try to establish whether or not the world's climate is in a state of global warming. Here, a compositional modeller might provide a repository of models and help expose what assumptions are made to support the arguments of different sides in the debate.

- *Model-based control* applications try to regulate the behaviour of a system such that it remains within a certain performance specification. As with model-based monitoring, the role of a compositional modeller is to provide the models that must extrapolate future states of the systems. In the particular case of model-based control, the models should reflect the effects of actions by the controller. In this way, the most appropriate course of action can be determined by extrapolating the future states of the system under different control behaviours. Control is also an important application in ecology. For example, in pest control, ecologists try to keep the population of an unwanted species as small as possible by introducing chemicals or natural predators in the eco-system. Currently, the evaluation of alternative strategies is performed manually, but a model-based control application using a compositional modeller could significantly speed up this cycle of strategy formulation, model building and simulation.

- *Model-based diagnosis* tries to determine the causes of a certain unwanted behaviour of a system, in order to generate a remedy. Here, compositional modellers are used to produce models describing plausible alternative hypothesised

states. The diagnosis application aims to find a state that implies a model whose predicted behaviour matches the observed behaviour. As in the physical systems domain, model-based diagnosis can be utilised in ecology to find an explanation for an undesirable state of the eco-system people live in. The potential benefits of such applications have been demonstrated in [77, 166], where compositional modelling is used to find the causes for inadequate water supplies in developing communities in India.

9.2.5 Generalised applications of compositional modelling

On the applications side, this work has focussed on system dynamics models in ecology. As a modelling formalism, system dynamics has also been successfully adopted in other areas, such as bio-technology [126], business process re-engineering [176], chemicals [161], decision support [104], energy [55, 127], financial services [41, 159], health care [84], macro-economics [59, 60], telecommunications [111], transport [66, 161] and utilities [23, 24]. It is very interesting to examine the applicability of the techniques developed in this thesis to these domains as well.

The compositional modelling techniques devised here are domain-independent. Hence, they should be applicable to other suitable representational formalisms as well. In particular, the preliminary application of compositional modelling to OOP models discussed in [74] was very encouraging. This work produced a compositional modelling based approach to translate ecological models from one formalism to another. To that end, two compositional modelling knowledge bases are used, each employing a different formalism (here, system dynamics and OOP models). By means of one knowledge base, this approach translates a given model into a scenario and a set of assumptions and by means of the second knowledge base, a new model, based on a different formalism, is constructed.

The compositional modelling techniques may also be applicable to modelling conventional physical systems. The experience gained with the extraction of the domain theory contained in the ModMed *n*-species model by means of compositional modelling provides a sensible foundation for this. When typical physical system's features were encountered in the *n*-species model, such as a component hierarchy (more specifically, the taxonomy of components of vegetations in the *n*-species model), the com-

positional modeller was capable of representing this knowledge and reasoning with it, as discussed in section 8.2.

To aid more generalised applications, improvements to the knowledge representation and inference engine should be made. For pragmatic reasons, this work has made distinctions between the discovery of inconsistent (i.e. non-composable) assignment relations and properties, and those between equation processing of composable assignments and the model fragment library. However, with extensions to the inference engine, the concept of inconsistent assignment relations can be specified as a property:

```
(defproperty inconsistent
  :source-participants ((?m :type variable))
  :structural-conditions ((?r1 :definition (== ?m *1))
                          (?r2 :definition (== ?m *2))
                          (not (= *1 *2))
                          (not (composable *1 *2)))
  :property (inconsistent ?r1 ?r2))
```

where `?r1` and `?r2` are two relations that respectively assign formulae `*1` and `*2` to a variable `?m`, such that `*1` and `*2` are not equal and not composable. With similar improvements to the inference engine, the composition operation of composable relations (see section 3.3.1) could be generalised as well. Currently, the composable relations in the model space that that selected during the model construction process are composed into complete relations by means of the rules summarised in table 3.1. These rules are hardcoded in the system and may have to change if a different application domain requires new types of composable relation. However, by extending the inference engine such that it can perform meta-reasoning about relations and parts of relations, such rules can be coded by means of relation definitions. For example, the composable addition rule of table 3.1 might be encoded in the knowledge base as follows:

```
(defRelation composable-addition
  :<= ((?r1 :definition (== ?m (C+ *1)))
       (?r2 :definition (== ?m (C+ *2))))
  :=> ((assert (== ?m (C+ (+ *1 *2))))
       (retract ?r1)
       (retract ?r2)))
```

This relation definition takes two composable relations `(== ?m (C+ *1))` and `(== ?m (C+ *2))` and replaces it by a new relation `(== ?m (C+ (+ *1 *2)))`. Obviously, by formally representing the knowledge employed to determine what partial models

are inconsistent and how composable relations are composed, it should be easier to extend the current compositional modeller to suit new application domains.

In conjunction with the generalisation of the existing knowledge representation and the inference engine, another useful development would involve enabling the use of macros. Macros translate new types of statement into the existing language. As such, they are commonly used in high-level programming languages, such as lisp, to extend the set of available constructs. For example, the functional decomposition of components in subcomponents could be generalised by a purpose built construct. As such, this could become a particularly powerful feature that would encourage the enrichment of the existing language with suitable domain dependent features.

9.2.6 Generalised analysis of compositional modellers

Constraints and preferences are both powerful and domain independent approaches to represent and solve complex problems. Although the existing compositional modelling approaches in the literature rarely use constraint satisfaction techniques, all compositional modellers find scenario models by applying one or more search algorithms to a representation of the space of all models. Therefore, an interesting area of research would involve finding answers to the following questions:

- How can existing compositional modellers be represented by a constraint satisfaction problem and resolved using the corresponding solution technique?
- What are the characteristics of these constraint satisfaction problems?
- What features of the existing compositional modelling problems can be translated into a certain type of CSP, and what features are domain-dependent and can not be detached from the original problem specification (and must therefore be resolved using domain-dependent algorithms)?

Answers to these questions would expose the many commonalities that exist between the inference engines employed by compositional modellers. As such, they would help in the development of more domain independent tools for automated model construction and encourage the reuse of existing knowledge bases.

Appendix A

Proofs of the Theorems

Theorem 4.2. If the labels $\mathcal{L}(n(d_1))$ and $\mathcal{L}(n(d_2))$ are complete and sound, the labels $\mathcal{L}(n(d_1 \wedge d_2))$, $\mathcal{L}(n(d_1 \vee d_2))$ and $\mathcal{L}(n(\neg d_1))$, as respectively computed in (4.1), (4.2) and (4.3) are complete.
Proof: The proof of this theorem is broken down in three parts that respectively demonstrate that $\mathcal{L}(n(d_1 \wedge d_2))$, $\mathcal{L}(n(d_1 \vee d_2))$ and $\mathcal{L}(n(\neg d_1))$ are complete.

- $\mathcal{L}(n(d_1 \wedge d_2))$ is complete, or $\forall E_i, (\wedge_{a \in E_i} a) \vdash d_1 \wedge d_2 \rightarrow (\exists E_j \in \mathcal{L}(n(d_1 \wedge d_2)), E_j \subseteq E_i)$:

  Assume that E_i is a set of nodes such that $(\wedge_{a \in E_i} a) \vdash d_1 \wedge d_2$. Because the labels $\mathcal{L}(n(d_1))$ and $\mathcal{L}(n(d_2))$ are complete, there must exist a $E_k \in \mathcal{L}(n(d_1))$ and a $E_l \in \mathcal{L}(n(d_2))$ such that

  $$(\wedge_{a \in E_k} a) \wedge (\wedge_{a \in E_l} a) \rightarrow (\wedge_{a \in E_i} a) \tag{A.1}$$

  The implication (A.1) requires that $E_k \cup E_l \subseteq E_i$. According to (4.1), $E_k \cup E_l$ is a member of $\mathcal{L}(n(d_1 \wedge d_2))$. Hence, for each E_i such that $(\wedge_{a \in E_i} a) \vdash d_1 \wedge d_2$, there exists a subset of E_i in the label $\mathcal{L}(n(d_1 \wedge d_2))$.

- $\mathcal{L}(n(d_1 \vee d_2))$ is complete, or $\forall E_i, (\wedge_{a \in E_i} a) \vdash d_1 \vee d_2 \rightarrow (\exists E_j \in \mathcal{L}(n(d_1 \vee d_2)), E_j \subseteq E_i)$:

  Assume that E_i is a set of nodes such that $(\wedge_{a \in E_i} a) \vdash d_1 \vee d_2$. Because the labels $\mathcal{L}(n(d_1))$ and $\mathcal{L}(n(d_2))$ are complete, there must exist a $E_k \in \mathcal{L}(n(d_1))$ and a $E_l \in \mathcal{L}(n(d_2))$ such that

  $$(\wedge_{a \in E_k} a) \vee (\wedge_{a \in E_l} a) \rightarrow (\wedge_{a \in E_i} a) \tag{A.2}$$

  The implication (A.2) requires that $(E_k \subseteq E_i) \vee (E_l \subseteq E_i)$. According to (4.2), E_k and E_l are a member of $\mathcal{L}(n(d_1 \vee d_2))$. Hence, for each E_i such that $(\wedge_{a \in E_i} a) \vdash d_1 \vee d_2$, there exists a subset of E_i in the label $\mathcal{L}(n(d_1 \vee d_2))$.

- $\mathcal{L}(n(\neg d_1))$ is complete, or $\forall E_i, (\wedge_{a\in E_i} a) \vdash \neg d_1 \rightarrow (\exists E_j \in \mathcal{L}(n(\neg d_1)), E_j \subseteq E_i)$:

  Assume that E_i is a set of nodes such that $(\wedge_{a\in E_i} a) \vdash \neg d_1$. Because the label $\mathcal{L}(n(d_1))$ is complete and sound, one of the assumptions in each of the labels must false in order to deduce that d_1 is indeed false. Hence, there must exist a $a_{1k_1} \in E_{11}, \ldots, a_{p_1k_{p_1}} \in E_{1p_1}$ such that:

  $$(a_{1k_1} \wedge \ldots \wedge a_{p_1k_{p_1}} \rightarrow (\wedge_{a\in E_i} a) \tag{A.3}$$

  The implication (A.3) requires that $\{a_{1k_1}, \ldots, a_{p_1k_{p_1}}\} \subseteq E_i$. According to (4.3), $\{a_{1k_1}, \ldots, a_{p_1k_{p_1}}\}$ is a member of $\mathcal{L}(n(\neg d_1))$. Hence, for each E_i such that $(\wedge_{a\in E_i} a) \vdash \neg d_1$, there exists a subset of E_i in the label $\mathcal{L}(n(\neg d_1))$.

■

Theorem 4.3. If the labels $\mathcal{L}(n(d_1))$ and $\mathcal{L}(n(d_2))$ are complete and sound, the labels $\mathcal{L}(n(d_1 \wedge d_2))$, $\mathcal{L}(n(d_1 \vee d_2))$ and $\mathcal{L}(n(\neg d_1))$, as respectively computed in (4.1), (4.2) and (4.3) are sound.
Proof: The proof of this theorem is broken down in three parts that respectively demonstrate that $\mathcal{L}(n(d_1 \wedge d_2))$, $\mathcal{L}(n(d_1 \vee d_2))$ and $\mathcal{L}(n(\neg d_1))$ are sound.

- $\mathcal{L}(n(d_1 \wedge d_2))$ is sound, or $\forall E \in \mathcal{L}(n(d_1 \wedge d_2)), (\wedge_{a\in E} a) \vdash (d_1 \wedge d_2)$:

  Because $\mathcal{L}(n(d_1))$ and $\mathcal{L}(n(d_2))$ are sound, it is known that for each pair of environments (E_1, E_2) such that $E_1 \in \mathcal{L}(n(d_1))$ and $E_2 \in \mathcal{L}(n(d_2))$

  $$(\wedge_{a\in E_1} a) \vdash d_1 \text{ and} \tag{A.4}$$
  $$(\wedge_{a\in E_2} a) \vdash d_2 \tag{A.5}$$

  From (A.4) and (A.5), it follows that

  $$(\wedge_{a\in E_1} a) \wedge (\wedge_{a\in E_2} a) \vdash (d_1 \wedge d_2) \tag{A.6}$$

  According to (4.1), for each environments $E \in \mathcal{L}(n(d_1 \wedge d_2))$, there exists an environment $E_1 \in \mathcal{L}(n(d_1))$ and an environment $E_2 \in \mathcal{L}(n(d_2))$ such that $E = E_1 \cup E_2$. Hence, following (A.6) for each $E \in \mathcal{L}(n(d_1 \wedge d_2))$ it follows that:

  $$(\wedge_{a\in E} a) \vdash (d_1 \wedge d_2)$$

  The label $\mathcal{L}(n(d_1 \wedge d_2))$ is therefore sound.

- $\mathcal{L}(n(d_1 \vee d_2))$ is sound, or $\forall E \in \mathcal{L}(n(d_1 \vee d_2)), (\wedge_{a\in E} a) \vdash (d_1 \vee d_2)$:

Because $\mathcal{L}(n(d_1))$ and $\mathcal{L}(n(d_2))$ are sound, it is known that for each pair of environments (E_1, E_2) such that $E_1 \in \mathcal{L}(n(d_1))$ and $E_2 \in \mathcal{L}(n(d_2))$

$$(\wedge_{a \in E_1} a) \vdash d_1 \text{ and} \tag{A.7}$$
$$(\wedge_{a \in E_2} a) \vdash d_2 \tag{A.8}$$

From (A.7) and (A.8), it follows that

$$(\wedge_{a \in E_1} a) \vee (\wedge_{a \in E_2} a) \vdash (d_1 \vee d_2) \tag{A.9}$$

According to (4.2), for each environments $E \in \mathcal{L}(n(d_1 \vee d_2))$, there exists an environment $E_1 \in \mathcal{L}(n(d_1))$ such that $E = E_1$ or an environment $E_2 \in \mathcal{L}(n(d_2))$ such that $E = E_2$. Hence, following (A.9) for each $E \in \mathcal{L}(n(d_1 \vee d_2))$ it follows that:

$$(\wedge_{a \in E} a) \vdash (d_1 \vee d_2)$$

The label $\mathcal{L}(n(d_1 \vee d_2))$ is therefore sound.

- $\mathcal{L}(n(\neg d_1))$ is sound, or $\forall E \in \mathcal{L}(n(\neg d_1)), (\wedge_{a \in E} a) \vdash (\neg d_1)$:

Because $\mathcal{L}(n(d_1))$ is complete and sound, it is known that d_1 can be disproven if at least one of the assumptions of each environment in $\mathcal{L}(n(d_1))$ is false. Because each environment $E \in \mathcal{L}(n(\neg d_1))$ is constructed by taking an assumption from each environment in $\mathcal{L}(n(d_1))$ and negating it. Hence, from each environment $E \in \mathcal{L}(n(\neg d_1))$, it follows that $\neg d_1$ is true and the label $\mathcal{L}(n(\neg d_1))$ is therefore sound.

■

Theorem 4.4. Given that the constraint propagator of GDE has generated a set of nodes in an ALTMS that corresponding to hypothesised values of connections $x : i$ and these nodes have been justified by the sets of components that must function correctly for the hypothesis to be true, then:

1. the ALTMS can produce the minimal set of fault candidates in the same way as the GDE by computing the label of a node hypotheses, which has been justified such that:
$$\neg \wedge_{n, n=n(x:j), j \neq i} n \rightarrow \text{hypotheses}(x : i)$$
2. the ALTMS can produce the minimal set of fault candidates that remain if a measurement of component x yields i, in the same way as the GDE, by computing the label of a node hypotheses$(x : i)$, which has been justified such that:
$$n(x : i) \wedge \neg(n_\perp) \rightarrow \text{hypotheses}(x : i)$$

Proof:

1. **the ALTMS can produce the minimal set of fault candidates in the same way as the GDE by computing the label of a node hypotheses, which has been justified such that:**

$$\neg \wedge_{n,n=n(x:j),j\neq i} n \rightarrow \textbf{hypotheses}(x:i)$$

Let $\mathcal{L}(n_\perp)$ be $\{E_1,\ldots,E_q\}$. Because the node hypotheses is justified such that $\neg(n_\perp) \rightarrow$ hypotheses and because of 4.3, the label of the hypotheses node is computed as as:

$$\mathcal{L}(\text{hypotheses}) = \mathcal{L}(\neg n_\perp) = \text{consistent}(\text{minimise}(\{\{\neg a_1,\ldots,\neg a_q\} \mid a_1 \in E_1,\ldots,a_q \in E_q\})) \tag{A.10}$$

where

$$\text{minimise}(L) = \{E \in L \mid \nexists E' \in L, E' \subset E\} \text{consistent}(L) \quad = \{E \in L \mid \nexists E' \in \mathcal{L}(n_\perp), E' \subset E\}$$

Because all assumptions in a GDE are of the form $\text{valid}(c_1)$, all justifications are of the form $\text{valid}(c_1) \wedge \ldots \wedge \text{valid}(c_n) \rightarrow x:i$ and all inconsistencies are of the form $x:i \wedge x:j \rightarrow n_\perp$, it follows that $\mathcal{L}(n_\perp$ consists entirely of environments of assumptions $\text{valid}(c_k)$. That is, none of the assumptions in a nogood environment are themselves negations of assumptions. Therefore, all assumptions in $\mathcal{L}(\text{hypotheses})$ are negations of assumptions and:

$$\begin{aligned}\text{consistent}(\text{minimise}(\{\{\neg a_1,\ldots,\neg a_q\} \mid a_1 \in E_1,\ldots,a_q \in E_q\}) = \\ \text{minimise}(\{\{\neg a_1,\ldots,\neg a_q\} \mid a_1 \in E_1,\ldots,a_q \in E_q\}\end{aligned}$$

In [37], the conflict set is defined such that:

$$\text{GDE-conflict-set} = (\text{minimise}(\{\{a_1,\ldots,a_q\} \mid a_1 \in E_1,\ldots,a_q \in E_q\}))$$

where the negation of the assumption is implicitly assumed. That is a hypothesis $H \in$ GDE-conflict-set denotes:

$$\wedge_{a\in H} \neg a$$

Therefore, GDE produces the same conflict set as the label of the node hypotheses is justified such that $\neg(n_\perp) \rightarrow$ hypotheses

2. **The ALTMS can produce the minimal set of fault candidates in the same way as the GDE by computing the label of a node hypotheses, which has been justified such that:**

$$\neg \wedge_{n,n=n(x:j),j\neq i} n \rightarrow \textbf{hypotheses}(x:i)$$

This part can be proven in the same way as part 1.

■

Theorem 4.7. If a model M can be derived from a model space Δ, M can also be derived from Δ' representing Δ extended by the application of a disaggregation fragment df.

Proof: Let A be the smallest set of assumptions such that $A, \Delta \vdash M$, A_{df} denote the instantiation of the assumptions of the disaggregation fragment and $\neg A_{df}$ denote the set of assumptions corresponding to the negation of the assumptions in A_{df}. The only change ApplyDF makes to the original model space consists of extending the justifications of certain nodes with $\neg A_{df}$. As the assumption specifications in A_{df} do not refer to assumption instances that are referred to by the model fragments in the knowledge base, and hence in Δ (definition 3.8), $A \cup \neg A_{df} \nvdash \bot$. Therefore, a set of assumptions $A' \subseteq A \cup A_{df}$ exists such that $A', \Delta' \vdash M$. ■

Theorem 4.8. Given that (i) Δ is a model space, (ii) Δ' is the model space resulting from extending Δ by the application of a disaggregation fragment df to a set of model fragment nodes Mf, (iii) A is a set of assumptions such that $A, \Delta \nvdash \bot$, and (iv) M_A is a model such that $A, \Delta \vdash M_A$, then the model M_D derived by $A \cup A_{df}, \Delta' \vdash M_D$ is a disaggregate model of M_A

Proof: For a given set of assumptions X, let $P(X)$ denote the set of participants $\{p \mid X, \Delta' \vdash p\}$, and $R(X)$ denote the set of relations $\{r \mid X, \Delta' \vdash r\}$. Thus, $M_D = \langle P(A \cup A_{df}), R(A \cup A_{df})\rangle$. It follows from theorem 4.7 that each aggregate model M_A that follows from the set of assumptions A in model space Δ also follows from $A \cup \neg A_{df}$, and hence $M_A = \langle P(A \cup \neg A_{df}), R(A \cup \neg A_{df})\rangle$.

As $P(A) \subseteq P(A \cup B)$ and $R(A) \subseteq R(A \cup B)$, $P(A)$ (or $R(A)$) is a set of participants (or relations) that M_A and M_D have in common. The nodes in $P(A \cup A_{df}) - P(A)$ (or $R(A \cup A_{df}) - R(A)$) are derived from the nodes that depend on the model fragment instances to which the disaggregation fragment is applied. They can be of two types:

- Some nodes represent participants or relations that are disaggregated according to the disaggregation fragment. For each of the participants $p \in P_{\text{dom}(\delta_P)}$ (with $P_{\text{dom}(\delta_P)} = P(A \cup A_{df}) - P(A) \cap \text{dom}(\delta_P)$), the set of participants $P_{\text{range}(\delta_P)} = \{\delta_P(\ldots, n_i, \ldots, p) \mid n_i \in N_i\}$ is created. The other participants (denoted P_a) are copied into new nodes whose label is the combination of the original label and A. Similarly, for each of the relations $r \in R_{\text{dom}(\delta_R)}$ (with $R_{\text{dom}(\delta_R)} = R(A \cup A_{df}) - R(A) \cap \text{dom}(\delta_R)$), a new set of relations $R_{\text{range}(\delta_R)} = \{\delta_R(\ldots, n_i, \ldots, r) \mid n_i \in N_i\}$ is created. Following definition 3.8, based on the remaining relations in $r(p^a_q, \ldots, p^a_r) \in R_a$ (with $R_a = R(A \cup \neg A_{df}) - \text{dom}(\delta_R)$), the set $R_d = \{r(\delta'_P(\ldots, n_i, \ldots, p^a_q), \ldots, \delta'_P(\ldots, n_i, \ldots, p^a_r)) \mid n_i \in N_i\}$ is created.

- The remaining nodes are the participants and relations that are newly instantiated from the target-participants and postconditions respectively. They are denoted by the sets P' and R'.

From this, it follows that $M_A = \langle P(A) \cup P_a \cup P_{\text{dom}(\delta_P)}, R(A) \cup R_a \cup R_{\text{dom}(\delta_R)}\rangle$ and that $M_D = \langle P(A) \cup P_a \cup P_{\text{range}(\delta_P)} \cup P', R(A) \cup R_{\text{range}(\delta_R)} \cup R_d \cup R'\rangle$. For the substitu-

tions $P^c = P(A) \cup P^a$, $P^a = P_{\text{dom}(\delta_P)}$, $P^d = R_{\text{dom}(\delta_R)}$, $R = R(A)$, $R^a = R_a \cup R_{\text{dom}(\delta_R)}$ and $R^d =_{\text{range}(\delta_R)} \cup R_d$, definition 3.3 applies. ■

Theorem 4.10. The combined disaggregation of the model space is a commutative operation. In other words,

$$D_\theta(\Delta, d_2 \circ d_1, M_1 \cup M_2 \cup M_{12}) = D_\theta(\Delta, d_1 \circ d_2, M_1 \cup M_2 \cup M_{12})$$

Proof: Following definition 4.9, this theorem needs to prove that, given two disaggregation fragments d_1 and d_2, a model space Δ and a partition $\{M_{12}, M_1, M_2, M\}$ of those nodes in Δ which represent model fragment instances, where $M_{12} \cup M_1$ and $M_{12} \cup M_2$ are minimal sets for model fragments to which d_1 and d_2 can be respectively applied,

$$\begin{aligned}
&d_\theta(d_\theta(\Delta, d_1, M_1 \cup M_{12}), d_2, d_M(\Delta, d_1, M_{12}) \cup M_2) \cup \\
&d_\theta(d_\theta(\Delta, d_1, M_1 \cup M_{12}), d_1, P^t(d_2, M_2 \cup M_{12})) \cup \\
&d_\theta(d_\theta(\Delta, d_1, M_1 \cup M_{12}), d_1, \Phi^t(d_2, M_2 \cup M_{12})) = \\
&d_\theta(d_\theta(\Delta, d_2, M_2 \cup M_{12}), d_1, d_M(\Delta, d_2, M_{12}) \cup M_1) \cup \\
&d_\theta(d_\theta(\Delta, d_2, M_2 \cup M_{12}), d_2, P^t(d_1, M_1 \cup M_{12})) \cup \\
&d_\theta(d_\theta(\Delta, d_2, M_2 \cup M_{12}), d_2, \Phi^t(d_1, M_1 \cup M_{12}))
\end{aligned}$$

Based on these notations, the following properties can be established:

Property 1. For each pair of sets of model fragment instances M_a and M_b, such that $M_a \cap M_b = \emptyset$,

$$d_\theta(\Delta, d, M_a \cup M_b) = d_\theta(\Delta, d, M_a) \cup d_\theta(\Delta, d, M_b)$$

When compared to Δ, $d_\theta(\Delta, d, M)$ consists of a new sub-hypergraph for each model fragment instance $m \in M$, containing the disaggregations (according to d) of the consequents of m. It is equivalent whether to apply this operation to the set M as a whole, or to apply it to the sets of a partition of M is equivalent.

Property 2. For each pair of disaggregation fragments d_a and d_b and each pair of sets of model fragments M_a and M_b, such that $M_a \cap M_b = \emptyset$, to which d_a and d_b are applicable,

$$\begin{aligned}
&d_\theta(d_\theta(\Delta, d_b, M_b), d_a, d_M(\Delta, d_b, M_a)) \\
=&d_\theta(d_\theta(\Delta, d_a, M_a), d_b, d_M(\Delta, d_a, M_b)) \\
=&d_\theta(\Delta, d_a, M_a) \cup d_\theta(\Delta, d_b, M_b)
\end{aligned}$$

As d_a (or d_b) is applied to a set of model fragments M_a (or M_b) that does not intersect with the model fragments to which the other disaggregation fragment d_b (or d_a) is applied, $d_M(\Delta, d_a, M_b) = M_b$ (or $d_M(\Delta, d_b, M_a) = M_a$). Thus, the order in which the disaggregation fragments are applied is irrelevant.

Property 3. For each pair of disaggregation fragments d_a and d_b that are applied to the same set of model fragments M in a model space Δ,

$$\begin{aligned} & d_\theta(d_\theta(\Delta, d_b, M), d_a, d_M(\Delta, d_b, M)) \cup d_\theta(\Delta, d_a, M) \\ =& d_\theta(d_\theta(\Delta, d_a, M), d_b, d_M(\Delta, d_a, M)) \cup d_\theta(\Delta, d_b, M) \end{aligned}$$

The application of the first disaggregation fragment, say $d_\theta(\Delta, d_a, M)$, creates a new set of sub-hypergraphs with roots $d_M(\Delta, d_a, M)$. The second disaggregation fragment will copy these into a new set of sub-hypergraphs with roots $d_M(d_\theta(\Delta, d_a, M), d_b, d_M(\Delta, d_a, M))$ in which the consequents are disaggregated according to d_a and d_b. The reverse order of application of disaggregation fragments also produces a new sub-hypergraphs containing the consequents of M disaggregated according to d_a and d_b, but through the different intermediate $d_\theta(\Delta, d_b, M)$. Therefore

$$\begin{aligned} & d_\theta(d_\theta(\Delta, d_b, M), d_a, d_M(\Delta, d_b, M)) - d_\theta(\Delta, d_b, M) \\ =& d_\theta(d_\theta(\Delta, d_a, M), d_b, d_M(\Delta, d_a, M)) - d_\theta(\Delta, d_a, M) \end{aligned}$$

Theorem A.1. Given two disaggregation fragments d_1 and d_2, a model space Δ and a partition $\{M_{12}, M_1, M_2, M\}$ of those nodes in Δ which represent model fragment instances, where $M_{12} \cup M_1$ and $M_{12} \cup M_2$ are minimal sets for model fragments to which d_1 and d_2 can be respectively applied,

$$\begin{aligned} & d_\theta(d_\theta(\Delta, d_1, M_{12} \cup M_1), d_2, d_M(\Delta, d_1, M_{12} \cup M_2)) \\ =& d_\theta(d_\theta(\Delta, d_2, M_{12} \cup M_2), d_1, d_M(\Delta, d_2, M_{12} \cup M_1)) \end{aligned}$$

Based on the previous properties:

$$\begin{aligned} & d_\theta(d_\theta(\Delta, d_1, M_{12} \cup M_1), d_2, d_M(\Delta, d_1, M_{12} \cup M_2)) \\ =\ & \left.\begin{array}{l} d_\theta(d_\theta(\Delta, d_1, M_{12}), d_2, d_M(\Delta, d_1, M_{12})) \cup \\ d_\theta(d_\theta(\Delta, d_1, M_1), d_2, d_M(\Delta, d_1, M_{12})) \cup \\ d_\theta(d_\theta(\Delta, d_1, M_{12}), d_2, M_2) \cup \\ d_\theta(d_\theta(\Delta, d_1, M_1), d_2, M_2) \end{array}\right\} \quad \text{(due to property 1)} \\ =\ & \left.\begin{array}{l} d_\theta(d_\theta(\Delta, d_1, M_{12}), d_2, d_M(\Delta, d_1, M_{12})) \cup \\ d_\theta(\Delta, d_1, M_1) \cup d_\theta(\Delta, d_1, M_{12}) \cup \\ d_\theta(\Delta, d_2, M_2) \cup d_\theta(\Delta, d_2, M_{12}) \end{array}\right\} \quad \text{(due to property 2)} \end{aligned} \tag{A.11}$$

Following the same line for reasoning, it can be proven that:

$$\begin{aligned} & d_\theta(d_\theta(\Delta, d_2, M_{12} \cup M_2), d_1, d_M(\Delta, d_2, M_{12} \cup M_1)) \\ =\ & d_\theta(d_\theta(\Delta, d_2, M_{12}), d_1, d_M(\Delta, d_2, M_{12})) \cup \\ & d_\theta(\Delta, d_1, M_1) \cup d_\theta(\Delta, d_1, M_{12}) \cup \\ & d_\theta(\Delta, d_2, M_2) \cup d_\theta(\Delta, d_2, M_{12}) \end{aligned}$$

From property 3

$$\begin{aligned} & d_\theta(d_\theta(\Delta,d_1,M_{12}),d_2,d_M(\Delta,d_1,M_{12}))\cup d_\theta(\Delta,d_2,M_{12}) \\ =& d_\theta(d_\theta(\Delta,d_2,M_{12}),d_1,d_M(\Delta,d_2,M_{12}))\cup d_\theta(\Delta,d_1,M_{12}) \end{aligned}$$

it follows that:

$$\begin{aligned} & d_\theta(d_\theta(\Delta,d_1,M_{12}\cup M_1),d_2,d_M(\Delta,d_1,M_{12}\cup M_2)) \\ =& d_\theta(d_\theta(\Delta,d_2,M_{12}\cup M_2),d_1,d_M(\Delta,d_2,M_{12}\cup M_1)) \end{aligned}$$

which proves this sub-theorem. ■

Property 4. For each pair of model spaces (or parts of model spaces) Δ_a and Δ_b, such that $\Delta_a \cap \Delta_b = \emptyset$,

$$d_\theta(\Delta_a \cup \Delta_b, d, M) = d_\theta(\Delta_a, d, M) \cup d_\theta(\Delta_b, d, M) \tag{A.12}$$

Now, the required proof can proceed. From definition 4.6 and property 4:

$$\begin{aligned} & D_\theta(D_\theta(\Delta,d_1,M_{12}\cup M_1),d_2,d_M(\Delta,d_1,M_{12}\cup M_2))\cup \\ & d_\theta(P^t(\Delta,d_2,M_{12}\cup M_1),d_1,M_{12}\cup M_1\cup M_2)\cup \\ & d_\theta(\Phi^t(\Delta,d_2,M_{12}\cup M_1),d_1,M_{12}\cup M_1\cup M_2) \qquad (A.13) \\ =& D_\theta(d_\theta(\Delta,d_1,M_{12}\cup M_1)\cup \\ & P^t(d_\theta,d_1,M_{12}\cup M_1)\cup\Phi^t(d_\theta,d_1,M_{12}\cup M_1),d_2,d_M(\Delta,d_1,M_{12}\cup M_2))\cup \\ & d_\theta(P^t(\Delta,d_2,M_{12}\cup M_1),d_1,M_{12}\cup M_1\cup M_2)\cup \\ & d_\theta(\Phi^t(\Delta,d_2,M_{12}\cup M_1),d_1,M_{12}\cup M_1\cup M_2) \\ =& D_\theta(d_\theta(\Delta,d_1,M_{12}\cup M_1),d_2,d_M(\Delta,d_1,M_{12}\cup M_2))\cup \\ & D_\theta(P^t(d_\theta,d_1,M_{12}\cup M_1),d_2,d_M(\Delta,d_1,M_{12}\cup M_2))\cup \\ & D_\theta(\Phi^t(d_\theta,d_1,M_{12}\cup M_1),d_2,d_M(\Delta,d_1,M_{12}\cup M_2))\cup \\ & d_\theta(P^t(\Delta,d_2,M_{12}\cup M_1),d_1,M_{12}\cup M_1\cup M_2)\cup \\ & d_\theta(\Phi^t(\Delta,d_2,M_{12}\cup M_1),d_1,M_{12}\cup M_1\cup M_2) \qquad (A.14) \end{aligned}$$

For the same reasons, it can be proven that:

$$\begin{aligned} & D_\theta(D_\theta(\Delta,d_2,M_{12}\cup M_2),d_1,d_M(\Delta,d_2,M_{12}\cup M_1))\cup \\ & d_\theta(P^t(\Delta,d_1,M_{12}\cup M_2),d_2,M_{12}\cup M_2\cup M_1)\cup \\ & d_\theta(\Phi^t(\Delta,d_1,M_{12}\cup M_2),d_2,M_{12}\cup M_2\cup M_1) \qquad (A.15) \\ =& D_\theta(d_\theta(\Delta,d_2,M_{12}\cup M_2),d_2,d_M(\Delta,d_2,M_{12}\cup M_1))\cup \\ & D_\theta(P^t(d_\theta,d_2,M_{12}\cup M_2),d_1,d_M(\Delta,d_2,M_{12}\cup M_1))\cup \\ & D_\theta(\Phi^t(d_\theta,d_2,M_{12}\cup M_2),d_1,d_M(\Delta,d_2,M_{12}\cup M_1))\cup \\ & d_\theta(P^t(\Delta,d_1,M_{12}\cup M_2),d_2,M_{12}\cup M_2\cup M_1)\cup \\ & d_\theta(\Phi^t(\Delta,d_1,M_{12}\cup M_2),d_2,M_{12}\cup M_2\cup M_1) \qquad (A.16) \end{aligned}$$

Because of theorem A.1, (A.14) is equivalent to (A.16). It follows that (A.13) is equivalent to (A.15). This proves that commutativity holds for the combined applications of disaggregation fragments. ■

Theorem 5.5.

$$P_1 \prec P_2 \rightarrow \forall p_a \in \mathbb{B}, (P_1 \prec_{p_a} P_2) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_1 \prec_{p_b} P_2)$$

Proof: If $\forall p_a \in \mathbb{B}, (P_1 \prec_{p_a} P_2) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_1 \prec_{p_b} P_2)$ does not hold, it is possible that:

- $P_1 = P_2$, if $\forall q \in \mathbb{B}, f_{P_1}(q) = f_{P_2}(q)$, or
- $P_2 \prec P_1$, if $\forall p_a \in \mathbb{B}, (P_2 \prec_{p_a} P_1) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_2 \prec_{p_b} P_1)$, or
- $P_1 ? P_2$ in all other cases due to (5.8).

Therefore, $\forall p_a \in \mathbb{B}, (P_1 \prec_{p_a} P_2) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_1 \prec_{p_b} P_2)$ is the only case wherein $P_1 \prec P_2$. ■

Theorem 5.7. $\text{path}+(O,x,y) = \text{path}-(O,y,x)$.
Proof: According to definition 5.6, a path is defined by any set of quantities $\{q_1, q_2, \ldots, q_{n-1}, q_n\}$ such that $(x,q_1) \in O$, $(q_1,q_2) \in O, \ldots$, $(q_{n-1},q_n) \in O$ and $(q_n,y) \in O$, and such a set $\{q_1, q_2, \ldots, q_{n-1}, q_n\}$ defines both a positive path $\text{path}+(O,x,y)$ and a negative path $\text{path}-(O,y,x)$. ■

Theorem 5.15. The combination of labels of a BPQ is unique to that BPQ.
Proof: For each BPQ q and for at least one of the orderings, there exists a cross-over quantity q_c such that there is a path exists between q and q_c, but no path between q and q_c contains a cross-over quantity itself. A strand $\text{strand}(O,q_c,q')$ exists such that $q \in \text{strand}(O,q_c,q')$. Only a single path exists between q_c and q that is empty or contains BPQs that are members of $\text{strand}(O,q_c,q')$. Hence, there is only one path to follow in computing the distance between q and q_c following $\text{strand}(O,q_c,q')$ and therefore, only one BPQ q is at a distance d from q_c on $\text{strand}(O,q_c,q')$. Because $\langle \text{strand}(O,q_c,q'),d,1\rangle \in L_{(\mathbb{B},O)}(q)$ and $\langle \text{strand}(O,q_c,q'),d,1\rangle \notin L_{(\mathbb{B},O)}(q'')$, where $q'' \neq q$ the combination of labels of q is unique to q. ■

Theorem 5.20. $\forall q_1,q_2,q_3 \in Q, (q_1 \succcurlyeq_O q_2) \wedge (q_2 \succcurlyeq_O q_3) \rightarrow (q_1 \succcurlyeq_O q_3)$
Proof:

According to definition 5.19, $(q_2 \succcurlyeq_O q_3)$ implies that:

$$\begin{aligned}&\forall \langle s_3,d_3,1\rangle \in L_{(Q,O)}(q_3), [\exists \langle s_2,d_2,1\rangle \in L_{(Q,O)}(q_2),\\ &((s_2,d_2) \succ_O (s_3,d_3)) \vee ((s_2 = s_3 \neq \text{strand} \circ (O,c,q)) \wedge (d_2 = d_3))]\end{aligned} \tag{A.17}$$

According to definition 5.19, $(q_1 \succcurlyeq_O q_2)$ implies that:

$$\begin{aligned}&\forall \langle s_2,d_2,1\rangle \in L_{(Q,O)}(q_2), [\exists \langle s_1,d_1,1\rangle \in L_{(Q,O)}(q_1),\\ &((s_1,d_1) \succ_O (s_2,d_2)) \vee ((s_1 = s_2 \neq \text{strand} \circ (O,c,q)) \wedge (d_1 = d_2))]\end{aligned} \tag{A.18}$$

From (A.17) and (A.18), it follows that:

$$\begin{aligned}&\forall\langle s_3,d_3,1\rangle \in L_{(Q,O)}(q_3),\\&[\exists\langle s_2,d_2,1\rangle \in L_{(Q,O)}(q_2),\\&[\exists\langle s_1,d_1,1\rangle \in L_{(Q,O)}(q_1),\\&[((s_2,d_2)\succ_O(s_3,d_3))\vee((s_2=s_3\neq \text{strand}\circ(O,c,q))\wedge(d_2=d_3))]\wedge\\&[((s_1,d_1)\succ_O(s_2,d_2))\vee((s_1=s_2\neq \text{strand}\circ(O,c,q))\wedge(d_1=d_2))]]]\end{aligned} \tag{A.19}$$

Because of theorem 5.17, (A.19) implies that

$$\begin{aligned}&\forall\langle s_3,d_3,1\rangle \in L_{(Q,O)}(q_3),[\exists\langle s_1,d_1,1\rangle \in L_{(Q,O)}(q_1),\\&((s_1,d_1)\succ_O(s_3,d_3))\vee((s_1=s_3\neq \text{strand}\circ(O,c,q))\wedge(d_1=d_3))]\end{aligned} \tag{A.20}$$

In turn, (A.20) implies that $q_1 \succcurlyeq_O q_3$. ■

Theorem 5.21. $\forall q_1,q_2 \in Q, \text{path}+(O,q_2,q_1) \leftrightarrow q_1 \succcurlyeq_O q_2$
Proof:

- $\text{path}+(O,q_2,q_1) \rightarrow q_1 \succcurlyeq_O q_2$:
  Two cases must be distinguished:

Case 1: $q_1 = q_2$: in this case $L_{(Q,O)}(q_1) = L_{(Q,O)}(q_2)$ and there exists a positive path $\text{path}+(O,q_1,q_2)$ according to definition 5.6.

Case 2: $\text{path}+(O,q_2,q_x)$, $q_x \succcurlyeq_O q_2$ and $(q_x,q_1) \in O$. In this case, all strand, distance, number triplets $\langle s_x,d_x,1\rangle$ in the label of q_x should be considered and compared to a strand, distance, number triplet $\langle s_1,d_1,1\rangle$ in the label of q_1. For the values $\langle s_x,d_x,1\rangle \in L_{(Q,O)(q_x)}$, it can be shown that definition 5.19 applies, and hence, that $q_1 \succcurlyeq_O q_x$. Because of theorem 5.20, this implies that $q_1 \succcurlyeq_O q_2$.

In any label $L_{(Q,O)}(q_x)$, three different kinds of strand, label, number triplets $\langle s_x,d_x,1\rangle$ will be found:

1. $s_x = \text{strand}(O,c,q_s)$ and a $\text{path}+(O,c,q_x)$ exists, or
2. $s_x = \text{strand}(O,c,q_s)$ and a $\text{path}-(O,c,q_x)$ exists, or
3. $s_x = \text{strand}\circ(O,c,q_s)$.

Each of these case is considered below:

Case a: If $s_x = \text{strand}(O,c,q_s)$ and a $\text{path}+(O,c,q_x)$ exists, then a $\text{path}+(O,q_s,q_x)$ exists (due to definition 5.11. From $q_x \in \text{strand}(O,c,q_s)$, $\text{path}+(O,q_s,q_x)$, $\text{path}+(O,q_x,q_1)$ and from definitions 5.6 and 5.11, it follows that $q_1 \in \text{strand}(O,c,q_s)$. Also, $d_{\max}(c,q_1,O) > d_{\max}(c,q_x,O)$ because of definition 5.10.
Because $q_1 \in \text{strand}(O,c,q_s)$ and $d_{\max}(c,q_1,O) > d_{\max}(c,q_x,O)$ it follows from definition 5.13 that $\exists\langle s_1,d_1,1\rangle \in L_{(Q,O)}(q_1)$ such that $s_1 = \text{strand}(O,c,q_s)$ and $d_1 > d_x$. This implies that $(s_1,d_1) \succ_O (s_x,d_x)$, using definition 5.16.

Case b: If $s_x = \text{strand}(O,c,q_s)$ and a path $-(O,c,q_x)$ exists, then $d_x < 0$ and there is a path $+(O,c,q_1)$ or there is a path $-(O,c,q_1)$ or there is no path (positive or negative) between c and q_1. Each of these is considered separately:

- If there is a path $+(O,c,q_1)$, then there exists a strand $+(O,c,q_{s1})$ such that $q_1 \in s_1 = \text{strand}+(O,c,q_{s1})$ (definition 5.11), and there exists $d_1 > 0$ such that $\langle s_1, d_1, 1\rangle \in L_{(Q,O)}(q_1$ (definition 5.13). Because a path $-(O,c,q_x)$ exists, $s_x = \text{strand}(O,c,q_s) = \text{strand}-(O,c,q_s)$ (definitions 5.8[1] and 5.11). Because $s_1 \succ s_x$, it follows that $(s_1,d_1) \succ_O (s_x,d_x)$ (definition 5.16).
- If there is a path $-(O,c,q_1)$, then it can be proven along the same lines of reasoning of case a, but applied to negative paths instead of positive paths, that there exists a $\langle s_1,d_1,1\rangle \in L_{(Q,O)}(q_1)$ such that $(s_1,d_1) \succ_O (s_2,d_2)$.
- If there is no path between c and q_1, then there exists a triplet $\langle \text{strand} \circ (O,c,q_s), d_x, 1\rangle \in L_{(Q,O)}(q_1)$ because of definitions 5.12 and 5.13. It follows from definition 5.16 that $(\text{strand} \circ (O,c,q_s), d_x) \succ (\text{strand}(O,c,q_s), d_x)$

Case c: If $s_x = \text{strand} \circ (O,c,q_s)$, then there are two possibilities:

- If $\langle \text{strand}(O,c,q_s), d_1, 1\rangle \in L_{(Q,O)}(q_1)$, then $d_1 < 0$ because $(q_x,q_1) \in O$. It follows from definition 5.16 that $(\text{strand}(O,c,q_s),d_1) \succ_O (\text{strand} \circ (O,c,q_s),d_1)$ in this case.
- In all other cases, $\langle \text{strand} \circ (O,c,q_s), d_1, 1\rangle \in L_{(Q,O)}(q_1)$, and definition 5.16 is not affected by this tuple. However, because the algorithm for computing labels considers all strands of all cross-over quantities, because $(q_x,q_1) \in O$ and following definitions 5.9, 5.10 and 5.11, it is easy to show that there must at least be one strand and cross-over quantity that matches one of the previous cases.

It has been shown that:

$$q \succcurlyeq_O q \qquad \text{(case 1)}$$

$$[\text{path}+(O,q_2,q_x)] \wedge [q_x \succcurlyeq_O q_2] \wedge [(q_x,q_1) \in O] \quad \rightarrow q_1 \succcurlyeq_O q_2 \qquad \text{(case 2)}$$

Because $path+(O,q_2,q_2)$ and $q_2 \succcurlyeq_O q_2$ (case 1), $(q_2,q_1) \in O \rightarrow q_1 \succcurlyeq_O q_2$ (case 2). Similarly, it can be proven, via case 2, that $q_1 \succcurlyeq_O q_2$ if $(q_2,q_x),(q_x,q_1) \in O$, etc. Because any positive path is based on a sequence of ordered pairs of quantities, it has been shown that $\text{path}+(O,q_2,q_1) \rightarrow q_1 \succcurlyeq_O q_2$

- $\text{path}+(O,q_2,q_1) \leftarrow q_1 \succcurlyeq_O q_2$:
  If $q_1 \succcurlyeq_O q_2$, for each strand distance pair (i.e. the strand distance (s,d) combination in each triplet $\langle s,d,1\rangle$) in the label of q_2, a strand distance pair exists in

[1] Note that the order of preference magnitude scale is assumed to be consistent.

the label of q_1 that is ranked higher than or equal to that of q_2 (definition 5.19). According to definition 5.16, this implies path$+(O,q_2,q_1)$.

■

Theorem 5.22. Given two BPQs $q_1,q_2 \in \mathbb{B}$ such that $q_1 \sim q_2$, then $L_{(\mathbb{B},O_{\ll})}(q_1) = L_{(\mathbb{B},O_{\ll})}(q_2)$.
Proof: Assume a strand distance pair (s_1,d_1) exists such that $q_1 \in (s_1,d_1)$ and $q_2 \notin (s_1,d_1)$. In this case, q_2 must be a member of a strand distance pair that is ranked higher than (s_1,d_1), or q_2 must be a member of a strand distance pair that is ranked lower than (s_1,d_1), or q_2 is not a member of a strand distance pair that is ranked higher or lower than (s_1,d_1). For the first two cases, it is straightforward to show that q_1 and q_2 can not be ordered equivalently with respect to all $q \in \mathbb{B}$. In the third and last case, q_1 and q_2 are incomparable and, therefore, obviously not of the same order of magnitude. ■

Theorem 5.25. The combination of labels of an OMP is unique to that OMP.
Proof: As shown in theorem 5.15, each combination of labels of a BPQ contains a tuple with a strand distance pair that is unique to that BPQ. Hence, following definition 5.24, for each BPQ q the label of an OMP P contains a tuple $\langle s,d,x\rangle$ with a strand distance pair (s,d) unique to of the constituent BPQs. Because no other BPQ exists whose label contains $\langle s,d,1\rangle$, x indicates the number of occurrences of q in the specification of P, or $x = f_P(q)$. Therefore, the labels of an OMP P unique identifies the BPQ that constitute P and hence P itself. ■

Theorem 5.26. Given an OMP P and an ordering of BPQs O, then

$$\forall\langle s,d,x\rangle \in L_{(\mathbb{B},O)}(P), x = \sum_{p,\langle s,d,1\rangle \in L_{(\mathbb{B},O)}(p)} f_P(p)$$

Proof: Assume that the OMP $P = p_1 \oplus \ldots \oplus p_n$. Let $g_{(O,s,d)}$ be a function:

$$g_{(O,s,d)} : \mathbb{B} \to \{0,1\} : p \mapsto g_{(O,s,d)}(p) = \begin{cases} 0 & \text{if } \nexists\langle s,d,1\rangle \in L_{(\mathbb{B},O)}(p) \\ 1 & \text{if } \exists\langle s,d,1\rangle \in L_{(\mathbb{B},O)}(p) \end{cases}$$

Now, it follows from definition 5.24 that

$$\forall\langle s,d,x\rangle \in L_{(\mathbb{B},O)}(P), x = \sum_{p=p_1,\oplus,p_n} g_{(O,s,d)}(p)$$

$$\Downarrow$$

$$\forall\langle s,d,x\rangle \in L_{(\mathbb{B},O)}(P), x = \sum_{p\in\mathbb{B}} f_P(p) \times g_{(O,s,d)}(p)$$

$$\Downarrow$$

$$\forall\langle s,d,x\rangle \in L_{(\mathbb{B},O)}(P), x = \sum_{p,\langle s,d,1\rangle \in L_{(\mathbb{B},O)}(p)} f_P(p)$$

■

Theorem 5.30.

$$\text{maxDS}(L_{(\mathbb{B},O_{\ll})}(P_1)) \prec\prec \text{maxDS}(L_{(\mathbb{B},O_{\ll})}(P_2)) \leftrightarrow L_{(\mathbb{B},O_{\ll})}(P_1) \prec\prec L_{(\mathbb{B},O_{\ll})}(P_2) \tag{A.21}$$

Proof: $L_{(\mathbb{B},O_{\ll})}(P_1) \prec\prec L_{(\mathbb{B},O_{\ll})}(P_2)$ is defined as the case where:

$$\forall\langle s_1,d_1,x_1\rangle \in L_{(\mathbb{B},O_{\ll})}(P_1), \exists\langle s_2,d_2,x_2\rangle \in L_{(\mathbb{B},O_{\ll})}(P_2), (s_1,d_1) \prec_{O_{\ll}} (s_2,d_2)$$

Due to definition 5.27:

$$\begin{aligned}\forall\langle s_i,d_i,x_i\rangle \in L_{(\mathbb{B},O_{\ll})}(P_i), \langle s_i,d_i,x_i\rangle \notin \text{maxDS}(L_{(\mathbb{B},O_{\ll})}(P_i) \rightarrow \\ \exists\langle s_j,d_j,x_j\rangle \in L_{(\mathbb{B},O_{\ll})}(P_i), (s_i,d_i) \prec_{O_{\ll}} (s_j,d_j)\end{aligned} \tag{A.22}$$

Clearly, the tuples $\langle s_i,d_i,x_i\rangle$ in expression (A.22) do not affect the truth of expression (A.22) and hence, they can be ignored. ■

Theorem 5.31. Given two OMPs P_1 and P_2 such that $L_{(\mathbb{B},O_{\ll})}(P_1) \prec\prec L_{(\mathbb{B},O_{\ll})}(P_2)$, then $P_1 \prec P_2$.
Proof: Following definition 5.26, $L_{(\mathbb{B},O_{\ll})}(P_1) \prec\prec L_{(\mathbb{B},O_{\ll})}(P_2)$ implies that

$$\forall\langle s_1,d_1,x_1\rangle \in L_{(\mathbb{B},O_{\ll})}(P_1), \exists\langle s_2,d_2,x_2\rangle \in L_{(\mathbb{B},O_{\ll})}(P_2), (s_2,d_2) \succ_{O_{\ll}} (s_1,d_1) \tag{A.23}$$

Following theorem 5.18, there is a positive path defined by $O_{\ll}$ from each BPQ that could be one to the constituent BPQs of P_1 to a constituent BPQ of P_2. Hence, expression (5.7) is true for each possible combination of BPQs that defines OMPs P_1 and P_2 with corresponding labels $L_{(\mathbb{B},O_{\ll})}(P_1)$ and $L_{(\mathbb{B},O_{\ll})}(P_2)$ such that $L_{(\mathbb{B},O_{\ll})}(P_1) \prec\prec L_{(\mathbb{B},O_{\ll})}(P_2)$. Hence, $P_1 \prec P_2$ is true by definition $P_1 \prec P_2$. ■

Theorem 5.32. Given two OMPs P_1 and P_2 such that $\forall p_1 \in P_1, f_{P_1}(p_1) > 0, \exists p_2 \in P_2, f_{P_2}(p_2) > 0, p_1 \ll p_2$, $L_{(\mathbb{B},O_{\ll})}(P_1)$ can each be partitioned into two sets L_{11} and L_{12} and $L_{(\mathbb{B},O_{\ll})}(P_2)$, can each be partitioned into two sets L_{21} and L_{22} such that $L_{11} \prec\prec L_{21}$, $L_{L_{12}\rightarrow(\mathbb{B},O_{<})}(P_1) = \emptyset$ and $L_{L_{22}\rightarrow(\mathbb{B},O_{<})}(P_2) = \emptyset$.
Proof: From $\forall p_1 \in P_1, f_{P_1}(p_1) > 0, \exists p_2 \in P_2, f_{P_2}(p_2) > 0, p_1 \ll p_2$, it follows that $\forall p_1 \in P_1, f_{P_1}(p_1) > 0, \exists p_2 \in P_2, \text{path}(O_{\ll}, p_1, p_2)$. For a given pair of BPQs p_1 and p_2, such that $\text{path}(O, p_1, p_2)$, it follows from theorem 5.21 that for each strand distance pair (s_1,d_1) in a tuple from the label of p_1, a strand distance pair (s_2,d_2) in a tuple of the label of p_2 exists such that $(s_1,d_1) \preccurlyeq_O (s_2,d_2)$. If $(s_1,d_1) \prec_O (s_2,d_2)$, the theorem is satisfied. If $(s_1,d_1) \prec_O (s_2,d_2)$, both strand distance pairs do not uniquely identify the BPQs and hence, by definition 5.28, their specific $O_<$ labels are empty. ■

Theorem 5.34. Given two OMPs $P_1 = p_{11} \oplus \ldots \oplus p_{1n_1}$ and $P_2 = p_{21} \oplus \ldots \oplus p_{2n_2}$ such that $p \sim p_{11} \sim \ldots \sim p_{1n_1} \sim p_{21} \sim \ldots \sim p_{2n_2}$, then

$$P_1 \preccurlyeq_p P_2 \leftrightarrow L_{(\mathbb{B},O_{<})}(P_1) \preccurlyeq L_{(\mathbb{B},O_{<})}(P_2)$$

Proof:

The proof presented herein consists of two parts. First, it will be shown that $P_1 \preccurlyeq_p P_2 \leftarrow L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2)$ and then, it will be shown that $P_1 \preccurlyeq_p P_2 \rightarrow L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2)$.

1. $\mathbf{P_1 \preccurlyeq_p P_2 \leftarrow L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2)}$:
Assume the hypothetical case where P_1 and P_2 are defined in such a way that $L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2)$ is true and that $P_1 \preccurlyeq_p P_2$ is false. By demonstrating that such a case is impossible, the first part of the theorem will be proven.

   Because of (5.6) and $\neg(P_1 \preccurlyeq_p P_2)$, there exists a $p_i \in \{p_{11},\ldots,p_{1n_1}\}$ such that:

$$(f_{P_1}(p_i) + \sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_1}(p_j)) > (f_{P_2}(p_i) + \sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_1}(p_j)) \tag{A.24}$$

   In (A.24), the first terms on both sides of the inequality can be ignored as they can always be incorporated in the respective second terms by considering a BPQ just below p_i in the $O_<$ ordering. Hence,

$$\sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_1}(p_j) > \sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_1}(p_j) \tag{A.25}$$

   From (A.25), it is possible to deduce the following:

$$\begin{aligned}
&\sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_1}(p_j) > \sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_1}(p_j) \\
&\qquad\qquad \Downarrow \quad \text{definition 5.6} \\
&\sum_{p_j \in \mathbb{B}, \text{path}+(O_<,p_i,p_j)} f_{P_1}(p_j) > \sum_{p_j \in \mathbb{B}, \text{path}+(O_<,p_i,p_j)} f_{P_1}(p_j) \\
&\qquad\qquad \Downarrow \quad \text{theorem 5.21} \\
&\sum_{p_j \in \mathbb{B}, p_j \succcurlyeq_{O_<} p_i} f_{P_1}(p_j) > \sum_{p_j \in \mathbb{B}, p_j \succcurlyeq_{O_<} p_i} f_{P_1}(p_j) \\
&\qquad\qquad \Downarrow \quad \text{definition 5.19} \\
&\sum_{\substack{p_j \in \mathbb{B}, \forall \langle s,d,1\rangle \in L_{(\mathbb{B},O)}(p_i), \\ [\exists \langle s_j,d_j,1\rangle \in L_{(\mathbb{B},O)}(p_j), \\ ((s,d) \succ_O (s_j,d_j)) \vee \\ ((s = s_j \neq \text{strand} \circ (O,c,q)) \wedge (d = d_j))]}} f_{P_1}(p_j) > \sum_{\substack{p_j \in \mathbb{B}, \forall \langle s,d,1\rangle \in L_{(\mathbb{B},O)}(p_i), \\ [\exists \langle s_j,d_j,1\rangle \in L_{(\mathbb{B},O)}(p_j), \\ ((s,d) \succ_O (s_j,d_j)) \vee \\ ((s = s_j \neq \text{strand} \circ (O,c,q)) \wedge (d = d_j))]}} f_{P_2}(p_j)
\end{aligned} \tag{A.26}$$

   Following definition 5.33, $L_{(\mathbb{B},O_<)}(P_1) \prec L_{(\mathbb{B},O_<)}(P_2)$ implies that:

$$\begin{aligned}
&\forall \langle s,d,x_1\rangle \in L_{(\mathbb{B},O_<)}(P_1), \\
&\sum_{\substack{\langle s_i,d_i,x_i\rangle \in L_{(\mathbb{B},O_<)}(P_1), \\ [((s_i,d_i) \succ_{O_<} (s,d)) \vee \\ (s_i = s \neq \text{strand} \circ (O_<,q,c) \wedge d_i = d)]}} x_i \leq \sum_{\substack{\langle s_i,d_i,x_i\rangle \in L_{(\mathbb{B},O_<)}(P_2), \\ [((s_i,d_i) \succ_{O_<} (s,d)) \vee \\ (s_i = s \neq \text{strand} \circ (O_<,q,c) \wedge d_i = d)]}} x_i
\end{aligned} \tag{A.27}$$

According to theorem 5.26,

$$\sum_{\substack{\langle s_i,d_i,x_i\rangle \in L_{(\mathbb{B},O_<)}(P_k),\\ [((s_i,d_i)\succ_{O_<}(s,d))\vee\\ (s_i=s\neq \text{strand}\circ(O_<,q,c)\wedge d_i=d)]}} x_i = \sum_{\substack{\langle s_i,d_i,x_i\rangle \in L_{(\mathbb{B},O_<)}(P_k),\\ [((s_i,d_i)\succ_{O_<}(s,d))\vee\\ (s_i=s\neq \text{strand}\circ(O_<,q,c)\wedge d_i=d)]}} \sum_{p_j,\langle s_i,d_i,1\rangle\in L_{(\mathbb{B},O)}(p_j)} f_{P_k}(p_j)$$

and hence:

$$\sum_{\substack{\langle s_i,d_i,x_i\rangle \in L_{(\mathbb{B},O_<)}(P_k),\\ [((s_i,d_i)\succ_{O_<}(s,d))\vee\\ (s_i=s\neq \text{strand}\circ(O_<,q,c)\wedge d_i=d)]}} x_i = \sum_{\substack{p_j\in\mathbb{B},\forall\langle s,d,1\rangle\in L_{(\mathbb{B},O)}(p_i),\\ [\exists\langle s_j,d_j,1\rangle\in L_{(\mathbb{B},O)}(p_j),\\ ((s,d)\succ_O(s_j,d_j))\vee\\ ((s=s_j\neq \text{strand}\circ(O,c,q))\wedge(d=d_j))]}} f_{P_k}(p_j) \tag{A.28}$$

From (A.27) and (A.28), it follows that:

$$\begin{aligned}&\forall\langle s,d,x_1\rangle\in L_{(\mathbb{B},O_<)}(P_1),\\ &\sum_{\substack{p_j\in\mathbb{B},\forall\langle s,d,1\rangle\in L_{(\mathbb{B},O)}(p_i),\\ [\exists\langle s_j,d_j,1\rangle\in L_{(\mathbb{B},O)}(p_j),\\ ((s,d)\succ_O(s_j,d_j))\vee\\ ((s=s_j\neq \text{strand}\circ(O,c,q))\wedge(d=d_j))]}} f_{P_1}(p_j) \leq \sum_{\substack{p_j\in\mathbb{B},\forall\langle s,d,1\rangle\in L_{(\mathbb{B},O)}(p_i),\\ [\exists\langle s_j,d_j,1\rangle\in L_{(\mathbb{B},O)}(p_j),\\ ((s,d)\succ_O(s_j,d_j))\vee\\ ((s=s_j\neq \text{strand}\circ(O,c,q))\wedge(d=d_j))]}} f_{P_2}(p_j)\end{aligned} \tag{A.29}$$

Because (A.26) and (A.29) contradict each other, the original hypothesis that $P_1 \preccurlyeq_p P_2$ is false when $L_{(\mathbb{B},O_<)}(P_1) \prec L_{(\mathbb{B},O_<)}(P_2)$ has been shown to be incorrect. Therefore, $P_1 \preccurlyeq_p P_2 \leftarrow L_{(\mathbb{B},O_<)}(P_1) \prec L_{(\mathbb{B},O_<)}(P_2)$.

2. $\mathbf{P_1 \preccurlyeq_p P_2 \rightarrow L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2)}$:
In order to prove this part of the theorem, the hypothesis will be made that P_1 and P_2 are defined such that $P_1 \preccurlyeq_p P_2$ is true and that $L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2)$ is false. It will be demonstrated that such a situation is impossible, and hence, $L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2)$ must be true if $P_1 \preccurlyeq_p P_2$ is true.
Because of definition 5.33, $\neg(L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2))$ implies that there exists a tuple $\langle s,d,x_1\rangle \in L_{(\mathbb{B},O_<)}(P_1)$ such that:

$$\begin{aligned}&\forall\langle s,d,x_1\rangle\in L_1,\\ &\sum_{\substack{\langle s_i,d_i,x_i\rangle\in L_1,[((s_i,d_i)\succ_{O_<}(s,d))\vee\\ (s_i=s\neq \text{strand}\circ(O_<,q,c)\wedge d_i=d)]}} x_i > \sum_{\substack{\langle s_i,d_i,x_i\rangle\in L_2,[((s_i,d_i)\succ_{O_<}(s,d))\vee\\ (s_i=s\neq \text{strand}\circ(O_<,q,c)\wedge d_i=d)]}} x_i\end{aligned}$$

By employing definition 5.33 and theorem 5.26 it can be shown in the same way as in part 1 of this theorem that:

$$\begin{aligned}&\exists\langle s,d,x_1\rangle\in L_{(\mathbb{B},O_<)}(P_1),\\ &\sum_{\substack{p_j\in\mathbb{B},\forall\langle s,d,1\rangle\in L_{(\mathbb{B},O)}(p_i),\\ [\exists\langle s_j,d_j,1\rangle\in L_{(\mathbb{B},O)}(p_j),\\ ((s,d)\succ_O(s_j,d_j))\vee\\ ((s=s_j\neq \text{strand}\circ(O,c,q))\wedge(d=d_j))]}} f_{P_1}(p_j) > \sum_{\substack{p_j\in\mathbb{B},\forall\langle s,d,1\rangle\in L_{(\mathbb{B},O)}(p_i),\\ [\exists\langle s_j,d_j,1\rangle\in L_{(\mathbb{B},O)}(p_j),\\ ((s,d)\succ_O(s_j,d_j))\vee\\ ((s=s_j\neq \text{strand}\circ(O,c,q))\wedge(d=d_j))]}} f_{P_2}(p_j)\end{aligned} \tag{A.30}$$

By employing definitions 5.6 and 5.19 and theorem 5.21 it can be shown, in the same way as in part 1 of this theorem, that $P_1 \preccurlyeq_p P_2$ implies that for all $p_i = p_{11}, \ldots, p_{1n_1}$:

$$\sum_{\substack{p_j \in \mathbb{B}, \forall \langle s,d,1\rangle \in L_{(\mathbb{B},O)}(p_i),\\ [\exists \langle s_j,d_j,1\rangle \in L_{(\mathbb{B},O)}(p_j),\\ ((s,d) \succ_O (s_j,d_j)) \vee\\ ((s = s_j \neq \text{strand} \circ (O,c,q)) \wedge (d = d_j))]}} f_{P_1}(p_j) \leq \sum_{\substack{p_j \in \mathbb{B}, \forall \langle s,d,1\rangle \in L_{(\mathbb{B},O)}(p_i),\\ [\exists \langle s_j,d_j,1\rangle \in L_{(\mathbb{B},O)}(p_j),\\ ((s,d) \succ_O (s_j,d_j)) \vee\\ ((s = s_j \neq \text{strand} \circ (O,c,q)) \wedge (d = d_j))]}} f_{P_2}(p_j) \tag{A.31}$$

Because (A.30) and (A.31) contradict each other, the original hypothesis that $L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2)$ can be false when $P_1 \preccurlyeq_p P_2$ is true has been shown to be incorrect. Therefore, the hypothesis must be withdrawn and part 2 of the theorem is proven.

■

Theorem 5.35. Given two OMPs $P_1 = p_{11} \oplus \ldots \oplus p_{1n_1}$ and $P_2 = p_{21} \oplus \ldots \oplus p_{2n_2}$ such that no pair of BPQs p_{ij} and p_{kl} can be taken from $p_{11}, \ldots, p_{1n_1}, p_{21}, \ldots, p_{2n_2}$ such that $p_{ij} \ll p_{kl}$, then

$$P_1 \preccurlyeq_p P_2 \leftrightarrow L_{(\mathbb{B},O_<)}(P_1) \preccurlyeq L_{(\mathbb{B},O_<)}(P_2) \tag{A.32}$$

Proof: This theorem is a generalisation of theorem 5.34 since it does not assume that all the constituent BPQs of OMPs P_1 and P_2 are within of equivalent order of magnitude. Because the $\sim$ relation is commutative and transitive (this is a direct consequence of (5.5)), two BPQs are not of equivalent order of magnitude if one is an order of magnitude greater than the other or if they are not related at all. Because the former is deemed false in the theorem specification, BPQs that are not of equivalent order of magnitude are unrelated and hence, they can not be members of the same strand and their labels are incomparable. Therefore, the BPQs $p_{11}, \ldots, p_{1n_1}$, $p_{21}, \ldots, p_{2n_2}$ can partitioned according to the distinct orders of magnitude. Because theorem 5.34 holds within each partition, and because the BPQs in different partitions are unrelated to one another, (A.32) holds in this more general case as well. ■

Theorem 5.36.

$$\begin{aligned}
P_1 = P_2 &\leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_<, P_1, P_2) = \text{equal-to}\\
P_1 \prec P_2 &\leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_<, P_1, P_2) = \text{smaller-than}\\
P_1 \succ P_2 &\leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_<, P_1, P_2) = \text{greater-than}\\
P_1 ? P_2 &\leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_<, P_1, P_2) = \text{incomparable}
\end{aligned}$$

Proof:

Below, it will be shown that:

$$P_1 = P_2 \leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{equal-to}$$
$$P_1 \prec P_2 \rightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{smaller-than}$$
$$P_1 \succ P_2 \rightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{greater-than}$$
$$P_1 ? P_2 \rightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{incomparable}$$

As these cases cover all possible outcomes of comparing OMPs, and the algorithm comes to the same conclusion for each of these cases, the following is also true:

$$P_1 \prec P_2 \leftarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{smaller-than}$$
$$P_1 \succ P_2 \leftarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{greater-than}$$
$$P_1 ? P_2 \leftarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O_{<}, P_1, P_2) = \text{incomparable}$$

1. $\mathbf{P_1 = P_2} \leftrightarrow \text{COMPARE-OMP}(\mathbb{B}, \mathbf{O}_{\ll}, \mathbf{O}_{<}, \mathbf{P_1}, \mathbf{P_2}) = \text{equal-to}$:
   Because of (5.7), $P_1 = P_2$ if all of its BPQs are equal. The algorithm 5.1 deems two OMPs equal if and only if their labels are identical. According to theorem 5.25, a combination of labels uniquely identifies an OMP. In other words, two OMPs can only have identical labels if they consist of the same combination of constituent BPQs. Therefore, the algorithm returns "equal-to" if and only if the OMPs are identical.

2. $\mathbf{P_1} \prec \mathbf{P_2} \rightarrow \text{COMPARE-OMP}(\mathbb{B}, \mathbf{O}_{\ll}, \mathbf{O}_{<}, \mathbf{P_1}, \mathbf{P_2}) = \text{smaller-than}$:
   Due to 5.5, $P_1 \prec P_2$ implies that:

$$\forall p_a \in \mathbb{B}, (P_1 \prec_{p_a} P_2) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_1 \prec_{p_b} P_2) \tag{A.33}$$

   Let $\mathbb{B}(P)$, with $i = P_1, P_2$, denote the subset of $\mathbb{B}$ such that $\forall q \in \mathbb{B}(P), f_P > 0$. The BPQs in $\mathbb{B}(P_1)$ and $\mathbb{B}(P_2)$ can each be partitioned into three sets $\mathbb{B}(P, \prec)$, $\mathbb{B}(P, =)$, and $\mathbb{B}(P, \text{rest})$, with $P = P_1, P_2$, such that:

$$\mathbb{B}(P_1, \prec) = \{p \in \mathbb{B}(P_1) \mid P_1 \prec_p P_2\}$$
$$\mathbb{B}(P_1, =) = \{p \in \mathbb{B}(P_1) \mid (p \notin \mathbb{B}(P_1, \prec)) \wedge (f_{P_1}(p) = f_{P_2}(p))\}$$
$$\mathbb{B}(P_1, \text{rest}) = \{p \in \mathbb{B}(P_1) \mid (p \notin \mathbb{B}(P_1, \prec)) \wedge (p \notin \mathbb{B}(P_1, =))\}$$
$$\mathbb{B}(P_2, \prec) = \{p \in \mathbb{B}(P_2) \mid P_1 \prec_p P_2\}$$
$$\mathbb{B}(P_2, =) = \{p \in \mathbb{B}(P_2) \mid (p \notin \mathbb{B}(P_2, \prec)) \wedge (f_{P_1}(p) = f_{P_2}(p))\}$$
$$\mathbb{B}(P_2, \text{rest}) = \{p \in \mathbb{B}(P_2) \mid (p \notin \mathbb{B}(P_2, \prec)) \wedge (p \notin \mathbb{B}(P_2, =))\}$$

   Let $P_{i,=}$, with $i = 1, 2$, be defined as:

$$P_{i,\prec} = \oplus_{p \in \mathbb{B}(P_i, \prec)} f_P(p) \times p$$

As shown in part 1, the parts of the label generated for BPQs in $\mathbb{B}(P_1,=)$ and $\mathbb{B}(P_2,=)$ are identical and can therefore be ignored. Because of (A.33), for each $p \in \mathbb{B}(P,\text{rest})$, there exists a $p' \in \mathbb{B}(P,\prec)$ such that $p \ll p'$. Therefore, the BPQs $p \in \mathbb{B}(P,\text{rest})$, it is shown that:

$$P_{1,\prec} \prec P_{2,\prec} \rightarrow \text{COMPARE-OMP}(\mathbb{B}, O_{\ll}, O) <, P_{1,\prec}, P_{2,\prec}) = \text{smaller-than}$$

Let $\mathbb{B}(P,\prec)$, with $i = P_1, P_2$, denote the subset of $\mathbb{B}$ such that $\forall q \in \mathbb{B}(P,\prec)$, $f_P(q) > 0$. The BPQs in $\mathbb{B}(P_1,\prec)$ and $\mathbb{B}(P_2,\prec)$ can each be partitioned into two sets $\mathbb{B}(P,\ll)$, $\mathbb{B}(P,<)$, with $P = P_1, P_2$, such that:

$$\begin{aligned}
\mathbb{B}(P_1,\ll) &= \{p \in \mathbb{B}(P_1,\prec) \mid \exists p' \in \mathbb{B}(P_2,\prec), p \ll p'\} \\
\mathbb{B}(P_1,<) &= \{p \in \mathbb{B}(P_1,\prec) \mid p \notin \mathbb{B}(P_1,\ll)\} \\
\mathbb{B}(P_2,\ll) &= \{p \in \mathbb{B}(P_2,\prec) \mid (\nexists p \in \mathbb{B}(P_1,\prec), p \sim p') \wedge (\exists p' \in \mathbb{B}(P_1,\prec), p' \ll p)\} \\
\mathbb{B}(P_2,<) &= \{p \in \mathbb{B}(P_2,\prec) \mid p \notin \mathbb{B}(P_2,\ll)\}
\end{aligned}$$

Let $P_{i,\ll}$ and $P_{i,<}$, with $i = 1,2$ be defined as:

$$\begin{aligned}
P_{i,\ll} &= \oplus_{p \in \mathbb{B}(P_i,\ll)} f_P(p) \times p \\
P_{i,<} &= \oplus_{p \in \mathbb{B}(P_i,<)} f_P(p) \times p
\end{aligned}$$

Clearly $P_{1,\ll} \prec P_{2,\ll}$. Because of theorem 5.32, the labels of $P_{i,\ll}$, with $i = 1,2$ can be partitioned into L_{i1} and L_{i2} such that $L_{11} \prec\prec L_{21}$ and $L_{L_{12} \rightarrow (\mathbb{B},O_<)}(P_1) = \emptyset$. From $L_{11} \prec\prec L_{21}$, the algorithm will deduce that P_1 is smaller than P_2. The specific $O_<$ labels derived from L_{12} and L_{22} will not contradict that.

Following the above definitions, it is also obvious that $\forall p_1 \in \mathbb{B}(P_1,<), \exists p_2 \in \mathbb{B}(P_2,<), p_1 \sim p_2$. Therefore a comparison will not be possible based on a comparison of $L_{(\mathbb{B},O_\ll)}(\mathbb{B}(P_1,<))$ and $L_{(\mathbb{B},O_\ll)}(\mathbb{B}(P_2,<))$. Therefore, the algorithm will make a comparison with respect to $O_<$. From theorem 5.35 and $\forall p_1 \in \mathbb{B}(P_1,<), \exists p_2 \in \mathbb{B}(P_2,<), p_1 \sim p_2$, $P_{1,<} \prec P_{2,<}$ implies that $L_{(\mathbb{B},O_<)}(\mathbb{B}(P_1,<)) \prec L_{(\mathbb{B},O_<)}(\mathbb{B}(P_2,<))$.

The combination of both comparisons leads the algorithm to conclude that $P_{1,\prec}$ is smaller that $P_{2,\prec}$

3. $\mathbf{P_1} \succ \mathbf{P_2} \rightarrow \text{COMPARE-OMP}(\mathbb{B}, \mathbf{O}_{\ll}, \mathbf{O}_<, \mathbf{P_1}, \mathbf{P_2}) = \text{greater-than}$:
   As part 2.

4. $\mathbf{P_1}?\mathbf{P_2} \rightarrow \text{COMPARE-OMP}(\mathbb{B}, \mathbf{O}_{\ll}, \mathbf{O}_<, \mathbf{P_1}, \mathbf{P_2}) = \text{incomparable}$:
   Proof is similar to parts 2 and 3, with the exception that the sets of BPQs $\mathbb{B}(P,\prec)$ are partitioned into a subset from which P_1 seems smaller than P_2 and a subset from which P_1 seems greater than P_2. From the combination of the labels of both, the algorithm concludes that the OMPs are incomparable.

■

Theorem 6.2. Any A* algorithm that is applied to a DPCSP and stores the nodes n it creates in the set O in descending order of a heuristic $\widehat{PP}(n)$, such that $\widehat{PP}(n) \succcurlyeq PP(n)$, is admissible and it is guaranteed to find a node n which maximises $CP(n)$.
Proof: It follows from definition 6.1 that $PP(n)$, and by extension that $\widehat{PP}(n)$ is greater than or equal to the combined preference of any value-assignment of unassigned attributes that is consistent with the partial solution of n. In this BFS, the nodes n are maintained in a priority queue in descending order of $\widehat{PP}(n)$. Let δ be a distance function that reverses the preference ordering such that $\delta(P_1) \prec \delta(P_2) \leftarrow P_1 \succ P_2$. The algorithm can then be described as a BFS guided by $\delta(\widehat{PP}(n)) = \delta(CP(n)) \oplus \delta(h(n))$, where the nodes n are maintained in a priority queue in ascending order of $\delta(\widehat{PP}(n))$ and where $\delta(h(n))$ is a lower bound on the distance between n and the optimal solution. Hence, as shown in [72], this A* algorithm is admissible and it is guaranteed to find a solution solution node n with a minimal $\delta(CP(n))$ or a maximal $CP(n)$. ■

Theorem 6.3. $\forall n, \widehat{PP}_{\text{basic}}(n) \succcurlyeq PP(n)$
Proof: According to definition 6.1 and equation (6.5):

$$\widehat{PP}(n) = (\oplus_{x:d\in S(n)} P(x:d)) \oplus (\oplus_{x\in X_{nd}(n)} \max_{d\in D(x)} P(x:d)) \tag{A.34}$$

According to definition 6.1:

$$PP(n) = [\oplus_{x:d\in S(n)} P(x:d)] \oplus \left[\max_{d_i\in D(x_i),\ldots,d_j\in D(x_j), S(n)\cup\{x_i:d_i,\ldots,x_j:d_j\}\mathbf{C},\mathbf{A}\nvdash\perp} (P(x_i:d_i)\oplus\ldots\oplus P(x_j:d_j))\right]$$

where $\{x_i,\ldots,x_j\} = X_{nd}(n)$. That is,

$$PP(n) = [\oplus_{x:d\in S(n)} P(x:d)] \oplus [P(x_i:d_i)\oplus\ldots\oplus P(x_j:d_j)]$$

where $\{x_i:d_i,\ldots,x_j:d_j\}$ is a set of assignments to the attributes of X_{nd} such that $S(n)\cup\{x_i:d_i,\ldots,x_j:d_j\}$ is consistent with the compatibility constraints $\mathbf{C}$ and the activity constraints $\mathbf{A}$. For each attribute $x \in X_{nd}(n)$, a preference $P(x:d)$, with $d \in D(x)$, is added to $PP(n)$ and a preference $\max_{d\in D(x)} P(x:d)$ is added to $\widehat{PP}(n)$. It is clear that

$$P(x_i:d_i) \preccurlyeq \max_{d\in D(x_i)} P(x_i:d) \quad \ldots \quad P(x_i:d_i) \preccurlyeq \max_{d\in D(x_i)} P(x_i:d)$$

It follows that, $\widehat{PP}(n) \succcurlyeq PP(n)$. ■

Theorem 6.4. $\forall n, \widehat{PP}_{\text{basic}}(n) \succcurlyeq \widehat{PP}_{\text{fc}}(n) \succcurlyeq PP(n)$
Proof: By means of definition 6.1, and equations (6.5) and (6.6), the properties

$\forall n, \widehat{PP}_{\text{basic}}(n) \succcurlyeq \widehat{PP}_{\text{fc}}(n)$ and $\forall n, \widehat{PP}_{\text{fc}}(n) \succcurlyeq PP(n)$ can be proven in the same way as theorem 6.3. ■

Theorem 6.6. $\forall n, \widehat{PP}_{\text{basic}}(n) \succcurlyeq \widehat{PP}_{\text{mpb}}(n) \succcurlyeq PP(n)$
Proof:
By means of definition 6.1, and equations (6.5) and (6.12), the properties $\forall n, \widehat{PP}_{\text{basic}}(n) \succcurlyeq \widehat{PP}_{\text{mpb}}(n)$ and $\forall n, \widehat{PP}_{\text{mpb}}(n) \succcurlyeq PP(n)$ can be proven in the same way as theorem 6.3. ■

Theorem 6.7. $\forall n, [\widehat{PP}_{\text{fc}}(n) \succcurlyeq \widehat{PP}_{\text{mpb-fc}}(n)] \wedge [\widehat{PP}_{\text{mpb}}(n) \succcurlyeq \widehat{PP}_{\text{mpb-fc}}(n)] \wedge [\widehat{PP}_{\text{mpb-fc}}(n) \succcurlyeq PP(n)]$
Proof:
By means of definition 6.1, and equations (6.6), (6.12) and (6.13), the properties $\forall n, \widehat{PP}_{\text{fc}}(n) \succcurlyeq \widehat{PP}_{\text{mpb-fc}}(n)$, $\forall n, \widehat{PP}_{\text{mpb}}(n) \succcurlyeq \widehat{PP}_{\text{mpb-fc}}(n)$, and $\forall n, \widehat{PP}_{\text{mpb-fc}}(n) \succcurlyeq PP(n)$ can be proven in the same way as theorem 6.3. ■

Appendix B

Framework for Automated Modelling

Models are approximate conceptual representations of real-world systems that enable effective and efficient problem solvers, as they formalise only the relevant relations between the concepts of interest. When modelling a given system, which may be an object or activity, the following issues must be addressed:

- a *representation formalism*: a (symbolic) language that allows the representation of relevant aspects of the system and their relations,
- a *problem*: a task that must be solved with respect to the system, such as design, diagnosis, explanation, monitoring or prediction, and
- a *problem solver*: an inference procedure capable of generating one or more solutions for the problem given a description of the system in the appropriate representation formalism.

A model is therefore an instance of the representation formalism that exhibits certain aspects of the behaviour or properties possessed by the system it models and that enables the problem solver to apply its inference procedures. Consequently, modelling is a process that requires the means to perceive a system's properties and behaviour, to propose promising models, and to validate the model by generating its properties and behaviour and comparing these with those of the system itself. From this description, it follows that modelling may involve both perception and symbolic reasoning. Unfortunately, such an integrated modelling process is beyond the scope of any existing automated modeller. In practice, automated modellers require the user to present a formalised perception of the system or its behaviour.

This appendix presents a brief overview of automated modelling and proposes a broad classification in order to relate compositional modelling to alternative approaches to model construction. A complete and exhaustive classification is beyond the scope of this work. In general, automated modellers differ with respect to the set of target systems they aim at modelling, the representation formalisms they employ in the process, the types of problem task specifications they hope to tackle and the associated problem solving procedures.

Compositional modelling research is not specific to a single class of target systems. Hence, the present classification does not consider this criterion. The *representation formalisms* significantly affect the range of models a modeller can represent. This criterion is elaborated in section B.1. The set of problem tasks and problem solvers for which an automated modeller can formulate models constitutes the basis of the second criterion. Although the problem settings differ amongst compositional modellers, they all provide a specific means of composing models from first principles. Hence, this classification focuses on the deductive or inductive nature of modellers, which is discussed in section B.2. Finally, in section B.3, the important classes of automated modellers according to these two main criteria are identified.

B.1 Representations

Because models are approximate conceptual descriptions they may differ between one another. Model classification frameworks provide sets of dimensions, such as precision, resolution, scope, granularity, etc. that allow a categorisation of the features of different models [27]. The purpose of model construction is to find a model with suitable feature given the problem at hand. The expressive power that enabled the distinctions between alternative models according to various dimensions in the first place lies in the representation formalisms used during the model construction process. Each class of representation formalisms has its own unique characteristics that allow it to express models with different features along a specific subset of model classification dimensions.

The present discussion aims at providing a general overview of automated modellers and hence, a rather broad classification of representation formalisms is proposed, which could be extended and enhanced to suit more specific purposes. Formalisms are categorised according to three levels of abstraction, as inspired by [19] and illustrated in figure B.1: technical, conceptual and mathematical.

B.1.1 Technical level representations

Representations at the *technical level* model a domain system by relating visually, or mentally (if the system in question can not be perceived visually, e.g. an economy), distinguishable elements (usually components or processes) of the system. Thus, technical models explicitly represent the structure of a system of interest. Consequently, modelling choices made at this level primarily affect the topology and scope of the resulting model.

Many kinds of problem solving, including those tasks within which compositional modelling is applied, require the conciseness and precision of a certain rigorous mathematical formalism whereas the domain expert's most basic understanding and intuition lies in technical models. In this respect, technical models are the closest to the real-world system in the mind of the domain expert and could be considered the least abstract. In other tasks, such as design, the modeller may have a better understand-

ing of the (required) mathematical properties of the system whereas a technical level model is required. In such cases, the technical level representation is said to be the most abstract. Within the remainder of this text, abstraction will be considered from the perspective of compositional modelling and hence, technical level representations are considered to be the least abstract.

Technical models include *geometric models with structural annotations* and *component-connection models*. The former are simplified graphical representations of systems annotated with information such as properties of the elements in the graphical representation, how and by what means they are connected, etc. For example, a simplified graphical representation of a U-Tube, together with an indication of the content of the tube being fluid, the tube being solid and rigid, the direction of gravity and the two position indicators of the fluid content's height, constitutes a geometric model with structural annotations. Component-connection models represent a system as a set of interconnected components. Each component is equivalent to a subsystem of the entire system and the connections model the input and output terminals between these subsystems.

Technical level representations are well suited for many types of problem specification by the user because they should be relatively easily produced by people with a minimal amount of experience in the domain of discourse. For this reason they are often used as the input model to an automated modeller. Component-connection models require some domain specific knowledge to recognise the components and how they are connected (see [12] for a detailed discussion). However this should not pose a problem for modellers with average knowledge of the domain of discourse, even though such a representation requires the user to resolve a number of the representational issues manually. For example, the level of detail at which a system's components are represented must be indicated by the user. As an alternative, geometric models on the other hand require fewer of these representational issues to be resolved, but any practical application relies on structural annotations to enrich the specifications [3]. Such annotations introduce new modelling decisions that need user intervention.

B.1.2 Conceptual level representations

Models at the *conceptual level* represent the general notions underlying classes of subsystems or phenomena that enable the system being modelled to perform its function, including influences exerted between the subsystems. Modelling choices at this level primarily affect the granularity and order of the resulting model and has an impact upon the resolution and form of the model.

Concepts that represent a class of technical components that perform a certain function are called conceptual components and are the type of concepts used in *conceptual component models*. Similar to component-connection models, conceptual component models consist of a set of interconnected components, but instead of representing technically distinct subsystems (e.g. an electrical motor), the components represent domain theory concepts (e.g. a gyrator). In the example, a gyrator is a specific type of conceptual means of transferring energy from one domain to another whereas an electrical

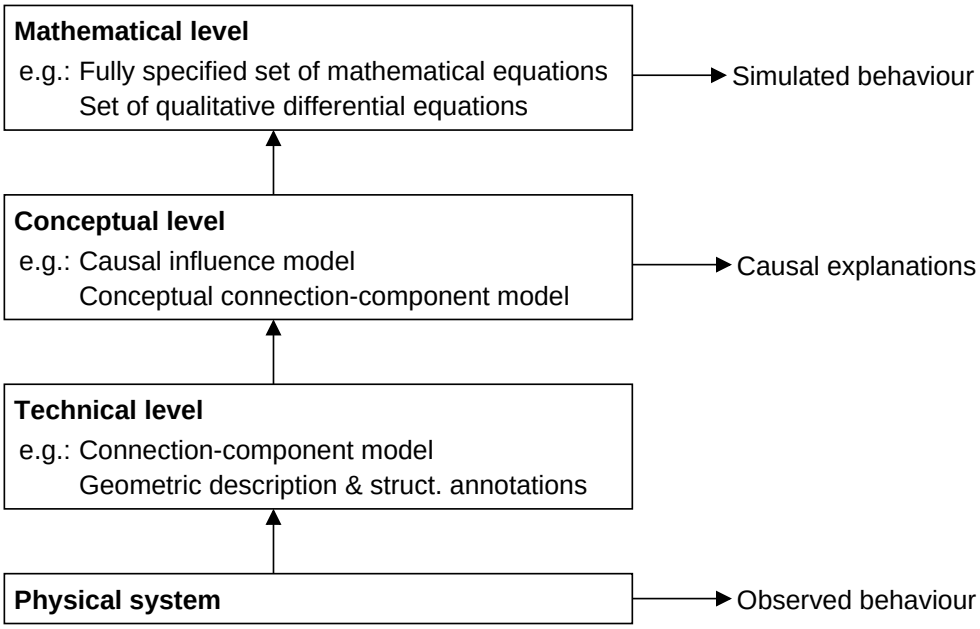


Figure B.1: Classification of representation formalisms

motor is one example of a piece of equipment that performs the function of gyrator. Since technical level components and conceptual components are entirely different, the former will be referred to as "components" and the latter will be referred to as "conceptual components" hereafter. Alternatively, the concepts may be phenomena or properties of a domain. In *causal influence models*, these properties are represented by means of mathematical variables of the domain theory, such as current or velocity. The relations between these variables are causal influences representing how variables affect one another.

Bond graphs [17, 18, 63] are an example of conceptual component models. The connections, called bonds, in these models represent transfer of energy. Energy E is itself modelled by two variables: flow f and effort e, such that $E \equiv f \times e$. The flow and effort variables and their derivatives have specific meanings and dimensions for each energy domain. Additionally, a small number of conceptual components can model specific constraints between effort, flow and their respective derivatives or model transfers between different energy domains. Again, these constraints represent specific functions for each energy domain. As such, bond graphs are able to represent a wide variety technical engineering systems within a uniform framework and by means of a small number of concepts.

Examples of causal influence models include models developed using domain theories such as *qualitative process theory* (QPT) [50]. In QPT, the relations between concepts are modelled by means of direct and indirect influences. A direct or differential influence $I^{\pm}(X_i, Y)$ means that an increase of the influencer X_i tends to in-

crease/decrease the rate of change of the influencee Y:

$$I^{+}(X_i, Y) \equiv \frac{dY}{dt} = f(X_1, \ldots, X_i, \ldots, X_n) \qquad \text{with} \frac{\partial f}{\partial X_i} > 0$$

Similarly, an indirect or functional influence, or qualitative proportionality, $Q^{\pm}(X_i, Y)$ means that an increase of X_i tends to increase/decrease Y:

$$Q^{+}(X_i, Y) \equiv Y = f(X_1, \ldots, X_i, \ldots, X_n) \qquad \text{with} \frac{\partial f}{\partial X_i} > 0$$

Extensions of QPT provide additional expressive power. *Qualitative influence diagrams* [76], for example, can handle more specific types of functional dependencies, more complex types of influence combinations. This work also introduces operators to automatically generate abstraction and approximations which are very useful in automated modelling.

B.1.3 Mathematical level representations

Mathematical level representations describe a system's behaviour by means of a set of equations. The behaviour captured by such a model can be reproduced by the use of a *simulator*. Numeric mathematical representations are the most precise type of model, in that they generate a single, uniquely specified, behaviour descriptions per model and the description is presented in real values. For example, a continuous simulator, such as DYNAMO [143], can produce a system's behaviour from a set of *ordinary differential equations (ODEs)* by extrapolating initial values based on the continuity and differentiability assumptions. Modelling choices at this level primarily affect precision, uncertainty, form and resolution [105]. As opposed to other types of models, however, numerical models require parameter value estimates. Parameter estimation is a problem that demands a set of observations and an assumed equation structure between variables.

Qualitative differential equations (QDEs) on the other hand encompass an entire class of behaviours (see [103] for a detailed discussion). In this representation, each variable takes on a qualitative state, rather than a value. A qualitative state typically consists of a quantity and a rate of change. The quantities are taken from quantity spaces which are ordered sets of so-called landmark values representing mathematically odd points in physically significant values. Rates of change are often denoted by the sign of the derivative of the variable in question, thereby representing one of $\{$Increasing, Steady, Decreasing$\}$. Relations between variables are modelled by qualitative constraints.

Given a number of variables, qualitative constraints on these variables and the initial qualitative states for all or some of the variables, a qualitative simulator, e.g. the QSIM algorithm [101], extrapolates all possible and qualitatively distinct behaviours from the initial state, also based on continuity and differentiability assumptions regarding the system's behaviour. Impossible behavioural descriptions generated are

then removed by checking the consistency between the QDEs and the generated states. Discontinuities require a new model and therefore new QDEs valid within an operating region, which is a range of qualitative values for each of the variables. The set of lists of subsequent qualitative states produced in this way for each variable in the set of QDEs that models the system is called the *envisionment*.

The equations in mathematical models enable the domains of the variables to be constrained to anything between a single value or a range of values. Therefore, mathematical models are more suitable for simulation, which is a prerequisite for many applications such as prediction, monitoring, diagnosis, etc. The conceptual counterparts of these mathematical models are not well equipped for this purpose. In general, conceptual relations (such as QPT influences) are not based on the closed-world assumptions, and hence no generic statements on their behaviour can be made since there may be interactions with unknown conceptual relations. Also, some conceptual formalisms (such as bond graphs) implicitly combine several variables in a single concept and hence their granularity is too low. However, the absence of a closed-world assumption makes the relations highly composable and useful as an intermediate representation formalism for some automated modellers [46]. The relative simplicity of the conceptual formalisms whilst containing the essence of the mechanics of the underlying domain theory makes them suitable as a representation for supporting explanation generation.

It is worth noting that there are alternative approaches to the representation of QDEs for qualitative simulation. For instance, the Mycroft system [158, 28] represents and infers the behaviour information about a physical system using a layered notation such that qualitative descriptions of higher-order derivatives can be obtained by differentiating QDEs represented at lower levels. Nevertheless, at each layer the explicit information regarding the systems structure and behaviour is also expressed at the mathematical level.

B.2 Modelling task descriptions

In addition to the use of the criteria regarding model representations, modellers can be classified according to the descriptions of modelling tasks. Although AI models have been applied to a variety of application domains, AI research into automated modellers focuses on constructing models for physical systems (or more generally on application of the natural sciences: chemical systems, environmental systems, etc.). The modellers usually assist in a design, diagnosis, explanation, monitoring or prediction task. A modeller's success at assisting such a task partially depends on the representation it is capable of using. It also depends on how well the model may be embedded in the application architecture of the problem solver. The interface that allows an automated modeller to be embedded in a problem solver typically prescribes whether the modeller works by reasoning deductive, inductive or hybrid. These approaches are discussed here.

B.2.1 Deductive modellers

The deductive approach to modelling transforms a model of one type into a model at another level of abstraction by instantiating generic laws of the domain theory. Deductive modellers differ with respect to the organisation of the knowledge base and the associated model deduction algorithms.

As mentioned earlier, *compositional modellers* use a knowledge base of composable and generic model fragments. The input usually consists of a technical (component-connection) model, although this is not required. This input allows compositional modellers to instantiate a large number of components for the elements of the input component. The allowed combinations of instantiated model fragments are restricted by various structural, operational and domain constraints. The task of a compositional modeller is to find such an allowed combination and possibly an optimal one with respect to some performance indicator, such as a relevance or simplicity measure.

As an example, the function-sharing program presented in [172] shows a knowledge based approach to simple design tasks. The input consists of a specification of *desired or required* behaviour. Based on this specification, it searches a component-connection model of a system that can produce the required behaviour. The actual search algorithm proceeds in two phases. First, a bond graph that exhibits the desired behaviour is searched. Next, the conceptual components of the bond graph are assembled into technical components. What makes this approach deductive rather than inductive (see below) is that the behavioural data are specifications rather than random observations and that the model is constructed by applying principles to specifications rather than discovering principles from a case.

Pure deductive modelling can be a difficult task. Little work exists, to validate the generated deductive models by means of observed behaviour. Therefore, a deductive modeller may need to be combined with other software to validate its results (see section B.2.3). However, certain problem settings, such as tutoring and design, are deductive by nature. As discussed in [53], deductive compositional modelling has been successfully applied to these domains. Deductive modelling may involve the use of large knowledge bases. This implies that the deductive process may produce a large space of possible models that would require pruning, for example, by considering expected behaviours. Again, in such cases, a hybrid modeller may be more appropriate.

B.2.2 Inductive modellers

Inductive modellers generalise *observed* behaviour of a system into an actual model of the system that explains the observed behaviour. In literature, this task is often named *systems identification* [93]. As with deductive modelling, purely inductive system identification is a very complex task because, in theory, the search space is infinite. This problem is particularly acute when the precise model of a complex behaviour must be determined, since that involves adding many new variables and the construction of complex equations. Also, parsimonious models only approximate observations and

therefore, these model constructors need a way to determine what approximations are adequate. Distinguishing noise from interesting behavioural characteristics without any underlying theory is likely to suffer from misjudgements [155].

The observations used by an inductive modeller can be as precise as real-valued time series data or consist of more general time dependent constraints over a number of mathematical properties of the system (e.g. a QSIM envisionment that represents the observed behaviour of the system). Conventional system identifiers also need additional knowledge (e.g. the structure of the mathematical equations) to evaluate model quality. Recently, a number of qualitative system identifiers have been proposed that operate by focusing on more specific but important families of models. This *inductive bias* [121] is achieved by restricting the operators used in a model's equations and by handling incomplete knowledge on relevant parameters, parameter values and measurement errors with qualitative reasoning techniques.

In [146] an approach is proposed to learn models of QDEs over multiple operating regions from a QSIM envisionment. This is achieved by: 1) locating the discontinuities in the behaviour trace so as to determine the operating regions, 2) learning a set of QDEs for the behaviours within each operating region, 3) identifying the operating conditions of each region, and 4) trying to unify operating regions with identical QDEs or operating conditions. Within a single postulated operating region, a set of QDEs can be learned (via step 2) by programs such as MISQ [148] and Qualitative System Identification (QSI) [155]. QSI, for example, first runs a constraint determination algorithm on the envisionment. This algorithm searches through all syntactically possible constraints for a set that applies to the envisionment. The set of constraints found in this way is then used for qualitative simulation (e.g. QSIM). This is called the depth test. If the model that consists of this set of constraints can reproduce the exact envisionment used as input, it is sufficiently "deep" to accurately represent the internal mechanism of the system being identified. Otherwise, the model is said to be too "shallow" and must be deepened by extending it with new variables. These variables are defined by relating them to variables that are already in the model (e.g. a new variable could the derivative of an existing variable). Using these constraints, and a set of heuristics that prefer minimal change in consecutive qualitative states, the qualitative states of these new variables are computed for the time points and intervals of the observation input. Each model extension phase is followed by a new constraint determination phase and a depth test. This procedure is repeated until a sufficiently deep qualitative model is found.

In [181] a system of two algorithms is proposed to discover basic mathematical laws that make up a domain theory, with the results represented as first-principle equations. In this approach, the first algorithm proposes a set of parameterised first-principle equation candidates using a set of variables and information on their scale and dimension. The second algorithm refines the findings of the first by fitting the parameters and testing the result.

A pure inductive modeller does not use any first principles to validate the models it produces. Therefore, inductive models are sensitive to errors in the observations. Noise filtering algorithms may reduce this problem but these risk eliminating significant fea-

tures of the behaviour as well. Additionally, the underlying assumptions of pure inductive models can not be determined. As a result, evaluation of models in terms of user requirements, completeness with respect to relevant phenomena and components of interest are not possible. Finally, the computational complexity and usefulness of the machine-proposed variables and constraints, resulting from qualitative system identification, may be prohibitive in practical applications. In large systems, certain variables are naturally related because of the underlying laws of the domain theory or because of the way the system has been engineered. Observations do not contain such information and hence, all variables are potentially related with one another via any number of intermediate variables.

B.2.3 Hybrid modellers

Being a compromise between inductive and deductive modellers, hybrid modellers use both an initial model and consequent data.

B.2.3.1 Deductive modellers that validate with intended behaviour

These approaches construct models using deductive techniques, but validate each constructed model against a specification of the expected behaviour provided by the user. For example, a minority of compositional modellers (e.g. [136]), though of a great significance, use an expected behaviour description as part of their input. Validation encompasses a comparison of the result of applying continuous simulation onto a candidate model, from one or more initial states, against the specification.

B.2.3.2 Deductive model construction and inductive validation

These methods deduce an intermediate model from a given scenario description but validate the model by means of actual observations. The MM program [3], for example, produces QDEs from geometric or component-connection model through bond graphs. It uses a rule base to generate potential bond graphs that describe the presumed mechanism of the system. The generated bond graph is then translated into a set of QDEs. A variant of QSIM generates the model's envisionment. If the resulting (qualitative) behaviour matches the given observations, the model is retained. Otherwise, a number of model correction rules are attempted or another model is constructed. This generate-and-test loop continues until a model is found that matches the observations.

B.2.3.3 Graph of Models

The graph of models (GoM) [1] is an early and influential approach to automated modelling that selects a model from a graph relating alternative models in terms of their underlying assumptions. The modelling task tackled by GoM consists of searching the simplest model in the graph that sufficiently approximates observed behaviour.

The vertices in a graph of models represent the alternative models and assumptions that justify their use. These nodes are connected by directed edges that represent how one model can be replaced by an alternative one. All edges are annotated by a set of assumptions that must be added and a set of assumptions that must be retracted, when substituting the model from which the edge originates by the model at which the edge is directed.

The assumptions are grouped in so-called assumption classes. Each assumption class represents a dimension along which alternative models can be classified and from which a single assumption must be chosen. As such, the assumptions of an assumption class are mutually exclusive and, for the sake of completeness, always include an assumption that represents the irrelevance of the assumption class. This is similar to certain approaches of compositional modelling.

In GoM, any chosen model may be deemed invalid because of various conflicts. Internal conflicts arise when the parameter values predicted by the model are inconsistent with the constraints defined in the underlying assumptions of the model. For example, an internal conflict occurs in a model that considers $sin(\theta) = \theta$ under the assumption that $-1.9 < \theta < 1.9$ and that predicts $\theta = 2.4$. Empirical conflicts are the result of discrepancies between predicted and observed values beyond the allowed tolerance boundaries. Parameter change rules dictate the effect of assumption transitions on the various parameter values in a model and these are used to find the assumption transitions that rectify empirical conflicts.

Based on the conflicts detected in a model, a set potential transitions can be derived that potentially reduce are alleviate one or more inconsistencies. Heuristics are proposed in [1] to choose the model transition that is likely to resolve most conflicts, and hence to focus the search. The task of model switching in a GoM based on discrepancies between observed and predicted behaviour is formalised by work on approximation reformulations [182, 183].

B.2.3.4 Induction from a model space

This class of modellers take a data set, a set of specifications of candidate models and methods to explicate the candidate models and evaluate them. In conventional system identification systems, the data set is collected according to a purpose built experimental design [49]. The models in the candidates set are then fit to the collected observations and evaluated using standard statistical techniques [137]. A detailed overview of the techniques involved in conventional system identification is presented in [107].

PRET [16] is an alternative approach based on the conventional system identification problem setting, specialised in selecting ordinary differential equations (ODEs) from a set of possibly conflicting hypotheses. However, it uses general ODE theory and domain specific knowledge to identify and instantiate the hypotheses that match the observations rather than statistical techniques.

Semi-QUantitative system IDentification (SQUID) [93] also falls into this category. SQUID represents the space of models in multiple levels. At the structural level the form of the differential equations is described. Such structural differential equations

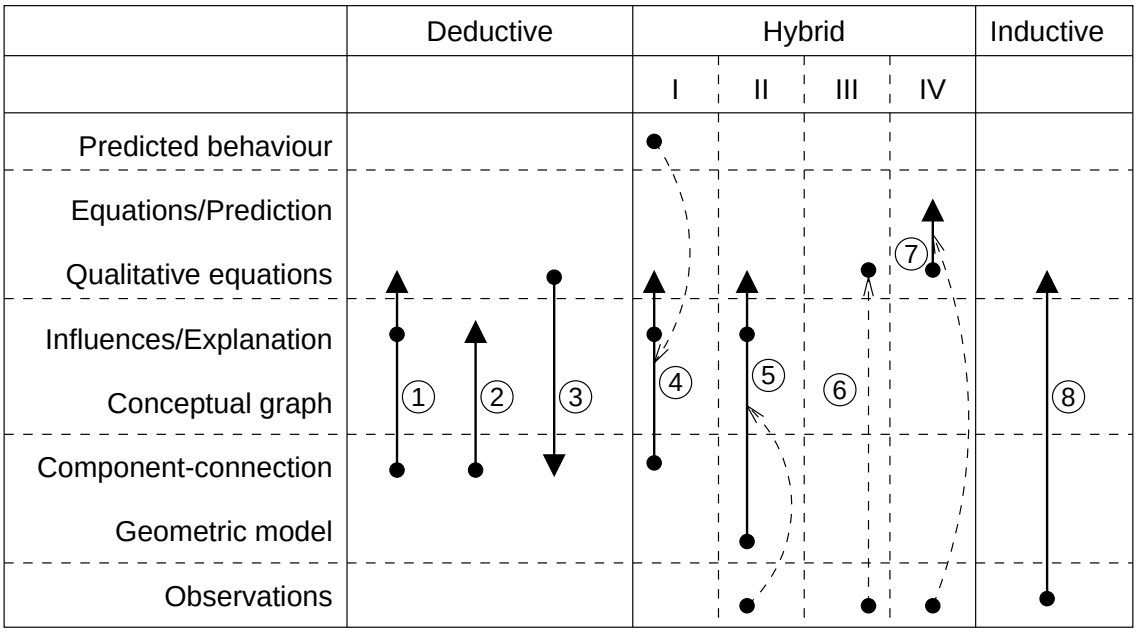
are initially given. The qualitative level breaks each model variable of the structural level into an ordered set of landmarks and adds information on the nature of the functions concerned. The semi-quantitative level adds further information by assigning numeric boundaries to the landmarks and by specifying a pair of boundary functions within which the actual ODE must lie. The approach taken by SQUID is to refine models through the different levels. A refinement operators specific to each level further specify a (qualitative, semi-quantitative or ordinary, depending on the level in question) differential equation and hence refute parts of the model space that do not match the observations. Continuous simulators, again specific to each level, are used to produce the range of behaviours implied by a differential equation that must be matched with the observations. For instance, at the qualitative level, QSIM is used to simulated the model represented at the qualitative level.

B.3 Summary

This section has presented the different issues involved in modelling with respect to a problem solving task. Model construction, or the transformation of one problem representation into another representation more suitable for problem solving, has been identified as the sub-task of modelling solved by most automated modellers. An overview was presented of important classes of algorithms and systems for automated model construction. These automated modellers have been categorised with respect to two important criteria: the representation formalisms used for input, intermediate computations and output and the deductive vs. inductive nature of the model construction process. Figure B.2 illustrates this categorisation for a number of automated modellers discussed in this section. The arrows represent the transformations between representation formalisms that are performed by the different automated modellers. The classes of representation formalisms in this figure are the same as those of section B.1. The columns indicate whether the automated modellers are deductive, inductive or hybrid.

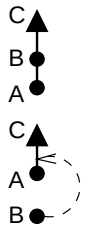
Both criteria classify automated modellers with respect to certain properties of the types of problems these modellers can solve. The possibilities of adapting a problem specification such that it fits into another category are often limited. Therefore, any comparison of automated modellers beyond the boundaries of a single category should be put into this perspective. It is advisable that any application of automated modelling be preceded by initially identifying the purpose of modelling, the available domain knowledge, the nature of available data and the required output. Such an initial analysis may be helpful in finding the appropriate automated modelling technique for a specific problem and, more importantly, may prevent the developer from excessive costs by choosing an unsuitable tool for the task at hand.

However, not all automated modellers fit only in a single category. Compositional modelling, in particular, has been applied to a wide variety of problems. Although the technology is predominantly knowledge-based, both purely deductive and hybrid variants exist. As the next sections show, it has also been applied to transformations between different types of representation formalisms. Consequently, compositional



I Deductive modellers that validate with intended behaviour
II Deductive model construction with inductive completion
III Graph of Models
IV Induction from a model space

1 QPC, and many other CMs
2 CM to generate causal explanations
3 Function-sharing program
4 typical hybrid CM
5 MM
6 Graph of Models
7 PRET
8 QSI, MISQ



Transformation from representation formalism A to B, and from B to C

Transformation from representation formalism A to C, guided by information of type C

Figure B.2: Classification of some important automated modellers

modelling constitutes an important and broad class of automated modellers.

Appendix C

Knowledge Base for the ModMed n Species Model

```
(defEntity variable)

(defEntity stock
  :subclass-of (variable)
)

(defEntity flow
  :subclass-of (variable)
)

(defEntity parameter
  :subclass-of (variable)
)

(defEntity pot-seasonal-parameter
  :subclass-of (parameter)
)

(defEntity integer
  :subclass-of (parameter)
)

(defEntity event-switch
  :subclass-of (variable)
)

(defEntity object)

(defEntity vegetation-component
  :subclass-of (object)
)

(defEntity vegetation-live-component
  :subclass-of (vegetation-component)
)

(defEntity vegetation-dead-component
  :subclass-of (vegetation-component)
)

(defEntity vegetation
  :subclass-of (vegetation-live-component)
)
```

```
(defproperty endogenous-1
  :source-participants ((?m :type variable))
  :structural-conditions ((== ?m *))
  :property (endogenous ?m)
);defproperty

(defproperty endogenous-2
  :source-participants ((?m :type variable))
  :structural-conditions ((d/dt ?m *))
  :property (endogenous ?m)
);defproperty

(defproperty exogenous
  :source-participants ((?m :type variable))
  :structural-conditions ((not (endogenous ?m)))
  :property (exogenous ?m)
);defproperty


;-----------------------------------------------------------------------------
; If the lifeform of a vegetation is relevant, include a variable to represent
; this property.
;-----------------------------------------------------------------------------
(defModelFragment vegetation-lifeform
  :source-participants
    ((?vegetation :type vegetation))
  :assumptions
    ((relevant lifeform ?vegetation))
  :target-participants
    ((?lifeform :type variable :name lifeform))
  :postconditions
    ((lifeform-of ?lifeform ?vegetation))
)

;-----------------------------------------------------------------------------
; Model fragments for partioning a vegetation into components
; (The approach taken is one that can be generalised to all modelling problems
; that require a part-of hierarchy.  The postconditions (e.g.
; (above-ground-vegetation ?above-ground-veg))indicating the function
; of the components could have been replaced by specialised types in a
; type taxonomy (e.g. an above-ground-vegetation is a vegetation-live-component).
;
; Here: a vegetation may consist of an above-ground and a below-ground
; partition.  The above-ground partition may consist of greens and woods.
; The below-ground partition may consist of woody roots and relocatable
; resources.
;-----------------------------------------------------------------------------
(defModelFragment vegetation-is-part-of-itself
  :source-participants
    ((?veg :type vegetation))
  :postconditions
    ((part-of ?veg ?veg))
);defModelFragment

(defModelFragment vegetation->above-and-below-ground-partitions
  :source-participants
    ((?veg :type vegetation))
  :assumptions
    ((model ?veg ground-partitioning))
  :target-participants
    ((?above-ground-veg :type vegetation-live-component :name above-ground-vegetation)
     (?below-ground-veg :type vegetation-live-component :name below-ground-vegetation)
    )
  :postconditions
```

```
    ((partitioning ?veg ?above-ground-veg ?below-ground-veg)
     (above-ground-vegetation ?above-ground-veg)
     (below-ground-vegetation ?below-ground-veg)
     (part-of ?above-ground-veg ?veg)
     (part-of ?below-ground-veg ?veg)
    )
);defModelFragment

(defModelFragment above-ground-vegetation->green-and-wood-partition
  :source-participants
    ((?veg :type vegetation)
     (?above-ground-veg :type vegetation-live-component))
  :structural-conditions
    ((above-ground-vegetation ?above-ground-veg)
     (part-of ?above-ground-veg ?veg)
    )
  :assumptions
    ((model ?above-ground-veg green-wood-partitioned))
  :target-participants
    ((?green :type vegetation-live-component :name green)
     (?wood :type vegetation-live-component :name wood)
    )
  :postconditions
    ((partitioning ?above-ground-veg ?green ?wood)
     (green ?green)
     (wood ?wood)
     (part-of ?green ?veg)
     (part-of ?wood ?veg)
    )
);defModelFragment

(defModelFragment below-ground-vegetation->root-partitions
  :source-participants
    ((?veg :type vegetation)
     (?below-ground-veg :type vegetation-live-component))
  :structural-conditions
    ((below-ground-vegetation ?below-ground-veg)
     (part-of ?below-ground-veg ?veg)
    )
  :assumptions
    ((model ?below-ground-veg wood-resource-partitioned))
  :target-participants
    ((?relocatable-resources :type vegetation-live-component :name relocatable-resources)
     (?woody-roots :type vegetation-live-component :name woody-roots)
    )
  :postconditions
    ((partitioning ?below-ground-veg ?relocatable-resources ?woody-roots)
     (relocatable-resources ?relocatable-resources)
     (woody-roots ?woody-roots)
     (part-of ?relocatable-resources ?veg)
     (part-of ?woody-roots ?veg)
    )
);defModelFragment

;-------------------------------------------------------------------------
; If interested in the growth of a vegetation, and assuming that a component
; of that vegetation is not to be partitioned further (i.e. it is model
; aggregate), then a stock should be created for that component
;-------------------------------------------------------------------------
(defModelFragment growth-phenomenon
  :source-participants
    ((?vegetation :type vegetation)
     (?component :type vegetation-live-component)
    )
  :structural-conditions
```

```
    ((part-of ?component ?vegetation))
  :assumptions
    ((relevant growth ?vegetation)
     (model ?component aggregated)
    )
  :target-participants
    ((?stock :type stock :name component-size))
  :postconditions
    ((size-of ?stock ?component))
);defModelFragment

;-----------------------------------------------------------------------------
; The sizes of the aggregation of two components equals the sum of the
; sizes of the components.
; (If I have some time, I would like to implement a wildcard option that could
; match zero, one or multiple arguments in a partition.  That way,
; (partitioning $? ?child $?) ($? being the wildcard) would match every single
; child.  Wildcards would enable the modelling of a partitioning into any
; number of components).
;-----------------------------------------------------------------------------
(defModelFragment size-aggregation
  :source-participants
    ((?parent :type vegetation-live-component)
     (?child-1 :type vegetation-live-component)
     (?child-2 :type vegetation-live-component)
     (?size-1 :type variable)
     (?size-2 :type variable)
    )
  :structural-conditions
    ((partitioning ?parent ?child-1 ?child-2)
     (size-of ?size-1 ?child-1)
     (size-of ?size-2 ?child-2)
    )
  :target-participants
    ((?parent-size :type variable :name aggregate-size))
  :postconditions
    ((== ?parent-size (+ ?size-1 ?size-2))
     (size-of ?parent-size ?parent)
    )
);defModelFragment

;-----------------------------------------------------------------------------
; If a vegetation is partitioned into two component vegetations, and a variable
; is available that represents the growth of the aggregate vegetation, a
; partitioning coefficient enables dividing the aggregate growth between the
; component vegetations.
;-----------------------------------------------------------------------------
(defModelFragment partitioning-growth
  :source-participants
    ((?biomass :type vegetation-live-component)
     (?growth :type variable)
     (?sub-biomass-1 :type vegetation-live-component)
     (?sub-biomass-2 :type vegetation-live-component)
    )
  :structural-conditions
    ((growth-of ?growth ?biomass)
     (partitioning ?biomass ?sub-biomass-1 ?sub-biomass-2)
    )
  :target-participants
    ((?sub-growth-1 :type variable :name growth)
     (?sub-growth-2 :type variable :name growth)
     (?partitioning-coeff :type pot-seasonal-parameter :name part-coeff)
    )
  :postconditions
    ((partitioning-coeff ?partitioning-coeff
```

```
                          ?biomass ?sub-growth-1 ?sub-growth-2
      )
      (== ?sub-growth-1
          (* ?growth (- 1 ?partitioning-coeff))
      )
      (== ?sub-growth-2
          (* ?growth ?partitioning-coeff)
      )
      (growth-of ?sub-growth-1 ?sub-biomass-1)
      (growth-of ?sub-growth-2 ?sub-biomass-2)
    )
);defModelFragment

;-------------------------------------------------------------------------------
; The seasonal-proportional-mismatch approach is currently specific to the
; partitioning into above ground and below ground partitions (I need to consult
; an ecologist to check whether this can be extended to the other
; partitionings).  The partitioning coefficient is computed based on the
; mismatch between a seasonally optimal proportional partitioning and the
; observed actual proportional partitioning.
;-------------------------------------------------------------------------------
(defModelFragment partitioning-coefficient-by-proportional-mismatch
  :source-participants
    ((?vegetation :type vegetation)
     (?above-ground-veg :type vegetation-live-component)
     (?below-ground-veg :type vegetation-live-component)
     (?above-ground-biomass :type variable)
     (?below-ground-biomass :type variable)
     (?above-ground-growth :type variable)
     (?below-ground-growth :type variable)
     (?partitioning-coeff :type pot-seasonal-parameter)
    )
  :structural-conditions
    ((partitioning-coeff ?partitioning-coeff
                         ?vegetation ?above-ground-growth ?below-ground-growth
     )
     (growth-of ?above-ground-growth ?above-ground-veg)
     (above-ground-vegetation ?above-ground-veg)
     (growth-of ?below-ground-growth ?below-ground-veg)
     (below-ground-vegetation ?below-ground-veg)
     (size-of ?above-ground-biomass ?above-ground-veg)
     (size-of ?below-ground-biomass ?below-ground-veg)
    )
  :assumptions
    ((model ?partitioning-coeff proportional-mismatch))
  :target-participants
    ((?actual-root-proportion :type variable :name actual-root-prop)
     (?optimal-root-proportion :type pot-seasonal-parameter :name optimal-root-prop)
     (?mismatch :type variable :name mismatch)
    )
  :postconditions
    ((== ?actual-root-proportion?
         (/ ?below-ground-biomass (+ ?above-ground-biomass ?below-ground-biomass))
     )
     (optimal-root-proportion ?optimal-root-proportion)
     (== ?mismatch
         (if (> ?optimal-root-proportion 0)
           (/ ?actual-root-proportion ?optimal-root-proportion)
           infty
         );if
     )
     (== ?partitioning-coeff
         (if (>= ?mismatch 1)
           (* ?optimal-root-proportion
              (/ 1 ?mismatch)
```

```
            )
            (+ ?optimal-root-proportion
              (* (- 1 optimal-root-proportion)
                 (- 1 mismatch)
              )
            )
          );if
      )
    )
);defModelFragment

;-----------------------------------------------------------------------------
; Model fragment about the relocation flow from greens to relocatable resources.
;-----------------------------------------------------------------------------
(defModelFragment green-to-roots-relocation
  :source-participants
    ((?vegetation :type vegetation)
     (?green :type vegetation-live-component)
     (?relocatable-resources :type vegetation-live-component)
     (?green-stock :type stock)
     (?rr-stock :type stock)
    )
  :structural-conditions
    ((green ?green)
     (part-of ?green ?vegetation)
     (relocatable-resources ?relocatable-resources)
     (part-of ?relocatable-resources ?vegetation)
     (size-of ?green-stock ?green)
     (size-of ?rr-stock ?relocatable-resources)
    )
  :assumptions
    ((relevant relocation ?green ?relocatable-resources))
  :target-participants
    ((?green-roots-flow :type flow :name green-roots-flow))
  :postconditions
    ((green-roots-flow ?green-roots-flow)
     (flow ?green-roots-flow ?green-stock ?rr-stock)
    )
);defModelFragment

(defModelFragment proportionate-green-to-roots-relocation
  :source-participants
    ((?green :type vegetation-live-component)
     (?relocatable-resources :type vegetation-live-component)
     (?green-stock :type stock)
     (?rr-stock :type stock)
     (?green-roots-flow :type flow)
    )
  :structural-conditions
    ((green-roots-flow ?green-roots-flow)
     (flow ?green-roots-flow ?green-stock ?rr-stock)
     (size-of ?green-stock ?green)
     (green ?green)
     (size-of ?rr-stock ?relocatable-resources)
     (relocatable-resources ?relocatable-resources)
    )
  :assumptions
    ((model ?green-roots-flow proportionate-relocation))
  :target-participants
    ((?relocation-proportion :type pot-seasonal-parameter :name relocation-prop))
  :postconditions
    ((== ?green-roots-flow
         (* ?relocation-proportion ?green-stock)
     )
     (relocation-proportion ?relocation-proportion)
```

```
    )
);defModelFragment

;-------------------------------------------------------------------------------
; Model fragment about the relocation flow from relocatable resources to greens.
;-------------------------------------------------------------------------------
(defModelFragment root-to-green-relocation
  :source-participants
    ((?vegetation :type vegetation)
     (?green :type vegetation-live-component)
     (?relocatable-resources :type vegetation-live-component)
     (?green-stock :type stock)
     (?rr-stock :type stock)
    )
  :structural-conditions
    ((green ?green)
     (relocatable-resources ?relocatable-resources)
     (part-of ?green ?vegetation)
     (part-of ?relocatable-resources ?vegetation)
     (size-of ?green-stock ?green)
     (size-of ?rr-stock ?relocatable-resources)
    )
  :assumptions
    ((relevant relocation ?relocatable-resources ?green))
  :target-participants
    ((?roots-green-flow :type flow :name roots-green-flow))
  :postconditions
    ((roots-green-flow ?roots-green-flow)
     (flow ?roots-green-flow ?rr-stock ?green-stock)
    )
);defModelFragment

;-------------------------------------------------------------------------------
; Model fragment for a proportionate roots to green flow.
;-------------------------------------------------------------------------------
(defModelFragment proportionate-roots-green-flow
  :source-participants
    ((?vegetation :type vegetation)
     (?green :type vegetation-live-component)
     (?relocatable-resources :type vegetation-live-component)
     (?woody-roots :type vegetation-live-component)
     (?green-stock :type stock)
     (?rr-stock :type stock)
     (?wr-stock :type variable)
     (?roots-green-flow :type flow)
    )
  :structural-conditions
    ((roots-green-flow ?roots-green-flow)
     (flow ?roots-green-flow ?rr-stock ?green-stock)
     (size-of ?green-stock ?green)
     (green ?green)
     (part-of ?green ?vegetation)
     (size-of ?rr-stock ?relocatable-resources)
     (relocatable-resources ?relocatable-resources)
     (part-of ?relocatable-resources ?vegetation)
     (part-of ?woody-roots ?vegetation)
     (woody-roots ?woody-roots)
     (size-of ?wr-stock ?woody-roots)
    )
  :assumptions
    ((model ?roots-green-flow proportionate-relocation))
  :target-participants
    ((?relocation-proportion :type pot-seasonal-parameter :name relocation-prop)
     (?green-roots-ratio :type pot-seasonal-parameter :name green-roots-ratio)
    )
```

```
  :postconditions
    ((== ?roots-green-flow
         (if (> ?relocation-proportion 0)
           (* ?relocation-proportion ?rr-stock)
           (if (and (= ?relocation-proportion 0)
                    (> ?green-stock 0)
                    (> ?rr-stock 0)
                    (< ?green-roots-ratio
                       (/ ?green-stock (+ ?rr-stock ?wr-stock))
                    )
               )
             (* 0.8 ?green-stock)
           );if
         );if
     )
    )
);defModelFragment

;-------------------------------------------------------------------------------
; Model fragment that introduces the actual growth variable.  Other model
; fragments may be triggered by this variable to construct an explanation for
; actual growth.
;-------------------------------------------------------------------------------
(defModelFragment introduce-growth-variable
  :source-participants
    ((?veg :type vegetation))
  :assumptions
    ((relevant growth ?veg))
  :target-participants
    ((?actual-gr :type variable :name actual-gr))
  :postconditions
    ((growth-of ?actual-gr ?veg))
);defModelFrament

;-------------------------------------------------------------------------------
; If a variable that describes the growth of a vegetation and a stock expressing
; the size of that vegetation exist, then a flow, equal to the growth and into
; the stock can be created.
;-------------------------------------------------------------------------------
(defModelFragment creation-flow
  :source-participants
    ((?vegetation :type vegetation-live-component)
     (?stock :type stock)
     (?growth :type variable)
    )
  :structural-conditions
    ((size-of ?stock ?vegetation)
     (growth-of ?growth ?vegetation)
    )
  :target-participants
    ((?flow :type flow :name growth-flow))
  :postconditions
    ((flow ?flow source ?stock)
     (== ?flow ?growth)
    )
);defModelFragment

;-------------------------------------------------------------------------------
; If interested in how a vegetation dies, a mortality event switch is needed.
;-------------------------------------------------------------------------------
(defModelFragment mortality-switch
  :source-participants
    ((?vegetation :type vegetation))
  :assumptions
    ((relevant death ?vegetation))
```

```
  :purpose-required
    ((endogenous ?mortality-event)
     (explanatory ?mortality-event)
    )
  :target-participants
    ((?mortality-event :type event-switch :name mortality))
  :postconditions
    ((mortality-event-of ?mortality-event ?vegetation)
     (== ?mortality-event (C-else 0))
    )
);defModelFragment


;-------------------------------------------------------------------------------
; If the mortality event and the lifeform feature are included in a model with
; respect to a vegetation, then the annual death of certain species should be
; included as a cause of the mortality event.
;-------------------------------------------------------------------------------
(defModelFragment mortality-event-1
  :source-participants
    ((?vegetation :type vegetation)
     (?mortality-event :type event-switch)
     (?lifeform :type variable)
     (?season :type variable)
    )
  :structural-conditions
    ((mortality-event-of ?mortality-event ?vegetation)
     (lifeform-of ?lifeform ?vegetation)
     (season ?season)
    )
  :target-participants
    ((?death-season :type variable :name death-season))
  :postconditions
    ((== ?mortality-event
         (C-if (= ?season ?death-season)
           1
         );C-if
    )
    )
);defModelFragment

;-------------------------------------------------------------------------------
; If there is a mortality event, and variables that represent total above
; ground biomass and total relocatable resources are available, then a
; mortality event must be added for the case where available relocatable
; resources are insufficient to sustain the amount of above ground biomass.
;-------------------------------------------------------------------------------
(defModelFragment mortality-event-2
  :source-participants
    ((?vegetation :type vegetation)
     (?mortality-event :type event-switch)
     (?above-ground-vegetation :type vegetation-live-component)
     (?relocatable-reserves :type vegetation-live-component)
     (?above-ground-biomass :type variable)
     (?rr-size :type variable)
    )
  :structural-conditions
    ((mortality-event-of ?mortality-event ?vegetation)
     (above-ground-vegetation ?above-ground-vegetation)
     (part-of ?above-ground-vegetation ?vegetation)
     (size-of ?above-ground-biomass ?above-ground-vegetation)
     (relocatable-resources ?relocatable-reserves)
     (part-of ?relocatable-reserves ?vegetation)
     (size-of ?rr-size ?relocatable-reserves)
    )
```

```
  :target-participants
    ((?reserves-threshold :type pot-seasonal-parameter :name reserves-threshold))
  :postconditions
    ((reserves-threshold ?reserves-threshold)
     (== ?mortality-event
         (C-if (and (> ?above-ground-biomass 0)
                    (< (/ ?relocatable-reserves ?above-ground-biomass)
                       ?reserves-threshold
                    );<
               );and
           1
         );C-if
     )
    )
);defModelFragment

;-------------------------------------------------------------------------------
; If the mortality event and the cumulative growth are included in a model with
; respect to a vegetation, then not attaining the minimal required growth should
; be included as a cause of the mortality event.
;-------------------------------------------------------------------------------
(defModelFragment mortality-event-3
  :source-participants
    ((?vegetation :type vegetation)
     (?mortality-event :type event-switch)
     (?cumulative-growth :type variable)
    )
  :structural-conditions
    ((mortality-event-of ?mortality-event ?vegetation)
     (cumulative-growth-of ?cumulative-growth ?vegetation)
    )
  :target-participants
    ((?minimal-growth-requirement :type variable :name min-growth-req))
  :postconditions
    ((== ?mortality-event
         (C-if (< ?cumulative-growth ?minimal-growth-requirement) 1)
     )
    )
);defModelFragment

;-------------------------------------------------------------------------------
; If the mortality event and the drought effect are included in a model with
; respect to a vegetation, then the drought effect should be included as a
; cause of the mortality event.
;-------------------------------------------------------------------------------
(defModelFragment mortality-event-4
  :source-participants
    ((?vegetation :type vegetation)
     (?mortality-event :type event-switch)
     (?drought-effect :type variable)
    )
  :structural-conditions
    ((mortality-event-of ?mortality-event ?vegetation)
     (drought-effect-of ?drought-effect ?vegetation)
    )
  :target-participants
    ((?dessication :type pot-seasonal-parameter :name dessication))
  :postconditions
    ((dessication ?dessication-data)
     (== ?mortality-event
         (C-if (>= ?drought-effect ?dessication)
           1
         );C-if
     )
    )
```

```
);defModelFragment

;-----------------------------------------------------------------------------
; If the mortality event and death by fire are included in a model with
; respect to a vegetation, then the death by fire should be included as a
; cause of the mortality event.
;-----------------------------------------------------------------------------
(defModelFragment mortality-event-5
  :source-participants
    ((?vegetation :type vegetation)
     (?mortality-event :type event-switch)
     (?fire-death :type variable)
    )
  :structural-conditions
    ((mortality-event-of ?mortality-event ?vegetation)
     (fire-death-of ?fire-death ?vegetation)
    )
  :postconditions
    ((== ?mortality-event (C-if (= ?fire-death 1) 1)))
);defModelFragment

;-----------------------------------------------------------------------------
; Dying woody vegetation, if considered, can be modelled in two ways:
; - By a flow of disappearing live woody vegetation,
; - By a flow from live woody vegetation to dead wood.
;-----------------------------------------------------------------------------
(defModelFragment dying-wood
  :source-participants
    ((?vegetation :type vegetation)
     (?wood :type vegetation-live-component)
     (?stock :type stock)
    )
  :structural-conditions
    ((wood ?wood)
     (part-of ?wood ?vegetation)
     (size-of ?stock ?wood)
    )
  :assumptions
    ((relevant death ?vegetation))
  :target-participants
    ((?dead-wood :type vegetation-dead-component :name dead-wood))
  :postconditions
    ((dead-incarnation-of ?dead-wood ?wood)
     (part-of ?dead-wood ?vegetation)
    )
);defModelFragment

(defModelFragment dying-wood-without-storage
  :source-participants
    ((?wood :type vegetation-live-component)
     (?dead-wood :type vegetation-dead-component)
     (?stock :type stock)
    )
  :structural-conditions
    ((dead-incarnation-of ?dead-wood ?wood)
     (wood ?wood)
     (size-of ?stock ?wood)
    )
  :assumptions
    ((model ?dead-wood as-removal))
  :target-participants
    ((?flow :type flow :name dying-wood))
  :postconditions
    ((death-of ?flow ?wood)
     (flow ?flow ?stock sink)
```

```
    )
);defModelFragment

(defModelFragment dying-wood-with-storage
  :source-participants
    ((?wood :type vegetation-live-component)
     (?dead-wood :type vegetation-dead-component)
     (?live-stock :type stock)
    )
  :structural-conditions
    ((dead-incarnation-of ?dead-wood ?wood)
     (wood ?wood)
     (size-of ?live-stock ?wood)
    )
  :assumptions
    ((model ?dead-wood as-new-incarnation))
  :target-participants
    ((?flow :type flow :name dead-wood-in)
     (?dead-stock :type stock :name dead-wood-size)
    )
  :postconditions
    ((death-of ?flow ?wood)
     (growth-of ?flow ?dead-wood)
     (flow ?flow ?live-stock ?dead-stock)
     (size-of ?dead-stock ?dead-wood)
    )
);defModelFragment

;-------------------------------------------------------------------------------
; Dying green vegetation, i.e. littering, can be modelled in two ways:
; - By a flow of disappearing live green vegetation,
; - By a flow from live green vegetation to litter.
;-------------------------------------------------------------------------------
(defModelFragment littering
  :source-participants
    ((?vegetation :type vegetation)
     (?green :type vegetation-live-component)
     (?stock :type stock)
    )
  :structural-conditions
    ((green ?green)
     (part-of ?green ?vegetation)
     (size-of ?stock ?green)
    )
  :assumptions
    ((relevant death ?vegetation))
  :target-participants
    ((?litter :type vegetation-dead-component :name litter))
  :postconditions
    ((dead-incarnation-of ?litter ?green)
     (part-of ?litter ?vegetation)
    )
);defModelFragment

(defModelFragment littering-without-storage
  :source-participants
    ((?green :type vegetation-live-component)
     (?litter :type vegetation-dead-component)
     (?stock :type stock)
    )
  :structural-conditions
    ((dead-incarnation-of ?litter ?green)
     (green ?green)
     (size-of ?stock ?green)
    )
```

```
  :assumptions
    ((model ?litter as-removal))
  :target-participants
    ((?flow :type flow :name littering))
  :postconditions
    ((death-of ?flow ?green)
     (flow ?flow ?stock sink)
    )
);defModelFragment

(defModelFragment littering-with-storage
  :source-participants
    ((?green :type vegetation-live-component)
     (?litter :type vegetation-dead-component)
     (?live-stock :type stock)
    )
  :structural-conditions
    ((dead-incarnation-of ?litter ?green)
     (green ?green)
     (size-of ?live-stock ?green)
    )
  :assumptions
    ((model ?litter as-new-incarnation))
  :target-participants
    ((?flow :type flow :name littering)
     (?dead-stock :type stock :name litter-size)
    )
  :postconditions
    ((death-of ?flow ?green)
     (growth-of ?flow ?litter)
     (flow ?flow ?live-stock ?dead-stock)
     (size-of ?dead-stock ?litter)
    )
);defModelFragment

;-------------------------------------------------------------------------------
; Explain live-green death flow by seasonal effect.  If alternative ways of
; modelling this are available, a model assumption must be added.
;-------------------------------------------------------------------------------
(defModelFragment include-seasonal-green-death
  :source-participants
    ((?green :type vegetation-live-component)
     (?stock :type stock)
     (?flow :type flow)
     (?season :type variable)
     (?season-number :type integer)
    )
  :structural-conditions
    ((green ?green)
     (size-of ?stock ?green)
     (death-of ?flow ?green)
     (season ?season)
     (season-number ?season-number)
    )
  :target-participants
    ((?seasonal-effect :type (array (1 ?season-number) variable)))
  :postconditions
    ((== ?flow
         (C-if (> (?seasonal-effect ?season) 0)
           (* (?seasonal-effect ?season) ?stock)
           0
           :priority 20
         );C-if
     )
    )
```

```
);defModelFragment

(defModelFragment include-seasonal-green-death
  :source-participants
    ((?vegetation :type vegetation)
     (?green :type vegetation-live-component)
     (?stock :type stock)
     (?flow :type flow)
     (?frost-effect :type variable)
    )
  :structural-conditions
    ((frost-effect-of ?frost-effect ?vegetation)
     (green ?green)
     (part-of ?green ?vegetation)
     (size-of ?stock ?green)
     (death-of ?flow ?green)
    )
  :postconditions
    ((== ?flow
         (C-if (> ?frost-effect 0)
           (* (?seasonal-effect ?season) ?stock)
           :priority 10
         );C-if
     )
    )
);defModelFragment

;-----------------------------------------------------------------------------
; If vegetation-death is relevant, and relocatable resources are modelled,
; add flow from the relocatable-resources stock.
;-----------------------------------------------------------------------------
(defModelFragment dying-woody-roots
  :source-participants
    ((?vegetation :type vegetation)
     (?roots :type vegetation-live-component)
     (?stock :type stock)
    )
  :structural-conditions
    ((relocatable-resources ?roots)
     (part-of ?roots ?vegetation)
     (size-of ?stock ?roots)
    )
  :assumptions
    ((relevant death ?vegetation))
  :target-participants
    ((?flow :type flow :name resources-out))
  :postconditions
    ((death-of ?flow ?roots)
     (flow ?flow ?stock sink)
    )
);defModelFragment

;-----------------------------------------------------------------------------
; Explain relocatable-resources death flow by seasonal effect.  If alternative
; ways of modelling this are available, a model assumption must be added.
;-----------------------------------------------------------------------------
(defModelFragment include-seasonal-wood-death
  :source-participants
    ((?roots :type vegetation-live-component)
     (?stock :type stock)
     (?flow :type flow)
     (?season :type variable)
     (?season-number :type integer)
    )
  :structural-conditions
```

```
    ((relocatable-resources ?roots)
     (size-of ?stock ?roots)
     (death-of ?flow ?roots)
     (season ?season)
     (season-number ?season-number)
    )
  :target-participants
    ((?seasonal-effect :type (array (1 ?season-number) variable) :name param-table))
  :postconditions
    ((== ?flow
         (C-if (> (?seasonal-effect ?season) 0)
           (* (?seasonal-effect ?season) ?stock)
           0
           :priority 10
         );C-if
    )
    )
);defModelFragment

;-----------------------------------------------------------------------------
; If vegetation-death is relevant, and woody-roots are modelled, add flow
; from the woody-roots stock.
;-----------------------------------------------------------------------------
(defModelFragment dying-woody-roots
  :source-participants
    ((?vegetation :type vegetation)
     (?roots :type vegetation-live-component)
     (?stock :type stock)
    )
  :structural-conditions
    ((woody-roots ?roots)
     (part-of ?roots ?vegetation)
     (size-of ?stock ?roots)
    )
  :assumptions
    ((relevant death ?vegetation))
  :target-participants
    ((?flow :type flow :name woody-roots-out))
  :postconditions
    ((death-of ?flow ?roots)
     (flow ?flow ?stock sink)
    )
);defModelFragment

;-----------------------------------------------------------------------------
; Explain woody-roots death flow by seasonal effect.  If alternative ways of
; modelling this are available, a model assumption must be added.
;-----------------------------------------------------------------------------
(defModelFragment include-seasonal-wood-death
  :source-participants
    ((?roots :type vegetation-live-component)
     (?stock :type stock)
     (?flow :type flow)
     (?season :type variable)
     (?season-number :type integer)
    )
  :structural-conditions
    ((woody-roots ?roots)
     (size-of ?stock ?roots)
     (death-of ?flow ?roots)
     (season ?season)
     (season-number ?season-number)
    )
  :target-participants
    ((?seasonal-effect :type (array (1 ?season-number) variable) :name param-table))
```

```
  :postconditions
    ((== ?flow
         (C-if (> (?seasonal-effect ?season) 0)
           (* (?seasonal-effect ?season) ?stock)
           0
           :priority 10
         );C-if
     )
    )
);defModelFragment

;-----------------------------------------------------------------------------
; Model fragment that introduces of cumulative growth of greens.  I am assuming
; that this concept only applies to greens for the moment.
;-----------------------------------------------------------------------------
(defModelFragment cumulative-green-growth
  :source-participants
    ((?vegetation :type vegetation)
     (?green :type vegetation-live-component)
     (?green-growth :type variable)
     (?season :type variable)
     (?season-number :type integer)
    )
  :structural-conditions
    ((part-of ?green ?vegetation)
     (growth-of ?green-growth ?green)
     (green ?green)
     (season ?season)
     (season-number ?season-number)
    )
  :assumptions
    ((relevant cumulative-growth ?vegetation))
  :target-participants
    ((?cumulative-growth :type stock :name cumulative-growth)
     (?seasonal-growth :type flow :name shoots-in)
     (?annual-stock-disposal :type flow :name annual-disposal)
    )
  :postconditions
    ((cumulative-growth-of ?cumulative-growth ?vegetation)
     (flow ?seasonal-growth source ?cumulative-growth)
     (flow ?annual-stock-disposal ?cumulative-growth sink)
     (== ?seasonal-growth ?green-growth)
     (== ?annual-stock-disposal
         (if (= ?season ?season-number) ?cumulative-growth)
     )
    )
);defModelFragment

;-----------------------------------------------------------------------------
; Model fragment to create handle for death by fire
;
; IMPORTANT: If the fire phenomenon is added in more detail, it will be better
; to introduce the fire phenomenon upstream in the inference chain where the
; causes of fire are introduced.  The variables and relations introduced by
; this model fragment then become logical consequences of the existence of other
; variables and relations related to fire.
;-----------------------------------------------------------------------------
(defModelFragment create-fire-effect
  :source-participants
    ((?vegetation :type vegetation))
  :assumptions
    ((relevant fire ?vegetation))
  :target-participants
    ((?fire-death :type variable :name fire-death)
     (?fire-effect :type variable :name fire-effect)
```

```
     (?fire-threshold :type variable :name fire-threshold)
    )
  :postconditions
    ((if (> ?fire-effect ?fire-threshold)
       1
       ?fire-effect
     );if
     (fire-effect-of ?fire-effect ?vegetation)
     (fire-death-of ?fire-death ?vegetation)
    )
);defModelFragment

;-----------------------------------------------------------------------------
; If the fire-death effect is modelled for some vegetation, then certain
; components of that vegetation are deemed burnable.
;-----------------------------------------------------------------------------
(defModelFragment green-is-burnable
  :source-participants
    ((?vegetation :type vegetation)
     (?fire-death :type variable)
     (?green :type vegetation-live-component)
    )
  :structural-conditions
    ((fire-death-of ?fire-death ?vegetation)
     (part-of ?green ?vegation)
     (green ?green)
    )
  :postconditions
    ((burnable ?green))
);defModelFragment

(defModelFragment wood-is-burnable
  :source-participants
    ((?vegetation :type vegetation)
     (?fire-death :type variable)
     (?wood :type vegetation-live-component)
    )
  :structural-conditions
    ((fire-death-of ?fire-death ?vegetation)
     (part-of ?wood ?vegation)
     (wood ?wood)
    )
  :postconditions
    ((burnable ?wood))
);defModelFragment

(defModelFragment dead-wood-is-burnable
  :source-participants
    ((?vegetation :type vegetation)
     (?fire-death :type variable)
     (?live-inc :type vegetation-live-component)
     (?dead-inc :type vegetation-dead-component)
    )
  :structural-conditions
    ((fire-death-of ?fire-death ?vegetation)
     (part-of ?live-inc ?vegetation)
     (dead-incarnation-of ?dead-inc ?live-inc)
    )
  :postconditions
    ((burnable ?dead-inc))
);defModelFragment

;-----------------------------------------------------------------------------
; If a component of a vegetation is burnable, a variable representing the
; fire-death effect for that vegetation is available and a stock representing
```

```
; the size of the component-vegetation is avaible, then losses due to fire
; are modelled as follows:
; - A flow is added from the stock into a sink to represent the losses
; - Losses a proportional to the stock size.  The proportion lost is expressed
;   by ?proportional-losses
;-----------------------------------------------------------------------------
(defModelFragment fire-losses
  :source-participants
    ((?vegetation :type vegetation)
     (?component :type vegetation-component)
     (?stock :type stock)
     (?fire-death :type variable)
    )
  :structural-conditions
    ((part-of ?component ?vegetation)
     (size-of ?stock ?component)
     (burnable ?component)
     (fire-death-of ?fire-death ?vegetation)
    )
  :target-participants
    ((?fire-loss-flow :type flow :name fire-losses)
     (?logistic-Xo-data :type variable :name log-X0-data)
     (?logistic-b-data :type variable :name log-b-data)
     (?proportional-losses :type variable :name prop-fire-losses)
    )
  :postconditions
    ((flow ?fire-loss-flow ?stock sink)
     (== ?fire-loss-flow (* ?stock ?proportional-losses))
     (== ?proportional-losses
         (^ (/ 1 (+ 1 (/ ?fire-death ?logistic-Xo-data)))
            ?logistic-b-data
         )
     )
    )
);defModelFragment


;-----------------------------------------------------------------------------
; Modelling actual-growth when it is considered dependent on the season (i.e.
; growth does not necessarily occurs in every season) and on the environment
; (i.e. other variables that set the environment in the model must be taken into
; account).
;-----------------------------------------------------------------------------
(defModelFragment season-and-environment-dependent-actual-GR
  :source-participants
    ((?vegetation :type vegetation)
     (?actual-gr :type variable)
     (?season :type variable)
     (?season-number :type integer)
    )
  :structural-conditions
    ((growth-of ?actual-gr ?vegetation)
     (season ?season)
     (season-number ?season-number)
    )
  :assumptions
    ((model ?actual-gr season-and-environment-dependent-growth))
  :target-participants
    ((?gr-season-table :type (array (1 ?season-number) variable) :name gr-season-table)
     (?gr-season :type variable :name gr-season)
     (?quantum-effect-table :type (array (1 ?season-number) variable) :name quantum-effect-table)
     (?max-quantum-effect :type variable :name max-quantum-effect)
     (?species-APAR :type variable :name species-APAR)
     (?growth-environment :type variable :name growth-env)
    )
```

```
  :postconditions
    ((max-quantum-effect-of ?max-quantum-effect ?vegetation)
     (species-APAR-of ?species-APAR ?vegetation)
     (growth-environment-of ?growth-environment ?vegetation)
     (== ?gr-season (?gr-season-table ?season))
     (== ?max-quantum-effect (?quantum-effect-table ?season))
     (== ?actual-gr
         (if (and (= ?gr-season 1)
                  (= ?growth-environment 1)
             );and
           (* ?max-quantum-effect ?species-APAR (- 1 ?growth-environment))
         )
     )
    )
);defModelFragment

;-------------------------------------------------------------------------------
; If a growth environment and a specific above ground vegetation distinction are
; part of the model, then the proportion of above ground biomass of a
; sustainable maximum is added to the growth environment.
;-------------------------------------------------------------------------------
(defModelFragment fbiomass-effect-on-growth
  :source-participants
    ((?vegetation :type vegetation)
     (?above-ground-vegetation :type vegetation-live-component)
     (?above-ground-biomass :type variable)
     (?growth-environment :type variable)
    )
  :structural-conditions
    ((above-ground-vegetation ?above-ground-vegetation)
     (part-of ?above-ground-vegetation ?vegetation)
     (size-of ?above-ground-biomass ?above-ground-vegetation)
     (growth-environment-of ?growth-environment ?vegetation)
    )
  :target-participants
    ((?fbiomass :type variable :name fbiomass)
     (?max-biomass :type variable :name max-biomass)
    )
  :postconditions
    ((== ?growth-environment (C+ (- 1 ?fbiomass)))
     (== ?fbiomass
         (if (< ?above-ground-biomass ?max-biomass)
           (/ ?above-ground-biomass ?max-biomass)
           0 ;shouldn't this be 1???
         );if
     );==
    )
);defModelFragment

;-------------------------------------------------------------------------------
; If a growth environment and water storage are part of the model, then a check
; whether the available water is within given limits must be added to the growth
; environment
;-------------------------------------------------------------------------------
(defModelFragment moisture-effect-on-growth
  :source-participants
    ((?vegetation :type vegetation)
     (?growth-environment :type variable)
     (?water-supply :type variable)
     (?water-storage :type water-storage)
    )
  :structural-conditions
    ((growth-environment-of ?growth-environment ?vegetation)
     (size-of ?water-supply ?water-storage)
    )
```

```
  :target-participants
    ((?gr-moist-min :type variable :name gr-moist-min)
     (?gr-moist-max :type variable :name gr-moist-max)
     (?gr-moisture :type variable :name gr-moisture)
    )
  :postconditions
    ((== ?growth-environment (C+ ?gr-moisture))
     (== ?gr-moisture
         (cond
           ((< ?water-supply ?gr-moist-min) 0)
           ((> ?water-supply ?gr-moist-max) 1)
           (t (/ (- ?water-supply ?gr-moist-min)
                 (- ?gr-moist-max ?gr-moist-min)
              );
           )
         );cond
     )
    )
)

;-----------------------------------------------------------------------------
; If a growth environment and mean input temperature are part of the model, then
; the suitability of the environment must be modelled with respect to four given
; temperatures: two boundaries for the optimal temperature range and two
; boundaries for the required temperature range.
;-----------------------------------------------------------------------------
(defModelFragment temperature-effect-on-growth
  :source-participants
    ((?vegetation :type vegetation)
     (?growth-environment :type variable)
     (?temperature :type variable)
    )
  :structural-conditions
    ((growth-environment-of ?growth-environment ?vegetation)
     (mean-input-temperature ?temperature)
    )
  :target-participants
    ((?gr-temp-1 :type variable :name gr-temp-min)
     (?gr-temp-2 :type variable :name gr-temp-lo-mid)
     (?gr-temp-3 :type variable :name gr-temp-hi-mid)
     (?gr-temp-4 :type variable :name gr-temp-max)
     (?gr-temp :type variable :name gr-temp)
    )
  :postconditions
    ((== ?growth-environment (C+ ?gr-temp))
     (== ?gr-temp
         (cond
           ((< ?temperature ?gr-temp-1) 0)
           ((and (> ?gr-temp ?gr-temp-1)
                 (< ?gr-temp ?gr-temp-2)
            );and
            (/ (- ?temperature ?gr-temp-1)
               (- ?gr-temp-2 ?gr-temp-1)
            )
           )
           ((and (> ?gr-temp ?gr-temp-2)
                 (< ?temperature ?gr-temp-3)
            );and
            1
           )
           ((and (> ?temperature ?gr-temp-3)
                 (< ?temperature ?gr-temp-4)
            );and
            (/ (- ?gr-temp-4 ?temperature)
               (- ?gr-temp-4 ?gr-temp-3)
```

```
            )
           )
           ((> ?temperature ?gr-temp-4) 0)
         );cond
     )
    )
);defModelFragment

;-----------------------------------------------------------------------------
; If there is a water storage in the model, then the drought effect must be
; modelled.
;
; It is assumed herein that only one model of drought is available.  Should
; additional drought-effect models be added, then a seperate model fragment
; that creates the drought-effect variable is necessary and one model fragment
; per drought-effect model should then be added.
;
; Note: copied from ModMed n-species model n-spp2.sml and may still contain an
; error.
;-----------------------------------------------------------------------------
(defModelFragment drought-effect
  :source-participants
    ((?vegetation :type vegetation)
     (?water-storage :type water-storage)
     (?water-supply :type variable)
    )
  :structural-conditions
    ((size-of ?water-supply ?water-storage))
  :target-participants
    ((?drought-effect :type variable :name drought-effect)
     (?drought-min :type variable :name drought-min)
     (?drought-max :type variable :name drought-max)
    )
  :postconditions
    ((drought-effect-of ?drought-effect ?vegetation)
     (== ?drought-effect
         (if (< ?water-supply ?drought-min)
           0
           (if (> ?water-supply ?drought-max)
             1
             (/ (- ?water-supply ?drought-min)
                (- ?drought-max ?drought-min)
             )
           );if
         );if
     )
    )
);defModelFragment

;-----------------------------------------------------------------------------
; If there is a minimum-input-temperature in the model, then the frost effect
; must be modelled.
;
; It is assumed herein that only one model of frost is available.  Should
; additional frost-effect models be added, then a seperate model fragment
; that creates the frost-effect variable is necessary and one model fragment
; per frost-effect model should then be added.
;
; Note: corrected potential error in ModMed n-species model n-spp2.sml.
;-----------------------------------------------------------------------------
(defModelFragment frost-effect
  :source-participants
    ((?vegetation :type vegetation)
     (?minimum-input-temperature :type variable)
    )
```

```
  :structural-conditions
    ((minimum-input-temperature ?minimum-input-temperature))
  :target-participants
    ((?frost-effect :type variable :name frost-effect)
     (?frost-min :type variable :name frost-min)
     (?frost-max :type variable :name frost-max)
    )
  :postconditions
    ((frost-effect-of ?frost-effect ?vegetation)
     (== ?frost-effect
         (if (> ?minimum-input-temperature ?frost-min)
           0
           (if (< ?minimum-input-temperature ?frost-max)
             1
             (/ (- ?minimum-input-temperature ?frost-max)
                (- ?frost-min ?frost-max)
             )
           );if
         );if
     )
    )
);defModelFragment

;=============================================================================
; Model features external to the species model
;=============================================================================

;-----------------------------------------------------------------------------
; Landscape variables:
; - mean temperature
; - minimal temperature (to establish whether frost has occurred)
;-----------------------------------------------------------------------------
(defModelFragment mean-temperature
  :assumptions
    ((relevant mean-input-temperature))
  :target-participants
    ((?temperature :type variable :name mean-input-temperature))
  :postconditions
    ((mean-input-temperature ?temperature))
);defModelFragment

(defModelFragment min-temperature
  :assumptions
    ((relevant minimal-input-temperature))
  :target-participants
    ((?temperature :type variable :name min-input-temperature))
  :postconditions
    ((minimal-input-temperature ?temperature))
);defModelFragment

;-----------------------------------------------------------------------------
; Season model
;-----------------------------------------------------------------------------
(defModelFragment seasons
  :assumptions
    ((relevant seasons))
  :target-participants
    ((?season :type variable :name season)
     (?season-number :type integer :name season-number)
    )
  :postconditions
    ((season ?season)
     (season-number ?season-number)
    )
);defModelFragment
```

```
(defModelFragment temperate
  :source-participants
    ((?season :type variable)
     (?season-number :type integer)
    )
  :structural-conditions
    ((season ?season)
     (season-number ?season-number)
    )
  :assumptions
    ((model ?season temperate-model))
  :postconditions
    ((== ?season-number 4))
);defModelFragment

(defModelFragment tropical
  :source-participants
    ((?season :type variable)
     (?season-number :type integer)
    )
  :structural-conditions
    ((season ?season)
     (season-number ?season-number)
    )
  :assumptions
    ((model ?season tropical-model))
  :postconditions
    ((== ?season-number 2))
);defModelFragment

(defModelFragment monthly
  :source-participants
    ((?season :type variable)
     (?season-number :type integer)
    )
  :structural-conditions
    ((season ?season)
     (season-number ?season-number)
    )
  :assumptions
    ((model ?season monthly-model))
  :postconditions
    ((== ?season-number 12))
);defModelFragment

;-------------------------------------------------------------------------------
; Modelling potentially seasonal parameters as seasonally changing parameters
;-------------------------------------------------------------------------------
(defModelFragment seasonally-changing-parameters
  :source-participants
    ((?parameter :type pot-seasonal-parameter)
     (?season :type variable)
     (?season-number :type integer)
    )
  :structural-conditions
    ((season ?season)
     (season-number ?season-number)
    )
  :assumptions
    ((model ?parameter seasonal-table))
  :target-participants
    ((?data-table :type (array (1 ?season-number) variable) :name table))
  :postconditions
    ((== ?parameter (?data-table ?season)))
```

```
);defModelFragment

;-----------------------------------------------------------------------------
; Modelling potentially seasonal parameters as constants (not seasonally
; changing).  This could be useful if no seasonal data could be gathered.
;-----------------------------------------------------------------------------
(defModelFragment constant-seasonal-parameters
  :source-participants
    ((?parameter :type pot-seasonal-parameter))
  :assumptions
    ((model ?parameter constant))
  :target-participants
    ((?constant :type variable :name constant))
  :postconditions
    ((== ?parameter ?constant))
);defModelFragment

;-----------------------------------------------------------------------------
; Water storage model
;-----------------------------------------------------------------------------
(defModelFragment water-storage
  :assumptions
    ((relevant water-storage))
  :target-participants
    ((?water-storage :type water-storage :name water-storage))
);defModelFragment

(defModelFragment simplest-water-storage-model
  :source-participants
    ((?water-storage :type water-storage))
  :assumptions
    ((model ?water-storage single-variable))
  :target-participants
    ((?water-supply :type variable :name available-water))
  :postconditions
    ((size-of ?water-supply ?water-storage))
);defModelFragment

(defModelFragment default-water-storage
  :source-participants
    ((?water-storage :type water-storage))
  :assumptions
    ((model ?water-storage stock-flow-system))
  :target-participants
    ((?water-supply :type stock :name available-water)
     (?water-inputs :type flow :name water-inputs)
     (?drainage :type flow :name drainage)
    )
  :purpose-required
    ((endogenous ?water-inputs))
  :postconditions
    ((size-of ?water-supply ?water-storage)
     (flow ?water-inputs source ?water-supply)
     (input-flows-of ?water-inputs ?water-storage)
     (flow ?drainage ?water-supply sink)
     (drainage-of ?drainage ?water-storage)
    )
);defModelFragment

(defModelFragment rain
  :source-participants
    ((?water-storage :type water-storage)
     (?water-inputs :type variable)
   )
  :structural-conditions
```

```
    ((input-flows-of ?water-inputs ?water-storage))
  :assumptions
    ((relevant rain))
  :target-participants
    ((?rain :type parameter :name rain))
  :postconditions
    ((== ?water-inputs (C+ ?rain))
     (rain-flow ?rain)
    )
);defModelFragment

(defModelFragment overland-flow
  :source-participants
    ((?water-storage :type water-storage)
     (?water-inputs :type variable)
    )
  :structural-conditions
    ((input-flows-of ?water-inputs ?water-storage))
  :assumptions
    ((relevant overland-water-flow))
  :target-participants
    ((?overland-flow :type parameter :name overland-flow))
  :postconditions
    ((== ?water-inputs (C+ ?overland-flow))
     (overland-water-flow ?overland-flow)
    )
);defModelFragment

(defModelFragment underground-flow
  :source-participants
    ((?water-storage :type water-storage)
     (?water-inputs :type variable)
    )
  :structural-conditions
    ((input-flows-of ?water-inputs ?water-storage))
  :assumptions
    ((relevant underground-water-flow))
  :target-participants
    ((?underground-flow :type parameter :name underground-flow))
  :postconditions
    ((== ?water-inputs (C+ ?underground-flow))
     (underground-water-flow ?underground-flow)
    )
);defModelFragment

(defModelFragment proportional-drainage
  :source-participants
    ((?water-storage :type water-storage)
     (?water-supply :type variable)
     (?drainage :type variable)
    )
  :structural-conditions
    ((drainage-of ?drainage ?water-storage)
     (size-of ?water-supply ?water-storage)
    )
  :assumptions
    ((model ?drainage proportional))
  :target-participants
    ((?drainage-coeff :type parameter :name drainage-coeff))
  :postconditions
    ((== ?drainage (* ?drainage-coeff ?water-supply)))
);defModelFragment

(defModelFragment evapo-transpiration
  :source-participants
```

```
    ((?water-storage :type water-storage)
     (?water-supply :type stock)
    )
  :structural-conditions
    ((size-of ?water-supply ?water-storage))
  :assumptions
    ((relevant evapo-transpiration ?water-storage))
  :target-participants
    ((?evapo-transpiration :type flow :name evapo-transpiration))
  :postconditions
    ((flow ?evapo-transpiration ?water-supply sink)
     (evapo-transpiration-of ?evapo-transpiration ?water-storage)
    )
  :purpose-required
    ((endogenous ?evapo-transpiration))
);defModelFragment

(defModelFragment proportional-transpiration
  :source-participants
    ((?water-storage :type water-storage)
     (?evapo-transpiration :type variable)
     (?vegetation :type vegetation)
     (?green :type vegetation-component)
     (?green-biomass :type variable)
     (?mean-input-temperature :type variable)
    )
  :structural-conditions
    ((evapo-transpiration-of ?evapo-transpiration ?water-storage)
     (mean-input-temperature ?mean-input-temperature)
     (green ?green)
     (part-of ?green ?vegetation)
     (size-of ?green-biomass ?green)
    )
  :assumptions
    ((model ?evapo-transpiration temp-green-proportional))
  :target-participants
    ((?evapo-coeff :type parameter :name evapo-coeff))
  :postconditions
    ((== ?evapo-transpiration (C+ (* ?evapo-coeff ?green-biomass ?mean-input-temperature))))
);defModelFragment
```

Bibliography

[1] Addanki, S., Cremonini, R., and Penberthy, J.S. Graph of models. *Artificial Intelligence*, 51:145–177, 1991.

[2] S. Amarel. On representations of problems of reasoning about actions. *Machine Intelligence*, 3:131–171, 1968.

[3] J. Amsterdam. *Automated Qualitative Modeling of Dynamic Physical Systems*. Ph.D. dissertation, Massachusetts Institute of Technology, 1992.

[4] Bacchus, F. and van Beek, P. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 311–318, 1998.

[5] Baker, J. Adaptive selection methods for genetic algorithms. In Grefenstette, J.J., editor, *Proceedings of the First International Conference on Genetic Algorithms*. Erbaum, 1985.

[6] Baker, J. Reducing bias and inefficiency in the selection algorithm. In Grefenstette, J.J., editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*. Erbaum, 1987.

[7] Balzter, H. Markov chain models for vegetation dynamics. *Ecological Modelling*, 126:139–154, 2000.

[8] Barros, L.C., Bassanezi, R.C., and Tonelli, P.A. Fuzzy modelling in population dynamics. *Ecological modelling*, 128:27–33, 2000.

[9] Berjak, S.G. and Hearne, J.W. An improved cellular automaton model for simulatin fire in a spatially heterogeneous savanna system. *Ecological Modelling*, 148:133–151, 2002.

[10] Biris, E. and Shen, Q. Automatic modelling using bayesian networks for explanation generation. In *Proceedings of the 13th International Workshop on Qualitative Reasoning about Physical Systems*, pages 19–26, 1999.

[11] Bistarelli, S., Montanari, U., and Rossi, F. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[12] Biswas, G., Manganaris, S., and Yu, X.W. Extending component connection modeling for analyzing complex physical systems. *IEEE Expert*, 8(1):48–57, 1993.

[13] Bobrow, D., Falkenhainer, B., Farquhar, A., Fikes, R., Forbus, K.D., Gruber, T.R., Iwasaki, Y., and Kuipers, B.J. A compositional modeling language. In *Proceedings of the 10th International Workshop on Qualitative Reasoning about Physical Systems*, pages 12–21, 1996.

[14] Boutilier, C. Toward a logic for qualitative decision theory. In *Proceedings of the International Conference on Knowledge Representation and Reasoning*, pages 75–86, 1994.

[15] Bouwers, A. and Bredeweg, B. VisiGarp: Graphical representations of qualitative simulation models. In *Proceedings of the 15th International Workshop on Qualitative Reasoning about Physical Systems*, pages 142–149, 2001.

[16] Bradley, E. and Stolle, R. Automatic construction of accurate models of physical systems. *Annals of Mathematics and Artificial Intelligence*, 17:1–28, 1996.

[17] P.C. Breedveld. *Physical Systems Theory in Terms of Bond Graphs*. Ph.D. dissertation, Twente University, 1984.

[18] Breedveld, P.C. Multibond-graph elements in physical systems theory. *Journal of the Franklin Institute*, 319(1/2):1–36, 1985.

[19] Breunese, A., Top, J.L., Broenink, J.F., and Akkermans, H. Libraries of reusable models: Theory and application. *Simulation*, 71(1):7–22, 1998.

[20] Brewka, G. Logic programs with context-dependent preferences. In Dubois, D., Klement, E.P., and Prade, H., editors, *Fuzzy Sets, Logics and Reasoning about Knowledge*, pages 381–394. Kluwer Academic Publishers, 1999.

[21] Brewka, G., Benferhat, S., and Le Berre, D. Qualitative choice theory. In *to appear in Proceedings of the 8th International Conference on Knowledge Representation and Reasoning*, 2002.

[22] Brilhante, V. and Robertson, D. Metadata-supported automated ecological modelling. In Rautenstrauch, C. and Patig, S., editors, *Environmental Information Systems in Industry and Public Administration*. Idea Group Publishing, Hershey, PA, USA, 2001.

[23] Bunn, D.W and Larsen, E.R. Sensitivity of reserve margin to factors influencing investment in the UK electricity market. *Energy Policy*, 20(5):490–502, 1992.

[24] Bunn, D.W and Larsen, E.R. Assessment of the uncertainty in future UK electricity investment using an industry simulation model. *Utilities Policy*, 4(3):229–236, 1994.

[25] Chi, D. and Iwasaki, Y. Abstraction framework for compositional modeling. In *Proceedings of the International Workshop on Qualitative Reasoning about Physical Systems*, 1994.

[26] Clarke, K.C. and Gaydos, L.J. Loose-coupling a cellular automaton model and gis: long-term urban growth prediction for san francisco and washington/baltimore. *International Journal of Geographic Information Science*, 12(7):699–714, 1998.

[27] Coghill, G. and Shen, Q. On the specification of multiple models for diagnosis of dynamic systems. *AI Communications*, 14(2):93–104, 2001.

[28] Coghill, G.M. and Chantler, M.J. Constructive and non-constructive asynchronous qualitative simulation. In *Proceedings of the 13th International Workshop on Qualitative Reasoning about Physical Systems*, pages 51–61, 1999.

[29] Crawford, J., Farquhar, A., and Kuipers, B.J. QPC: A compiler from physical models into qualitative differential equations. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 365–372, San Mateo, CA, 1990. Morgan-Kaufmann.

[30] Dagorn, L., Menczer, F., Bach, P., and Olson, R.J. Co-evolution of movement behaviours by tropical pelagic predatory fishes in response to prey environment: a simulation model. *Ecological Modelling*, 134:325–341, 2000.

[31] P. Dague. Numeric reasoning with relative orders of magnitude. In *Proceedings of the National Conference on Artificial Intelligence*, pages 541–547, 1993.

[32] P. Dague. Symbolic reasoning with relative orders of magnitude. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1509–1514, 1993.

[33] Davis, R. and Hamscher, W. Model-based reasoning: troubleshooting. In Shrobe, H., editor, *Exploring Artificial Intelligence*, pages 297–346. Morgan-Kaufmann, 1988.

[34] de Kleer, J. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

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

[36] de Kleer, J. and Brown, J.S. A qualitative physics based on confluences. In Bobrow, D.G., editor, *Qualitative Reasoning about Physical Systems*, pages 7–84. The MIT Press, 1985.

[37] de Kleer, J. and Williams, B.C. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[38] Dechter, A. and Dechter, R. Removing redundancies in constraint networks. In *Proceedings of the National Conference on Artificial Intelligence*, pages 105–109, 1987.

[39] Dechter, R. and Dechter, A. Belief maintenance in dynamic constraint networks. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 37–42, 1988.

[40] Dechter, R. and Pearl, J. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989.

[41] Doman, A., Glucksman, M., Mass, N., and Sasportes, M. The dynamics of managing a life insurance company. In *Proceedings of the 1994 System Dynamics conference*, 1994.

[42] Dreyfus-Leon, M. and Kleiber, P. A spatial individual behaviour-based model approach of the yellowfin tuna fishery in the eastern pacific ocean. *Ecological Modelling*, 146:47–56, 2001.

[43] Dubois, D., Fargier, D., and Prade, H. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.

[44] J. Elster. *Solomonic Judgements: Studies in the Limitations of Rationality*. Cambridge University Press, New York, 1989.

[45] Falkenhainer, B. and Forbus, K.D. Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

[46] Farquhar, A. *Automated Modeling of Physical Systems in the Presence of Incomplete Knowledge*. Ph.D. dissertation, University of Texas at Austin, 1993.

[47] Farquhar, A. A qualitative physics compiler. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 1168–1174, 1994.

[48] Farquhar, A. and Brajnik, G. A semi-quantitative physics compiler. In *Proceedings of the 8th International Workshop on Qualitative Reasoning about Physical Systems*, pages 81–89, 1994.

[49] V.V. Fedorov. *Theory of Optimal Experiments*. Academic Press, 1972.

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

[51] Forbus, K.D. The qualitative process engine. In Weld, D. and de Kleer, J., editors, *Readings in Qualitative Reasoning about Physical Systems*. Morgan-Kaufmann, Los Altos, CA, 1990.

[52] Forbus, K.D. and Falkenhainer, B. Self-explanatory simulations: An integration of qualitative and quantitative knowledge. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 380–387, 1990.

[53] Forbus, K.D., Whalley, P.B., Everett, J.O., Ureel, L., Brokowski, M., Baher, J., and Kuehne, S.E. CyclePad: An articulate virtual laboratory for engineering thermodynamics. *Artificial Intelligence*, 114:297–347, 1999.

[54] Ford, A. *Modeling the Environment - An Introduction to System Dynamics Modeling of Environmental Systems*. Island Press, 1999.

[55] Ford, A., Bull, M., and Nail, R. Bonneville's conservation policy analysis models. *Energy Policy*, 15(2):109–124, 1987.

[56] C.L. Forgy. RETE: A fast algorithm for the many pattern/many object pattern matching problem. *Artificial Intelligence*, 19:17–37, 1982.

[57] J.W. Forrester. *Industrial Dynamics*. MIT Press, 1961.

[58] J.W. Forrester. *Principles of Systems*. Wright-Allen Press, Cambridge, MA, USA, 1968.

[59] J.W. Forrester. *Urban Dynamics*. MIT Press, 1968.

[60] J.W. Forrester. *World Dynamics*. MIT Press, 1971.

[61] E.C. Freuder. Partial constraint satisfaction. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 278–283, 1989.

[62] Wallace, R.J. Freuder, E.C. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[63] P. Gawthrop. *Metamodelling: Bond Graphs and Dynamic Systems*. Prentice Hall, 1996.

[64] Goldberg, D.E. A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 4:445–460, 1990.

[65] Goldberg, D.E. and Deb, K. A comparative analysis of selection schemes used in genetic algorithms. In Rawlings, G., editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991.

[66] P. Gottschalk. A system dynamics model for long range planning in a railroad. *European Journal of Operational Research*, 14(2):156–162, 1983.

[67] Green, D. and Shapiro, I. *Pathologies of Rational Choice Theory*. Yale University Press, New Haven, 1995.

[68] Green, D.G. and Klomp, N.I. Environmental informatics - a new paradigm for coping with complexity in nature. *Complexity International*, 6:36–44, 1998.

[69] Hamscher, W., Console, L., and de Kleer, J., editors. *Readings in Model-Based Diagnosis*. Morgan-Kaufmann, San Mateo, 1992.

[70] Haralick, R.M. and Elliot, G.L. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[71] Hargrove, W.W., Gardner, R.H., Turner, M.G., Romme, W.H., and Despain, D.G. Simulating fire patterns in heterogeneous landscapes. *Ecological Modelling*, 135:243–263, 2000.

[72] Hart, P.E., Nilsson, N.J., and Raphael, B. A formal basis for the heuristic determination of minimal cost paths. *IEEE Transactions on Systems, Science and Cybernetics*, SSC-4(2):100–107, 1968.

[73] Hart, P.E., Nilsson, N.J., and Raphael, B. Correction to "a formal basis for the heuristic determination of minimal cost paths". *SIGART Newsletter*, 37:28–29, December 1972.

[74] Hassan, M.F. Compositional model conversion. Project report, University of Edinburgh, 2001.

[75] Heller, U. and Struss, P. Qualitative modeling for environmental decision support. In *Proceedings of the 10th International Symposium "Informatics for Environmental Protection"*, pages 358–367, 1996.

[76] Heller, U. and Struss, P. Transformation of qualitative dynamic models - application in hydro-ecology. In *Proceedings of the 10th International Workshop on Qualitative Reasoning about Physical Systems*, pages 83–92, 1996.

[77] Heller, U. and Struss, P. Diagnosis and therapy recognition for ecosystems - usage of model-based diagnosis techniques. In *Proceedings of the 12th International Symposium "Computer Science for Environment Protection"*, 1998.

[78] Holland, J.H. *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[79] C.S. Holling. Some characteristics of simple types of predation and parasitism. *Canadian Entomologist*, 91:385–398, 1959.

[80] Holst, N., Axelsen, J.A., Olesen, J.E., and Ruggle, P. Object-oriented implementation of the metabolic pool model. *Ecological Modelling*, 104:175–187, 1997.

[81] Horowitz, E. and Sahni, S. *Fundamentals of Computer Algorithms*. Computer Science Press, Rockville, MD, 1978.

[82] Houghton, J.T., Ding, Y., Griggs, D.J., Noguer, M., van der Linden, P.J., and Xiaosu, D., editors. *Climate Change 2001: The Scientific Basis*. Cambridge University Press, 2001.

[83] N. Howard. *Paradoxes of Rationality: Theory of Metagames and Political Behaviour*. MIT Press, Cambridge, 1971.

[84] P.T. Ittig. A model for planning ambulatory health services. *Management Science*, 24(10):1001–1010, 1976.

[85] Iwasaki, Y., Farquhar, A., Saraswat, V., Bobrow, D., and Gupta, V. Modeling time in hybrid systems: How fast is "instantaneous"? In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1773–1780, 1995.

[86] Iwasaki, Y. and Levy, A.Y. Automated model selection for simulation. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 1183–1190, 1994.

[87] Iwasaki, Y. and Low, C.M. Device modeling environment: an integrated model-formulation and simulation environment for continuous and discrete phenomena. In *Proceedings of the Conference on Intelligent Systems Engineering*, pages 141–146, 1992.

[88] Iwasaki, Y. and Simon, H.A. Causality and model abstraction. *Artificial Intelligence*, 67(1):143–194, 1994.

[89] Kahneman, D. and Tversky, A. Prospect theory: an analysis of decision under risk. *Econometrica*, 47:263–291, 1979.

[90] Kahneman, D. and Tversky, A. The psychology of preference. *Scientific American*, 246:160, 1982.

[91] Kaleva, O. Fuzzy differential equations. *Fuzzy Sets and Systems*, 24:301–317, 1987.

[92] Kampichler, C., Barthel, J., and Wieland, R. Species density of foliage-dwelling spiders in field margins: a simple, fuzzy rule-based model. *Ecological Modelling*, 129:87–99, 2000.

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

[94] Keppens, J. and Shen, Q. Towards compositional modelling of ecological systems via dynamic flexible constraint satisfaction. In *Proceedings of the 14th International Workshop on Qualitative Reasoning about Physical Systems*, pages 74–82, 2000.

[95] Keppens, J. and Shen, Q. Disaggregation in compositional modelling of ecological systems via dynamic constraint satisfaction. In *Proceedings of the 15th International Workshop on Qualitative Reasoning about Physical Systems*, pages 21–28, 2001.

[96] Keppens, J. and Shen, Q. On compositional modelling. *Knowledge Engineering Review*, 16(2):157–200, 2001.

[97] Keppens, J. and Shen, Q. A calculus of partially ordered preferences for compositional modelling and configuration. In *Proceedings of the AAAI Workshop on Preferences in AI and CP: Symbolic Approaches*, pages 39–46, 2002.

[98] Keppens, J. and Shen, Q. Compositional modelling as a dynamic constraint satisfaction problem involving order-of-magnitude preferences. Submitted for journal publication, 2002.

[99] Keppens, J. and Shen, Q. On supporting dynamic constraint satisfaction with order of magnitude preferences. In *Proceedings of the 16th International Workshop on Qualitative Reasoning about Physical Systems*, pages 75–82, 2002.

[100] Keppens, J. and Shen, Q. A relative order-of-magnitude calculus and its application for valued constraint satisfaction problems. Submitted for journal publication, 2002.

[101] Kuipers, B.J. Qualitative simulation. *Artificial Intelligence*, 29:289–338, 1986.

[102] Kuipers, B.J. Abstraction by time scale in qualitative simulation. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 621–625, 1987.

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

[104] Larsen, E., van Ackere, A., and Warren, K. The growth of service and the service of growth: Using system dynamics to understand service quality and capital allocation. *Decision Support Systems*, 19:271–287, 1997.

[105] Leitch, R.R., Shen, Q., Coghill, G.M., and Chantler, M.J. Choosing the right model. *IEE Proceedings on Control Theory and Applications*, 146:435–449, 1999.

[106] Levy, A.Y., Iwasaki, Y., and Fikes, R. Automated model selection for simulation based on relevance reasoning. *Artificial Intelligence*, 96:351–394, 1997.

[107] L. Ljung. *System Identification: Theory for the User*. MIT Press, Cambridge, MA, 1999.

[108] Logofet, D.O. and Lesnaya, E.V. The mathematics of Markov models: what Markov chains can really predict in forest successions. *Ecological Modelling*, 126:285–298, 2000.

[109] Lorek, H. and Sonnenschein, M. Object-oriented support for modelling and simulation of individual-oriented ecological models. *Ecological Modelling*, pages 77–96, 1998.

[110] A.J. Lotka. *Elements of physical biology*. Williams & Wilkins Co., Baltimore, 1925.

[111] M.H. Lyons. Modelling the interactions between new telecommunications services. *British Telecom Technology Journal*, 12(2):109–113, 1994.

[112] T.R. Malthus. An essay on the principle of population. printed for J. Johnson in St. Paul's Church Yard, London, England, 1798.

[113] Manel, S., Dias, J-M, and Ormerod, S.J. Comparing discriminant analysis, neural networks and logistic regression for predicting species distributions: a case study with a himalayan river bird. *Ecological Modelling*, 120:337–347, 1999.

[114] Marcot, B. G., Holthausen, R.S., Raphael, M.G., Rowland, M., and Wisdom, M. Using bayesian belief networks to evaluate fish and wildlife population viability under land management alternatives from an environmental impact statement. *Forest Ecology and Management*, 153(1–3):29–42, 2001.

[115] Mavrovouniotis, M.L. and Stephanopoulos, G. Reasoning with orders of magnitude and approximate relations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 626–630, 1987.

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

[117] Miguel, I. *Dynamic Flexible Constraint Satisfaction and its Application to AI Planning*. PhD thesis, The University of Edinburgh, 2001.

[118] Miguel, I. and Shen, Q. Dynamic flexible constraint satisfaction. *Applied Intelligence*, 13(3):231–245, 2000.

[119] Miguel, I. and Shen, Q. Solution techniques for constraint satisfaction problems: Advanced approaches. *Artificial Intelligence Review*, 15(4):269–293, 2001.

[120] Minton, S., Johnston, M.D., Philips, A.B., and Laird, P. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[121] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[122] Mitchell, M. *An introduction to genetic algorithms*. MIT Press, 1996.

[123] Mittal, S. and Falkenhainer, B. Dynamic constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 25–32, 1990.

[124] L. Mohr. *The Causes of Human Behaviour: Implications for Theory and Method in the Social Sciences*. University of Michigan Press, Ann Arbor, 1996.

[125] Moore, R.E. Methods and applications of interval analysis. In *SIAM studies in applied mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[126] Morecroft, J.D.W., Lane, D.C., and Vita, P.S. Modelling growth strategy in a biotechnology startup firm. *System Dynamics Review*, 6(2):93–116, 1991.

[127] Morecroft, J.D.W. and van der Heijden, K.A. Modelling the oil producers - capturing oil industry knowledge in a behavioural simulation model. *European Journal of Operational Research*, 59(1):102–122, 1992.

[128] Moura Pires, J. and Prade, H. Logical analysis of fuzzy constraint satisfaction problems. In *Proceedings of the 7th IEEE International Conference on Fuzzy Systems*, pages 957–962, 1998.

[129] Muetzelfeldt, R.I. and Taylor, J. The suitability of ame for agroforestry modelling. *Agroforestry Forum*, 8:7–9, 1997.

[130] Muetzelfeldt, R.I. and Yanai, R.D. Model transformation rules and model disaggregation. *Science of the Total Environment*, 183:25–31, 1996.

[131] S.S. Murthy. Qualitative reasoning at multiple resolutions. In *Proceedings of the National Conference on Artificial Intelligence*, pages 296–300, 1988.

[132] Nayak, P.P. Order of magnitude reasoning using logarithms. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 201–210, 1992.

[133] Nayak, P.P. Order of magnitude reasoning using logarithms. In *Proceedings of the International Workshop on Qualitative Reasoning about Physical Systems*, 1993.

[134] Nayak, P.P. Causal approximations. *Artificial Intelligence*, 70:277–334, 1994.

[135] Nayak, P.P. *Automated Modeling of Physical Systems*. Lecture Notes in Artificial Intelligence. Springer, 1995.

[136] Nayak, P.P. and Joskowicz, L. Efficient compositional modeling for generating causal explanations. *Artificial Intelligence*, 83:193–227, 1996.

[137] Neter, J., Kutner, Michael H., Nachtsheim, C.J., and Wasserman, W. *Applied Linear Statistical Models*. Irwin, 1996.

[138] Nicholson, A.J. and Bailey, V.A. The balance of animal populations. *Proceedings of the Zoological Society of London*, 1:551–598, 1935.

[139] Nilsson, N.J. *Principles of Artificial Intelligence*. Tioga Publishing, 1980.

[140] Parsons, S. and Hunter, A. A review of uncertainty handling formalisms. In Parsons, S. and Hunter, A., editors, *Applications of Uncertainty Formalisms*. Springer-Verlag, 1998.

[141] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufmann, 1988.

[142] Prügel-Bennett, A. and Shapiro, J.L. An analysis of genetic algorithms using statistical mechanics. *Physical Review Letters*, 72(9):1305–1309, 1994.

[143] A.L. Pugh. *DYNAMO User's Manual*. MIT Press, 1976.

[144] O. Raiman. Order of magnitude reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 100–104, 1986.

[145] O. Raiman. Order of magnitude reasoning. *Artificial Intelligence*, 51:11–38, 1991.

[146] Ramachandran, S., Mooney, R.J., and Kuipers, B.J. Learning qualitative models for systems with multiple operating regions. In *Proceedings of the 8th International Workshop on Qualitative Reasoning about Physical Systems*, pages 212–223, 1994.

[147] Raphael, B. A* algorithm. In Shapiro, S.C., editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 1–3. John Wiley & Sons, 1990.

[148] Richards, B.L., Kraan, I., and Kuipers, B.J. Automated abduction of qualitative models. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 723–728, 1992.

[149] Rickel, J. and Porter, B. Automated modeling for answering prediction questions: Selecting the time scale and system boundary. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 1191–1198, 1994.

[150] Rickel, J. and Porter, B. Automated modeling of complex systems to answer prediction questions. *Artificial Intelligence*, 93:201–260, 1997.

[151] Robertson, D., Bundy, A., Muetzelfeldt, R.I., Haggith, M., and Uschold, M. *Eco-Logic. Logic-Based Approaches to Ecological Modelling*. MIT Press, 1991.

[152] Rogers, D.J. Random search and insect population models. *Journal of Animal Ecology*, 41:369–383, 1972.

[153] Salles, P. and Bredeweg, B. Constructing progressive learning routes through qualitative simulation models in ecology. In *Proceedings of the 15th International Workshop on Qualitative Reasoning about Physical Systems*, pages 82–89, 2001.

[154] Salski, A. Fuzzy knowledge-based models in ecological research. *Ecological modelling*, 63:103–112, 1992.

[155] Say, A.C. and Kuru, S. Qualitative system identification: deriving structure from behavior. *Artificial Intelligence*, 83:75–141, 1996.

[156] T. Schiex. Possibilistic constraint satisfaction problems, or how to handle soft constraints. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 268–275, 1992.

[157] Schiex, T., Fargier, H., and Verfaillie, G. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 631–637, 1995.

[158] Scott, J.A. and Coghill, G.M. Qualitative euler integration with continuity. In *Proceedings of the 12th International Workshop on Qualitative Reasoning about Physical Systems*, pages 114–122, 1998.

[159] Senge, P. and Sterman, J.D. Systems thinking and organizational learning: Acting locally and thinking globally in the organization of the future. *European Journal of Operational Research*, 59(1):137–150, 1992.

[160] Shen, Q. and Leitch, R.R. Fuzzy qualitative simulation. *IEEE Transactions on Systems, Man, and Cybernetics*, 23:1038–1061, 1993.

[161] J.D. Sterman. *People Express Management Flight Simulator*. MIT Sloan School of Managment, 1988.

[162] S.S. Stevens. On the theory of scales of measurement. *Science*, 103:677–680, 1946.

[163] S.S. Stevens. Mathematics, measurement and psychophysics. In S.S. Stevens, editor, *Handbook of experimental psychology*, pages 1–49. Wiley, New York, 1951.

[164] Struss, P. Artificial intelligence for nature - why knowledge representation and problem solving should play a key role in environmental decision support. In Haasis, H.D. and Ranze, K.C., editors, *Computer Science for the Environmental Protection '98*. Metropolis Verlag, 1998.

[165] Struss, P. and Heller, U. Process-oriented modeling and diagnosis - revising and extending the theory of diagnosis from first principles. In *Proceedings of the 9th International Workshop on Principles of Diagnosis*, 1998.

[166] Struss, P. and Heller, U. Model-based support for water treatment. In *Proceedings of the IJCAI'99 Workshop on Qualitative and Model Based Reasoning for Complex Systems and their Control*, pages 84–90, 1999.

[167] Thompson, W.R. On the relative value of parasites and predators in the biological control of insect pests. *Bull. Etnomol. Res.*, 19:343–350, 1929.

[168] Travé-Massuyès, L. and Piera, N. The orders of magnitude models as qualitative algebras. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1261–1266, 1989.

[169] Tsang, E. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.

[170] Tsang, E. and Warwick, T. Applying genetic algorithms to constraint satisfaction optimization problems. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 649–654, 1990.

[171] Tversky, A. and Thaler, R.H. Anomalies preference reversal. *Journal of Economic Perspectives*, 4:201–211, 1990.

[172] K. Ulrich. Computation and pre-parametric design. Technical Report 1043, Massachusetts Institute of Technology, 1988.

[173] Uschold, M. *The Use of Typed Lambda Calculus for Comprehension and Construction of Simulation Models in the Domain of Ecology*. PhD thesis, The University of Edinburgh, 1990.

[174] Uschold, M. The use of the typed lambda calculus for guiding naive users in the representation and acquisition of part-whole knowledge. *Data and Knowledge Engineering, Special Issue on Parts and Wholes*, 20:385–404, 1996.

[175] Usher, M.B. Modelling successional processes in ecosystems. In Gray, A.J., Crawley, M.J., and Edwards, P,J., editors, *Colonization, Succession and Stability*, pages 31–56. Blackwell, Oxford, 1987.

[176] van Ackere, A., Larsen, E., and Morecroft, J.D.W. Systems thinking and business process re-design. *European Management Journal*, 11(4):412–423, 1993.

[177] Verfaillie, G. and Schiex, T. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 307–312, 1994.

[178] P. Verhulst. Recherches mathématiques sur la loi d'accroissement de la population. *Nouveaux mémoires de l'académie royale des sciences et belles-lettres de Bruxelles*, 18:1–38, 1838.

[179] V. Volterra. Fluctuations in the abundance of a species considered mathematically. *Nature*, 118:558–560, 1926.

[180] G. Walter. *Compartmental Modeling with Networks*. Birkhauser Boston, 1999.

[181] Washio, T. and Motoda, H. Discovery of first-principle equations based on scale-type-based and data-driven-reasoning. *Knowledge-Based Systems*, 10:403–411, 1998.

[182] Weld, D.S. Approximation reformulations. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 407–412, 1990.

[183] Weld, D.S. and Addanki, S. Query directed approximations. In Faltings, Boi and Struss, Peter, editors, *Recent Advances in Qualitative Physics*. MIT Press, Cambridge, MA, 1991.

[184] Wortmann, J., Hearne, J.W., and Adams, J.B. Evaluating the effects of freshwater inflow on the distribution of estuarine macrophytes. *Ecological Modelling*, 106:213–232, 1998.

Index of Authors

Index of Concepts