Author/s:
BLOM, MICHELLE

Title:
Arguments and actions: decoupling preference and planning through argumentation

Date:
2011

Citation:
Blom, M. (2011). Arguments and actions: decoupling preference and planning through argumentation. PhD thesis, Engineering - Computer Science and Software Engineering, The University of Melbourne.

Persistent Link:
https://hdl.handle.net/11343/35953

Arguments and Actions

Decoupling Preference and Planning through Argumentation

Michelle Blom

Submitted in total fulfillment of the requirements of the degree of

Doctor of Philosophy

Department of Computer Science and Software Engineering
THE UNIVERSITY OF MELBOURNE

April 2011

Produced on archival quality paper

Abstract

Automated decision-making and planning is a capability that lays the foundation for the development of intelligent decision-support. These systems range from sophisticated multi-agent solutions for corporate decision-support, to decision recommendation for the general consumer. The ideas, algorithms, and tools developed within this thesis cater for this latter category – a class of approaches that includes online travel planners, smart environments, context-aware navigational aides, and personal shopping assistants.

Operating on behalf of a human user, these tools are charged with the recommendation of a decision that best serves their interests – characterised by their preference over available choices. The mission of an automated decision-maker is to extract and use this knowledge in the selection of a decision. Across the range of problem-domain scenarios faced by a human decision-maker, the appropriate means by which their preference is communicated varies – from the quantitative to qualitative, logical to heuristic, and simple to complex. The capacity of these tools to select choices on the basis of such wide ranging classes of preference increases their utility as human decision aides.

This thesis considers the decoupling of preference from the decision-making and planning algorithms that underly such systems. These algorithms operate on this preference as if it were an instance of an abstract type – supporting the use of general expressions of preference while abstracting away from its representational detail. Such algorithms can not only be reused across a range of decision-making problems faced by an agent or user, but cater for the diversity of human users on whose behalf decision-making is taking place. This thesis presents three key contributions within this domain.

The existing body of work within the field of automated decision-making and planning with preference is critiqued, identifying the field of computational argumentation as a promising vehicle for the discovery of decoupled decision-making. The argumentation-based approach to decision-making allows preference to be captured by structures – structures that are manipulated by the decision-making process as abstract entities. This thesis develops a decoupled algorithm for the selection of choices on the basis of general mechanisms of choice comparison – a feat not achieved within existing work.

Attention is then shifted from decision-making over single choices or actions to the problem of planning with preference. Algorithms are developed that chart a course of action for an agent or human user to follow in pursuit of their goals. This thesis focuses on a specific planning formalism – the GOLOG family of agent programming languages. High level instructions in the form of programs are interpreted to discover a sequence of actions for an agent or human user to perform. Two interpreters are devised for the discovery of preferred program executions – the first providing support for only one form of preference expression; the second presenting a decoupled approach to program interpretation with general preference. These interpreters go beyond the capabilities of existing works, with improved heuristic guidance in the form of a relaxed lookahead and support for decoupling through argumentation-based program interpretation.

Declaration

This is to certify that:

1. The thesis comprises only my original work towards the PhD;
2. Due acknowledgement has been made in the text to all other material used;
3. The thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Michelle Blom, April 2011

Acknowledgements

I would like to express my gratitude to all those who have supported me in the completion of this thesis. First, and foremost, I would like to thank my family: Helen, Victor, Suzanne, and Matthew Blom, for their continued support. I am especially indebted to my supervisor Adrian R. Pearce for his continued guidance, support, and mentoring throughout my candidature, and to my supervisory committee Liz Sonenberg and Tim Baldwin for their advice. I would also like to thank Peter McBurney, Katie Atkinson, and Iyad Rahwan for their guidance and for hosting my visits to the British University in Dubai and the University of Liverpool. I wish to thank Sebastian Sardiña for our discussions on the topics of my thesis, and my friends, colleagues, and office mates at the University of Melbourne: Dana Zhang, Jayant Baliga, Mohammed Jubaer Arif, Ryan Kelly, Jian Huang, Vijay Vijayalayan, Yang Liao, Andrey Kan, and Masud Moshtagi. These individuals have kept me sane and have provided me with a wonderful PhD experience.

Preface

The content of this thesis comprises only my original work which was conducted solely during my PhD candidature and has not been submitted for any other qualifications or degrees. Parts of this thesis have been extracted and published in various venues in collaboration with my supervisor Dr. Adrian R. Pearce, as noted below:

1. M. Blom. Optimising the Interpretation of GOLOG Programs with Argumentation. In *European Workshop on Multi-Agent Systems (EUMAS)*, 2008, pp. 597–611.

2. M. Blom and A. Pearce. An Argumentation-Based Interpreter for GOLOG Programs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2009, pp. 690–695.

3. M. Blom and A. Pearce. Relaxing Regression for a Heuristic GOLOG. In T. Ågnotes, editor, *Proceedings of the Fifth Starting AI Researchers Symposium (STAIRS 2010)*, pp. 37-49. IOS Press, 2010.

In the above works, I developed most of the ideas, conducted all of the described experiments, and was responsible for the writing of each paper.

Contents

Chapter 1
Introduction

THE capacity to support general or arbitrary preference within tools for automated decision-making and planning is important in the development of flexible decision support. Such flexibility increases its usefulness as an aide to human decision-making, and our satisfaction in the performance of the intelligent agents that employ it.

The range of applications in which user or agent preference must play a role in their operation is seemingly endless. Travel planning agents have been designed to construct personalised itineraries for cross-country travellers [Ambite et al. (2002); Knoblock (2004)]. A travel planner, designed to organise flight and road transportation between two destinations, should tailor its bookings to meet the preferences of its client. An agent that arranges flights and transport with carriers undesirable to a user is unlikely to engender a customer following. In the construction of ore stockpiles within the domain of mine operations [Smith and Dimitrakopoulos (1999)], block selection that leads to a desired stockpile quality is paramount. In the control of a spacecraft for celestial target observation [Smith et al. (2000)], the goal is to maximise scientific return while respecting constraints on resources and time (requiring judicious selection of targets to observe).

Existing techniques within the umbrella of preference-based decision-making and planning have a common limitation. Representation of user preference in these approaches is not decoupled from, or independent of, the algorithmic process used to make decisions. To successfully operate, each algorithm requires a knowledge of the representational detail of the preferences supplied to it. Traditional decision-theoretic approaches,

for instance, assume that the preference of a decision-maker is expressed as a utility function [von Neumann and Morgenstern (1944)]. The preferred evaluation of decisions with respect to a set of prioritised criteria (such as cost, reliability, and quality) guides decision-making in multi-criteria methods [Figueira et al. (2005)]. Across the range of existing preference-based planners, preference is expressed in temporal logics [Baier et al. (2009); Bienvenu et al. (2006)], as soft goals and constraints [Brafman and Chernyavsky (2005); Benton et al. (2009); Edelkamp and Kissmann (2009)], or metric functions and action costs [Borowsky and Edelkamp (2008); Benton et al. (2005); Keyder and Geffner (2008)].

The development of such a wide range of techniques, each designed to cater for a different style of preference expression, is indicative of the fact that people express preference in different ways when faced with different problem scenarios and domains. When purchasing goods, preference with a ceteris paribus "all else being equal" interpretation [Doyle and Wellman (1994); Boutilier et al. (2004); Hansson (1996)] may be appropriate. We may prefer, for example, a square table over a round when all other attributes are equivalent [Hansson (1996)] – an example of a ceteris paribus preference. Numeric evaluation functions, that must be maximised or minimised, are natural when money and resource utilisation are the burning issues. Temporal preference may be fitting when system evolution during plan execution is important. We can, for example, prefer plans in which properties hold always, sometimes, never, or until alternative properties take hold.

The term 'decoupling' is used in this thesis to refer to the separation of two entities such that one is independent of the other. In the field of psychology, cognitive decoupling is the process of separating beliefs about a hypothetical reality from those concerning the actual world [Leslie (1987)]. Decoupling in software systems identifies the system components that should not or need not invoke the functionality of the other, and removes those dependencies. In this thesis, a decision-making or planning algorithm is decoupled from the preferences supplied to it if, in its operation, it uses and manipulates these preferences as if they were instances of an abstract type. Each instance of this abstract type is an arbitrary preference in some format – a series of prioritised goals or a utility function, for example – whose representational detail is hidden from the algorithm operating on it. This decoupling is not simply a separation of preference into a structure (such as a utility

function) that can be accessed by a decision-making algorithm. Decoupling of preference from the decision-making and planning process, in this thesis, allows different forms of preference – from soft goals to ceteris paribus or temporal logic statements – to guide decision-making, alone or in combination, within a single algorithm.

The decoupling of preference from the algorithms that drive automated decision-making and planning is advantageous in a number of respects. Such decoupled algorithms can be applied to address different decision-making and planning problems in different domains – where the appropriate means by which preference should be represented varies. These systems are able to cater for a wide range of potential users – each user an individual with their own preferences and preferred means of expressing them. Such systems have potential widespread application in tools typically used by the general consumer: online shopping applications; travel planners; and navigational aides.

1.1 Thesis Contributions

This thesis asks the question: can the decoupled decision-making and planning algorithms described above be developed, and how? As I describe in this thesis, algorithms of this type do not currently exist. In this thesis, I present three major contributions:

1. A decoupled argumentation-based decision-making framework for the selection of a most preferred choice from a set of choices (denoted the ADM);
2. An integration of heuristic search within the interpretation of GOLOG programs, based on the use of a relaxed lookahead. This relaxed lookahead employs relaxed situation calculus regression – a technique developed in this thesis;
3. A new approach to search – using argumentation-based evaluation of available paths – used to create an argumentation-based GOLOG interpreter (AR-GOLOG).

My first contribution is a decoupled decision-making framework (ADM) for the selection of a most preferred choice. This ADM employs techniques within the field of computational argumentation (described in Chapter 3), and is presented in Chapter 4.

The second and third of these contributions transition into the study of planning with preference. The second contribution of this thesis is an interpreter for GOLOG programs

[Levesque et al. (1997); de Giacomo et al. (2000)] that is guided by a relaxation-based heuristic. This heuristic incorporates a process of relaxed situation calculus regression, a technique that I develop in Section 6.3. My heuristic GOLOG interpreter is, in addition, presented in Chapter 6 and evaluated in Chapter 7 on a number of case studies.

My third thesis contribution presents an argumentation-based approach to goal-directed search, denoted ABS. This search technique is presented in Chapter 8. This chapter presents, in addition, an application of ABS in the form of an argumentation-based interpreter for GOLOG programs, denoted AR-GOLOG. This interpreter represents a novel decoupled approach to planning with preferences in the GOLOG space. These contributions are important as they provide a means of flexible decision-support within applications designed to interact with human users, and for intelligent agents.

I begin by considering the problem of selecting a single decision from a set on the basis of a collection of user preferences whose form is mixed amongst the different expressions we often employ (from numerical functions, ceteris paribus comparatives, prioritised goals or criteria, to logical representations). This study forms Part I of this thesis.

1.1.1 Part I

In the collection of chapters that represent Part I of this thesis, I describe the spectrum of preference handling techniques that have been employed within existing tools for automated decision-making (Chapter 2). I have found that no existing representation is expressive enough to capture each of the different styles in which user preference can be described. Consequently, the development of a decision-making system tailored to even the most expressive of these representations will not result in the same freedom of preference expression attainable through the use of a decoupled approach. I have discovered that the class of decision-making approaches that comes closest to achieving a decoupling of preference and algorithm are those that wrap user preferences into structures – structures that can be manipulated by a decision-making algorithm as if they were abstract entities. These approaches lie within the field of computational argumentation.

On the back of growing popularity in the research of non-monotonic logics during the 1980s, argumentation presented an alternative approach to belief revision and decision-

making in dynamically changing environments. Reflective of the manner in which humans often engage in decision-making, these techniques allowed agents to engage in debates on the relative merits of choices, and the likelihood of certain states-of-affairs over others. I present a historical perspective of the argumentation field, with an emphasis on formalisms for argumentation-based decision-making and planning, in Chapter 3.

These argumentation-based systems allow an agent to construct arguments in support of or against statements of interest or points of view. These points of view may be beliefs regarding the state of its environment, decisions that should be selected or rejected on the basis of a collection of preferences, or relationships that exist between a set of available choices. These arguments conflict and engage in attack when one disputes the validity of another. Acceptable arguments define the points of view that the agent can believe without contradiction. The process of finding acceptable arguments varies between argumentation formalisms. These acceptable arguments provide an intelligent agent with all the information it requires to form its beliefs, and commit to its decisions.

In argumentation-based decision-making, preference is captured in arguments that support the selection of one choice over others – the acceptable subset used to infer a preferred collection of choices. Decoupling is encouraged by allowing an agent to abstract away from the content of arguments, and thus the types of preference used in their construction, when ascertaining an acceptable subset. In the mid-1990s, the seminal work of Dung (1995) introduced an abstract argumentation framework in which the discovery of acceptable arguments required knowledge of only the attack relationships that existed between them. A number of existing argumentation-based decision-makers employ this framework. In this thesis, I consider the use of argumentation as a vehicle for the delivery of a decoupled approach to decision-making with preference (Chapter 4).

While a range of argumentation-based decision-making and planning methods have been developed in the past few decades, none have delivered a decoupled system. The manipulation of arguments as abstract entities is not a sufficient condition to guarantee the decoupling of preference and algorithmic process within these approaches – an additional condition must hold. The structure of these arguments must not be restricted in ways that limits the range of preference types they can capture when supporting one

choice over others. As I describe in Chapter 3, this is not a feature of existing work.

I develop in this thesis a system, employing the abstract framework of Dung (1995), that meets each of the two required conditions for decoupling outlined above. I place this decoupled approach to decision-making in context with alternative work in the vast landscape of automated decision-making research, highlighting the advantages of argumentation-based methods over more traditional techniques. These systems not only reason over the means by which choices are compared, but critically analyse the information upon which this comparison takes place. The decoupled argumentation-based decision-making system I develop represents the first key contribution of this thesis.

The second and third contributions of this thesis depart from the study of decision-making over single actions or choices, and consider the task of planning with preference. The collection of chapters containing this work forms Part II of this thesis.

1.1.2 Part II

The type of planning studied in this thesis is the process of finding a sequence of actions that, when performed by an agent, achieves a collection of desired goals. These goals may be the completion of a task or a set of activities, or the achievement of a desirable state of affairs. A wide variety of formalisms exist in which an agent can conduct planning.

Within the classical planning formalism, planning discovers a finite linearly ordered sequence of actions that transitions a planner from an initial state to a desired goal state in a domain that is modelled by a restricted state-transition system [Ghallab et al. (2004)][1]. Planning in the hierarchical task network (HTN) approach [Ghallab et al. (2004)] starts with a partially ordered set of tasks to be performed by an agent – these tasks then iteratively decomposed into simpler subtasks. The resulting partially ordered set of primitive operations defines a number of possible concrete plans. In logical formalisms, such as the GOLOG family of agent programming languages [Levesque et al. (1997); de Giacomo et al. (2000)], planning is defined as the search for an execution of a program. Each program is a set of high level instructions that tell an agent how to derive a sequence of actions that, when performed, will achieve a set of goals or complete a required task.

[1] The restrictions that characterise these systems include an assumption of deterministic state transitions, finite states, and a static environment (an environment that changes only in response to actions).

This thesis considers planning within the GOLOG formalism. An agent that employs this formalism achieves its goals by finding and performing an execution of an appropriate GOLOG program. It does so through the use of an interpreter. The role of a GOLOG interpreter is to take a program, and the current state-of-affairs of the agent, and find a sequence of actions for this agent to perform. The (non-deterministic) constructs available in the formation of a GOLOG program provide an interpreter with choice in its selection of actions. Consequently, each GOLOG program has a potentially large number of possible executions. A traditional or standard interpreter is not guided by preference in its search for one of these executions. The GOLOG language is described in Chapter 5.

Within the GOLOG literature there exists a range of interpreters developed for the purpose of finding a most preferred program execution. The types of preference supported by these interpreters range from utilities [Boutilier et al. (2000)], expressions in temporal logic [Sohrabi et al. (2006)], to a combination of the two [Fritz and McIlraith (2005)]. I have identified two limitations in the current crop of preference-based interpreters (Chapter 6). The first limiting aspect of these works is their use of naive heuristic guidance. The second is that existing techniques designed to search for preferred executions of a GOLOG program are not decoupled from the preferences supplied to them.

Heuristic guidance within an interpreter directs search toward choices that are most likely to lead to a most preferred (optimal) execution. The aim of this guidance is to reduce the space of executions explored in the search for a most preferred execution, or improve the quality of solutions found where optimality is not required. Across the range of interpreters that employ a heuristic estimation of the desirability of available choices, this estimation is naive. The future possibilities that may arise in the pursuit of a path are not considered when determining its merit. The development of more informed heuristic techniques has the potential to further reduce the search effort required within optimal interpreters, and improve the quality of executions found by the sub-optimal.

I present in this thesis two distinct preference-guided interpreters for GOLOG programs. The second of these interpreters represents a decoupled approach to the preference-guided interpretation of a GOLOG program. The first offering considers and addresses the first of my two identified limitations. In doing so, I consider a specific form

of preference representation – the numeric evaluation or utility function – to characterise the most preferred execution of a program. Each evaluation function, when applied to a program execution, returns a numeric assessment of its desirability. I develop an interpreter whose aim is to find a program execution that minimises this function, while employing an informed (lookahead-based) heuristic to guide its search. This heuristic is adapted from a class of relaxation-based heuristics [Bonet and Geffner (2001); Hoffman and Nebel (2001)] – often used in the realm of classical planning. These heuristics use the solutions of relaxed versions of a planning problem to guide search toward a desired solution. I explore the benefit of using these more informed heuristic guidance techniques relative to those characteristic of existing work. The resulting heuristic interpreter forms the second major contribution of this thesis, and is presented in Chapter 6.

The second preference-guided GOLOG interpreter I present in this thesis returns its focus to the decoupling of preference and planning. The experience gained and the techniques developed in my first two thesis contributions lay the foundation for a decoupled approach to preference-based GOLOG interpretation. This second offering employs my argumentation-based decision-making system (ADM), presented in Chapter 4, in a search algorithm designed to discover a most preferred execution of a GOLOG program.

At each instance in which this GOLOG interpreter faces a choice, the ADM of Chapter 4 is applied to the range of available choices. Arguments are generated to support the exploration of one path in the interpretation of a program over others. The decisions made by this interpreter are formulated on the basis of which choices are selected by this ADM as being the most preferred. Argumentation-based interpretation inherits the advantages of the ADM it employs – allowing this interpreter to find an execution of a GOLOG program while operating on an abstract representation of preference. This interpreter represents the third contribution of this thesis and is presented in Chapter 8.

1.2 Overview

This section presents a chapter-by-chapter breakdown of the remainder of this thesis. I provide a brief synopsis of each of these chapters in the sections that follow.

1.2.1 Part I

Chapter 2 In this chapter the role of preference in decision-making is explored, with the aim of giving insight into the wide spectrum of preference expressions used in the fields of decision theory and analysis. A decoupled decision-making system must support each of these types of preference. The conditions used to test for the decoupling of preference are defined.

Chapter 3 This chapter presents a historical perspective of, and a selective background on, the field of computational argumentation – a field whose tools and techniques are featured prominently in this thesis.

Chapter 4 An argumentation-based decision-making system is presented that operates on preferences as if they were members of an abstract type, achieving a decoupling between preference and algorithmic process. This system, denoted the ADM, forms the first contribution of this thesis.

1.2.2 Part II

Chapter 5 The first chapter of Part II of this thesis introduces the GOLOG family of agent programming languages and the situation calculus upon which they are based [McCarthy and Hayes (1969); Reiter (2001)].

Chapter 6 A heuristic GOLOG interpreter for the discovery of preferred program executions is presented in this chapter. This interpreter employs a heuristic that 'looks ahead' when evaluating available choices in the search for a program execution. This heuristic relies upon a relaxed variation of situation calculus regression – developed within this thesis as relaxed regression. This different style of regression, and the heuristic interpreter that it characterises, form the second contribution of this thesis.

Chapter 7 In this chapter an experimental evaluation of the heuristic interpreter of Chapter 6 is provided. This interpreter is experimentally evaluated on a range of domains (case studies), and compared to a collection of interpreters that are representative of existing and related work.

Chapter 8 This chapter presents an argumentation-based interpreter for GOLOG programs, denoted AR-GOLOG. This interpreter employs the argumentation-based decision-maker of Chapter 4 in the search for a most preferred execution of a GOLOG program. This interpreter is experimentally evaluated on a travel planning domain, and a subset of the domains considered in Chapter 7 – its results compared with the heuristic interpreter of Chapter 6. This work is the third contribution of this thesis.

Chapter 9 In this chapter, I examine the degree to which I have been successful in achieving my goal of developing decoupled algorithms for decision-making and planning with preference. I conclude by describing a number of directions in which the presented work can be taken.

1.2.3 Appendices

Appendix A An extended listing of the results of a subset of the experiments conducted within this thesis is provided.

Appendix B In this chapter, I provide a number of definitions that complement the material in Chapter 8. These definitions further characterise the travel planning domain used to evaluate my AR-GOLOG interpreter.

Part I

Argumentation and Decision-Making

Introduction to Part I

IN our daily lives we are charged with the responsibility of making a vast array of decisions. From the purchase of commodities; the planning and arrangement of travel itineraries; to the selection of a movie to see on the weekend. In these situations we make or select a decision that, from our perspective, best serves our interests. These interests are characterised by our preferences over the properties of available choices.

The mission of an automated decision-making system is to extract these preferences from a human user and make decisions on their behalf. Such automated systems have endless applications: from the development of personal shopping assistants [Menczer et al. (2002); Kim et al. (2009)]; online travel agents and planners [Camacho et al. (2001); Yim et al. (2004)]; context-aware navigational aides, travel guides, and user interfaces [Peintner et al. (2008)]; to autonomous robots and smart environments [Cloos (2005); Hardy-Vallèe (2009); Cook and Das (2007); Youngblood et al. (2005)].

One can trace the study of preference in decision-making to the work of Greek philosopher Aristotle (Book III of Topics) [Hansson (2001); Pickard-Cambridge (2010)]. These writings tell us that each individual brings to the table their own set of guidelines, their own 'science', with which to make decisions. These guidelines are principles that describe how alternatives are to be compared – on the basis of justice, for example, or on the basis of their consequences, induced pleasure, pain, happiness, or wealth.

In Chapter 2, the means by which these principles of Aristotle are expressed within computational theories of decision-making are discussed. In doing so, the first challenge of decoupling preference and decision-making is tackled – an ability to express preference in a general form. A range of different representations and their ability to support general preference (the comparison of choices by any desired means) are discussed, and an approach to general representation identified. A range of different decision-making approaches are described, including argumentation-based techniques. These methods, as highlighted in Chapter 2, have potential in the realisation of decoupling. In this chapter, the conditions that determine whether such approaches are decoupled are defined.

I provide, in Chapter 3, an introduction to the field of computational argumentation. This chapter will take you on a journey through its history – charting a course from its

beginnings in the 1980s and early 1990s to the current state of the art. I first discuss some of the ideas that have inspired its conception, and then focus on the use of argumentation in decision-making and planning. These methods are imbued with distinct advantages over more classical approaches within decision theory and analysis.

The most common design of an argumentation-based decision-maker constructs arguments to support or reject the selection of a choice, on the basis of user preference. The acceptable subset of these arguments defines a ranking over or a partitioning of the choice set, in a manner that varies across formalisms. Where these approaches encourage decoupling is that they allow an agent to abstract away from the content of arguments, and as such the types of preference used in their construction, when ascertaining an acceptable subset. Not one of the existing range of such approaches, described within Chapter 3, achieves my desire for the decoupling of preference, however. Within Chapter 3, I subject each of these approaches to the conditions for decoupling of Chapter 2.

I present an argumentation-based approach to decoupled decision-making in Chapter 4. In doing so, I employ a range of features present within existing work. In contrast to many of these existing approaches, my system requires an agent to construct arguments in support of pair-wise preference relationships over the set of available choices, in place of arguments in support of or against single decisions. Arguments within this system are derived from general preference – general means by which these choices are compared. The decision-making procedure this system adopts abstracts away from the contents of these arguments, ensuring that the type of comparison they exhibit is unrestricted. I evaluate this system with respect to the conditions for decoupling of Chapter 2.

Chapter 2

Preference Handling in Decision-Making

THE scientific study of decision-making as we know it today originated in the form of operational research during the World War II era [Tsoukiàs (2008)] – a response to the need for optimal solutions to complex military planning and decision-making problems. Over the following 60 years, the field diverged into three classes of decision-making tools and theories. Normative approaches enforced an ideal of rational behaviour – principles that a decision-maker must adhere to as a rational being. Descriptive decision-making considered not how a person should make decisions, but how they do make decisions. Prescriptive tools provided decision-support to users without a prescribed ideal of rationality or a desire to replicate the human decision-making process.

The representation of decision-maker preference in a computational form is of vital importance to the success of these devices. Without some representation of what a user prefers, an automated decision-maker cannot operate effectively on their behalf. In the vast landscape of literature on preference handling, two distinguishing labels are used when classifying representations. Delgrande et al. (2007) distinguish two forms of preference representation – absolute (describing the desirability of a particular state-of-affairs) and comparative (describing the desirability of one state-of-affairs relative to others). Similar distinctions are made throughout the literature on preference – often referring to absolute representations as *monadic* and comparatives as *dyadic* [Hansson (2001)].

The favoured model of preference representation within economics and early deci-

sion theory has been the (absolute) expected utility function [von Neumann and Morgenstern (1944)]. When applied to a choice this function returns a numeric evaluation of its desirability. It does so by adding the utilities associated with each consequence or outcome of the choice multiplied by the probability of their occurrence. Maximisation of this function determines which is the best choice. A variety of techniques exist for the elicitation of utility functions from clients or users [Wakker and Deneffe (1996); Farquhar (1984); Keeney and Raiffa (1976); Keeney (1982); Saaty (1980)]. These range from the simple elicitation of numeric values to associate with choice outcomes, to more sophisticated methods involving the comparison of two-outcome lotteries with certain outcomes[1].

The utility function has been criticised as being atypical of the way humans express or desire to express preference. Domshlak (2008) describes the difficulties inherent in the elicitation of a utility function, and identifies the circumstances under which qualitatively expressed preference is more appropriate. The elicitation of a utility function is a time consuming process, requiring the supply of a great deal of information from a client or user. The formation of a utility function from the comparison of lotteries with certain outcomes, for example, typically involves a large number of comparisons. In Chapter 1, I identify a number of applications in which the decoupled decision-making tools I aim to develop in this thesis have potential. In these applications, a user is likely to prefer a means of expressing their preference that requires minimal effort.

A wide range of qualitative or informal forms of preference expression have been developed in response to these criticisms. Ceteris paribus expressions [Wellman and Doyle (1991); Doyle and Wellman (1994)] are qualitative rules which define the properties that (all else being equal) distinguish one choice as better than another. Starmer (1996) describes a class of rationalisation strategies (rules of thumb) representing heuristic decision guides. These guides reflect the preference of a user by describing desired behaviour, such as "Never walk alone at night" or "Always tip 10% at a restaurant". Doyle and Wellman (1994) assert that humans place value on compact representations of

[1] In the probability-equivalence method, a lottery $(x, p; z)$ with outcome x (occurring with probability p) and outcome z (occurring with probability $1 - p$) is compared with a probabilistically certain outcome y [Wakker and Deneffe (1996)]. The probability p is varied until the client expresses indifference between the lottery and y. The utility of y ($u(y)$) can be found from the equation $u(y) = pu(x) + (1 - p)u(z)$. The process of elicitation begins with x and z representing outcomes with minimal and maximal utilities of 0 and 1.

information "that directly entail the most important conclusions". Superlatives (such as "fastest", "cheapest", and "largest") and comparatives (such as "faster", "tastier", and "happier") are recognised as being common ways in which humans describe preference.

In Section 2.1, I consider the three classes of decision-making, and present a number examples within each class. The number of tools in existence for automated decision-making is too large to permit a reference to them all. Hence, I focus on the properties that are typically exhibited by decision-making tools within each class – describing some of the ways in which these techniques often handle preference. This analysis highlights which types of tools have potential in the creation of a decoupled decision-maker.

This is followed, in Section 2.2, with a survey of representations that have been conceived to express our preferences for the purpose of automated decision-making. This survey considers a broad spectrum of representations: from qualitative to quantitative; logical to heuristic; and simple to complex. Representations that have been employed in existing decision-making tools are considered in addition to those that describe preference in a form suitable for manipulation by an automated process. The list presented in this chapter is not exhaustive, but is representative and reflective of the means by which preference is described in the wider philosophical study of decision-making.

The goal of this chapter is to highlight that there is a wide range of techniques by which we can express our preferences (how we would like to compare choices), and a wide range of tools for automated decision-making. The conditions that I have identified as characterising the decoupling of preference and decision-making are outlined in Criterion 2.1.1, presented below in Section 2.1. On the basis of these conditions, I describe in this chapter which representations and tools I believe have the most promise in the development of a decoupled decision-making (or planning) system.

2.1 Decision-Making – A Classification

Decision-making research has introduced to us three classes of tools for decision selection. These classes are normative, descriptive and prescriptive [Bell et al. (1988)]. I highlight in this section a number of techniques within each class, and discuss how each class of approach tends to handle the preferences of a user. The reader is referred to two re-

views that explore this classification in greater detail [Bell et al. (1988); Tsoukiàs (2008)].

Before proceeding with this discussion on the classes of decision-making, I present the conditions (questions) that I will subject these approaches to when determining whether they are decoupled from the preferences supplied to them. To achieve decoupling, an approach must be able to operate on the basis of general preference, and abstract away from its detail during this operation. These conditions are used to analyse each of the preference-based decision-making and planning approaches described in this thesis.

Criterion 2.1.1 A decision-making or planning algorithm is evaluated with respect to the following questions, capturing the required properties of a decoupled approach:

1. Is this approach capable of selecting a choice or plan on the basis of general preference – any representation that infers a preference for one entity or choice over another solely on the basis of their individual or relative properties[2] – reflecting the variety of approaches with which preference can be expressed by a human decision-maker or agent and choices compared?
2. Does this approach abstract away from the detail of the preferences supplied to it – requiring no knowledge of their type or content – during decision-making[3]?

If these questions are answered in the affirmative for a given decision-making or planning algorithm, it is classified as decoupled.

Much of the existing decision-making and planning tools described in this thesis satisfy one or both of the conditions highlighted in Criterion 2.1.1 to a certain extent – exhibiting a degree of abstractness (differing degrees of knowledge required of the type and content of preference) and an ability to support a varying range of preferences. In applying the above criterion to such an approach, I consider whether there exists a type of preference (of the kind described in Criterion 2.1.1) that it does not support, and whether it requires any information about the content or type of supplied preferences during the

[2]Each of the types of preference described in this chapter fall into this category. Tversky and Kahneman (1981) describe the framing of a decision problem and its impact on the choices that a human decision-maker selects. The context, or alternative decisions available to the decision-maker, is in some circumstances believed to influence (eg. reverse) the 'relative desirability' of choices. Representations of preference that depend on such a context are not considered in this thesis – I consider only those representations that depend on no more than the individual or relative properties of two choices under comparison.

[3]The decision-making process is the sequence of steps performed between the instance of input being supplied (for example, choices and preference information) and the return of a choice or subset of choices.

decision-making process. The goal of this thesis is to develop tools for decision-making and planning that not only advance this state-of-the-art – developing approaches that are both more abstract and able to support a wider range of preference representations – but move beyond this in an attempt to create approaches that are decoupled (as per Criterion 2.1.1, the advantages of which are stated in the Introduction to this thesis, and below).

As described in Chapter 1, decoupling preference from the algorithms that underlie automated tools for decision-making is advantageous. Importantly, it promotes their re-use: across problem domains where different styles of preference expression are appropriate; and across potential users, each with their own preferred means of preference expression. Such flexibility increases the utility of these tools as decision-making aides.

The decoupling of preference is not always desirable – namely, where preference is not an input to the decision-making process, but an emergent property. In group decision-making, for example, the preferences of one individual may be dependent on those of the others – preferences may be elicited, shared, and incrementally altered during the decision-making process. In this thesis, I consider and develop decision-making processes in which preference is provided as an input, and is not an emergent entity.

2.1.1 Normative Decision-Making

Normative methods are based on the belief that there is an ideal standard of rationality that should guide our choices. A prominent approach within this class of decision-making is expected utility theory [von Neumann and Morgenstern (1944)]. Within this theory, the behaviour of a rational decision-maker adheres to a set of axioms that govern the nature of its preferences[4]. These axioms guarantee that the preferences of a decision-maker can be expressed in the form of a utility function. A rational decision-maker in this setting is one that makes decisions which maximise its expected utility – the sum of the utilities of possible decision outcomes multiplied by the probability of their occurrence.

The maximisation of utility is the cornerstone of many normative decision-making approaches. Altering the set of axioms that characterise decision-maker behaviour has

[4]These axioms prescribe transitivity, completeness, independence, and continuity. Completeness requires that the decision-maker is able to determine which of a pair of choices is preferred. Transitivity, meanwhile, requires that if a choice A is preferred to a choice B, and B preferred to a choice C, then A is preferred to C.

produced a number of variations within expected utility theory. Fishburn (1982), for example, considers the removal of axioms requiring transitivity and independence[5]. These variations have been designed to address axiom violations (to transitivity and independence, for example) discovered by a range of researchers [Allais (1953); Tversky (1969)]. Expected utility maximisation approaches have also been developed to operate given subjective probability assessments. Subjective expected utility theory comes in a number of different varieties, the first complete theory presented by Savage (1954).

A number of approaches that fall within the umbrella of multi-criteria decision analysis (MCDA) have normative foundations. MCDA methods are designed to make decisions when faced with a set of choices characterised by a set of criteria. Multi-attribute utility theory (MAUT) [Keeney and Raiffa (1976)] is a commonly used method in the solution of multi-criteria discrete alternative problems (in which decision-making ranges over a finite set of alternatives). Within MAUT, each decision is evaluated with respect to a finite set of criteria. A multi-attribute (expected) utility is determined on the basis of single-attribute von Neumann and Morgenstern (1944) utilities derived on each criterion. These single-attribute utilities are weighted on the basis of their relative importance, and linear or non-linear methods of aggregation are applied. A number of alternative MCDA approaches exist for the solution of such problems [Roy (1991); Brans et al. (1986); Saaty (1990); e Costa and Vansnick (1994)], a selection of which are described in Section 2.1.3.

Recall the questions for decoupling shown in Criterion 2.1.1. The approaches I have described in this section do not answer the first in the affirmative. These approaches support the expression of preference only in terms of utility functions and prioritised criteria which, as this chapter describes, are just two of many types of representation.

2.1.2 Descriptive Decision-Making

Enforcing a model of ideal rationality – a theory of how we should make decisions – is often at odds with real human decision-making. A contrasting branch of research is the study of descriptive decision-making [Bell et al. (1988)]. Descriptive approaches analyse human decision-making behaviour with the goal of understanding or replicating it.

[5]The independence axiom requires that given three choices A, B, and C, if $A \geq B$ (A is at least as good as B) then $pA + (1-p)C \geq pB + (1-p)C$ for all C and $p \in (0,1)$ [von Neumann and Morgenstern (1944)].

Kahneman and Tversky (1979) describe a model of decision-making amongst risky alternatives (choices whose outcomes occur with a given probability) called prospect theory. Through experiment, Kahneman and Tversky (1979) discover that a real decision-maker often violates the principles of expected utility theory. The expected utility of a decision, as described in Section 2.1.1, is the sum of the utilities of its possible outcomes weighted by the probability with which they will occur. A real decision-maker is found to act as if (probabilistically) certain outcomes are overweighted, and less certain outcomes are underweighted, in this calculation. Prospect theory assesses choice outcomes in terms of received gains and losses (from a reference point), weighting each by a function of their probability in the computation of expected gain (or loss). This theory is one amongst many that feature descriptive decision-making elements [Gigerenzer (2008); von Winterfeldt and Edwards (1986); Poulton (1994); Shafir et al. (1993)].

Rules and heuristics are often viewed as a means to an end in human decision-making. Starmer (1996) considers the use of rationalisation strategies in the development of an alternative (descriptive) theory of decision-making. These strategies represent rules of thumb or choice patterns that provide guidance or instruction for use in specific decision-making scenarios. Such rules, conceived often from personal experience, guide a reasoner toward making choices that keep them safe, encourage gains, and prevent loss. Gigerenzer (2008) asserts that humans reason with an adaptive toolbox of heuristics – each able to solve particular kinds of problems in particular kinds of environments.

In Section 2.1.1, an MCDA approach with a normative foundation was considered. Greco et al. (2001) present a rule-based technique for multi-criteria decision-making that has descriptive features. Their approach is designed to learn decision-rules from historical records of decision-maker behaviour within specific domains. These records identify the choices that have been selected by a decision-maker as the most preferred within given collections. Each decision-rule is an if . . . then structure that matches characteristic attributes of available choices with their place in an overall ranking or ordering. These rules can identify the value (numeric or symbolic) of a choice on the basis of its attributes and properties, or a preference relationship that exists between two choices.

The objective of a decision-making tool within this setting is to suggest decisions

that would have been made by the human decision-maker in question. Descriptive approaches to decision-making are concerned with replicating human decision-maker behaviour, often by analysing the past experience of these individuals. These approaches determine what this behaviour reveals about the preferences of the decision-maker, and employs that information in the selection of a decision. Preference is never elicited in any explicit form, and hence the question of representation is not relevant. The decision-rules of Greco et al. (2001), the rationalisation strategies of Starmer (1996), and the heuristics of Gigerenzer (2008), are reflective of a range of preferences a user may hold.

Within these descriptive approaches, how a decision-maker prefers to express their preference is irrelevant – it is revealed in a model of their behaviour. Acquiring this model requires access to historical records of past decision-making, which may not be available. Moreover, from the perspective of a user, eliciting this information is just an alternative approach to the communication of their preference. While some may prefer to describe how they would act (have acted) in a range of contexts, others may prefer to express preference in one of the forms described in this chapter. The questions of Criterion 2.1.1 are not answered in the affirmative for the approaches described in this section. The rules of Starmer (1996) can be employed to compare choices by general means, but a procedure with which to select a decision on the basis of these rules is not presented.

2.1.3 Prescriptive Decision-Making

A prescriptive decision-making tool or aide is designed to help a human user make better decisions, while not necessarily reflecting the processes that the user performs in their every-day decision-making or adhering to any axiomatic account of rational behaviour. Prescriptive guides can take the form of instructions, provided to a human user to follow in order to make good decisions. Alternatively, a prescriptive tool can be an automated system that follows these instructions and suggests a decision to a human user.

In Sections 2.1.1 and 2.1.2, MCDA has been considered from a normative and descriptive perspective. Not all MCDA methods, however, are descriptive or grounded in a normative theory of rational behaviour. In the ELECTRE methods of Roy (1991), each criterion that characterises choices generates an outranking relation over a finite set of decision alternatives. Not necessarily complete or transitive, this relation describes which

choices are at least as good as others with respect to the given criterion. For each pair of alternatives, some of these relations will infer that one is better than the other, some will infer the reverse, and some will infer indifference. The sets of criteria that agree (or disagree) with a given outranking relationship between two alternatives, and their importance, determine the degree to which this relationship is supported (or rejected). These numeric assessments are used to decide which relationships hold in a comphrehensive outranking relation over choices. Decisions are made on the basis of this relation.

These ELECTRE methods belong to the European school of decision-making [Roy and Vanderpooten (1996)], while approaches with a normative foundation such as MAUT [Keeney and Raiffa (1976)] and expected utility theory [von Neumann and Morgenstern (1944)] belong to the American school. Within the American school, there is a notion of a pre-existing axiomatically defined ideal form of behaviour which should be achieved or approximated by a decision-making tool. Approaches within the European school aim to create the 'conviction' that a decision is the best, where such an ideal is not pre-existing.

Within the PROMETHEE range of MCDA approaches [Brans et al. (1986)], preference is elicited from a user in terms of pair-wise comparisons. For each pair of available alternatives, a user must specify how they differ in evaluation with respect to each criterion. These differences are aggregated to determine the degree to which each alternative is preferred to the other. These degrees are then employed to construct a comprehensive outranking relation over the set of available choices. The process used varies across the different PROMETHEE techniques. In a similar vein, MACBETH [e Costa and Vansnick (1994)] elicits qualitative assessments of the relative attractiveness of alternative choice pairs on a set of criteria. These judgements, however, are used in the construction an additive utility model – a utility function for each criterion and its weighting (on the basis of importance) – revealing a normative purpose. Like ELECTRE, the spectrum of PROMETHEE methods lie within the European school of decision-making.

Argumentation-based approaches fall within the prescriptive partition of decision-making. In Section 3.6 of Chapter 3 a range of these techniques are described. Several of these argumentation-based decision-makers construct arguments in support of (pro) and/or against (con) the selection of available choices. These arguments are derived from

a series of criteria, properties or features of the alternatives in question – characteristics that lead to conflicting evaluations of which decisions are the most preferred[6]. In the method of Amgoud and Prade (2004a), the acceptable of these pros and cons are used to weigh each available decision against all others – for example, on the basis of the strength of each pro and weakness of each con. Preference representation within these approaches ranges from prioritised goals and values [Amgoud and Prade (2004a); Atkinson et al. (2006, 2005c); Atkinson and Bench-Capon (2007a)], abstract (absolute) representations [Amgoud et al. (2008b)], to comparison schemes [Ouerdane et al. (2008)].

The approaches described in this section each prescribe a method of preference expression, ranging from prioritised criteria in MCDA tools to pair-wise judgements on relative attractiveness. In Chapter 3, a range of existing argumentation-based decision-making methods are evaluated with respect the questions of Criterion 2.1.1. The range of techniques for representing and handling preference used within these prescriptive decision-making approaches is diverse. In the following section a survey of preference representation within the context of automated decision-making is presented.

2.2 Preference Representation – A Survey

I consider in this section the ceteris paribus preferences of Wellman and Doyle (1991), Doyle and Wellman (1994), and Boutilier et al. (2004, 1999); the temporal logic formulae of Bienvenu et al. (2006) and Son and Pontelli (2004); the logical language of comparatives of Delgrande et al. (2007); the value-based schemes of Atkinson et al. (2005c, 2006); the reasoning patterns of Ouerdane et al. (2008); and the goal and criteria-based evaluations of Amgoud and Prade (2004a) and Amgoud et al. (2005b), amongst others.

I demonstrate that each of these representations have both strengths and limitations. I do not consider the expected utility functions of von Neumann and Morgenstern (1944), and the rule or heuristic-based approach to capturing decision-maker behaviour. These ideas have been described earlier in this chapter (Sections 2.1.1 and 2.1.2).

[6] As an aside, MCDA problems are natural candidates for an argumentation-based treatment. Each criterion and the ranking it generates can form the basis of arguments constructed for and against the selection of a choice. A variety of MCDA approaches incorporate argumentation-based elements in the form of such bipolar arguments [Dubois et al. (2008); Grabisch et al. (2008); Tsoukiàs et al. (2002)].

2.2.1 Ceteris Paribus Preference

Ceteris Paribus is Latin phrase, meaning 'all things being equal'. A preference with a ceteris paribus interpretation expresses an inclination toward certain properties over others where all unmentioned properties are the same in each choice under comparison [Wellman and Doyle (1991)]. The CP-nets formalism of Boutilier et al. (2004, 1999) defines a graphical preference network composed of such ceteris paribus preference statements.

Definition 2.2.1 A ceteris paribus preference in the CP-nets approach is a propositional expression of the form $\phi : a_1 \succ a_2$ denoting that a choice in which the proposition a_1 holds is preferred to that in which a_2 holds if the formula ϕ holds and 'all else is equal'.

Example 2.2.1 Consider the preference *hot* : *water* $\succ$ *coffee* over beverages and two choices that share the same attributes (except for the values of *water* and *coffee*, which describe whether water or coffee is drunk by the decision-maker). The alternative in which *water* holds is preferred to that in which *coffee* holds (assuming the weather is *hot*).

CP-nets represent, in the form of a network, which variables within a domain influence a preference over others. These nets are used to both find an optimal assignment to a set of variables, and to determine when one assignment is preferred to another. To achieve the former the net is traversed from parents to children, assigning to each variable its most preferred value given the assignments of its parents. To achieve the latter, a strategy of flipping sequences can be employed – in which an assignment Φ_1 is preferred to another Φ_2 if Φ_2 can be reached from Φ_1 by a series of worsening flips, each altering a single variable assignment to form a less preferred assignment [Boutilier et al. (2004)].

An advantage of ceteris paribus preferences is that they are easily expressed by human decision-makers. It is often the case, as highlighted by Doyle and Wellman (1994), that we express preference in terms of comparatives with a ceteris paribus interpretation. There are some limitations associated with the CP-nets representation [Boutilier et al. (2004)], however. The propositional nature of these nets limits the expressivity of the preferences used in their construction. It cannot express, for example: preference over how variables change over time (temporal preference); properties of object classes (requiring predicate logic); quantitative preference; and priorities over preferences.

Several extensions of the CP-nets formalism have been developed to address some of these complaints (TCP-nets extend CP-nets with the ability to prioritise preferences over the values of propositions [Brafman and Domshlak (2002)], and FCP-nets incorporate the use of objective or metric functions [Gavenelli and Pini (2008)]). CI-nets [Bouveret et al. (2009)] represent a further extension or relative of the CP-net formalism. CI-nets define preference networks composed of preference statements over sets of items or goods (for example, "I would prefer the {Mario Kart, Zelda} Wii bundle over the {Mario Galaxy, Wii Play} bundle"). The mechanisms employed to reason with CI-nets are similar to their counterparts in the CP-nets formalism, and are described by Bouveret et al. (2009).

2.2.2 A Prototypical Preference Logic

Bienvenu et al. (2010b) describe a prototypical preference logic (PL) designed to combine features of existing representations, including the CP-nets formalism of Section 2.2.1. The building blocks of the logic are statements that describe preference over propositional formulae – each associated with a set of formulae whose interpretations must be fixed across each choice under comparison. These statements can then be combined through the use of boolean connectives ($\neg$, $\vee$ and $\wedge$) to produce preference formulae.

Example 2.2.2 Consider two propositional formulae α and β. Preference for a choice satisfying the formula α over a choice that satisfies β is described in PL by the expressions: $\alpha \rhd \beta || F$ (strict) or $\alpha \trianglerighteq \beta || F$ (non-strict), where F is a set of propositional formulae whose interpretations do not differ between the two choices under comparison.

Bienvenu et al. (2010b) demonstrate that fragments of this logic represent the CP-nets [Boutilier et al. (2004)], CI-nets [Bouveret et al. (2009)], and prioritised goal base [Benferhat et al. (1993)] formalisms. Prioritised goal bases are described in Section 2.2.4.

2.2.3 Preferences in Temporal Logic

Bienvenu et al. (2006, 2010a) describe a temporal preference representation (denoted $\mathcal{LLP}$) that extends the language $\mathcal{PP}$ [Son and Pontelli (2004)]. This representation is applied in decision-making scenarios where the available choices are courses of action.

The $\mathcal{PP}$ formalism of Son and Pontelli (2004) describes a hierarchy of preference constructs: basic desire (or trajectory property) formulae outlining temporal logic constraints on plan trajectories (for example, "sometime before eating cake, I'd like to drink water"); atomic preferences defining an ordering over basic desires according to their importance (for example, "eventually drinking water" is more important than "eventually drinking beer"); and general preference formulae that combine atomic preferences with operators analogous to $\wedge$, $\vee$, and $\neg$. Bienvenu et al. (2006) extend $\mathcal{PP}$ with additional (lexicographic) mechanisms of preference combination and assign weights to basic desires in atomic preference formulae. These weights describe the degree to which a formula is more important than others, and are used to compute a numeric value for each available choice – an aggregation of the weights of the preferences it satisfies.

This temporal logic preference representation is in some respects more expressive than CP-nets [Boutilier et al. (2004)]. Preferences over the temporal aspects of plans can be expressed, together with relationships over and properties of object classes. It cannot, however, express comparative preference (an advantage of the CP-nets approach), or those involving metric functions such as total cost or time. Preferences in the work of Bienvenu et al. (2006, 2010a) are limited to those that can be evaluated with respect to a single choice, and are either satisfied or violated. $\mathcal{LLP}$ is used in the PPLAN planner [Bienvenu et al. (2006)] and the preferred diagnoses generator of Sohrabi et al. (2010).

2.2.4 Prioritised Goals, Values, and Criteria

The use of prioritised goals is prevalent in decision-making applications, theories, and preference logic formulations. Sardiña and Shapiro (2003) use a prioritisation over agent goals in the definition of a rational search operator within the INDIGOLOG language (Section 5.5 of Chapter 5). Benferhat et al. (1993) employ a prioritisation over belief bases for the purpose of finding preferred consistent sub-theories or bases of their inconsistent parents. Brewka (2004) describes a preference representation language defined in terms of a prioritised belief and goal base for the purpose of model selection.

In Chapter 3, a range of argumentation-based tools for practical decision-making (decision-making over action) are described that involve the use of prioritised values.

In the work of Atkinson et al. (2006, 2005c), the performance of an action realises a goal that in turn promotes a value. Arguments are constructed in support of actions, highlighting the values they promote. The ranking a decision-maker holds over these values plays an integral role in the acceptance or rejection of these arguments and in the ultimate selection of an action. Atkinson and Bench-Capon (2007a,b) provide an alternating transition semantics for the formalism of Atkinson et al. (2006, 2005c) in which actions cause transitions from one state to another while promoting or demoting a value.

In the argumentation-based decision aide of Amgoud and Prade (2004a), a prioritised goal base provides the content of arguments in support of and against the selection of an action to perform. These arguments highlight the priorities of goals that are achieved and unfulfilled, respectively, by each action. Similar prioritised goal bases are employed in the work of Amgoud and Prade (2006) and Amgoud et al. (2005a). In many works, goal bases are prioritised in a bipolar fashion in which desirable goals are distinguished from goals to be avoided. Where choices are evaluated with respect to criteria, a degree of satisfaction beyond a neutral point is desired, while that below is not (in this case the criteria are dissatisfied). Amgoud et al. (2005b) employ this method of choice evaluation. Benferhat et al. (2002, 2006) consider the distinction between positive preferences (aspirations) and negative preferences (things to be rejected) – expressed in possibilistic logic – and the processes by which they can be aggregated in the evaluation of a choice.

Prioritised goals, values and criteria are simple in their representation and require little effort in their elicitation. Many techniques within the domain of multi-criteria decision analysis (MCDA), described in Section 2.1, operate on the basis of weightings assigned to such criteria, highlighting their relative importance. This form of representation, however, is less expressive than the earlier described CP-nets formalism of Boutilier et al. (2004) and the temporal logic preferences of Bienvenu et al. (2006, 2010a).

2.2.5 A Logic of Comparatives

Delgrande et al. (2007) express preference in terms of comparison formulae over plans (courses of action). Formulae are composed in a (first order) logical language that combines queries (over plans) with logical connectives (such as $\wedge$ and $\vee$). Each query is built

from fluents and atoms that describe the properties of a plan at various time points, in conjunction with logical connectives, arithmetic symbols, and quantifiers.

A preference for one plan over another exists, with respect to a preference formula Φ, if and only if Φ holds. The formula Φ evaluates a series of queries with respect to each plan during its interpretation, as shown in Example 2.2.3.

Example 2.2.3 Imagine that plans (or action histories) are compared as follows [Delgrande et al. (2007)]. An action history H_h is preferred to a history H_l if a fluent f holds at some time point i during H_h but never holds during the course of H_l. This preference is expressed by the formula: $\Phi = (\mathbf{h} : \exists i f(i)) \wedge (\mathbf{l} : \forall i \neg f(i))$, where $f(i)$ holds if fluent f holds at time point i and the labels **h** and **l** refer to the histories H_h and H_l (ie. given $(\mathbf{h} : \phi)$, the formula ϕ is evaluated with respect to the action history **h**).

These formulae are able to model the preferences of $\mathcal{PP}$ [Son and Pontelli (2004)], but do not consider prioritisation over preference formulae as explored in Bienvenu et al. (2006, 2010a). The representation of the latter is more expressive in its ability to combine preference formulae in different manners (lexicographically and conditionally, for example), but the formulae of Delgrande et al. (2007) allow the comparison of plans by metric functions and the expression (to a certain extent) of comparative preference.

Despite the aim of developing a representation for general or arbitrary preferences, there are comparative rules that cannot be expressed by the preference formulae of Delgrande et al. (2007). One example is the majority rule [Tversky (1969)]. This rule infers a preference for one plan over another if the former is rated better on a majority of the attributes that characterise these plans. Such a comparison cannot feasibly be expressed by a logical formula that does not allow quantification over more than time points.

2.2.6 Comparison Schemes and Abstractions

Ouerdane et al. (2008) consider a variety of schemes with which to compare choice pairs – each scheme describing a particular method of representing and reasoning with the preferences of a decision-maker. Schemes developed by Ouerdane et al. (2008) include a majority principle scheme, and a lexicographic scheme (see Examples 2.2.4 and 2.2.5).

Example 2.2.4 When given a series of equally important criteria, and a choice pair, a majority scheme asserts a preference for the choice that is better on a majority of criteria.

Example 2.2.5 A lexicographic scheme operates on a linearly ordered set of criteria and selects one of a pair of choices as preferred following a lexicographic comparison. This scheme compares choices on the most important criterion first, and proceeds to less important criteria if and only if both choices are equally good with respect to it.

The approach of Ouerdane et al. (2008) is a shift away from decision-making aides that employ a language for the representation of preference. Here, preferences are encapsulated in schematic reasoning patterns that can be defined using a variety of different methods of choice comparison. These schemes can use one or more of the preference representations described in the body of literature on preference handling – which are chosen and how they are employed can be varied. Ouerdane et al. (2008) propose an argumentation-based decision-making aide in which these schemes are used to generate arguments in support of the selection of one choice over another. Questioning of these arguments through the use of critical questions (as in the work of Atkinson et al. (2005c, 2006)[7]) and interaction with a human user is considered as a means of selecting a preferred choice. This process is preliminary, however, and not fully specified[8].

The schemes of Ouerdane et al. (2008) have not been applied within a decision-making tool that abstracts away from their contents (answering question two of Criterion 2.1.1 in the negative). A number of argumentation-based techniques operate on arguments as abstract entities. Amgoud et al. (2008b) present a method that encapsulates reasons to select a choice (on the basis of user preference) into arguments that support its selection. The abstract framework of Dung (1995) is applied to find the acceptable subset of these arguments, and consequently which choices are the best. This approach, and a range of other argumentation-based methods, are described in Chapter 3. As shall be discovered in Chapter 3, these arguments can capture only absolute preference in the support of a choice selection. These arguments are constructed with respect to individual choices and their properties, and manipulated as reasons to select these choices above all others.

[7] This work is described in greater detail within Section 3.6.4 of Chapter 3.
[8] A justification of this statement in provided Section 3.6.4, where this formalism is described.

Comparative style preferences, such as those considered in Sections 2.2.1 and 2.2.5, can be employed to justify the selection of one choice only over certain others.

2.3 Concluding Remarks – Toward An Ideal Representation

The schemes of Ouerdane et al. (2008) are perhaps an ideal method of capturing preference (while not being an explicit representation). These schemes describe how a pair of decision alternatives can be compared to determine which is preferred. Each of the different styles of preference described in this chapter can be used in a scheme that compares two choices. Such schemes, to my knowledge, have not been employed within a decision-making tool that: is able to select one of a set of choices; and abstracts away from the content of these schemes. The techniques considered in this chapter that manipulate abstract structures during decision-making – argumentation-based methods are prime examples – restrict these structures in ways that prevent the use of these schemes.

The combination of tools for abstract argumentation and comparison schemes that capture diverse means of choice comparison is a promising avenue for the development of decoupled systems for automated decision-making. For this reason, I have employed comparison schemes as the manner in which preference is described within an argumentation-based decision-making framework (presented in Chapter 4) – a framework that has been designed to represent a decoupled approach. Before introducing this system, I introduce its foundation – the field of computational argumentation. The chapter that follows presents an introduction to this field, paying particular attention to existing argumentation-based approaches for decision-making and planning with preferences. I explain that not one of these approaches represents a decision-making tool that is decoupled from the preferences supplied to it (as per Criterion 2.1.1).

Chapter 3
Computational Argumentation

Truth springs from argument amongst friends

David Hume

AT the heart of human intelligence is our ability to reason and argue. Humans use argumentation, both individually and collaboratively, to make the best decisions in the face of many alternatives. Day-by-day, decision-by-decision, we present reasons or arguments for making one choice over an alternative. These arguments form a case or support for that choice and they convince us to make our decision one way or the other. In the presence of uncertainty and conflicting information, we use argumentation to determine the most likely conclusion. In the absence of quantitative fact, we use argumentation to analyse the qualitative information at hand. Argumentation is a natural human ability, but a foreign concept to an artificial computational entity – a computer.

It has become increasingly important to impart our unique decision-making and reasoning capabilities to a computational entity – an intelligent agent. In the field of computational argumentation, techniques are developed that allow an agent to make decisions through the generation and analysis of arguments. The field received its legs at a time when classical techniques of deduction were reaching their limits when applied in complex environments. In these environments information is incomplete (new information is constantly arriving) and there are no hard truths but only uncertain beliefs.

In the 1980s and 90s, non-monotonic reasoning became a hot topic. No longer bound by the shackles of classical logics, non-monotonic formalisms (such as Reiter's default logic [Reiter (1980)], Moore's autoepistemic logic [Moore (1984)], and McCarthy's cir-

cumscription [McCarthy (1980)]) allowed inferences to be revised in the presence of new information. In a similar vein, truth maintenance systems [Doyle (1979)] conducted belief revision but with an argumentation-based flavour. In tangent to this development in non-monotonic reasoning, the use of argumentation in the study of artificial intelligence was taking shape. Birnbaum et al. (1980) described a computational process for the construction of argument graphs – graphs that highlighted support and dispute between propositional nodes. The work of Pollock (1987, 1990, 1992, 1994, 1995) brought existing philosophical work on defeasible reasoning and models of argument (for example, see [Toulmin (1958); Pollock (1974); Rescher (1977); Toulmin et al. (1979)] to name a few) into the realm of computational argumentation – describing processes and models of argument interaction, attack, and defeat. McGuire et al. (1981), Birnbaum (1982), Flowers et al. (1982), and Flowers (1982), described how the recognition of reasoning patterns within arguments allowed the construction of a relevant counterpoint or attack.

As the development of formal models of argument and argumentation continued, these ideas found many applications within tools for decision-support. A prominent example is the Oxford System of Medicine (OSM) – an expert system designed to aid general practitioners in the formation of diagnoses and the selection of treatments [Fox et al. (1987, 1990)]. On the basis of qualitative arguments formed in support of and against each of the candidate solutions identified for a diagnosis or treatment plan, this system analysed the relative merits of candidates to determine the most appropriate[1]. Within such applications was a need to reason in the presence of uncertainty, motivating the development of a logic of argumentation [Fox et al. (1992); Krause et al. (1995a)] in which arguments have a numeric or symbolic level of confidence, and an aggregation of such measures determines the conclusive force of arguments for or against a position.

Reflective of much of the existing work in argumentation-based decision-making, these ideas spawned a number of similar systems for decision-support in the biosciences: cancer diagnosis and therapy construction in the Bordeaux Oncology Support System [Renaud-Salis and Taylor (1990)]; carcinogenicity analysis of chemical compounds in

[1]The PRO*forma* language was subsequently developed to construct an executable model of this process – providing structures for the representation of argument, guidelines and protocols to be followed in the making of decisions, and an engine for the execution of these protocols [Fox et al. (1997)].

STAR [Krause et al. (1995b)]; and drug prescription in CAPSULE [Walton et al. (1997)].

The development of formal systems of argumentation, such as the defeasible inference system of Loui (1987), the abstract argumentation system of Lin and Shoham (1989), and the defeasible reasoning implementation of Simari and Loui (1992), presented a challenge to the existing range of formalisms for non-monotonic reasoning. Argument structures within the work of Lin and Shoham (1989) – consistent collections of facts, tree-structured arguments, and conclusions – reflected the extensions of a range of non-monotonic logics. In the mid-1990's, Dung published his seminal work on abstract argumentation frameworks [Dung (1993a, 1995)] – incorporating a notion of argument attack and acceptability not present in the earlier work of Lin and Shoham (1989). This work not only represented a non-monotonic approach, but was shown to encapsulate many of the popular non-monotonic formalisms of the day. Much of the current state-of-the-art in argumentation-based systems for intelligent agents employs the work of Dung (1995).

The following decade saw many extensions of the Dung (1995) framework, including the incorporation of preferences [Amgoud and Cayrol (2002a)] and values [Bench-Capon (2003a)] as a representation of argument strength. Early defeasible reasoning formalisms [Pollock (1992); Vreeswijk (1997)] presented an alternative to the abstract framework of Dung (1995) and its predecessors, while an analysis of argument structure and its influence in the argumentation process led to a variety of new frameworks [Verheij (1996a); Bondarenko et al. (1997)]. An exploration of argument interaction gave rise to contrasting theories of defeat, attack, and coordination [Krause et al. (1995a); Verheij (1996a); Pollock (2001); Prakken and Sartor (2002); Prakken (2005a)]. An uprising in the study of dialectical models of argumentation [Walton (1998); Prakken and Sartor (1998)], set in motion by the legal dispute generation of Ashley (1990), encouraged its use in the development of multi-agent systems. Argumentation has played a role in the design of effective negotiation between agents [Sycara (1989); Kraus et al. (1993, 1998); Parsons and Jennings (1996); Parsons et al. (1998); Tambe and Jung (1999); Rahwan et al. (2004)], flexible information access [Doutre et al. (2005)], and scientific inquiry [McBurney and Parsons (2001a)].

As we leave the 20^{th} century behind us and enter the 21^{st}, new and exciting branches of computational argumentation emerge. Meta-argumentation [Modgil (2007); Modgil

and Bench-Capon (2008)] pushed the boundaries established by existing formalisms of preference-based argumentation[2] by supporting defeasible preferences. The fusion of argumentation and game theoretic ideas [Rahwan and Larson (2008a); Bench-Capon et al. (2009); Rahwan et al. (2009)] brought strategy in argumentation to the forefront. The development of the semantic web resulted in a number of tools supporting online interactive argumentation [Rahwan (2008)] and exchange of information [Chesñevar et al. (2006)]. Conventional utility-maximising techniques for decision-making received an argumentation-based overhaul [Kakas and Moraitis (2003); Amgoud and Prade (2004a); Ferretti et al. (2008); Amgoud et al. (2008b)]. The combination of automated planning and argumentation, meanwhile, has produced novel techniques for reasoning about action in complex domains [Hulstijn and van der Torre (2004); Atkinson et al. (2005c, 2006); Rahwan and Amgoud (2006); Atkinson and Bench-Capon (2007a); Belesiotis et al. (2010)].

In this chapter each of these developments is discussed. Particular attention is payed to argumentation-based approaches for decision-making and planning – I evaluate each with respect to the criteria for decoupling of Chapter 2. As stated in Chapter 1, a decision-making (or planning) algorithm is decoupled from the preferences supplied to it if it operates on these preferences as though they are instances of an abstract type. For each of these decision-makers and planners, a set of questions is asked (Criterion 2.1.1) that consider their ability to: support general means of choice or plan comparison; and abstract away from the details of these comparisons during decision-making.

The reader is referred to the following reviews for further reading [Carbogim et al. (2000); Chesñevar et al. (2000); Rahwan (2005)].

3.1 The Rise and Rise of Non-Monotonic Reasoning

The desire to reason in the presence of incomplete information, and adapt results upon the discovery of new knowledge, has resulted in the development of non-monotonic reasoning formalisms – dominating the AI scene in the 1980s and 90s. Classical logic, which is monotonic in nature, was found to be inadequate for such a task. The monotonicity

[2]The term preference-based argumentation, as used here, refers to the preference-based argumentation frameworks (PAFs) of Amgoud and Cayrol (2002a). In this thesis, I use the term more generally to refer to systems that use preference over choices to form arguments in the selection of a best choice.

of classical logic requires that a statement inferred true by an information set is inferred true by all supersets – restricting adaptiveness to new knowledge.

3.1.1 Default Logic

In the early 80s, Reiter (1980) introduced a famous non-monotonic reasoning formalism known as default logic. In this formalism, knowledge consists of facts (such as "Tweety is a bird") and default rules (such as "Birds tend to fly") describing relationships that typically (but do not always) hold. These default rules allow conclusions to be inferred by default, but retracted when new information is received.

Default logic is used to find extensions of incomplete logical theories – consistent belief sets (maximal with respect to set inclusion) that include logical deductions and information derivable by default rules. A default rule consists of justifications and a conclusion. If these justifications are consistent with known facts and deduced information at the time of rule application then the conclusion is inferred. Default inferences are retracted if new information can be used to logically deduce inconsistencies with respect to the justifications of the default rule used. An incomplete logical theory can have several extensions [Pearl (1988)] – each resulting from a different order of rule application.

Default Logic, Argumentation, and Truth Maintenance

Contradictory beliefs can be present in different extensions of a logical theory. Techniques developed to decide which beliefs to put ones faith in have incorporated a distinct argumentation-based influence. Poole (1988), for example, views defaults as hypotheses that explain why a proposition should be believed. Brewka (1989) extends this work by ascribing varied levels of reliability to justifications, inferences, and default rules. A proposition that exists in all extensions of a theory is more reliable than one existing in at least one but not all extensions. This variation can be viewed as an early argumentation system. A default together with its justification represents a reason why we should accept its conclusion. In the presence of inconsistent conclusions across extensions, we accept the conclusion with the strongest (most reliable) supporting reason.

Prior to the publication of default logic [Reiter (1980)], Doyle introduced the truth

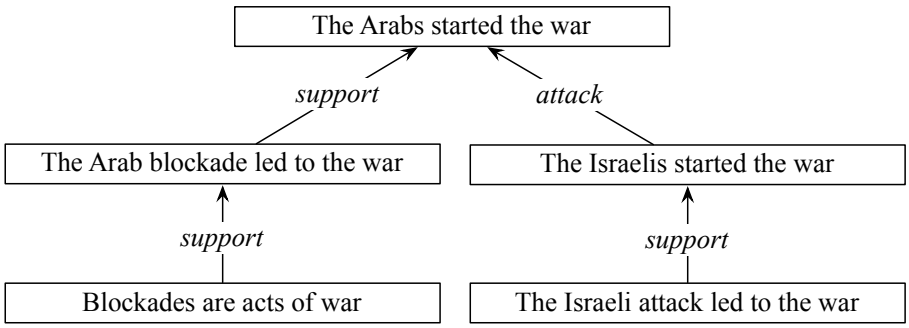


Figure 3.1: Initiation of the 1967 Arab-Israeli war [Birnbaum et al. (1980)].

maintenance system (TMS) [Doyle (1978, 1979)]. In a TMS, beliefs are associated with a *well-founded support* consisting of the facts, beliefs (derived information), assumptions (beliefs that have not been proved false), and rules used to derive them. Upon receipt of new information, a TMS considers the knowledge that can be deduced from it. If this derived information is inconsistent with an assumption in the support of a currently held belief, that belief is retracted to maintain consistency.

A TMS embodies the concept of constructing arguments in support of beliefs, and rejecting certain lines of reasoning when inconsistencies arise.

3.1.2 Early Argumentation Frameworks

While default logic [Reiter (1980)] and truth maintenance systems [Doyle (1978, 1979)] introduced belief revision with an argumentation-based flavour, models of computational argumentation were in the process of development. Birnbaum et al. (1980) presented an early model of argumentation in which an argument graph is constructed to represent the relationships of support and dispute (edges) that exist between a series of propositions (nodes). Figure 3.1 demonstrates an inference graph argument presented by Birnbaum et al. (1980) analysing the question of who started the 1967 Arab-Israeli war.

This tree-like representation of arguments is also used in the work of Pollock (1987), Loui (1987), and Lin and Shoham (1989). Pollock (1987) employed two modes of attack between arguments – undercutting and rebuttal – variations of which would often be used in later frameworks and techniques. Each argument is a collection of propositions (premises) that are linked to the propositions (conclusions) they infer. In the work of

Pollock (1987), an argument a_1 undercuts another a_2 if a_1 refutes an inference that occurs within a_2 – a link between a premise and a conclusion. The most common interpretation of undercutting used within the argumentation community, however, is the denial of a premise – an argument undercuts another by denying a premise within it. An argument a_1 rebuts another a_2 if a_1 is a reason for denying the conclusion of a_2 – reflecting the definition of rebuttal within the Toulmin (1958) model of argument (an 'exceptional condition' that refutes the conclusion of an argument). This formulation of defeasible argumentation formed part of the implemented OSCAR system [Pollock (1987, 1990)], and was further developed by Pollock in later work [Pollock (1994, 1992, 2001)].

3.2 The Abstract Argumentation System

At a time when computational argumentation techniques were taking off, the idea of abstracting away from the structure and content of an argument took hold. Lin and Shoham (1989) introduced an abstract argumentation system that, while continuing the tradition of graphical argument structures, did not enforce the adoption of a specific language for the representation of premises and conclusions. The seminal work of Dung (1995) viewed arguments as atomic elements, providing abstraction in both structure and language. The work of Dung (1995) focused on the semantics of argument acceptability.

The abstract argumentation framework of Dung (1995) consists of two key elements: a set of atomic arguments and a (neither symmetric nor asymmetric) binary attack relation over those arguments. An instantiation of the framework can be depicted as a (possibly disconnected) graph in which each vertex represents an argument, and each directed edge denotes an attack from one argument against another.

Definition 3.2.1 An argumentation framework is a pair $AF = \langle AR, \mathcal{X} \rangle$ where AR is a set of arguments, and $\mathcal{X} \subseteq AR \times AR$ is an attack relation over those arguments. An argument $a_1 \in AR$ attacks an argument $a_2 \in AR$ if and only if $(a_1, a_2) \in \mathcal{X}$.

Example 3.2.1 Consider the collection of abstract arguments $AR = \{a_1, a_2, a_3, a_4, a_5, a_6\}$ and the attack relation $\mathcal{X} = \{(a_1, a_2), (a_2, a_3), (a_3, a_4), (a_4, a_1), (a_5, a_6)\}$. A graphical depiction of the argumentation framework $\langle AR, \mathcal{X} \rangle$ is shown in Figure 3.2.

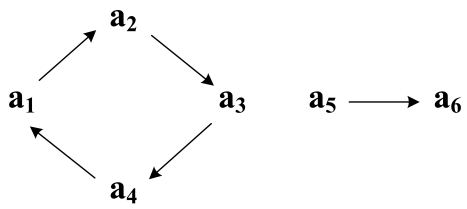


Figure 3.2: An example of a Dung (1995) argumentation framework.

Dung (1995) presents the concepts of conflict free and admissible sets, laying the foundation for a definition of argument acceptability. These two concepts are defined and demonstrated in Definition 3.2.2 and Example 3.2.2, respectively.

Definition 3.2.2 Consider an argumentation framework $AF = \langle AR, \mathcal{X} \rangle$:

1. A set of arguments $S \subseteq AR$ is conflict free if and only if there are no two arguments $a_1, a_2 \in S$ such that $(a_1, a_2) \in \mathcal{X}$;
2. An argument $a_1 \in AR$ is acceptable with respect to a set of arguments $S \subseteq AR$ if and only if for every $a_2 \in AR$ where $(a_2, a_1) \in \mathcal{X}$, there is an argument $b_1 \in S$ such that $(b_1, a_2) \in \mathcal{X}$ (in which case, we say that a_1 is defended by the set S);
3. A set of arguments $S \subseteq AR$ is admissible if and only if it is conflict free and each argument in S is acceptable with respect to S (given AR).

Example 3.2.2 Consider the argumentation framework defined in Example 3.2.1. There are many conflict free subsets of AR, including: $\{a_1, a_3, a_5\}$, $\{a_5, a_4, a_2\}$, $\{a_1\}$, and $\{a_5\}$. The admissible subsets of AR are: $\{a_5\}$, $\{a_1, a_3\}$, $\{a_2, a_4\}$, $\{a_1, a_3, a_5\}$, and $\{a_2, a_4, a_5\}$.

In default logic [Reiter (1980)] an extension of an incomplete logical theory (of facts and default rules) is a collection of derivations that can be believed simultaneously without being inconsistent or contradictory. Dung (1995) defines a variety of extension semantics for the abstract argumentation framework in a similar spirit.

Definition 3.2.3 Consider an argumentation framework $AF = \langle AR, \mathcal{X} \rangle$:

1. An admissible subset of arguments in AR that is maximal (with respect to set inclusion) is a preferred extension of AF (preferred semantics);
2. The characteristic function $F_{AF} : 2^{AR} \rightarrow 2^{AR}$ of AF is defined as follows:

$$F_{AF}(S) = \{A \mid A \in AR \text{ and } A \text{ is acceptable with respect to } S\}$$

3. The grounded extension of AF is the least fixed point of F_{AF} (grounded semantics).

A number of additional types of extension exist (such as complete and stable), and are highlighted by Dung (1995). Additional extension semantics defined in the argumentation literature since the work of Dung (1995) include: semi-stable [Caminada and Dunne (2006)]; ideal [Dung et al. (2007)]; and super-stable [Dimopoulos et al. (2009)]. For the purposes of this thesis, a knowledge of these semantics is not required.

Example 3.2.3 Consider the argumentation framework defined in Example 3.2.1. There are two preferred extensions of $AF = \langle AR, \mathcal{X} \rangle$. These extensions are $\{a_1, a_3, a_5\}$ and $\{a_2, a_4, a_5\}$. The grounded extension of the framework AF is $\{a_5\}$.

A preferred extension is a consistent position that cannot be extended without introducing a conflict. A grounded extension is a more sceptical position, in which we consider arguments not attacked (challenged) by others and the arguments they defend. There can be many preferred extensions of an abstract argumentation framework, but only one (possibly empty) grounded extension. These extension types define semantics (such as preferred, or grounded) under which argument acceptability is determined.

Definition 3.2.4 Consider an argumentation framework $AF = \langle AR, \mathcal{X} \rangle$, and its extensions E under a given semantics (such as preferred or grounded):

1. An argument $a_1 \in AR$ is credulously accepted under the given semantics if it appears in at least one extension in E (under that semantics);
2. An argument $a_1 \in AR$ is sceptically accepted under the given semantics if it appears in each extension of E (under that semantics).

Example 3.2.4 Consider the argumentation framework defined in Example 3.2.1 and its extensions, defined in Example 3.2.3. The arguments a_1, a_2, a_3, a_4, and a_5 are credulously accepted, while the argument a_5 is sceptically accepted under the preferred semantics.

Dung (1995) demonstrated that several popular non-monotonic reasoning formalisms of the day (default logic [Reiter (1980)] and defeasible reasoning [Pollock (1987)]) could be viewed as argumentation instantiated within this abstract argumentation framework.

3.3 Argument Structure and its Role in Attack

The abstract argumentation framework of Dung (1995) provides a general framework that can be instantiated with any choice of argument structure and notion of argument attack. Alongside this trend toward abstraction, argumentation techniques were in development that studied forms of attack based on the structure and content of arguments.

In the formalisms described within this section, the acceptability of an argument is dependent on its strength relative to the arguments that attack it. The strength of an argument is characterised in terms of its structure and content. In Section 3.4, I examine how the framework of Dung (1995) can be instantiated so that the strength of arguments plays a role in determining their acceptability. The following sections consider the use of rules in argument construction (Section 3.3.1) and show how the structural properties of an argument are used to define relative argument strength (Section 3.3.2).

3.3.1 Rule-Based Argument Construction

The most general method of representing arguments is as deductions, or sequences of rule applications [Bench-Capon and Dunne (2005)]. A number of early systems employ this view. Prakken and Sartor (1997) define arguments as sets of strict and defeasible rules. A strict rule is an indisputable relationship, while a defeasible rule is a relationship that does not always hold. In the work of Loui (1987) and Verheij (1996a), defeasible rules are linked together to form arguments in support of a conclusion. Vreeswijk (1997) proposes a system of argumentation in which strict and defeasible rules are chained together to create tree-structured arguments. The assumption-based argumentation framework of Bondarenko et al. (1997) is defined with respect to a system of deduction – a logical language and a set of rules. Arguments are sets of assumptions that, given these rules and a base set of facts, infer a conclusion by classical deduction or inference.

Simari and Loui (1992) describe a formal system for reasoning with defeasible rules – subsets of which form argument structures that infer conclusions through defeasible derivation. Structures in disagreement (arguments that collectively infer a contradiction) are compared on the basis of specificity as per the definition of Poole (1985). In the extension of this work by Simari et al. (1994), arguments are justified or accepted on the basis

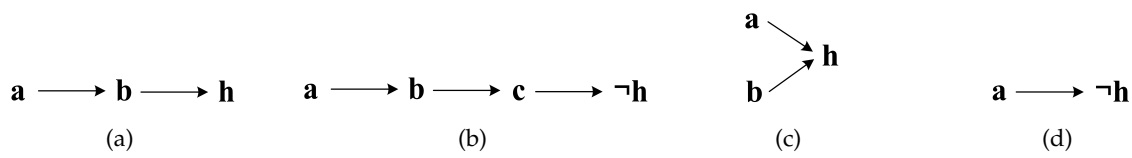


Figure 3.3: Four arguments in the framework of Loui (1987).

of a dialectical process – the construction of a tree of arguments, the branches of which represent lines of alternating supporting and opposing arguments[3]. These ideas in turn evolved to form part of the defeasible logic programming (DeLP) approach of García and Simari (2004) – this work supporting the declarative representation of strict and defeasible rules (with negation) in the form of logic programs, and the justification of arguments (warrant) through the construction of dialectical argument trees[4].

3.3.2 Structure and Strength

In the work of Loui (1987), the structure and content of a tree-structured argument defines its strength relative to others. Given two arguments in support of contradictory conclusions, the argument that is less direct or less specific is less strong. An argument is less direct than another if it involves longer chains of rule application and a superset of premises (the argument in Figure 3.3(b) is less direct than that of Figure 3.3(a)). An argument is less specific than another if it shares rules with the same conclusion, but a subset of premises (the argument in Figure 3.3(d) is less specific than that of Figure 3.3(c)).

In the work of Vreeswijk (1997), arguments have a length (depth of their tree), a size (number of nodes in their tree), and sub-arguments (the sub-trees of their tree). The relative strength of these arguments (constructed using strict and defeasible rules) is defined by an *order of conclusive force*. Vreeswijk (1997) considers several strategies in the definition of conclusive force, each of which consider the structure of the arguments being compared – from the number of defeasible rules used in their construction to their length and size. The work of Pollock (2001) assigns degrees of justification to the premises in

[3] A discussion concerning dialectical argumentation, the construction of dialectical argument trees, and its use within multi-agent systems is presented in Section 3.5.

[4] The DeLP approach of García and Simari (2004) is discussed further in Section 3.6.3 within the context of the work of Ferretti et al. (2008).

an argument. These degrees define its strength. Similarly, Krause et al. (1995a) use the uncertainty of information present in an argument to assess its strength. In earlier work, Elvang-Gøransson et al. (1993a,b) use the relationships of undercutting or rebuttal that exist between arguments, formed on the basis of argument structure, to separate arguments into varying levels of acceptability. Arguments not open to rebuttal or undercutting are more acceptable than those that are merely consistent collections of information. Degrees of certainty or strength – such as certain, confirmed, and plausible – are attributed to propositions on the basis of their membership within these classes.

Often there are many arguments that support the truth of a conclusion (or its negation). In a variety of formalisms, these arguments are grouped into an accrual and treated as a single argument [Prakken (2005a)] . Krause et al. (1995a) consider the supremum of the strengths of supporting arguments of a proposition to represent the probability that it is provable. Pollock (2001) considers how the degrees of justification (strengths) of constituent arguments determine the strength of their combination. Additional frameworks that accrue (combine) arguments and compare the result in the selection of which propositions to accept include Reason-Based Logic (RBL) [Hage (1995); Verheij et al. (1998)], CumulA [Verheij (1996a, 1995)], the subjective logic framework of Oren et al. (2007a), and the defeasible logic programming systems of Lucero et al. (2009a,b).

3.4 Early Dung-Based Frameworks

The framework of Dung (1995), in its basic form, does not consider how an attack relation amongst arguments is arrived at, or how differing argument strengths influences its formation. A number of more recent works have proposed strategies for constructing such an attack relation amongst arguments that vary in strength or persuasiveness. Amgoud and Cayrol (2002a) present a preference-based framework in which a preference relation over arguments determines which attack relationships hold. Bench-Capon (2003a) propose a framework in which arguments promote values of varying importance – these values, like the preferences of Amgoud and Cayrol (2002a), govern the formation of an attack relation amongst arguments. These two frameworks lay the foundation for a number of argumentation-based decision-making systems (described in Section 3.6).

In much existing work, conflict amongst arguments is modelled by an attack relation while the combination of conflict and strength forms one of defeat. This thesis employs a different convention – conflict between arguments is captured in a conflicts relation while conflict combined with strength is captured in one of attack. An attack relation is viewed in the same light as Dung (1995), as a relation that abstracts away from the use of argument strength in its characterisation. That an argument is in conflict with a weaker argument does not infer that the latter is ultimately defeated (not accepted) – hence, I believe the term 'attack' is more appropriate than 'defeat' in describing conflict and strength. The existing work within this chapter is described using this convention.

3.4.1 Preference-Based Argumentation

Amgoud and Cayrol (1998, 2002b,a) define a framework in which a preference relation over arguments plays a key role in determining argument attack and acceptability.

Definition 3.4.1 A preference-based argumentation framework (PAF) is a triple, denoted by $AF = \langle AR, \mathcal{C}, PREF \rangle$, where:

1. AR is a set of arguments;
2. $\mathcal{C}$ is a (neither symmetric nor asymmetric) conflict relation over the arguments in AR, such that $(a_1, a_2) \in \mathcal{C}$ if and only if $a_1, a_2 \in AR$ and a_1 conflicts with a_2;
3. $PREF$ is a partial or total (transitive and reflexive) preference relation over the arguments in AR such that for any two arguments $a_1, a_2 \in AR$, $(a_1, a_2) \in PREF$ if and only if argument a_1 is at least as preferred as argument a_2;
4. $PREF_s$ is the strict ordering associated with $PREF$;
5. $\mathcal{R}$ is an attack relation on AR, defined such that $(a_1, a_2) \in \mathcal{R}$ if and only if $(a_1, a_2) \in \mathcal{C}$ and $\neg(a_2, a_1) \in PREF_s$ (a_2 is not strictly preferred to a_1).

The extensions of AF under a given semantics (Definition 3.2.3) are equivalent to those of the Dung (1995) framework $\langle AR, \mathcal{R} \rangle$ under those same semantics.

Example 3.4.1 Consider the Dung argumentation framework $\langle AR, \mathcal{X} \rangle$ of Example 3.2.1. Let $\langle AR, \mathcal{X}, PREF \rangle$ be a preference-based argumentation framework with $PREF = \{(a_1, a_2), (a_4, a_3), (a_5, a_6)\}$. The relationships of attack amongst arguments are shown in Figure

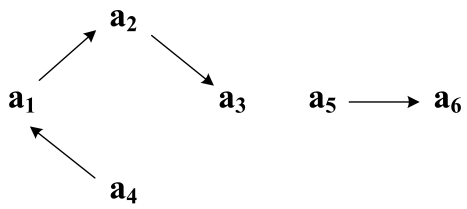


Figure 3.4: Attack relationships in a PAF [Amgoud and Cayrol (2002a)].

3.4. The framework $\langle AR, \mathcal{X}, PREF \rangle$ has a single preferred extension $\{a_2, a_4, a_5\}$.

PAFs have been used to reason about preferences that hold in contexts [Amgoud et al. (2000)], adapted to ensure strict acyclicity in their attack relations [Kaci et al. (2006)], used to merge inconsistent knowledge bases [Amgoud and Kaci (2007)], analysed to elicit their computational properties [Dimopoulos et al. (2008)], and generalised to include value-based frameworks [Bench-Capon (2003a,b)] as a special case [Bourguet et al. (2010)].

3.4.2 Value-Based Argumentation

Bench-Capon (2003a,b) defines value-based argumentation frameworks (VAFs) – in which the acceptability of an argument depends on the importance of the moral value it promotes, relative to those promoted by its conflicting arguments. A VAF considers the notion of persuadability in argumentation, and its dependence on the audience being subjected to it [Perelman (1980)]. Each audience has an individual preference ordering over values, and arguments are evaluated by an audience using this ordering. Let $\mathcal{P}$ denote a set of audiences. Definition 3.4.2 describes an audience-specific VAF (AVAF).

Definition 3.4.2 An audience-specific value-based argumentation framework (AVAF) is a 5-tuple, denoted by $AF = \langle AR, \mathcal{C}, V, \eta, \mathcal{I}_\alpha \rangle$, where:

1. $\alpha \in \mathcal{P}$ is an audience from a set of audiences $\mathcal{P}$;
2. AR and $\mathcal{C}$ are defined as in Definition 3.4.1, and V is a non-empty set of values;
3. $v = \eta(a_1)$ denotes the value $v \in V$ promoted by an argument $a_1 \in AR$;
4. $\mathcal{I}_\alpha$ is a partial or total (transitive, irreflexive, and asymmetric) preference ordering over V for audience α;
5. For $v_1, v_2 \in V$, $(v_1, v_2) \in \mathcal{I}_\alpha$ denotes that audience $\alpha \in \mathcal{P}$ prefers value v_1 to v_2;

6. $\mathcal{R}$ is an attack relation on AR, such that $(a_1, a_2) \in \mathcal{R}$ if and only if $(a_1, a_2) \in \mathcal{C}$ and $(\eta(a_2), \eta(a_1)) \notin \mathcal{I}_\alpha$ (a_2 does not promote a more important value).

The extensions of AF under a given semantics (Definition 3.2.3) are equivalent to those of the Dung (1995) framework $\langle AR, \mathcal{R} \rangle$ under those same semantics.

The acceptability of an argument in a VAF relies upon its acceptability for each audience in $\mathcal{P}$. An argument is credulously accepted if it is accepted by at least one audience within the framework. It is sceptically accepted if it is accepted by all audiences.

VAFs have been applied in the modelling of legal cases [Wyner et al. (2007)], the development of action selection [Nawab et al. (2008); Atkinson et al. (2005c, 2006)] and decision-making mechanisms [Bench-Capon et al. (2009)], analysed to elicit their computational properties [Dunne and Bench-Capon (2004)], used within hierarchical [Modgil (2006a)] and meta-argumentation frameworks [Modgil and Bench-Capon (2008)], and adapted to support multiple-valued arguments [Kaci and van der Torre (2008)].

3.5 Dialectics and Multi-Agent Systems

Alongside an increasing focus on the acceptability, logical structure and interaction of arguments within the computational argumentation community, a body of work on the process of argumentation and its dialectical nature was beginning to flourish. A limitation of the former pipeline approach to argumentation is its dissimilarity with the process by which humans argue. The latter development viewed and modelled argumentation as a dialogue, rather than a process of constructing arguments and finding extensions.

In the early 90s, argumentation dialogues were used to develop a program (HYPO) that generated legal disputes given past case information [Ashley (1990)]. Armed with a case in question and information about past cases and precedents, HYPO presented 3-ply arguments – an argument for either side of the dispute (in support of a claim, given a supporting precedent), a response from the opponent, and a rebuttal of this response. This work, extended in later years [Aleven (2003); Branting (2003)], influenced the formation of computational characterisations of dialectic argumentation. The interested reader is referred to Hage (2000) for a review of early dialectical models of argumentation.

Prakken and Sartor (1998) present a dialectical proof theory for testing the acceptability of a proposition – combining their earlier work [Prakken and Sartor (1996b)] with the dialogue moves of HYPO. A dialectical proof theory is designed to test the acceptability of a claim by determining if an undefeated argument in support of it exists. Determining the acceptability of an argument proceeds with the construction of a proof tree. The argument in question forms the root of the tree (representing the position of a proponent), and its children are the arguments that attack it (representing the position of an opponent). Their children are the arguments that attack them, and the tree is constructed to encapsulate all possible attacks on the arguments in its branches. The root argument is undefeated if each leaf of the tree represents an argument of the proponent.

Prakken and Sartor (1998) were not the first (or last) to develop a dialectical proof theory. Verheij (1996b) compare the extension-based semantics of Dung (1995) with a staged approach to argumentation (in which arguments are presented in stages). Defeasible rule-based argumentation gets a dialectic treatment by Prakken (1999), while Dunne and Bench-Capon (2003) consider two-party argumentation-based disputes. A dialectic treatment of the abstract argumentation framework of Dung (1995) has led to a number of proof theories [see Jakobovits and Vermeir (1999); Cayrol et al. (2001); Devred and Doutre (2007); Dung et al. (2006); Thang et al. (2009)].

3.5.1 Argumentation and Agents

Philosophical treatments of dialogue have laid the foundation for the development of protocols to govern agent behaviour in multi-agent systems. Walton (1998) characterises dialogues as: information seeking, persuasion, negotiation, deliberation, inquiry, and eristic. In information seeking dialogues participants ask others for information. Persuasion sees participants attempt to persuade each other to accept a proposition. Negotiation involves multiple parties proposing offers and counter-offers until an offer is accepted by all. Deliberation is the negotiation over an appropriate course of action. In an inquiry participants collaborate to find the answer to a question, while eristic dialogues are quarrels.

Doutre et al. (2005, 2007) demonstrate the use of dialectic argumentation in information seeking where permissions are required to access information. In this work, an agent

is able to argue with the holder of information to justify its access. McBurney and Parsons (2000, 2001a,b,c) describe a dialogue of scientific inquiry, demonstrating the use of dialectic argumentation in qualitatively representing and handling uncertainty in experimental evidence, in the combination of evidence from different sources, and in the conclusions drawn from that evidence. Rahwan et al. (2004) describe how the use of argumentation in conjunction with bargaining in negotiation has the potential to improve the quality of deals achieved over bargaining-only methods. Their approach uses argumentation as a means of persuading participants to reorder their preferences over offers by providing them with information they were not aware of. The benefits of applying argumentation in negotiation dialogues has been expressed in earlier work (see [Sycara (1989); Kraus et al. (1993, 1998); Parsons and Jennings (1996); Parsons et al. (1998)]), and has since been a topic of active research [Rahwan et al. (2003); Karunatillake et al. (2005, 2009); Amgoud et al. (2007); Rahwan et al. (2007a); Sierra and Noriega (2002); Amgoud and Prade (2004b,c); Tambe and Jung (1999); Oren et al. (2007b); Dung et al. (2008)]. Sycara (1989) introduced the practice of using argumentation to modify the beliefs or intentions of agents during negotiation (to encourage cooperation) in the PERSUADER system.

The forms of dialogue delineated by Walton (1998) occur in many typical problem solving scenarios involving multiple parties, and provide a basis for the development of protocols for agent behaviour in multi-agent systems [McBurney et al. (2002, 2003); McBurney and Parsons (2002, 2004); Amgoud and Hameurlain (2006); Amgoud et al. (2006, 2002); Parsons et al. (2007); Parsons and McBurney (2003)]. Forms of dialogue beyond those of Walton (1998) are considered in the command, and chance discovery, dialogues of Atkinson et al. (2008) and McBurney and Parsons (2003).

3.6 Argumentation-Based Decision-Making

This section considers the use of argumentation as a tool for agent-based decision-making and planning. This section describes the early pro and con-based systems of Fox et al. (1992), Brewka and Gordon (1994), Bonet and Geffner (1996), Fox and Parsons (1997), and Parsons and Green (1999); the decision-making under uncertainty approach of Amgoud and Prade (2004a) and its descendents [Amgoud and Prade (2006); Amgoud et al.

(2005a,b, 2008b)]; the decision aiding process of Ouerdane et al. (2008); and an approach grounded in defeasible logic programming (DeLP) [Ferretti et al. (2008)]. Also discussed are the ideas that underpin a range of early argumentation-based decision-support systems, such as: the Oxford System of Medicine [Fox et al. (1987, 1990)]; the Bordeaux Oncology Support System [Renaud-Salis and Taylor (1990)]; the STAR system for carcinogenicity analysis of chemical compounds [Krause et al. (1995b)]; and the CAPSULE system for the recommendation of drug prescriptions [Walton et al. (1997)].

The development of argumentation-based planning techniques is discussed, particularly within the context of BDI (belief-desire-intention) [Rao and Georgeff (1991)] agents. Amgoud (2003) and Amgoud et al. (2008a) consider the problem of practical decision-making and the search for consistent collections of intentions, with a number of other works supporting a similar goal [Rahwan and Amgoud (2006); Hulstijn and van der Torre (2004); Rotstein et al. (2007)]. This section considers, in addition, the scheme-based formalisms for reasoning about action [Atkinson et al. (2006, 2005b,a,c); Bench-Capon et al. (2009)], together with the multi-agent iterative planning tool of Belesiotis et al. (2010).

For each of the decision-making and planning approaches described in this section, those that operate on the preferences of a user are critiqued. These approaches are subjected to the questions of Criterion 2.1.1. These questions consider whether each approach is able to support decision-making on the basis of general preference expressions (any representation that infers a preference for one entity or choice over another solely on the basis of their individual or relative properties), while abstracting away from the representational detail of these preferences. For each approach, I consider whether there is such a representation of preference it does not support, and if a knowledge of the content or type of supplied preference is required to select a decision. If both questions in Criterion 2.1.1 are answered in the affirmative, the approach is considered decoupled.

3.6.1 Early Argumentation-Based Decision-Making

Early work in argumentation-based decision-making [Bonet and Geffner (1996); Fox and Parsons (1997); Parsons and Green (1999)] has followed the lead of Fox et al. (1992) and Brewka and Gordon (1994). In this work, arguments in support of and against each deci-

sion are created and compared in the selection of a most preferred choice. This process is also employed in a range of early decision-support systems within the biosciences [Fox et al. (1987, 1990); Renaud-Salis and Taylor (1990); Krause et al. (1995b); Walton et al. (1997)]. Each of these approaches is described prior to a presentation of its evaluation.

Bonet and Geffner (1996) use argumentation to select a most preferred action given uncertain beliefs, prioritised positive (to achieve) and negative (to avoid) goals, and rules that infer the consequences of each action. The positive and negative goals achieved by an action form the basis of arguments for and against its selection. The strength of an argument is dependent on the likelihood of the rules, and the certainty of beliefs, used in its construction. A preference ranking over actions is determined as follows. An action a is better than an action b with respect to a positive goal g if there is a stronger reason for a achieving g than for b. Action a is better than b with respect to a negative goal g' if there is a stronger reason for b achieving g' than for a. If a is better than b with respect to a goal g, and b is not better than a on any higher priority goal, then a is better than b.

Fox and Parsons (1997) develop a framework for decision-making in which arguments highlight the expected (symbolic) positive or negative values promoted by actions. The arguments for and against each action are aggregated to determine its expected value – the action with the most desirable value selected as the best. Parsons and Green (1999) present an argumentation-based tool to reason about changes in the utility of a decision-maker in response to actions. This work is based on the qualitative probabilistic reasoners of Parsons (1998), designed to reason about changes in the probability of formulae.

Brewka and Gordon (1994) consider issue-based decision-making as part of the Zeno project [Gordon and Karacapilidis (1997)]. Zeno is a computational mediation system developed to aid users in the selection of a position (a problem solution or method of achieving a goal) to resolve an issue (a question to be answered or a goal to be achieved). Arguments highlight the propositional properties of each choice that lead the user to select or reject it. A qualitative value logic is used to express the relative value of properties. Arguments are compared, and decisions are made, on the basis of value maximisation.

The foundation of Zeno [Gordon and Karacapilidis (1997)] is the Issue Based Information System (IBIS) of Rittel and Webber (1973), which allows for the decomposition of

problems into: questions (or issues), answers to questions in the form of choices (positions), arguments for and against those choices, and preferences over the factors used in these arguments. The Zeno system facilitates multi-user argumentation, allowing questions to be raised over provided arguments and preferences. This iterative process constructs a tree in which the nodes are questions, positions, or preferences, and the branches are arguments. In a process that travels from the leaves of this tree to its root, arguments are balanced to determine which positions are winners and which are losers.

The Oxford System of Medicine (OSM) was an expert system designed to aid medical practitioners in the formation of diagnoses and the selection of treatments [Fox et al. (1987, 1990)]. In the formation of a diagnosis or the selection of a course of treatment, there are a range of candidates. Within this system, qualitative arguments are formed in support of and against each available candidate for a given problem. These arguments are generated from varying sources of knowledge and are aggregated to determine the relative merit of differing candidates. This process forms the core of a range of subsequent systems [Renaud-Salis and Taylor (1990); Krause et al. (1995b); Walton et al. (1997)]. A numeric or symbolic level of confidence is determined for each argument, an aggregation of which determines a score for each candidate. A decision is then selected on the basis of a comparison of these individual scores (for example, by maximisation).

An Evaluation of Early Argumentation-Based Decision-Making

The questions of Criterion 2.1.1 are now answered with respect to the approaches described in this section. The first question asks whether each approach is able to support the range of preference representations that can be defined in the comparison of two choices, while the second requires that it abstract away from their representational detail.

Each formalism described within this section prescribes a method of preference expression. In the work of Bonet and Geffner (1996), prioritised positive (to achieve) and negative (to avoid) goals encapsulate the preferences of an agent or human user. Fox and Parsons (1997) employ (symbolic) value functions to determine the benefit to a decision-maker of particular decision outcomes. Parsons and Green (1999) also capture the desirability of actions in terms of the value of their consequences – in this work, focus is placed on the change in value or utility that an action produces. Brewka and Gordon (1994)

employ a qualitative value logic to define the relative values of propositional choice properties with the aim of selecting a choice that maximises value. Fox et al. (1992) and the described range of biomedical-based decision-support tools [Fox et al. (1987, 1990); Renaud-Salis and Taylor (1990); Krause et al. (1995b); Walton et al. (1997)] use arguments to determine a numeric or symbolic score for each available choice – the comparison of these scores (by maximisation, for example) guides decision-making. Hence, in each of these approaches there are types of preference that are not supported – ceteris paribus comparatives (Section 2.2.1), rules of thumb, and comparison schemes (Section 2.2.6) are a few. The algorithms underlying each approach are tailored to the form in which preference has been expressed. Neither of the two questions are answered in the affirmative.

It should be noted that this evaluation, and those that accompany each approach discussed in this chapter, seek only to determine whether these existing works are decoupled (according to the guidelines expressed in Criterion 2.1.1). These evaluations do not expound upon the relative merits of these approaches outside of this criterion.

3.6.2 Decision-Making with Preference-Based Argumentation

This early work preceded many techniques that use arguments for (pro) and against (con) decisions as a basis for decision-making. Amgoud and Prade (2004a) develop an argumentation-based approach for decision-making under uncertainty – relying on the construction of arguments for and against available decisions.

Within this approach, two criteria (optimistic and pessimistic) are used in the selection of a decision. Under the optimistic criterion, a decision is preferred to another if the strongest reason against it is weaker than each reason against the other. Under the pessimistic criterion, a decision is preferred to another if there is a reason in support of it that is stronger than those in support of the other. This framework is described in detail within this section. This is followed with a look at a range of its variations, and I conclude with an evaluation of these works with respect to the questions of Criterion 2.1.1.

A Preference-Based Decision-Making Framework

A decision-making agent has a propositional belief and goal base, $\mathcal{K}$ and $\mathcal{G}$, defining what it believes to be true and the goals it desires to achieve. Each sentence in $\mathcal{K}$, and each goal

in $\mathcal{G}$, is respectively associated with a certainty level and a priority – each a member of a finite linearly ordered scale (with a top element of 1 and a bottom element of 0). This knowledge base allows the agent to construct arguments for and against available choices on the basis of the goals they achieve or neglect (Definitions 3.6.2 and 3.6.3).

The knowledge base of an agent is not assumed to be consistent – contradictory beliefs can be inferred and used in the construction of arguments for and against decisions. To determine which of these arguments are based on an accepted collection of consistent beliefs, an agent constructs arguments for and against the beliefs it can derive from $\mathcal{K}$. Definitions 3.6.1 to 3.6.3 are extracted from the work of Amgoud and Prade (2004a), where $\vdash$ denotes classical inference, $\bot$ a contradiction, $\mathcal{O}$ an available choice set, and $\mathcal{G}^*$ and $\mathcal{K}^*$ the goals and knowledge in $\mathcal{G}$ and $\mathcal{K}$ without priority and certainty weights.

Definition 3.6.1 An argument in support of a belief h is a pair $a_i = \langle H, h \rangle$ where:

1. $H \subseteq \mathcal{K}^*$ is the support of a_i;
2. $H \vdash h$;
3. $H \cup \{h\}$ is consistent ($\neg\exists S.\ S \subseteq H \wedge S \cup h \vdash \bot$);
4. No strict subset of H satisfies (1)–(3).

$H = \mathit{Support}(a_i)$ is the support of argument a_i, and $h = \mathit{Conc}(a_i)$ is the conclusion of a_i.

An agent constructs arguments in support of and against each decision in the set $\mathcal{O}$. A supporting argument highlights the goals that are satisfied by the decision, while an opposing argument describes the goals that are not.

Definition 3.6.2 An argument in support of a decision d is a triple $a_i = \langle H, G, d \rangle$ where:

1. $d \in \mathcal{O}$;
2. $H \subseteq \mathcal{K}^*$ is the support of a_i and $G \subseteq \mathcal{G}^*$;
3. $H \cup \{d\}$ is consistent ($\neg\exists S.\ S \subseteq H \wedge S \cup \{d\} \vdash \bot$);
4. $\forall g_i \in G,\ H \cup \{d\} \vdash g_i$;
5. No strict subset of H satisfies (1)–(4).

$H = \mathit{Support}(a_i)$ is the support of argument a_i, $G = \mathit{Goals}(a_i)$ are the goals realised by d, and $d = \mathit{Conc}(a_i)$ is the conclusion of a_i (the decision being supported).

Definition 3.6.3 An argument against a decision d is a triple $a_i = \langle H, G, d \rangle$ where:

1. $d \in \mathcal{O}$;
2. $H \subseteq \mathcal{K}^*$ is the support of a_i and $G \subseteq \mathcal{G}^*$;
3. $H \cup \{d\}$ is consistent ($\neg\exists S.\ S \subseteq H \wedge S \cup \{d\} \vdash \bot$);
4. $\forall g_i \in G, H \cup \{d\} \vdash \neg g_i$;
5. No strict subset of H satisfies (1)–(4).

$H = \mathit{Support}(a_i)$ is the support of argument a_i, $G = \mathit{Goals}(a_i)$ are the goals not realised by d, and $d = \mathit{Conc}(a_i)$ is the conclusion of a_i (the decision being rejected).

Let $\mathcal{A}_B$, $\mathcal{A}_P$, and $\mathcal{A}_C$ denote the set of arguments in support of beliefs, in support of a decision $d \in \mathcal{O}$, and against a decision $d \in \mathcal{O}$, respectively, that can be formed given a belief and goal base $\mathcal{K}$ and $\mathcal{G}$. The subset of practical arguments ($\mathcal{A}_C$ and $\mathcal{A}_P$) that are based on an acceptable set of consistent beliefs is found through the use of a preference-based argumentation framework (PAF, described in Section 3.4.1).

Definition 3.6.4 Let $AF = \langle AR, \mathcal{C}, PREF \rangle$ be a preference-based argumentation framework (as defined in Definition 3.4.1) in which $AR = \mathcal{A}_B \cup \mathcal{A}_C$ under an optimistic criterion, and $AR = \mathcal{A}_B \cup \mathcal{A}_P$ under a pessimistic criterion. In this framework:

1. The certainty level of an epistemic argument $a_1 = \langle H, h \rangle$ in $\mathcal{A}_B$ is denoted by: $L_B(a_1) = min\{\rho_i \mid k_i \in H \text{ and } (k_i, \rho_i) \in \mathcal{K}\}$. If $H = \emptyset$ then $L_B(a_1) = 1$;
2. $PREF$ is a preference relation over AR such that: $\forall a_1, a_2 \in \mathcal{A}_B$, $(a_1, a_2) \in PREF$ if and only if $L_B(a_1) \geq L_B(a_2)$;
3. $\mathcal{C}$ is a conflict relation over AR such that $\forall a_1 \in \mathcal{A}_B$ and $\forall a_2 \in AR$: $(a_1, a_2) \in \mathcal{C}$ if and only if $\exists h.\ [h \in \mathit{Support}(a_2) \vee h = \mathit{Conc}(a_2)] \wedge \mathit{Conc}(a_1) = \neg h$.

The set of acceptable arguments for and against decisions in AF are used to determine a preference ranking over decisions. The strength of these arguments, computed as described in Definitions 3.6.5 and 3.6.6, is used in this determination. In these definitions, the function $n(\cdot)$ is introduced, an order reversing mapping where $n(\sigma) = 1 - \sigma$. Definitions 3.6.5 to 3.6.7 are paraphrased from the work of Amgoud and Prade (2004a).

Definition 3.6.5 The strength of a pro argument $a_1 = \langle H, G, d \rangle$ is $\langle L_P(a_1), W_P(a_1) \rangle$:

1. $L_P(a_1)$ is the certainty level of a_1, such that: $L_P(a_1) = min\{\rho_i \mid k_i \in H \text{ and } (k_i, \rho_i) \in \mathcal{K}\}$. If $H = \emptyset$ then $L_P(a_1) = 1$;
2. $W_P(a_1)$ is denoted by: $W_P(a_1) = n(\beta)$ with $\beta = max\{\lambda_j \mid (g_j, \lambda_i) \in \mathcal{G} \text{ and } g_j \notin G\}$; If $\beta = 1$ then $W_P(a_1) = 0$ and if $G = \mathcal{G}^*$ then $W_P(a_1) = 1$.
3. $\forall a_1, a_2 \in \mathcal{A}_P$, a_1 is preferred to a_2 iff: $min\{L_P(a_1), W_P(a_1)\} \geq min\{L_P(a_2), W_P(a_2)\}$.

Definition 3.6.6 The weakness of a con argument $a_1 = \langle H, G, d \rangle$ is $\langle L_C(a_1), W_C(a_1) \rangle$:

1. $L_C(a_1)$ is the certainty level of a_1, such that: $L_C(a_1) = n(\psi)$ where $\psi = min\{\rho_i \mid k_i \in H \text{ and } (k_i, \rho_i) \in \mathcal{K}\}$. If $H = \emptyset$ then $L_C(a_1) = 0$;
2. $W_C(a_1)$ is denoted by: $W_C(a_1) = n(\beta)$ with $\beta = max\{\lambda_j \mid (g_j, \lambda_i) \in \mathcal{G} \text{ and } g_j \in G\}$;
3. $\forall a_1, a_2 \in \mathcal{A}_C$, a_1 is preferred to a_2 iff: $max\{L_C(a_1), W_C(a_1)\} \geq max\{L_C(a_2), W_C(a_2)\}$.

Decisions are compared on the basis of the relative strength (or weakness) of arguments for and against them, as described in Definition 3.6.7.

Definition 3.6.7 Let $\mathcal{A}^*$ denote the subset of acceptable arguments within the set AR, under either criteria, in the framework of Definition 3.6.4. A decision $d \in \mathcal{O}$ is preferred to $d' \in \mathcal{O}$, $d \neq d'$, if and only if there exists an argument $a_1 \in \mathcal{A}^*$ where $Conc(a_1) = d$ and $\forall a_2 \in \mathcal{A}^*$, where $Conc(a_2) = d'$, a_1 is preferred to a_2 (by Definition 3.6.5 or 3.6.6).

Extensions and Variations

There have been a number of variations of the preference-based decision-making approach of Amgoud and Prade (2004a). Amgoud and Prade (2006) extend this work to allow two types of arguments pro and con. An argument that supports a decision demonstrates the satisfaction of positive goals or the absence of negative goals. An argument against a decision infers its negative consequences, or the absence of good ones.

Amgoud et al. (2005a) develop a framework for multi-agent negotiation. Each agent constructs arguments, following the process of Amgoud and Prade (2004a), to support the offers they propose. Offers supported by strong arguments are made and accepted, while offers associated with a strong argument against them are refused and challenged. A framework for multi-criteria decision is considered by Amgoud et al. (2005b). In this

work, a decision-maker assesses a decision based on the degree to which it satisfies (or dissatisfies) a set of criteria. Arguments in support of a decision are derived from the criteria it satisfies, while arguments against are formed based on those it does not.

Amgoud et al. (2008b) consider the two step decision-making process of Amgoud and Prade (2004a) – in which acceptable arguments are balanced by strength or weakness to rank decisions – as not in keeping with the spirit of an argumentation system. Amgoud et al. (2008b) propose a framework in which acceptable decisions are inferred in a more direct manner from its acceptable arguments. This framework is composed of arguments that justify beliefs (epistemic arguments) and practical arguments that justify decisions (highlighting their positive characteristics). These arguments are viewed as abstract entities. Definitions 3.6.8 to 3.6.10 are paraphrased from the work of Amgoud et al. (2008b).

Definition 3.6.8 The decision-making framework of Amgoud et al. (2008b) is a 4-tuple, denoted by $AF = \langle \mathcal{A}_B, \mathcal{A}_P, \mathcal{C}, PREF \rangle$, where for a collection of choices $\mathcal{O}$:

1. $\mathcal{A}_B$ is a set of (abstract) epistemic arguments;
2. $\mathcal{A}_P$ is a set of (abstract) practical arguments, each $a_1 \in \mathcal{A}_P$ supporting a choice $d \in \mathcal{O}$ – the conclusion of each $a_1 \in \mathcal{A}_P$ is a choice ($d = Conc(a_1)$);
3. $\mathcal{C}$ is a conflict relation over the arguments in $\mathcal{A}_B \cup \mathcal{A}_P$ (as per Definition 3.6.9);
4. $PREF$ is a preference relation over $\mathcal{A}_B \cup \mathcal{A}_P$ (as per Definition 3.6.10).

The extensions of AF are the extensions of the PAF given by $\langle \mathcal{A}_B \cup \mathcal{A}_P, \mathcal{C}, PREF \rangle$.

Definition 3.6.9 A conflict relation $\mathcal{C}$ over arguments in the framework of Definition 3.6.8, $AF = \langle \mathcal{A}_B, \mathcal{A}_P, \mathcal{C}, PREF \rangle$, is the union of three relations $\mathcal{R}_B$, $\mathcal{R}_P$, and $\mathcal{R}_M$:

1. $\mathcal{R}_B$ is an (abstract) conflict relation over $\mathcal{A}_B$;
2. $\mathcal{R}_P$ is a conflict relation over $\mathcal{A}_P$ where $\forall a_1, a_2 \in \mathcal{A}_P$, $(a_1, a_2) \in \mathcal{R}_P$ if and only if they support different choices ($Conc(a_1) \neq Conc(a_2)$);
3. $\mathcal{R}_M$ is a conflict relation over $\mathcal{A}_B \cup \mathcal{A}_P$, where a pair $(a_1, a_2) \in \mathcal{R}_M$ if and only if $a_1 \in \mathcal{A}_B$, $a_2 \in \mathcal{A}_P$, and a_1 undermines (conflicts with) a_2[5].

Definition 3.6.10 A preference relation $PREF$ over arguments in the framework of Definition 3.6.8, $AF = \langle \mathcal{A}_B, \mathcal{A}_P, \mathcal{C}, PREF \rangle$, is the union of three relations $\geq_B$, $\geq_P$ and $\geq_M$:

[5]The means by which such arguments undermine or conflict with others remains abstract.

1. $\geq_B$ and $\geq_P$ are (abstract) preference relations over $\mathcal{A}_B$ and $\mathcal{A}_P$, respectively;
2. $\geq_M$ is a preference relation over $\mathcal{A}_B \cup \mathcal{A}_P$, where $\forall a_1 \in \mathcal{A}_B$, and $\forall a_2 \in \mathcal{A}_P$, $(a_1, a_2) \in \geq_M$ (each epistemic argument is preferred to all practical arguments).

Definitions 3.6.9 and 3.6.10 demonstrate that conflict and preference amongst arguments is treated in a more or less abstract manner. The principles of PAFs (Section 3.4.1) are used to find the acceptable arguments in the framework of Definition 3.6.8. Statuses are assigned to decisions on the basis of these arguments. A sceptical option, for example, is one that is supported by an argument in all extensions of the framework by a given semantics, while a credulous option is one that is supported by an argument in at least one extension. An ordering over these statuses yields an ordering over decisions.

An Evaluation of Preference-Based Decision-Making

The questions of Criterion 2.1.1 are now answered with respect to the approaches described in this section. As was the case in Section 3.6.1, each formalism discussed within this section prescribes a particular method of preference expression.

In the work of Amgoud and Prade (2004a), a decision-maker is equipped with a prioritised goal base. This goal base leads to the construction of arguments for and against decisions. Amgoud et al. (2005a) employ the framework of Amgoud and Prade (2004a) with added mechanisms for coordinating negotiation amongst decision-makers. Amgoud and Prade (2006) employ goal prioritisation as a means of preference representation, but additionally distinguish between positive goals (those a decision-maker desires to achieve) and negative goals (those a decision-maker seeks to avoid). Amgoud et al. (2005b) represent preference as prioritised criteria that a decision satisfies to varying degrees. Amgoud et al. (2008b) assume that preference is expressed in terms of positive (absolute) decision properties. The range of preferences supported by these approaches does not extend beyond the absolute (describing the desirability of individual choices, see Chapter 2), as each argument is assumed to represent a reason for selecting a choice over all others. Comparative style preference, as highlighted in Section 2.2.6, lays the foundation for arguments that support the selection of a choice only over certain others.

The framework of Amgoud et al. (2008b) abstracts away from the content of practical

arguments during decision-making – defining a single decision-making process for the range of preference it supports. For each of the approaches described in this section, it is not the case that both of the questions in Criterion 2.1.1 are answered in the affirmative.

3.6.3 Decision-Making with Choice Rules

Ferretti et al. (2008) combine choice rules and argumentation to create a decision-making system based on defeasible logic programming (DeLP) [García and Simari (2004)]. In this approach, choice rules applied to a collection of choices define its best subset. Each rule describes a set of conditions (literals) that must be warranted, and a set of conditions (literals) that must not be warranted, for the rule to apply. Examples of such rules are presented later in this section. DeLP is used to represent the knowledge of an agent, and build arguments in support of (and against) literals to determine warrant.

Definitions 3.6.11 to 3.6.14 are paraphrased from the work of Ferretti et al. (2008) and García and Simari (2004).

Definition 3.6.11 A defeasible logic program is a tuple $\Gamma = \langle F, \Delta \rangle$, where:

1. F is a set of ground literals (n-ary predicates or their negation); and,
2. Δ is a set of strict and defeasible rules, where each strict (defeasible) rule denotes a relationship that always (tends to) hold.

   A strict rule has the form $L_1, \ldots, L_n \rightarrow L_{n+1}$ where each $L_{i=1\ldots n,n+1}$ is a literal. A defeasible rule has the form $L_1, \ldots, L_n \Rightarrow L_{n+1}$, where each $L_{i=1\ldots n,n+1}$ is a literal. Any variables in these literals are universally quantified at outermost scope.

To determine if a literal L is warranted, arguments are constructed in support of and against L. A literal is warranted if there exists an undefeated argument that supports it. A literal is derived from a collection of facts and rules by classical deduction.

Definition 3.6.12 Given $\Gamma = \langle F, \Delta \rangle$, an argument supporting literal L is a pair $\langle A, L \rangle$:

1. $A \subseteq \Delta$;
2. L can be derived from $\langle F, A \rangle$ (by Definition 3.6.13);
3. There is no proper subset $A' \subset A$ such that A' satisfies (1)–(2).

Definition 3.6.13 Let Γ be a defeasible logic program (Definition 3.6.11), and $Ground(\Gamma)$ the set of ground instances of the rules in Δ of Γ. A derivation of a literal L from $\Gamma = \langle F, \Delta \rangle$ is a finite sequence $L_1, L_2, \ldots, L_n = L$ where each L_i is a ground literal such that:

1. $L_i \in F$; or,
2. There exists a rule $R \in Ground(\Gamma)$ with a head L_i and a body $B_1, B_2, \ldots, B_k$ such that each B_j is an element of the sequence appearing before L_i.

In the DeLP-program of an agent decision-maker is a set of strict and defeasible rules describing when one alternative is better than another. Each of these rules is based on a comparison literal, expressing a principle by which two alternatives can be compared.

Example 3.6.1 Consider a scenario involving a robot and a collection of boxes. The robot prefers to pick up the box that is closest to it. This preference is expressed in the rule:

$$CloserRobot(box_1, box_2) \Rightarrow Better(box_1, box_2)$$

where $B_1, \ldots, B_n \Rightarrow B_{n+1}$ is a defeasible rule, and the comparison literal is $CloserRobot$.

These *Better* literals form the basis of arguments that determine whether one choice $d_1 \in \mathcal{O}$ is better than another $d_2 \in \mathcal{O}$. The acceptability of these arguments is ascertained using the dialectical process of García and Simari (2004) – a proof tree construction procedure. An argument is deemed acceptable (warranted) if it withstands all possible attacks against it[6]. A tree of attacks is formed, the root of which is the argument in question (representing the position of a proponent), its children are the arguments that attack it (the positions of an opponent), their children are the arguments that attack them (proponent positions), and so on. In this process, an argument a_1 attacks another a_2 if it rebuts or undercuts it (undercutting and rebuttal are described in Section 3.1.2) and a_1 is preferred to a_2 (by a chosen comparison criterion). An argument is warranted if the leaves of this constructed argument tree are each a position of the proponent.

In the work of Ferretti et al. (2008), preference over arguments is based on an ordering over comparison literals. An argument is preferred to another if it is based on a more

[6]García and Simari (2004) use the term defeats and defeaters in place of attacks and attackers.

important comparison literal. The *Better* literals that are supported by an accepted (warranted) argument determine which choice rules – rules that identify a preferred subset of a choice set – apply (as shown in Definition 3.6.14 and Example 3.6.2).

Definition 3.6.14 For a set of decisions, $\mathcal{O}$, a choice rule defines a subset $S \subseteq \mathcal{O}$ of best choices. Γ_O denotes the set of choice rules for an agent. Each $r \in \Gamma_O$ is of the form:

$$S \overset{\mathcal{O}}{\leftarrow} P,\ not\ T$$

where: P and T are literal sets denoting the preconditions and constraints of the rule. If the conditions in P are warranted, and the constraints in T are not warranted, the rule r applies and is used to select a most preferred subset of choices.

Example 3.6.2 Given a collection of decisions, $\mathcal{O}$, the following rule states that choice $d_1 \in \mathcal{O}$ should be chosen if it is better than an alternative $d_2 \in \mathcal{O}$ and no alternative $d_3 \in \mathcal{O}$ is better than d_1. This rule is expressed as [Ferretti et al. (2008)]:

$$\{d_1\} \overset{\mathcal{O}}{\leftarrow} \{Better(d_1, d_2)\}, not\{Better(d_3, d_1)\}$$

Given a DeLP-program $\mathcal{P} = \langle F, \Delta \rangle$ (Definition 3.6.11), a collection of decisions $\mathcal{O}$, and a set of choice rules Γ_O (Definition 3.6.14), the set of acceptable decision alternatives within $\mathcal{O}$ is the union of the sets produced from the applicable choice rules in Γ_O.

An Evaluation of Decision-Making with Choice Rules

The questions of Criterion 2.1.1 are now answered with respect to the approach of Ferretti et al. (2008). This approach allows preference to be expressed in the form of comparison rules – each rule describing a different method of comparing two choices to determine which is the more desirable. Comparison rules can be used to compare choices by each of the means described in Chapter 2 – their implementation in this formalism, however, is more limiting. Each rule has a literal as its head and the conjunction of a set of literals as its antecedent. These rules are used to derive a literal through rule application, starting with a set of facts (ground literals or their negation). Consequently, it is not feasible to compare choices by, for example, employing the majority rule of Tversky (1969) (discussed in Section 2.2.5). If it is possible to evaluate a literal through an alternative means

– through the use of custom functions, for example, rather than rule application from a set of facts – choices can be compared using each of the methods presented in Chapter 2.

The two questions of Criterion 2.1.1 are not answered in the affirmative for the approach presented in this section.

3.6.4 Scheme-Based Decision-Making

One of the many strands of practical decision-making (reasoning about action) with argumentation is based on the argumentation schemes of Walton (1996). These schemes describe patterns of argument, and include two schemes for justifying the selection of an action to perform – a necessary condition and sufficient condition scheme. These schemes describe a typical pattern of reasoning – if g is a goal for an agent and α is an action that is necessary (sufficient) to achieve g, then the agent should perform action α.

Instantiations of these schemes represent reasons in support of performing an action. Each argument is subjected to critical questions to determine, for example, if there are alternative means of achieving the relevant goal, or whether the action in question can (legally) be performed. It is only the arguments that withstand such questioning that are deemed justified. These schemes, and the critical questions that assess them, have been considered in a number of works [Atkinson et al. (2006, 2005a,b,c, 2008); Bench-Capon et al. (2009); Atkinson and Bench-Capon (2008, 2007b); Tolchinsky et al. (2007); Nawab et al. (2008); Bench-Capon and Prakken (2006); Ouerdane et al. (2008)].

Atkinson et al. (2006, 2005c) use a modified sufficient scheme for practical reasoning [Walton (1996)] to construct arguments in support of performing an action, with an extended set of critical questions reflecting the change. This scheme describes the following pattern – in the current circumstances R, we should perform action α, which will realise goal g, which will in turn promote a value v. Atkinson and Bench-Capon (2007a,b) provide an alternating transition semantics for this formalism in which actions cause transitions from one state to another while promoting or demoting a value.

Ouerdane et al. (2008) describe a method of ascertaining the most preferred member of a pair of choices, a process designed to form part of a fully capable decision-making system. Argument schemes denoting common patterns of reasoning define how choices can be compared, and are used to generate arguments in support of the selection of one

choice over another. Questioning of these arguments through the use of critical questions (as in the work of Atkinson et al. (2005c, 2006) and Atkinson and Bench-Capon (2007a)) is introduced as a means of selecting the best of a pair of choices. This process, however, is preliminary and not fully specified. Ouerdane et al. (2008) limit their discussion to decision-making scenarios involving only two actions (choices), and the dialogue process designed to select a best choice (on the basis of schemes and critical questions) is, as Ouerdane et al. (2008) state 'the subject of ongoing work' and 'currently under development'. The schemes of Ouerdane et al. (2008) are described in Section 2.2.6.

An Evaluation of Scheme-Based Decision-Making

The questions of Criterion 2.1.1 are now answered with respect to the approaches described in this section. The value-based approaches described within this section (such as [Atkinson et al. (2005c, 2006)]) support only prioritised values as a means of preference expression. The schemes of Ouerdane et al. (2008) are constructed to compare two choices by any desired means. This approach supports general preference expression, while the value-based methods do not. The schemes of Ouerdane et al. (2008) can be constructed and defined in any manner desired by a decision-maker. Once constructed, however, arguments are subjected to critical questions specific to the scheme used in its formation by both the decision-aiding system and a user during an interactive dialogue. This process is not fully specified, and is the subject of ongoing work (as described above).

The two questions in Criterion 2.1.1 are not answered in the affirmative for the approaches presented in this section.

3.6.5 Modular Decision-Making Architectures

In the next section, attention is shifted from decision-making to argumentation and planning. Before leaving this discourse on decision-making, it must be noted that other argumentation-based tools for decision-making exist that I have not described in the preceding sections – notably the work of Kakas and Moraitis (2003) and their modular architecture for a decision-making agent. This work is used and extended in a variety of later works [Kakas et al. (2004); Morge and Stathis (2008); Kakas and Moraitis (2006); Witkowski and Stathis (2004)]. These works do not operate on the preferences of a user

in the selection of a best decision, and consequently are not focused on in this chapter.

3.7 Argumentation and Planning

The key advantage of argumentation within the domain of decision-making is its ability to manage conflict and inconsistency in: knowledge, preferences, and the rules by which a decision is made. These advantages extend to the realm of planning – the task of finding a course of action to achieve a set of goals (or complete a task). This task is a series of decision-making problems, the solutions of each forming a plan when combined.

In the past decade, a number of techniques for argumentation-based planning have been conceived. In this section the use of argumentation within BDI agent systems is discussed. A contrasting approach uses argumentation as a tool for determining which actions are applicable in a given state, arguing to determine the truth of action preconditions and consequences [Simari et al. (2004); García et al. (2007, 2008)]. Hulstijn and van der Torre (2004) use argumentation for the generation of agent goals, and consistent plans to realise those goals. These plans are represented by arguments, and a concept of attack allows the selection of compatible sets of plans (those that can be executed together). Another strand of research considers the use of argumentation in multi-agent planning. Belesiotis et al. (2010), for example, present an approach for multi-agent planning in which a collection of agents must agree on a plan to achieve a common goal.

3.7.1 BDI Agents and Argumentation

Argumentation-based tools for planning have predominately focused on BDI agents. Amgoud (2003) proposes a system that discovers consistent sets of intentions (desires to achieve) from a conflicting set of desires (desires that cannot all together be achieved). In this work, developed further by Amgoud et al. (2008a), the task is not to find the set of most preferred desires, but sets of desires that are together achievable. Rotstein et al. (2007) present similar work in desire and intention selection. In this section, two approaches are described that extend the work of Amgoud (2003) in the search for preferred plans – plans that support consistent (achievable) sets of preferred desires.

Rahwan and Amgoud (2006) use the work of Amgoud (2003), Amgoud and Maudet (2002) and Amgoud and Kaci (2004) to develop an argumentation-based system for agent

plan construction. In this system, rules define what an agent should desire in a given context (state-of-affairs) – a context that is subject to argumentation, with arguments constructed for and against beliefs. The application of these desire rules produces a set of 'explanatory arguments', each supporting a desire for the agent to achieve in its plan. A prioritisation over desires, and the certainty of beliefs used in the construction of explanatory arguments, determines their relative strength (a preference relation).

Preference over, conflict between, and the acceptability of belief and explanatory arguments is defined in the same vein as arguments within the work of Amgoud and Prade (2004a) (described in Section 3.6.2). An agent forms a plan to achieve the desires supported by the acceptable of these arguments. Planning rules are instantiated to form 'instrumental arguments' – each describing how a set of these desires can be achieved. Two instrumental arguments conflict if they represent plans that cannot both be implemented, while each argument is preferred over those that achieve less important desires. The acceptable of these arguments define a preferred plan for an agent to perform.

Amgoud et al. (2008a) consider an alternative approach – combining the two stages of argumentation present within the work of Rahwan and Amgoud (2006). Belief, explanatory, and instrumental arguments, are constructed in the work of Amgoud et al. (2008a), but are fed into a single argumentation framework. A prioritisation over desires forms the basis of a preference relation over arguments in both approaches.

An Evaluation of Argumentation-Based Planning for BDI Agents

Within this section, two approaches that use argumentation to discover a preferred plan have been described – the works of Rahwan and Amgoud (2006) and Amgoud et al. (2008a). For each of these approaches, the questions of Criterion 2.1.1 are not answered in the affirmative. Preference is expressed by a prioritisation over desires in both works, just as preference is represented in the form of a prioritised goal base in the work of Amgoud and Prade (2004a) (see Section 3.6.2). Moreover, the process by which plans are constructed is tailored toward this representation of preference. As is the case in the work of Amgoud and Prade (2004a), comparative style preference is not supported[7].

[7]See my evaluation of Amgoud and Prade (2004a), and related works, in Section 3.6.2 for further comment on comparative preference.

3.8 21^{st} Century Argumentation

The past decade has heralded many innovative applications of computational argumentation, beyond those discussed in this chapter. These modern developments have increased the power of this approach as a model of human reasoning.

The conceptualisation of the semantic web [Berners-Lee and Fischetti (1999)], for example, has spurred the development of a World Wide Argument Web [Rahwan and Sakeer (2006); Rahwan et al. (2007b)] for the publication, search, and evaluation of public argumentation. An increasing focus on strategy in argumentation has led to the combination of game theory and argumentation [Procaccia and Rosenchein (2005); Rahwan and Larson (2008a); Riveret et al. (2008); Matt and Toni (2008); Rahwan et al. (2009)]. The development of meta-argumentation tools has imbued agents with the ability to argue about the relationships between arguments [Wooldridge et al. (2005); Modgil (2006b)].

3.8.1 Meta-Argumentation

Wooldridge et al. (2005) recognise that argumentation is not merely the exchange of arguments, each supporting a viewpoint and potentially attacking others, but a meta-logical process. This process involves argumentation about arguments and the relationships that exist between them. Wooldridge et al. (2005) describe a hierarchical argumentation formalism in which a tower of arguments is constructed – each level referring to (or arguing about) the arguments that reside in the levels below, and their relationships.

In earlier work, Brewka (2001) describes a framework in which the protocol or process of argumentation is itself a topic of debate. Modgil (2006b) resolves conflicts between arguments in a Dung (1995) argumentation framework by subjecting the preferences that exist between arguments to debate. Modgil (2006a) considers debate over value-orderings within the value-based frameworks of Bench-Capon (2003a,b).

Boella et al. (2009) and Modgil (2009a) provide an overview of a wide range of tools and techniques for meta-argumentation.

3.8.2 Argumentation and the Semantic Web

The inspiration behind the semantic web [Berners-Lee and Fischetti (1999)] was the development of a framework in which data and services could be understood by and shared

between web-based applications. The argumentation community took the concept and initiated the development of the World Wide Argument Web [Rahwan and Sakeer (2006); Rahwan et al. (2007b)]. The goal was to allow the structured representation, sharing, and querying of opinions and viewpoints given by users of the World Wide Web.

Necessary in providing this structure was a standard argument format capable of expressing a wide variety of human argument. Chesñevar et al. (2006) propose an argument interchange format as a standard representation of argument. This format is extended by Rahwan et al. (2007b) to capture typical argumentation schemes [Walton and Krabbe (1995)] and Reed et al. (2008) to incorporate dialectic argumentation locutions or moves.

Rahwan et al. (2007b) develop ArgDF, a system in which a user can present arguments using the schemes of Walton and Krabbe (1995), query them using RDF (a semantic web query language) [Brickley and Guha (2004)], and form networks of linked arguments. The benefits of a World Wide Argument Web are further explored by Rahwan (2008) who highlights its potential to enable the search, evaluation and comparison of relationships that exist between arguments or opinions published on the web.

Rahwan and Banihashemi (2008) extend the work of Rahwan et al. (2007b) by incorporating the OWL ontology [McGuinness and van Harmelen (2010)] for argument representation and enhanced querying capability. Buckingham Shum (2008) describes an alternative system, designed to attract a web community with a balance between informal user-system interaction and a formal structure of argumentation.

3.8.3 Argumentation and Game Theory

While a significant degree of research within the computational argumentation community has focused on the semantics of argument acceptability, protocols for multi-agent argumentation, and mechanisms of attack and defeat, strategy in argumentation has been left on the back-burner [Rahwan and Larson (2008a)].

Rahwan and Larson (2008a) promote the use of game theory and mechanism design in argumentation as a means of enabling strategy in agent-based argumentation. A number of other works maintain a similar goal and consider this fusion of ideas. Two party games are considered by Matt and Toni (2008) in the assessment of argument strength, social welfare is used to determine argument acceptability in the work of Rahwan and

Larson (2008b), and Riveret et al. (2008) consider the utility of moves in a dialogue. In earlier work, Procaccia and Rosenchein (2005) introduce a pay-off function in a dialectic argumentation game. In this game, agents do not win or lose rounds of argumentation but receive a pay-off based on how well they perform. These pay-offs determine the victor of the game and allow the development of winning strategies.

In the work of Rahwan and Larson (2008a), an agent selects which arguments to put forth in a debate on the basis of a decision mechanism. A direct-revelation mechanism sees agents select which arguments to reveal simultaneously while attempting to maximise the number of their arguments accepted. A study of strategy proofness determines which mechanisms encourage an agent to reveal all of their arguments. Rahwan et al. (2009) extend the argumentation mechanisms of Rahwan and Larson (2008a) by removing the assumption that agents cannot lie about their arguments, but only hide a subset from presentation. In addition, a more realistic preference setting is incorporated in which an agent does not focus on maximising the number of their arguments accepted, but on achieving acceptance of a particular (focal) argument.

The semantics of argument acceptability varies between game theoretic argumentation frameworks. Rahwan and Larson (2008b) consider the use of social welfare in selecting which semantics to adopt. Riveret et al. (2008) consider the cost and likelihood of acceptance when putting forward arguments in a dialogue – preference over arguments is defined using the expected utility of moves, combining the probability of success of an argument with the cost of raising it. Rahwan and Tohmé (2010) consider the aggregation of differing agent views on the acceptability of an argument, examining a variety of aggregation techniques including argument-wise plurality voting.

3.9 Concluding Remarks

The field of computational argumentation, over the past few decades, has amassed a vibrant research community. In its early years, the focus of research was on the formation of arguments – defining relationships of support and attack between propositions [Birnbaum et al. (1980)]. Early argumentation frameworks defined the relationship between argument structure and their interaction [Pollock (1987); Loui (1987); Lin and Shoham

(1989); Vreeswijk (1997)], while the seminal framework of Dung (1995) described semantics for the evaluation of argument acceptability. Alternative formalisms in which the relative strength of arguments determined their tenability [Krause et al. (1995a); Verheij (1996a); Vreeswijk (1997)] collided with the abstract work of Dung (1995) to form a variety of extensions [Amgoud and Cayrol (1998, 2002b,a); Bench-Capon (2003a,b)].

The dissimilarity of the extension-based approach promoted by Dung (1995) with the process by which humans argue led to the popularity of dialectic argumentation formalisms [Prakken (1999); Prakken and Sartor (1998); Dunne and Bench-Capon (2003); Cayrol et al. (2001); Devred and Doutre (2007); Dung et al. (2006); Thang et al. (2009)]. Within these techniques, argumentation takes place as a dialogue in which arguments are gradually introduced into a debate. The test for argument acceptability does not involve the creation of an extension, but its defence from the arguments that attack it.

In the past decade, a great deal of research has considered the use of argumentation in the design of decision-making and planning mechanisms for agents. The compatibility of argumentation techniques with human reasoning results in an approach that is intuitive for the human user. In decision support, an argumentation-based system is able to explain the thought process it has followed in the recommendation of a choice or course of action through the presentation of its supporting arguments. Influential works in argumentation-based decision-making include the preference-based framework of Amgoud and Prade (2004a) and its extensions, the scheme-based reasoning of Atkinson et al. (2005c, 2006), and the BDI-agent focused work of Rahwan and Amgoud (2006).

In this chapter, a range of argumentation-based decision-making and planning approaches have been evaluated in terms of a criterion for preference decoupling. Two questions have been devised that, if answered in the affirmative for an approach, indicate that it satisfies this criterion. The first question asks whether such an approach is able to support the range of preference representations that can be defined in the comparison of two choices[8], while the second requires that it abstract away from their representational detail. It has been found that not one of the decision-making or planning tools discussed in this chapter answers both questions in the affirmative. This chapter has not

[8]Where this comparison solely depends on the individual or relative properties of these two choices.

sought to evaluate the merit or relative merits of these approaches, merely to determine if they achieve a decoupling of preference as outlined by the questions listed in Criterion 2.1.1 (the advantages of which are discussed in the Introduction to this thesis).

Recent developments in the computational argumentation community have charted a course for its future. The combination of argumentation and game theory allows agents to strategise when presenting arguments for debate – a reflection of the way argumentation takes place between humans [Rahwan and Larson (2008a); Matt and Toni (2008); Rahwan et al. (2009)]. Web-based tools for argumentation encourage the structured representation and public evaluation of user opinions and views [Rahwan and Sakeer (2006); Rahwan et al. (2007b); Buckingham Shum (2008)], while a characterisation of argumentation as a meta-logical process has increased its range of uses [Wooldridge et al. (2005); Modgil (2006b, 2007); Gabbay (2009); Modgil (2009b)].

In the next chapter, an argumentation-based decision-making tool – a tool that satisfactorily answers the questions for decoupling of Criterion 2.1.1 – is described. Many of the features of this approach are inspired by the work presented in this chapter.

Chapter 4

An Argumentation-Based Decision-Maker

Nothing is more difficult . . . than to be able to decide.

Napoleon Bonaparte

IN this chapter, I present an argumentation-based decision-making framework that lies within the prescriptive partition of decision-making. This framework is not designed to model the behaviour of a human decision-maker, or adhere to an axiomatic account of rationality. Following the style of the European school of decision-making [Roy and Vanderpooten (1996)], this framework aims to provide a human user or agent with a reason to believe that a given choice is one of the best, on the basis of their supplied preferences. Argumentation-based approaches allow the preferences of a human decision-maker to form the basis of arguments in support of or against available choices – arguments whose acceptability guides the selection of a most preferred choice.

A key advantage of argumentation-based methods over traditional decision-making approaches is that not only can we reason, argue or deliberate over the means by which choices are compared, but we can critically analyse the information upon which this comparison takes place. One of the goals of this thesis is to develop an automated decision-making system that is decoupled from the preferences supplied to it. Within such tools, these preferences are manipulated and viewed as instances of an abstract type. The ability of argumentation-based methods to treat arguments as abstract entities during decision-making has encouraged their use in this thesis to achieve this goal.

In Chapter 3, a range of techniques within the field of argumentation-based decision-making have been considered. Not one of these approaches, however, is decoupled from the preferences supplied to it. These approaches do not describe a general procedure that can be applied to select a choice, irrespective of the means by which they are compared by a user. In Criterion 2.1.1 of Chapter 2, a set of questions which determine whether such a decision-making tool is decoupled have been presented. I have found that not one of these existing approaches answers each of these questions in the affirmative. These questions highlight a number of characteristics that any argumentation-based decision-making system or framework developed in this thesis must strive to achieve.

Such a system must support the construction of arguments that express reasons to select one choice over others on the basis of general preference. In other words, it must be capable of distinguishing choices, through arguments, by a variety of means – reflecting the variety of means through which we can express preference (a description of a number of such methods is provided in Chapter 2)[1]. Last, but not least, the system must be capable of selecting a choice while treating constructed arguments as abstract entities – abstracting away from the representational detail of the preference they describe.

The evaluation of existing work, presented in Sections 3.6 and 3.7 of Chapter 3, considers a range of early argumentation-based approaches – namely, the pro and con-based systems of Fox et al. (1992), Brewka and Gordon (1994), Bonet and Geffner (1996), Fox and Parsons (1997), and Parsons and Green (1999). I evaluate the possibilistic approach of Amgoud and Prade (2004a) upon which numerous decision-making tools are based [such as, Amgoud and Prade (2006) and Amgoud et al. (2005a,b, 2008b)]. Additionally appraised is the logic programming-based method of Ferretti et al. (2008) and the scheme-based approaches of Ouerdane et al. (2008) and Atkinson et al. (2005c, 2006). Atkinson et al. (2005c, 2006) are considered as representative descriptions of the use of practical decision-making schemes and values in the justification of choices, explored in an array of alternative and later works (see Section 3.6.4 of Chapter 3).

In response to this evaluation, I construct and present in this chapter an alternative argumentation-based decision-making framework. In its development, I have extracted a

[1]To label an approach decoupled, as per Criterion 2.1.1, it must support any type of preference that relies solely on the individual and relative properties of choices under comparison.

number of features from the existing work of Chapter 3. Within my approach, arguments are constructed in support of preference relationships amongst choices. This is in contrast to much of this existing work, in which arguments are formed in support of or against the selection of a choice. Abstract comparison schemes, expressing mechanisms by which choices are compared, form the basis upon which these arguments are generated. This aspect of my approach is reflective of the work of Ouerdane et al. (2008) and provides support for the management of general preference expressions. The framework I develop in this chapter is designed to be used by an agent that interacts with a user, eliciting from them their preference, and instantiating each of the framework's components.

Within this framework, abstract conflict and preference relations over arguments are instantiated and employed to form a Dung (1995) argumentation framework. The acceptable arguments of this framework determine which preference relationships amongst choices are valid. These relationships form a dominance graph over available decision alternatives (a structure used in the MCDA[2] approach ELECTRE [Roy (1991)]), upon which a decision principle or social choice rule (in the style of those employed when selecting a winner of a group vote [Sen (1995); Brandt et al. (2007); Arrow et al. (2002)], and reminiscent of the work of Ferretti et al. (2008)) is applied to partition the choice space.

I begin by describing the range of concepts present in existing work that are used in the development of my framework. I follow this with its definition, a detailed description of each of its components, and the process by which it manipulates these components to select a preferred subset of choices. A running example involving the selection of a meal is used to demonstrate its operation. I conclude by highlighting a number of future directions in which this work can be taken. Moreover, I consider how this framework can be used within the kinds of applications that have motivated its creation. A number of such applications have been identified in the Introduction to Part I of this thesis.

4.1 Preliminaries

While no existing argumentation-based decision-making system or framework has all of the desired features characterised in Criterion 2.1.1 of Chapter 2, many display a subset

[2] A range of multi-criteria decision analysis (MCDA) approaches are described in Chapter 2.

of these. The framework I present in Section 4.2 of this chapter combines elements that appear in a variety of these existing works. I describe these elements below.

I consider the work of Amgoud et al. (2008b) and its distinction between abstract arguments in support of beliefs and those in support of choices, combined with an abstract definition of their interaction. These elements support the need to abstract away from the representational detail of arguments during decision-making. Moreover, they enable the management of uncertainty and inconsistency in knowledge (through the construction of arguments in support of and against beliefs derived from a knowledge base).

To support the general expression of preference, I incorporate ideas presented in the work of Ouerdane et al. (2008) – namely, their schematic patterns of choice comparison – in the construction of arguments. These schemes are able to capture the range of different ways in which choices can be compared. This feature, in combination with an ability to abstract away from the content of arguments during decision-making, allows a decision-making process to be developed that is decoupled from the preferences of a user.

The elements described thus far can be combined to develop a system that determines which of a pair of choices is the most preferred. I consider the additional concept of decision or social choice rules (as used in the development of voting criteria – rules that determine which of a set of candidates is a winner given the pair-wise rankings of a range of voters [Sen (1995); Brandt et al. (2007); Arrow et al. (2002)]) as a means of determining the most preferred subset of a set of choices given pair-wise preferences.

4.2 An Argumentation-Based Decision-Maker (ADM)

I begin with a definition of this argumentation-based decision-making framework in its entirety. I then proceed to delve deeper into the details of each of its components. The goal of this framework is to discover a most preferred subset of a finite set of choices of at least two elements. This framework is a 5-tuple, as shown in Definition 4.2.1. A decision-making agent instantiates each of its components, one of which is a collection of choices $\mathcal{O}$. These components are consequently manipulated as abstract entities by a process, described in this chapter, that selects from $\mathcal{O}$ a most preferred subset of choices. Preference within this framework is represented by a set of abstract comparison schemes.

Definition 4.2.1 The ADM framework is 5-tuple $AS_{ADM} = \langle \mathcal{O}, \mathcal{A}_e, \mathcal{A}_p, \mathcal{X}, RL \rangle$, where:

1. $\mathcal{O}$ is a finite set of available choices, containing at least two elements[3];
2. $\mathcal{A}_e$ is a finite set of epistemic arguments, each inferring a belief that the agent can choose to accept about its environment (Section 4.4);
3. $\mathcal{A}_p$ is a finite set of practical arguments – each argument $a \in \mathcal{A}_p$ in support of a pair-wise preference for one choice $o_i \in \mathcal{O}$ over another $o_j \in \mathcal{O}$ (Section 4.4);
4. $\mathcal{X} = \langle \mathcal{R}, \succeq, Sem \rangle$ defines: an irreflexive binary conflict ($\mathcal{R}$) relation, and a reflexive and transitive preference ($\succeq$) relation, over the arguments in $\mathcal{A}_e \cup \mathcal{A}_p$ (Section 4.5); and a Dung (1995) extension semantics *Sem* (as per Definition 3.2.3);
5. RL is a decision-rule (Section 4.6) that determines the subset of $\mathcal{O}$ that is most preferred given a set of pair-wise preference relationships over its choices.

The components of an ADM define: the collection of choices over which decision-making is taking place; a set of epistemic arguments debating the state-of-affairs of the decision-maker; a set of practical arguments capturing (general) decision-maker preference in the support of choices over their alternatives; a conflict and preference relation over these arguments; and a decision-rule that, when applied to a set of pair-wise preference relationships over a choice set, discovers its most preferred subset.

Definition 4.2.1 does not describe how this ADM finds a most preferred subset of the set of choices $\mathcal{O}$. Its epistemic $\mathcal{A}_e$ and practical $\mathcal{A}_p$ arguments, in conjunction with the given conflict $\mathcal{R}$ and preference $\succeq$ relation over them, form a preference-based argumentation framework (PAF, see Section 3.4.1). I define the acceptable subset of these arguments as those that are sceptically accepted under the given semantics *Sem* by this PAF. The acceptable arguments within the set $\mathcal{A}_p$, as per this framework, define a set of justified pair-wise preference relationships over the collection of available choices $\mathcal{O}$. The decision-rule, RL, determines the subset of choices that are most preferred when applied to these relationships. This process is formally described in the sections that follow.

In the following sections, I describe each of the components of AS_{ADM} and what, if anything, the ADM framework assumes about their structure and content. I make several assumptions of any agent employing this approach. I assume that an agent instantiating

[3]Each $o \in \mathcal{O}$ is viewed by the system as abstract – I do not prescribe its representational detail.

this framework has a deductive system – a system that captures what it knows about its environment, and how to derive new knowledge – and a collection of comparison schemes, each describing a means by which choices are compared (Definition 4.2.2).

Definition 4.2.2 An agent instantiating the framework of Definition 4.2.1 has:

1. A deductive system Γ, for which $\Gamma \rightsquigarrow k$ denotes that the piece of knowledge k (whose representation is not prescribed) is inferred to hold by the system Γ;
2. A finite and non-empty set of choice comparison schemes SCH (Section 4.3) – each scheme defining the conditions under which an $o_i \in \mathcal{O}$ is preferred to an $o_j \in \mathcal{O}$;
3. The ability to construct practical $\mathcal{A}_p$ and epistemic $\mathcal{A}_e$ arguments (as per Section 4.4) over a set of choices given Γ and its scheme set SCH;
4. The ability to determine when two arguments $a_1, a_2 \in \mathcal{A}_e \cup \mathcal{A}_p$ conflict and when one such argument a_1 is preferred to another a_2, as per Section 4.5;
5. A Dung (1995) extension semantics Sem (as per Definition 3.2.3) by which it believes the acceptability of arguments should be assessed;
6. A decision-rule RL which, when applied to a set of pair-wise preference relationships over a choice set determines its most preferred subset (Section 4.6).

In Section 4.7, I evaluate this framework with respect to the questions of Criterion 2.1.1 – both of which must be answered in the affirmative by an approach to classify it as decoupled. In the following section, I describe an example that shall be used throughout this chapter, describing an agent of the form shown in Definition 4.2.2. In this chapter, I show how this agent instantiates the components of AS_{ADM} in Definition 4.2.1, and how the ADM framework selects a preferred choice subset on the basis of this instantiation.

I consider, in Section 4.9, a strategy with which this framework can be implemented – as a service or system module – within an agent system.

4.2.1 A Running Example

Before describing each of the components of Definition 4.2.1 and how they are instantiated, I present a running example that shall be used to demonstrate various aspects of the ADM framework. I then define an agent of the form described in Definition 4.2.2.

Example 4.2.1 Consider the decision of selecting something to eat. Let $\mathcal{O}$ denote a finite (non-empty) set of possible food items. Preference for one item $o_i \in \mathcal{O}$ over another $o_j \in \mathcal{O}$ can be justified by two means: according to taste (o_i is preferred to o_j if o_i tastes better), and by how healthy they are (o_i is preferred to o_j if o_i is healthier). The item set, from which a most preferred subset must be selected, is $\mathcal{O} = \{Soup, Biscuits, Fish\}$.

Recall from Definition 4.2.2 that an agent using the ADM framework has a number of capabilities. I now define an agent called ALFRED, who I shall refer to in the examples that remain in this chapter. ALFRED's task is to select the item(s) in $\mathcal{O} = \{Soup, Biscuits, Fish\}$ of Example 4.2.1 that it most prefers. For exemplary purposes, I assume that ALFRED's deductive system is a defeasible logic program [García and Simari (2004)] as presented in Definition 3.6.11[4]. It is the case that $\Gamma \rightsquigarrow L$, Γ infers a literal L, if there is a derivation of L from Γ. Definition 3.6.13 describes the nature of a derivation.

In the sections that follow, I shall describe the comparison scheme set SCH of ALFRED, how it determines conflict and preference over arguments, the decision-rule RL it employs, and the content of its defeasible logic program. In addition, I describe each of the components of the ADM framework (identified in Definition 4.2.1). In Section 4.6, I demonstrate how the framework discovers a most preferred subset of choices.

4.3 Comparison Schemes

A comparison scheme encapsulates the circumstances under which one choice of a pair of choices is preferred over another. These schemes can be instantiated to represent comparison by any chosen method of preference handling or aggregation (a variety of which are described in Chapter 2) that can be described in terms of a comparison between two choices. Each scheme is applied to a pair of choices $o_1, o_2 \in \mathcal{O}$ and describes a set of conditions under which o_1 can be considered as preferred to o_2.

Definition 4.3.1 A comparison scheme sc for an agent of the form described in Definition 4.2.2 is a tuple $sc = \langle o_i, o_j, PR, o_i \succ_{sc} o_j \rangle$ where:

1. o_i and o_j denote a pair of distinct choices under comparison ($o_i \neq o_j$);

[4]The operation of the ADM does not depend on how an agent represents and deduces knowledge.

2. $o_i \succ_{sc} o_j$ denotes that o_i is preferred to o_j by scheme sc;
3. PR is a set of abstract premises (conditions), defined with respect to o_i and o_j, such that if $\forall p \in PR$, $\Gamma \rightsquigarrow p$ by Definition 4.2.2, then $o_i \succ_{sc} o_j$ can be inferred.

Let $PR = Premises(sc)$ denote the premises of sc, and $o_i \succ_{sc} o_j = Conc(sc)$ its conclusion.

I now reconsider Example 4.2.1, and show how the comparison of meals by taste and health can be represented in the form of comparison schemes (by Definition 4.3.1).

Example 4.3.1 Let $TastesBetter(o_i, o_j)$ denote that item o_i tastes better than o_j. A comparison scheme based on taste, applied to a choice pair $o_i, o_j \in \mathcal{O}$, is defined as:

$$sc_{taste} = \langle o_i, o_j, \{TastesBetter(o_i, o_j)\}, o_i \succ_{sc_{taste}} o_j \rangle$$

Example 4.3.2 Let $Healthier(o_i, o_j)$ denote that item o_i is healthier than o_j. A comparison scheme based on health, applied to a choice pair $o_i, o_j \in \mathcal{O}$, is defined as:

$$sc_{health} = \langle o_i, o_j, \{Healthier(o_i, o_j)\}, o_i \succ_{sc_{health}} o_j \rangle$$

These schemes form the set $SCH = \{sc_{taste}, sc_{health}\}$ for the decision-making agent ALFRED, introduced in Section 4.2.1. I now define ALFRED's deductive system (its defeasible logic program), describing how it infers the premises of the above schemes.

Example 4.3.3 The deductive system of ALFRED is a defeasible logic program $\Gamma = \langle F, \Delta \rangle$ as defined in Definition 3.6.11. The set of rules Δ contains the following[5]:

$$Expired(o_i) \wedge \neg Expired(o_j) \quad \rightarrow \quad \neg TastesBetter(o_i, o_j) \tag{4.1}$$

$$Sweeter(o_i, o_j) \quad \Rightarrow \quad TastesBetter(o_i, o_j) \tag{4.2}$$

where: $TastesBetter(o_i, o_j)$ is defined as in Example 4.3.1; $Sweeter(o_i, o_j)$ denotes that item o_i is sweeter than o_j; and $Expired(o_i)$ denotes that o_i is expired (past its use-by date). These rules describe that an expired item does not taste better than one that is within its

[5]Variables are denoted by lower case letters, constants by upper case letters.

use-by date, and that sweeter items tend to taste better than those that are less sweet. The decision-making agent ALFRED maintains the following facts within F[6]:

$$F = \left\{ \begin{array}{ccc} Healthier(Fish, Soup) & Sweeter(Biscuits, Fish) & TastesBetter(Soup, Fish) \\ \neg Expired(Fish) & Expired(Biscuits) & Healthier(Soup, Biscuits) \end{array} \right.$$

where $Healthier(o_i, o_j)$ is defined as in Example 4.3.2. With this deductive system, ALFRED can infer that $\neg TastesBetter(Biscuits, Fish)$ (as the *Biscuits* are expired, while the *Fish* is not), and that $TastesBetter(Biscuits, Fish)$ (as *Biscuits* tend to be sweeter than *Fish*).

4.4 The Construction of Arguments

As described in Definition 4.2.1, a decision-maker compares pairs of available choices through the application of a variety of comparison schemes (Definition 4.3.1). There is no guarantee that the collection of schemes a decision-maker has available for application will yield consistent results. One scheme may infer that a choice $o_i \in \mathcal{O}$ is preferred to another $o_j \in \mathcal{O}$, while a different scheme may infer the opposite conclusion. Moreover, in the presence of uncertainty and inconsistency in the knowledge of a decision-maker, the grounds upon which a scheme is applied (the premises used in its construction) can be disputed. For this reason, each scheme and its application represent an argument whose premises and conclusion support a preference for one choice over another.

I now introduce two types of argument that a decision-maker using my framework constructs, representing the components $\mathcal{A}_p$ and $\mathcal{A}_e$ in AS_{ADM} of Definition 4.2.1. The first type of argument is practical. These arguments are constructed in support of a preference relationship between a pair of choices and are instantiations of a comparison scheme (Definition 4.3.1). The set of these arguments is denoted $\mathcal{A}_p$, each treated as an abstract entity within an ADM (as shall be discovered in the sections that follow).

Definition 4.4.1 A practical argument in support of a preference for a choice $o_i \in \mathcal{O}$ over an alternative $o_j \in \mathcal{O}$ is a tuple $a = \langle sc, r \rangle$ where:

1. sc is a comparison scheme (Definition 4.3.1);

[6]The knowledge of the decision-maker is incomplete in this domain as it does not know about the relative taste and healthiness of all available food items.

2. Each $p \in Premises(sc)$ holds with respect to o_i and o_j (as per Definition 4.3.1);
3. $r = o_i \succ_{sc} o_j$ is the conclusion of the scheme sc.

Let $Premises(a) = Premises(sc)$ denote the premises of a, and $r = Conc(a)$ its conclusion.

Example 4.4.1 Consider the meal selection domain of Section 4.2.1, and Examples 4.3.1 to 4.3.3. ALFRED constructs the following practical arguments over the set $\mathcal{O} = \{Soup, Biscuits, Fish\}$, by Definition 4.4.1, using its deductive system Γ (of Example 4.3.3). The choices within this set are denoted $o_1 = Soup$, $o_2 = Biscuits$, and $o_3 = Fish$.

$$\begin{aligned}
a_1 &= \langle\langle o_1, o_2, \{Healthier(o_1, o_2)\}, o_1 \succ_{sc_{health}} o_2\rangle, o_1 \succ_{sc_{health}} o_2\rangle \\
a_2 &= \langle\langle o_1, o_3, \{TastesBetter(o_1, o_3)\}, o_1 \succ_{sc_{taste}} o_3\rangle, o_1 \succ_{sc_{taste}} o_3\rangle \\
a_3 &= \langle\langle o_3, o_1, \{Healthier(o_3, o_1)\}, o_3 \succ_{sc_{health}} o_1\rangle, o_3 \succ_{sc_{health}} o_1\rangle \\
a_4 &= \langle\langle o_2, o_3, \{TastesBetter(o_2, o_3)\}, o_2 \succ_{sc_{taste}} o_3\rangle, o_2 \succ_{sc_{taste}} o_3\rangle
\end{aligned}$$

Argument a_1 in Example 4.4.1 infers that $o_1 = Soup$ is preferred to $o_2 = Biscuits$ as $Soup$ is healthier. Argument a_2 infers that $Soup$ is preferred to $o_3 = Fish$ as $Soup$ tastes better, while a_3 infers that $Fish$ is preferred to $Soup$ because $Fish$ is healthier than $Soup$. Argument a_4 infers that $Biscuits$ are preferred to $Fish$ as $Biscuits$ taste better.

The second type of argument a decision-maker using the ADM framework forms is epistemic (Definition 4.4.2). These arguments are created in support of or against a belief, forming the set $\mathcal{A}_e$. This distinction between practical and epistemic arguments is used in the work of Amgoud et al. (2008b). In line with this work, these arguments are treated as abstract entities – the representation of their premises and conclusion are not restricted.

Definition 4.4.2 An epistemic argument in support of a belief h is a pair $a = \langle H, h\rangle$ where: $H = Premises(a)$ is an abstract set of premises which, by a chosen system of deduction, infer conclusion $h = Conc(a)$ ($h \neq o_i \succ_{sc_k} o_j$ for $o_i, o_j \in \mathcal{O}$ and $sc_k \in SCH$).

Example 4.4.2 Consider again the meal selection domain of Section 4.2.1, and Examples 4.3.1 to 4.4.1. The agent ALFRED constructs the following epistemic arguments, by Definition 4.4.2, using its deductive system Γ (of Example 4.3.3).

$$a_5 = \langle\{Sweeter(Biscuits, Fish), Equation\ 4.2\}, TastesBetter(Biscuits, Fish)\rangle$$
$$a_6 = \langle\{Expired(Biscuits), \neg Expired(Fish), Equation\ 4.1\}, \neg TastesBetter(Biscuits, Fish)\rangle$$

Argument a_5 in Example 4.4.2 supports the conclusion that the *Biscuits* taste better than the *Fish* (as the *Biscuits* are sweeter). Argument a_6 infers that the *Biscuits* do not taste better than the *Fish* (as the *Biscuits* are expired and the *Fish* is not).

4.5 Argument Attack and Interaction

To determine which of the epistemic ($\mathcal{A}_e$) and practical ($\mathcal{A}_p$) arguments formed by an agent using the ADM framework should be accepted – defining a justified set of beliefs and preference relationships – an attack relation is constructed over the set $\mathcal{A}_e \cup \mathcal{A}_p$. I mention in Section 4.2 that this attack relation is derived from a semi-abstract conflict and preference relation over $\mathcal{A}_e \cup \mathcal{A}_p$. This conflict $\mathcal{R}$ and preference $\succeq$ relation form two components of AS_{ADM} in Definition 4.2.1. Before describing this process, I consider the variety of ways in which existing argumentation systems define argument attack.

4.5.1 Argument Attack within Existing Work

Amgoud and Cayrol (1998, 2002b,a) define a preference-based argumentation framework (Section 3.4.1 of Chapter 3) in which a conflict and a (partial or total, reflexive and transitive) preference relation over a set of arguments leads to the formation of an attack relation. An argument attacks another if it is in conflict with it and it is not that case that the latter argument is strictly preferred (the strict relation being irreflexive and asymmetric). Amgoud et al. (2008b) describe attack amongst arguments in terms of an abstract (reflexive and transitive) preference relation employed in the same manner. In the value-based argumentation frameworks of Bench-Capon (2003a,b) (Section 3.4.2), each argument promotes a value and a partial ordering over values (an irreflexive, transitive, and asymmetric relation) determines an attack relation. An argument attacks another if it is in conflict with it, and it is not the case that the latter promotes a more important value.

The formalisations of attack described above are similar in the sense that the attack of one argument a_i against another a_j occurs if a_i conflicts with a_j and the latter argument a_j is not strictly preferred to a_i. The when and how of argument conflict is typically dependent on the structure and content of the arguments in question. Pollock (1987) defines two varieties of attack – undercutting and rebuttal (described in Section 3.1.2)

– variations of which are employed in a wide range of argumentation frameworks (see the surveys of Chesñevar et al. (2000), Prakken and Sartor (2002), Prakken (2005b), and Carbogim et al. (2000)). What constitutes an undercut and rebuttal by one argument against another – and consequently argument conflict – varies between frameworks.

Prakken and Sartor (1997) define arguments as sequences of ground strict and defeasible rules. An argument a_i conflicts with another a_j if there exists sequences of strict rules R_1 and R_2 such that $a_i \cup R_1$ infers a contradictory conclusion to that of $a_j \cup R_2$. Alternatively, a_i conflicts with a_j if the conclusion of a_i disputes an assumption employed in the construction of a_j (an assumption is a literal that appears in a rule of an argument). Prakken and Sartor (1996a) interpret conflict between these rule-based arguments differently – an argument a_i conflicts with an argument a_j if a_i contains a rule with a contradictory consequent to a rule of a_j, or the conclusion of a_i contradicts an assumption of a_j. Dung (1993b) views arguments as sets of assumptions which can be used to infer a conclusion through the application of rules in a logic program. An argument conflicts with another if the union of their assumptions infers a contradiction (rebuttal), or if one argument infers a conclusion that contradicts an assumption in the other (undercutting).

4.5.2 Argument Attack and the ADM

In light of this variation in the definitions of conflict described above, I take a leaf out of the book of Amgoud et al. (2008b) in the definition of attack within my framework. I treat the supplied conflict $\mathcal{R}$ and preference $\succeq$ relations over the arguments $\mathcal{A}_e \cup \mathcal{A}_p$ as abstract (with certain restrictions, as I shall describe). I prescribe the means by which practical arguments conflict, while leaving conflict between epistemic arguments, and between epistemic and practical arguments, to the desires of the agent instantiator.

A goal of the ADM framework is to determine which pair-wise preference relationships exist between available choices. These relationships, in conjunction with a decision-rule (Section 4.6), recommend to an agent a most preferred choice subset. The given set of practical arguments suggest which pair-wise preferences are valid. These arguments conflict, however, with several arguments potentially supporting opposing relationships (ie. $o_i \succ_{sc_i} o_j$ and $o_j \succ_{sc_j} o_i$ given $o_i, o_j \in \mathcal{O}$ and $sc_i, sc_j \in SCH$). The collection of epis-

temic arguments forms a basis for potentially rejecting certain relationships (for example, by disputing that a premise within the comparison scheme of an argument holds).

The selection of an acceptable subset of $\mathcal{A}_p$ within the ADM framework determines which preference relationships are justified. Definition 4.5.1 describes how an attack relation is formed over the collection of epistemic ($\mathcal{A}_e$) and practical ($\mathcal{A}_p$) arguments instantiated by a decision-making agent, drawing on the work of Amgoud et al. (2008b).

Definition 4.5.1 Let $AS_{ADM} = \langle \mathcal{O}, \mathcal{A}_e, \mathcal{A}_p, \mathcal{X}, RL \rangle$ be an ADM as defined in Definition 4.2.1. The component $\mathcal{X}$ is a triple $\langle \mathcal{R}, \succeq, Sem \rangle$ where:

1. $\mathcal{R}$ is an irreflexive (neither symmetric nor asymmetric) binary conflict relation over $\mathcal{A}_e \cup \mathcal{A}_p$ such that $\forall a_i, a_j \in \mathcal{A}_e \cup \mathcal{A}_p$, $(a_i, a_j) \in \mathcal{R}$ if and only if:
   (a) $a_i \in \mathcal{A}_e$, $a_j \in \mathcal{A}_e \cup \mathcal{A}_p$, and a_i is in conflict with a_j[7]; or
   (b) $a_i, a_j \in \mathcal{A}_p$, $Conc(a_i) = o_i \succ_{s(a_i)} o_j$ and $Conc(a_j) = o_j \succ_{s(a_j)} o_i$, where $s(a_k)$ denotes the comparison scheme used in argument a_k and $o_i, o_j \in \mathcal{O}$[8].
2. $\succeq$ is a reflexive and transitive preference relation over $\mathcal{A}_e \cup \mathcal{A}_p$, and consists of the union of the following three relations: $\succeq_e$, $\succeq_p$, $\succeq_m$ where:
   (a) $\succeq_e$ ($\succeq_p$) is a reflexive and transitive relation over $\mathcal{A}_e$ ($\mathcal{A}_p$) where $\forall a_i, a_j \in \mathcal{A}_e$ ($\mathcal{A}_p$), $(a_i, a_j) \in \succeq_e$ ($\succeq_p$) if and only if a_i is at least as good (preferred) as a_j;
   (b) $\succeq_m$ is an irreflexive and asymmetric relation over $\mathcal{A}_e \cup \mathcal{A}_p$ such that $(a_i, a_j) \in \succeq_m$ (a_i is preferred to a_j) if and only if $a_i \in \mathcal{A}_e$ and $a_j \in \mathcal{A}_p$.
3. *Sem* is defined as in Definition 4.2.1.

$\mathcal{W}$ is a binary attack relation over $\mathcal{A}_e \cup \mathcal{A}_p$. Let $\succ$ denote the strict (asymmetric) relation associated with $\succeq$. For each pair of arguments $a_i, a_j \in \mathcal{A}_e \cup \mathcal{A}_p$, $(a_i, a_j) \in \mathcal{W}$ if and only if $(a_i, a_j) \in \mathcal{R}$ (a_i conflicts with a_j) and $(a_j, a_i) \notin \succ$ (a_j is not strictly preferred to a_i).

The preference relation $\succeq$ in Definition 4.5.1 describes which arguments are stronger or more conclusive than others. This relation requires that epistemic arguments be strictly preferred to those of the practical variety. Consequently, for each argument $a_i \in \mathcal{A}_e$ that

[7] I do not prescribe how epistemic arguments conflict with each other and with practical arguments.

[8] Conflict amongst practical arguments is prescribed – accepting arguments that support opposing preferences defeats the purpose of the ADM as a tool to determine which relationships are to be accepted.

is in conflict with an argument $a_j \in \mathcal{A}_p$, a_i attacks a_j[9]. The attack relation $\mathcal{W}$ in Definition 4.5.1 is defined in the same manner as that within a PAF (as per Section 3.4.1).

Recall the agent ALFRED and the meals domain of Examples 4.2.1 to 4.4.2. Example 4.5.1, below, demonstrates how ALFRED might determine conflict amongst arguments.

Example 4.5.1 Let $a_i, a_j \in \mathcal{A}_e \cup \mathcal{A}_p$, $o_i, o_j \in \mathcal{O}$, and $sc_i, sc_j \in SCH$. ALFRED determines that argument a_i conflicts with a_j if and only if:

1. $a_i, a_j \in \mathcal{A}_e$, $Conc(a_i) = h$ and $Conc(a_j) = \neg h$ (rebuttal); or
2. $a_i, a_j \in \mathcal{A}_p$, $Conc(a_i) = o_i \succ_{sc_i} o_j$ and $Conc(a_j) = o_j \succ_{sc_j} o_i$ (rebuttal); or
3. $a_i \in \mathcal{A}_e$ and $a_j \in \mathcal{A}_e \cup \mathcal{A}_p$, $Conc(a_i) = h$ and $\neg h \in Premises(a_j)$ (undercutting).

ALFRED's definition of conflict between practical and epistemic arguments, shown in Example 4.5.1, is now applied to those it constructs in Examples 4.4.1 and 4.4.2.

Example 4.5.2 Consider the set of epistemic arguments $\mathcal{A}_e = \{a_5, a_6\}$ in Example 4.4.2, and practical arguments $\mathcal{A}_p = \{a_1, a_2, a_3, a_4\}$ in Example 4.4.1. Arguments a_5 and a_6 rebut as a_5 infers that $TastesBetter(Biscuits, Fish)$ while a_6 that $\neg TastesBetter(Biscuits, Fish)$. Argument a_6 undercuts argument a_4 as $TastesBetter(Biscuits, Fish)$ is one of the premises of a_4. Arguments a_2 and a_3 rebut as a_2 infers that $Soup$ is preferred to $Fish$ on the basis of taste, while a_3 infers the opposite on the basis of health. The conflict relation $\mathcal{R}$ formed by ALFRED over the arguments $\mathcal{A}_e \cup \mathcal{A}_p$ (as per Definition 4.5.1) is:

$$\mathcal{R} = \{(a_2, a_3), (a_3, a_2), (a_5, a_6), (a_6, a_5), (a_6, a_4)\}$$

To form an attack relation over the arguments in the meals domain (as per Definition 4.5.1), a method of assessing the relative strength of arguments must be proposed. I now define how ALFRED might determine preference over arguments. In doing so, $\succeq$ of Definition 4.5.1 is characterised, forming a component of AS_{ADM} in Definition 4.2.1.

Example 4.5.3 Let $\mathcal{I}_{SCH}$ denote a partial, reflexive, and transitive prioritisation over a set of comparison schemes SCH, such that $\mathcal{I}_{SCH}(sc_i, sc_j)$ if and only if $sc_i \in SCH$ is at least

[9] As described by Amgoud et al. (2008b) this avoids wishful thinking, where the truth of knowledge takes a back seat if it can be used to demonstrate a powerful reason for selecting a choice.

as high in priority as $sc_j \in SCH$. Assume that ALFRED has such a prioritisation over its schemes SCH. Let $s(a)$ be the scheme employed in the construction of argument $a \in \mathcal{A}_p$. ALFRED determines that an $a_i \in \mathcal{A}_e \cup \mathcal{A}_p$ is preferred to $a_j \in \mathcal{A}_e \cup \mathcal{A}_p$ if and only if:

1. $a_i, a_j \in \mathcal{A}_p$ and $\mathcal{I}_{SCH}(s(a_i), s(a_j))$; or
2. $a_i \in \mathcal{A}_e$ and $a_j \in \mathcal{A}_p$; or
3. $a_i, a_j \in \mathcal{A}_e$ and the number of defeasible rules employed in the construction of a_i is less than or equal to that of a_j.

ALFRED's definition of preference between practical and epistemic arguments, shown in Example 4.5.3, is now applied to those it constructs in Examples 4.4.1 and 4.4.2.

Example 4.5.4 Let $\mathcal{I}_{SCH} = \{(sc_{taste}, sc_{health}), (sc_{taste}, sc_{taste}), (sc_{health}, sc_{health})\}$. The strict form of the preference relation $\succeq$ formed by ALFRED over the arguments $\mathcal{A}_e \cup \mathcal{A}_p$ is:

$$\succ = \{(a_2, a_1), (a_2, a_3), (a_4, a_1), (a_4, a_3), (a_6, a_5), (a_5, a_1), (a_5, a_2), (a_5, a_3), (a_5, a_4), (a_6, a_1), (a_6, a_2), (a_6, a_3), (a_6, a_4)\}$$

Armed with a definition of conflict and preference amongst arguments within the meals domain, the formation of an attack relation $\mathcal{W}$, as described in Definition 4.5.1, over the arguments in $\mathcal{A}_e \cup \mathcal{A}_p$ can now be demonstrated.

Example 4.5.5 Given the conflict relation $\mathcal{R}$ of Example 4.5.2, and the strict relation $\succ$ (of $\succeq$) of Example 4.5.4, the attack relation $\mathcal{W}$ of Definition 4.5.1 in the meals domain is:

$$\mathcal{W} = \{(a_6, a_5), (a_2, a_3), (a_6, a_4)\}$$

4.6 Argument Acceptance and Choice Selection

In Sections 4.3 to 4.5, I have described the nature of arguments constructed by a decision-maker using the ADM framework and the formation of an attack relation over those arguments. I have defined this ADM as a 5-tuple $AS_{ADM} = \langle \mathcal{O}, \mathcal{A}_e, \mathcal{A}_p, \mathcal{X}, RL \rangle$ in Definition 4.2.1. In this section, I describe how this framework selects a most preferred subset of choices within the set $\mathcal{O}$ upon which its arguments ($\mathcal{A}_e \cup \mathcal{A}_p$) have been instantiated.

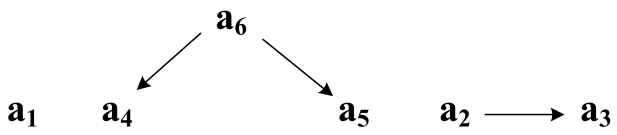


Figure 4.1: A Dung (1995) argumentation framework derived within the meals domain.

The conclusions of the set of practical arguments $\mathcal{A}_p$ describe a range of preference relationships available for acceptance. To determine which pair-wise preference relationships should be accepted, a Dung (1995) argumentation framework is formed given the arguments $\mathcal{A}_e \cup \mathcal{A}_p$ and the attack relation $\mathcal{W}$ over those arguments (as defined in Definition 4.5.1). Section 3.2 of Chapter 3 describes the Dung (1995) argumentation framework and its associated acceptability semantics. The ADM must select a consistent subset of these pair-wise preference relationships for acceptance – it cannot accept a preference for one choice over another while simultaneously accepting an opposing statement. For this reason, sceptical acceptance under a desired semantics of Dung (1995) is used.

Definition 4.6.1 Consider an ADM $AS_{ADM} = \langle \mathcal{O}, \mathcal{A}_e, \mathcal{A}_p, \mathcal{X}, RL \rangle$. An attack relation $\mathcal{W}$ over $\mathcal{A}_e \cup \mathcal{A}_p$ is derived from $\mathcal{X} = \langle \mathcal{R}, \succeq, Sem \rangle$ as described in Definition 4.5.1. To determine the acceptable subset of $\mathcal{A}_e \cup \mathcal{A}_p$, a Dung (1995) argumentation framework $\langle \mathcal{A}_e \cup \mathcal{A}_p, \mathcal{W} \rangle$ is formed. The acceptable subset of $\mathcal{A}_e \cup \mathcal{A}_p$ is the collection of these arguments that are sceptically accepted under the chosen semantics *Sem* [Dung (1995)].

Example 4.6.1 Consider the meals domain of Examples 4.2.1 to 4.5.5. The Dung (1995) argumentation framework formed in this domain is shown below, and in Figure 4.1.

$$AF = \langle \{a_1, a_2, a_3, a_4, a_5, a_6\}, \{(a_2, a_3), (a_6, a_5), (a_6, a_4)\} \rangle$$

Assume that the decision-making agent ALFRED chooses the preferred semantics as *Sem* in Definition 4.2.1. The acceptable subset of $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ – arguments sceptically accepted under the preferred semantics of Dung (1995) – is $\{a_1, a_2, a_6\}$.

Within the ADM framework, the conclusions of the acceptable subset of $\mathcal{A}_p$ form a dominance graph over the choices in $\mathcal{O}$. The vertices of this graph are the choices in $\mathcal{O}$,

Figure 4.2: A dominance graph over choices in the meals domain.

while its edges denote dominance relationships amongst them. A directed edge from one choice $o_i \in \mathcal{O}$ to another $o_j \in \mathcal{O}$ indicates that o_i dominates (is strictly preferred to) o_j.

Definition 4.6.2 A dominance graph is a directed graph $\mathcal{G} : \mathcal{V} \times \mathcal{E}$ of vertices $\mathcal{V}$ and edges $\mathcal{E}$. $\mathcal{V}$ is a collection of choices $\mathcal{O}$ and $\mathcal{E}$ is an asymmetric, irreflexive and potentially cyclic preference relation over $\mathcal{O}$. A directed edge from one choice $o_i \in \mathcal{O}$ to another $o_j \in \mathcal{O}$ exists in this graph if there exists an acceptable argument within an ADM instantiated with respect to $\mathcal{O}$ whose conclusion is $o_i \succ_{sc_i} o_j$. In this case, we say that o_i dominates o_j.

Example 4.6.2 Consider the acceptable subset of practical arguments $\mathcal{A}_p$ within the meals domain, $\{a_1, a_2\}$, discovered in Example 4.6.1. Given the set of available choices $\mathcal{O} = \{Soup, Biscuits, Fish\}$, these acceptable arguments result in the following dominance relationships amongst choices $\{(Soup, Biscuits), (Soup, Fish)\}$, as shown in Figure 4.2.

A dominance graph formed within the ADM does not define a ranking over choices – it is, as described in Definition 4.6.2, a potentially cyclic structure. In the ADM framework, the decision-rule RL is a function that when applied to a dominance graph over a collection of choices $\mathcal{O}$ returns the subset of most preferred choices within $\mathcal{O}$. I define these rules in the style of social choice rules considered within voting theory [Sen (1995); Brandt et al. (2007); Arrow et al. (2002)] – these rules determine which candidate (choice) of a candidate set should be selected given the pair-wise rankings of a range of voters.

Definition 4.6.3 Let GR denote a dominance graph formed over a set of choices $\mathcal{O}$ (as per Definition 4.6.2). The decision-rule RL of Definition 4.2.1 is a function $RL : GR \rightarrow \mathcal{O}'$ where $\mathcal{O}' \subseteq \mathcal{O}$ denotes the (non-empty) subset of most preferred choices within $\mathcal{O}$.

The definition of a dominance graph and decision-rule in Definitions 4.6.2 and 4.6.3 are not novel concepts. In the series of ELECTRE multi-criteria decision-making algorithms [Roy (1991)], outranking relations over choices take the form of a dominance

graph. In these methods, a decision-maker is presented with a set of alternatives $\mathcal{O}$ and has a set of criteria $\mathcal{C}_e$ with which to evaluate them. The evaluation of each choice $o \in \mathcal{O}$ with respect to a criterion $c_i \in \mathcal{C}_e$ yields a real number $c_i(o)$ describing the desirability of o with respect to c_i. An outranking relation OR_i associated with each criterion c_i is defined such that $OR_i(o_1, o_2)$ given $o_1, o_2 \in \mathcal{O}$ if and only if $c_i(o_1) \geq c_i(o_2)$ (o_1 is at least as good as o_2 with respect to c_i). A comprehensive or aggregate outranking relation OR over choices is defined such that $OR(o_1, o_2)$ if the majority of criteria support this assertion, and those that oppose it do not do so too strongly[10]. The outranking relation OR is not necessarily complete or transitive, and so forms a graph over choices rather than an ordering.

A rule or method is required to select preferred choices given this graph – for example, by selecting the minimal (with respect to set inclusion) set of choices such that (i) no choice within the set outranks another member, and (ii) each member outside the set is outranked by a member of the set. There are many alternative ways in which a decision-rule can be defined. One approach is to consider the literature on social choice theory.

4.6.1 Social Choice Theory and Decision-Rules

One aspect of social choice theory is the study of mechanisms by which a maximal or most preferred subset of choices can be determined on the basis of a dominance graph over them. This graph is often formed by aggregating individual pair-wise preferences over the choice set. Each preference may belong to a member of a group – the result of voting for several candidates in an election. An edge in this graph exists, for example, if a majority of individuals prefer one choice to another. There are a variety of decision (or choice) rules in existence [Brandt et al. (2007)] designed for such a purpose, one of which is the Schwartz set [Schwartz (1972)]. Example 4.6.3 demonstrates the use of the Schwartz set in the definition of a decision-rule of the style shown in Definition 4.6.3.

Example 4.6.3 Given a set of choices $\mathcal{O}$, and a dominance graph GR over $\mathcal{O}$:

1. A Schwartz set component of $\mathcal{O}$ is a subset $SCO \subseteq \mathcal{O}$ such that:
   (a) There does not exist a choice $o_i \in \mathcal{O}$ where $o_i \notin SCO$ that dominates (by Definition 4.6.2 and GR) an alternative $o_j \in SCO$;

[10] How this comprehensive outranking relation is formed varies across ELECTRE formalisms [Roy (1991)].

(b) No non-empty subset $SCO' \subset SCO$ satisfies property (a).

2. The union of all Schwartz set components of $\mathcal{O}$ is the Schwartz set of $\mathcal{O}$ ($Z(GR)$).

The Schwartz set decision-rule is denoted $RL(GR) = Z(GR)$.

Example 4.6.4 Within the meals domain, Example 4.6.2 shows a dominance graph GR over available choices $\mathcal{O}$. The Schwartz set of $\mathcal{O}$ ($Z(GR)$), as defined in Example 4.6.3, is $\{Soup\}$. This is also the set of undominated choices (choices for which no other choice dominates them) within $\mathcal{O}$. The Schwartz set applies even to cyclic dominance graphs.

In the next section, I evaluate the framework I have described in the preceding sections with respect to the questions of Criterion 2.1.1.

4.7 Evaluation: A Test for Decoupling

In Criterion 2.1.1, I have specified two questions that, if answered in the affirmative by a decision-making system or framework, indicate that it is decoupled from the user preferences it operates on. The first question asks whether such an approach is able to support the variety of preference representations that can be used in the comparison of two choices, while the second requires that it abstract away from their representational detail.

The ADM framework is a tuple of 5 components, that once instantiated are manipulated as abstract entities by a process that selects a most preferred subset of a given set of choices. The arguments that form one component of this tuple are derived from the application of comparison schemes – each scheme defining the conditions under which one choice is preferred to another. The use of abstract comparison schemes and the formation of arguments in support of preference relationships between choices allows this system to work with the desired range of user preference expressions. The preferences of a human decision-maker can be employed in the selection of a choice provided they can be expressed in terms of a comparison between two choices. Any of the approaches for preference representation discussed in Chapter 2 can be captured within the comparison schemes of Definition 4.3.1. In the selection of a most preferred subset of choices, the ADM does not require a knowledge of the content of these comparison schemes.

Criterion 2.1.1 requires that the ADM abstract away from the representational detail of arguments, and the schemes used in their construction, during decision-making. An ADM has provided to it a collection of practical and epistemic arguments, and a conflict and preference relation over those arguments. With this information, the framework applies a process to select a most preferred subset of a set of choices that treats these arguments and relations as abstract entities. So few restrictions are placed on the nature of: comparison schemes, epistemic arguments, conflict amongst arguments, and preference over arguments, that these entities can be configured in a vast number of different ways. The strategy I have used to achieve a decoupling of the decision-making process from the preferences of a user is to capture the tasks that require a knowledge of their detail into structures or functions that can be manipulated as abstract entities by this process.

I have answered each of the questions of Criterion 2.1.1 in the affirmative, demonstrating the decoupled nature of the ADM. I consider in the conclusion of this chapter (Section 4.9) how this system can be used in the development of the range of decision recommendation applications mentioned thus far in this thesis. These tools range from personal online shopping assistants [Menczer et al. (2002); Kim et al. (2009)] to online travel agents and planners [Camacho et al. (2001); Yim et al. (2004)].

4.8 A Comparison with Related Work

I have described a range of argumentation-based decision-making systems in Section 3.6 of Chapter 3. These systems vary in the way they capture the preferences of a decision-maker and the way they select or rank choices on the basis of arguments. The goal of this chapter has been to create a system that is capable of distinguishing choices, through arguments, by general mechanisms of comparison. It is on this basis that I have evaluated this system, and the existing frameworks of Section 3.6. Of these approaches, the ADM of this chapter is the only one that answers the questions of Criterion 2.1.1 satisfactorily.

I have not considered whether the ADM framework is able to replicate the behaviour of this existing work. Each approach uses a different method of ranking or selecting choices on the basis of arguments generated from decision-maker preferences. Amgoud et al. (2008b), for example, assign statuses to choices and an ordering over statuses results

in an ordering over choices. A sceptical option, for example, is one that is supported by a 'pro' argument in all extensions of their framework (by a given semantics). I have proposed what I believe to be a reasonable method of selecting choices on the basis of arguments that support preference relationships. This method reflects the way that a variety of tools within more traditional decision analysis operate – extracting preferred choices from a preference or dominance graph defined over them [see Roy (1991)].

Given the disparity across existing work with respect to the meaning of arguments, whether choices are ranked or partitioned into two sets, and how this ranking or partitioning takes place, it is unrealistic to imagine that one system could generalise each of these approaches. My goal has been to support the manner in which these methods compare choices, not the strategies they use in aggregating comparisons. In Section 4.9, I consider a range of future directions in which the ADM can be developed. One topic of future work is to determine which approaches this system is able to generalise.

4.9 Concluding Remarks and Future Work

In this chapter I have described a framework for argumentation-based decision-making, and demonstrated its use within a meal selection domain. I have developed this framework with a set of desirable features in mind. In Criterion 2.1.1, I present two questions that if answered in the affirmative with respect to the ADM framework indicate that it is decoupled from the preferences it operates on. The ADM framework is required to be capable of distinguishing choices by general means – supporting the range of ways in which a user expresses their preference and compares choices. Moreover, the ADM must be abstract – able to operate on arguments without a knowledge of, or restricting, the kinds of preference used in their construction. I have evaluated the ADM framework with respect to these questions and have found it to satisfactorily answer each.

To achieve these properties, the ADM framework was constructed by extracting elements from the prevailing state-of-the-art in argumentation-based decision-making. I discovered in Section 3.6 that no existing approach satisfactorily answered each question of Criterion 2.1.1, but some appropriately answered a subset. The use of abstract epistemic and practical arguments, and an abstract formulation of their interaction, as

it appears in the work of Amgoud et al. (2008b), was incorporated into the ADM to satisfy the requirement of abstractness. The schematic reasoning patterns of Ouerdane et al. (2008) were employed as a means of capturing the preferences of a user – each scheme based on a set of abstract premises. The ADM abstracts away from the content of practical arguments, constructed by instantiating these schemes, during decision-making.

The concept of decision or choice rules, as used in voting theory [Sen (1995); Brandt et al. (2007); Arrow et al. (2002)], has been employed within the ADM to select a most preferred subset of choices from a series of pair-wise preference relationships. A key difference between my approach and many existing argumentation-based decision-making systems, is that an agent using the ADM framework constructs arguments in support of pair-wise preference relationships (between choices), in place of arguments in support of or against single decisions only. Such arguments are able to reflect comparative style preferences that exist only between one decision and certain others.

Meta-argumentation frameworks employ argumentation to reason about the preferences that exist between arguments. Modgil (2006b) describe the application of a Dung (1995) argumentation framework to itself to determine the validity of its attacks. Where attacks have been determined on the basis of preference between arguments, these attacks can be attacked on the basis that this preference does not hold. The goal of the ADM can be viewed as similar to that of this higher level of argumentation – the ADM is designed to determine which preference relationships hold between decisions, while meta-argumentation can be employed to determine which preference relationships hold between arguments in a given framework. Meta-argumentation formalisms are often hierarchical (or 'levelled') applications of existing frameworks (such as that of Dung (1995), preference-based frameworks [Amgoud and Cayrol (1998, 2002b,a), Section 3.4.1], and value-based frameworks [Bench-Capon (2003a,b), Section 3.4.2]) – see the work of Modgil (2006a,b, 2007), and Modgil and Bench-Capon (2008, 2010) for examples.

The frameworks that form part of these meta-argumentation formalisms are not, by themselves, decision-making tools – they must be instantiated appropriately for this purpose. As described in Section 4.2, the ADM is an instantiation of a preference-based framework [Amgoud and Cayrol (1998, 2002b,a)] – designed to construct a network of prefer-

ences amongst choices – combined with a rule that operates on this network to select a preferred subset of choices. In alternative work, Amgoud et al. (2000) consider reasoning about the preferences that exist between arguments, but do so through the integration of differing preference orderings (each the result of different criteria or perspectives) – by, for example, aggregating these orderings on the basis of a priority relation over them. In the ADM, the means by which decisions are compared form the basis of arguments. These comparisons can be disputed if the premises they rely upon are suspect – this is not a feature of the work of Amgoud et al. (2000). Brewka (2001) considers meta-argumentation within a dialectic exchange (formalised in a prioritised default logic) in which default rules are used to assert and contradict propositions during a dialogue between multiple parties. The priority of default rules is also subject to debate – with default rules defined to assert preference over rules. While the work of Brewka (2001) describes a means of reasoning about preference between objects, it is not an abstract argumentation system.

The argumentation-based decision-making framework I have developed in this chapter plays an important role in the development of argumentation-based search – a technique I present in Chapter 8. This work represents a significant application of the ADM. The ADM framework is designed to be used within the type of applications discussed in Chapter 1 by a higher level system that is able to instantiate it appropriately. Its components, as described in Definition 4.2.1, are a choice set, arguments, a conflict and preference relation over arguments, and a decision-rule. These components can be instantiated in a flexible way by the higher level system, and the process of this chapter applied to select a preferred choice subset (while treating these components as abstract entities). The framework can be implemented as a service or system module that can be deployed within these kinds of applications – removing the need for such a higher level system to instantiate it. This service would be provided with a set of inputs – comparison schemes, a deductive system, and a set of choices – and a most preferred subset of these choices returned. I discuss this, and additional items of future work, in the paragraphs that follow.

There are a variety of additional directions in which the ADM framework can be developed. I have not considered within this chapter whether this approach is a generalisation of other systems within the literature on argumentation-based decision-making. I have

found in Section 3.6 that these approaches are quite disparate in terms of how they construct arguments, interpret their meaning, and use them in the selection or ranking of choices. While it is unlikely that any system or framework could capture the behaviour of each of these approaches, a determination of which offerings (if any) the ADM is a generalisation of, is a topic of future work. The application of the framework for use in multi-agent decision-making is another prime area of future work. I have demonstrated a link between the use of social choice in the selection of decisions on the basis of pair-wise preference relationships, and my use of dominance graphs and decision-rules. A key future direction in which the ADM can be taken is the potential integration and fusion of individual instantiations of the framework for use in multi-agent decision-making.

I believe the ADM framework can be implemented as a service or module that provides decision-support to an agent. In doing so, some of its flexibility would be restricted. This service would be supplied with the choices over which decision-making is taking place, a set of comparison schemes reflecting the preferences of the agent, a function that encapsulates its deductive system, and a prioritisation over comparison schemes. The service or module could then instantiate the ADM framework, constructing arguments by instantiating comparison schemes – feeding their premises into the given deductive system to determine if they hold. If the manner in which conflict occurs between epistemic arguments, and epistemic and practical arguments, is constrained, this deductive system can also be employed to derive a conflict relation over arguments. Restricting the nature of conflict to undercutting and rebuttal, the system need only require knowledge of which collections of premises oppose each other, or lead to the inference of opposing pieces of information. A decision-rule and semantics by which arguments are tested for acceptability can be implemented within the system as fixed (if not supplied as inputs).

This chapter brings us to the end of Part I of this thesis. In the next collection of chapters, I shift my focus away from decision-making over a set of choices (or actions) to consider automated planning with preferences.

Part II

Planning with Preferences

Introduction to Part II

THUS far in this thesis I have considered the problem of automated decision-making – the design of tools that select a preferred choice from a finite set of alternatives. Such approaches have application in a vast array of domains, from the recommendation of products to purchase [Wellman et al. (2002)], the arrangement of a meeting [Chalupsky et al. (2002); Tambe et al. (2008)], to the selection of a restaurant to frequent. In the chapters that follow, I transition into the study of planning with preferences. I consider problems in which a sequence of decisions must be made, charting a course of action for an agent or human user to follow in pursuit of their goals. I focus on a particular formalism of agent-based planning – the GOLOG family of agent programming languages [Levesque et al. (1997); de Giacomo et al. (2000)]. The goal in these chapters is to develop a decoupled approach to planning with preferences within the GOLOG space.

In Chapter 5, I introduce the situation calculus [McCarthy and Hayes (1969); Reiter (1991)] and the GOLOG family of agent programming languages. I first describe the key components of a situation calculus basic action theory – a theory that characterises the dynamics of a domain. I follow with an account of regression – a method of determining what holds in an environment as actions are performed. In addition, I highlight the expressiveness of basic action theories relative to that of the STRIPS [Fikes and Nilsson (1971)] and ADL [Pednault (1989)] action representations. These formalisms lay the foundation for a number of existing preference-based planning tools, discussed in Chapters 6 and 8. I conclude by considering the use of basic action theories in the search for an execution of a GOLOG program. These programs describe high level plans or instruction sets, each designed to achieve one or more goals. The executions of such programs, found by an interpreter, are concrete action sequences for robotic and agent control.

The GOLOG language has been used in a variety of agent-based applications, from the development of a robot for the delivery of mail [Lespérance et al. (1999)] to a personal banking assistant [Lespérance et al. (1997)]. Tam et al. (1997) consider the embedding of GOLOG within mobile robots, Burgard et al. (1998) develop a GOLOG-based robotic museum tour guide, Ferrein et al. (2005) employ GOLOG in the control of robotic soccer players, while McIlraith and Son (2002) and Sohrabi et al. (2006) consider the application

of GOLOG in the composition of web-services in travel planning domains. In each of these applications, the ability to find desirable or preferred program executions is important. A travel planner is not a useful tool unless it books flight, car, and hotel arrangements that meet a traveller's preferences over airlines, car models, and hotel chains. A museum tour guide will provide a more pleasant tour experience if it selects exhibits to visit on the basis of group interest. The ability to find preferred executions of a GOLOG problem is important. The decoupling of these preferences from the interpretation process allows these applications, as described in Chapter 1, to cater for a wide range of users within a wide range of decision-making and planning scenarios.

In Chapter 6, I take a detour from the main theme of this thesis – the creation of decoupled tools for decision-making. I devise a GOLOG interpreter that aims to find most preferred program executions with respect to a numeric evaluation function. I apply relaxation-based heuristics – often used in classical planning – to find (near) optimal executions of a GOLOG program. An optimal execution is one that minimises the given evaluation function. In doing do, I present a theory of relaxed regression for the approximate interpretation of a GOLOG program. This relaxed interpreter is used to heuristically evaluate the desirability of available choices in the search for a program execution.

I evaluate, in Chapter 7, the performance of the presented heuristic GOLOG interpreter (described in Chapter 6) across a range of domains – from spacecraft control, the automation of operations in a mine, to task scheduling. I devise a number of alternative interpreters, reflective of the state-of-the-art in preference-based GOLOG, and pit them against my heuristic interpreter across the considered domains. I evaluate these interpreters in terms of the quality of executions they discover. In doing so, the choices I have made in the design of this heuristic interpreter are experimentally justified.

Within Chapter 8, focus is returned to the decoupling of preferences during planning. I present the final contribution of this thesis – an algorithm for argumentation-based search. This algorithm is applied in the construction of an argumentation-based interpreter for GOLOG programs. Existing techniques developed to discover most preferred program executions, and indeed most preferred solutions to planning problems in general, are limited in terms of the kinds of preference (or means by which plans are com-

pared) that can be used to guide search for a solution. The presented interpreter applies the argumentation-based decision-making system of Chapter 4 to select paths to pursue in the search for a most preferred execution. To the best of my knowledge, this is the first work to integrate argumentation and the interpretation of GOLOG programs, and to use argumentation as a tool in search. Moreover, these works inherit the properties of this decision-making system to form a decoupled preference-based GOLOG interpreter.

Chapter 5
The Situation Calculus & GOLOG

THE situation calculus is a second order language for characterising and reasoning about dynamically changing domains, worlds, and environments [McCarthy and Hayes (1969)]. Within the situation calculus, these environments progress through a series of situations – each situation a history of actions performed since an initial state or position. In this chapter, I describe the language of the situation calculus, its foundational axioms, and the formation of basic action theories. These basic action theories express the properties of initial states or situations, the actions available to agents and their characteristics, and the impact these actions have on a world environment.

The determination of what environment properties, expressed as logical formulae, hold in a situation is achieved through the use of situation calculus regression (Section 5.1.6). The modus operandi of regression is to transform each formula under evaluation into a formula that involves no situation other than the initial. The result of this transformation is a formula that can be evaluated with a first order theorem prover, using only the information that characterises the initial situation. In Section 5.1.6, I describe the rules of regression, highlighting how a formula in the situation calculus, on the basis of its structure, can be shifted one step closer to being uniform in the initial situation.

These tools provide support for the development of expressive planning mechanisms, of which the GOLOG family of programming languages [Levesque et al. (1997)] forms a subset (Section 5.3). Grounded in the situation calculus, these languages harness its power for the purpose of agent control. GOLOG is a language for describing high level

task specifications in the situation calculus, originally designed as a tool for programming robots. The language has garnered a number of applications, not least among them its use in programming agents. Providing agents with a task specification – a set of actions that must be performed to achieve their goals and objectives – is fundamental in their construction. In the GOLOG formalism, such a specification takes the form of a high level program – a collection of high level actions connected by programming constructs.

Planning within the classical realm involves the search for a sequence of actions (a plan) that transitions the planner (an agent) from an initial state to a goal state (a state in which desired properties hold). In contrast, high level programs within the GOLOG formalism outline the actions an agent must perform to achieve these goals. These programs may be fully specified, or contain constructs involving choice. Given such a program, an agent must find an execution of it – a mapping between each high level action and choice to a concrete action. The result is a sequence of actions for the agent to perform. In Section 5.3, I describe the GOLOG language, the formation of programs, and their execution. In Sections 5.4 and 5.5, I describe two extensions of the GOLOG language – CONGOLOG [de Giacomo et al. (2000)] and INDIGOLOG [de Giacomo and Levesque (1999)]. CONGOLOG supports the concurrent execution of GOLOG programs, while INDIGOLOG provides support for sensing and online execution.

The remainder of this chapter is structured as follows. I begin with a description of the language of the situation calculus – the representation of environment properties as fluents, and its formal alphabet. I then consider the formation of basic action theories (Section 5.1.2), regression (Section 5.1.6), the STRIPS and ADL action description formalisms (Section 5.2), and the GOLOG language (Section 5.3). These concepts lay the foundation for the preference-guided GOLOG interpreters of Chapters 6 and 8.

5.1 The Situation Calculus

The situation calculus is a many-sorted second order language for reasoning about dynamically changing domains and environments, developed by McCarthy and Hayes (1969) and extended by Reiter (1991, 2001). I describe and employ the extended ver-

sion of Reiter (2001)[1]. In this version, the language of the situation calculus ($\mathcal{L}_{sitcalc}$) is composed of the following three sorts:

Actions	Functions denoting instantaneous events that may or may not change the state of an environment.
Situations	Terms that denote sequences of actions that have occurred since an initial situation S_0, where situations are built using the function: $$do : Action \times Situation \rightarrow Situation$$
Objects	Terms that represent domain objects.

Example 5.1.1 Consider the function $pickup(Dipsy)$. This function is an action in which $Dipsy$ the doll (an object) is picked up. When performed in situation S_0, this action yields the situation $s_1 = do(pickup(Dipsy), S_0)$. The situation that arises when another doll *Bert* is picked up in situation s_1 is $s_2 = do(pickup(Bert), do(pickup(Dipsy), S_0))$.

The properties of an environment are represented by fluents. These fluents come in two flavours – relational and functional. Each fluent is an n-ary predicate or function that takes as its final argument a situation (a history of actions) and returns (or evaluates to) a value. A relational fluent returns (or evaluates to) a true or false value. This value is the truth of the fluent after the actions in the situation have been performed. A functional fluent returns (or evaluates to) any type of value, as highlighted in Example 5.1.2.

Example 5.1.2 Consider the fluent $location(o, s)$ which takes two arguments – an object o and a situation s. This fluent returns the location of object o in situation s – a coordinate value (X, Y). This fluent is a functional fluent. Consider the fluent $Open(w, s)$ where w is a window and s is a situation. This fluent evaluates to a true or false value, and holds if and only if the window w is open in situation s. This fluent is relational.

Some properties of an environment are situation independent or rigid. For example, the relational predicate $Doll(d)$ is rigid and holds if and only if d is a doll. The rigid

[1]The convention used by Reiter (2001), and in this chapter, is that lower case roman letters denote variables, and upper case roman letters denote constants.

functional fluent $height(d)$ evaluates to the height of object d.

The language of the situation calculus ($\mathcal{L}_{sitcalc}$) is defined within the work of Pirri and Reiter (1999). It consists of the standard alphabet of logical symbols, connectives and quantifiers ($\wedge$, $\neg$, $\exists$, $\vee$, $\forall$, and $\rightarrow$) with their usual definitions. It also contains a countably infinite number of: variables of each sort; and n-ary predicate and function symbols denoting actions, constants, and relational and functional fluents. The reader is referred to the work of Pirri and Reiter (1999) for a definition of $\mathcal{L}_{sitcalc}$ in its entirety.

5.1.1 Foundational Axioms of the Situation Calculus

The nature of situations in the situation calculus is described by a series of foundational axioms [Levesque et al. (1998)]. These axioms express that situations are unique – no two situations are the same unless they denote the same action history. A second order induction axiom on situations states that any property belonging to the initial situation and to the successor of any situation which has that property, belongs to all situations. Finally, a situations as histories axiom expresses that no situation precedes the initial, and that new situations are created by performing actions in existing situations.

The reader is referred to the work of Levesque et al. (1998) for a formalisation of these axioms. It is not required to know their explicit form in the understanding of this thesis.

5.1.2 Basic Action Theories

The foundational axioms described in Section 5.1.1 form part of a basic action theory (BAT) which describes the dynamics of a domain. Definition 5.1.1 presents the collection of axioms that characterise a domain in the situation calculus.

Definition 5.1.1 A basic action theory $\mathcal{D}$ is the union of the following sets of axioms: the set of foundational situation calculus axioms ($\sum$); a set of action precondition axioms ($\mathcal{D}_{ap}$); a set of successor state axioms ($\mathcal{D}_{ss}$); a set of unique names axioms for actions ($\mathcal{D}_{una}$); and a set of axioms describing the values of fluents in the initial situation S_0 ($\mathcal{D}_{S_0}$).

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$$

In the following sections the axioms identified in Definition 5.1.1 are described. Action precondition axioms (Section 5.1.3) describe when actions can be performed in a situation, successor state axioms (Section 5.1.4) describe how the fluents in a domain change, and the initial situation axioms (Section 5.1.5) describe what holds in the initial situation. The set of unique name axioms for actions $\mathcal{D}_{una}$ expresses that no two distinct actions share the same name and that each action is referred to by only one name.

Many of these definitions refer to a concept of uniform formulae. The definition of Pirri and Reiter (1999) is replicated in two parts – the first describing the concept of uniform terms, and the second extending this definition to describe uniform formulae.

Definition 5.1.2 Let σ be a term of sort situation. The terms of $\mathcal{L}_{sitcalc}$ that are uniform in σ are the smallest set of terms such that:

1. All rigid (situation independent) terms are in the set;
2. σ is in the set;
3. All variables v not of sort situation are in the set;
4. All $g(t_1, \ldots, t_n)$ are in the set, where g is an n-ary fluent function symbol, and $t_1, \ldots, t_n$ are terms uniform in σ whose sorts are appropriate for g.

Definition 5.1.3 The set of formulae of $\mathcal{L}_{sitcalc}$ that are uniform in situation σ is the smallest set of formulae that contains:

1. $t_1 = t_2$ for each pair of terms t_1 and t_2 that are uniform in σ, and are terms of the same sort object or action;
2. $P(t_1, \ldots, t_n)$ for each n-ary fluent predicate symbol P of $\mathcal{L}_{sitcalc}$ where $t_1, \ldots, t_n$ are terms uniform in σ whose sorts are appropriate for P;
3. The formulae $\neg\phi$, $\phi \wedge \phi'$ and $\exists v.\phi$, where ϕ and ϕ' are formulae uniform in σ and v is a variable that is not of sort situation.

5.1.3 Action Precondition Axioms

Action precondition axioms define the conditions that must be satisfied in a situation before a particular action can be performed in it. The set $\mathcal{D}_{ap}$ of Definition 5.1.1 contains

one precondition axiom for each available action within the domain being modelled. The precondition axiom for an action $A(x_1, \ldots, x_n)$ takes the form:

$$Poss(A(x_1, \ldots, x_n), s) \equiv \Pi_A(x_1, \ldots, x_n, s) \tag{5.1}$$

where: $\Pi_A(x_1, \ldots, x_n, s)$ is uniform in the situation s and defines the conditions that must be satisfied in s for action A (a function) to be legally performable in s; and whose free variables (amongst $x_1, \ldots, x_n, s$) are implicitly universally quantified at outermost scope.

To demonstrate, I consider a domain in which there is a window that can be opened and closed. It can be opened in a situation only if it is closed in that situation. It can be closed in a situation only if it is open, as shown in Example 5.1.3.

Example 5.1.3 The precondition axiom for the action of closing a window w in the situation s, only possible if the window is open, is:

$$Poss(close(w), s) \equiv Open(w, s)$$

The formula $Open(w, s)$ is uniform in the situation s by Definition 5.1.3.

5.1.4 Successor State Axioms

Successor state axioms define, for each fluent in a domain, the ways in which its value changes through the performance of actions. These axioms take two forms for the two distinct types of fluent. A successor state axiom for a relational fluent takes the form:

$$F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s) \equiv \gamma^+(\vec{x}, a, s) \vee (F(\vec{x}, s) \wedge \neg\gamma^-(\vec{x}, a, s)) \tag{5.2}$$

Here, $F(\vec{x}, do(a, s))$ is a relational fluent, while $\gamma^+(\vec{x}, a, s)$ and $\gamma^-(\vec{x}, a, s)$ denote the conditions under which the fluent is made true, and false respectively, if action a is performed in situation s. $F(\vec{x}, s)$ denotes the truth of the fluent in the situation s. Each of $\gamma^+(\vec{x}, a, s)$, $\gamma^-(\vec{x}, a, s)$, and $F(\vec{x}, s)$ are uniform in the situation s by Definition 5.1.3.

Under the assumption that it is possible to perform the action a in the situation s, the successor state axiom for a relational fluent states that the fluent $F(\vec{x}, do(a, s))$ is true if

and only if the conditions expressed in $\gamma^+(\vec{x}, a, s)$ are satisfied, or the fluent is true in the situation s and the conditions in $\gamma^-(\vec{x}, a, s)$ do not arise to make it false.

Fluents of the functional type are now considered. A successor state axiom for a functional fluent takes the form:

$$\begin{aligned} G(\vec{x}, do(a,s)) = y &\equiv \Psi_G(\vec{x}, a, y, s) \\ &\equiv \delta_G(\vec{x}, y, a, s) \vee G(\vec{x}, s) = y \wedge \neg(\exists y').\delta_G(\vec{x}, y', a, s) \end{aligned} \tag{5.3}$$

Here, $G(\vec{x}, do(a,s))$ is the functional fluent in question, and y is its value. $G(\vec{x}, s)$ denotes the value of the fluent in situation s. The formula $\delta_G(\vec{x}, y, a, s)$, uniform in s by Definition 5.1.3, defines the conditions under which the fluent evaluates to y in the situation $do(a, s)$.

Under the assumption that it is possible to perform action a in situation s, the successor state axiom for a functional fluent $G(\vec{x}, do(a,s))$ states that it evaluates to y if and only if the conditions expressed in $\delta_G(\vec{x}, y, a, s)$ are satisfied, or the fluent evaluates to y in situation s and action a does not result in an alternative assignment ($\neg(\exists y').\delta_G(\vec{x}, y', a, s)$).

Example 5.1.4 Consider the relational fluent $Open(w, do(a,s))$ which holds if the window w is open in the situation $do(a, s)$. The successor state axiom for this fluent is:

$$Open(w, do(a,s)) \equiv a = open(w) \vee Open(w, s) \wedge \neg(a = close(w))$$

stating that the window w is open in situation $do(a, s)$ if and only if the action a opened it, or it was open prior to the action being performed and the action a did not close it.

Example 5.1.5 Consider the functional fluent $location(x, do(a,s))$ which evaluates to the location of object x in situation $do(a, s)$. The successor state axiom for this fluent is:

$$location(x, do(a,s)) = y \equiv a = move(x, y) \vee location(x, s) = y \wedge \neg\exists y'.(a = move(x, y'))$$

stating that object x is at location y in situation $do(a, s)$ if the action a moved x to y, or if x was at y in situation s and the action a did not move it away from y.

5.1.5 Initial Situation Axioms

Initial situation (or initial state) axioms are the axioms that describe what holds in the initial situation of a domain, and form the set $\mathcal{D}_{S_0}$ within a basic action theory (Definition

5.1.1). These axioms are uniform in S_0 by Definition 5.1.3.

Example 5.1.6 The axiom $location(X, S_0) = (0, 0)$ denotes that object X is at location $(0, 0)$ in the initial situation S_0. The axiom $\forall x.[parrot(x) \rightarrow bird(x)]$ expresses that all parrots are birds, while $Open(W, S_0)$ denotes that the window W is open in S_0.

5.1.6 Regression in the Situation Calculus

To determine if a property holds (or evaluates to a particular value) in a situation, situation calculus regression is employed. Regression, as we shall discover in Section 5.3, plays an important role in the search for an execution of a GOLOG program.

The idea behind regression is to take a regressable situation calculus formula, as defined in Definition 5.1.4, below, and express it in terms of no other situation but the initial. The transformed formula can then be proven with respect to the initial situation axioms $\mathcal{D}_{S_0}$ of a given basic action theory with a first order theorem prover. Definition 5.1.4 describes the properties that a formula must satisfy to be successfully regressed, and is replicated from the work of Pirri and Reiter (1999) and Reiter (2001).

Definition 5.1.4 A formula ϕ in $\mathcal{L}_{sitcalc}$ is regressable if and only if:

1. ϕ is first order;
2. Every term of sort situation mentioned in ϕ is rooted in the initial situation S_0[2];
3. In each atom of the form $Poss(a, s)$ mentioned in ϕ, action a is of the form $A(\vec{t})$ where A is an n-ary function symbol of $\mathcal{L}_{sitcalc}$;
4. ϕ does not quantify over situations;
5. ϕ does not involve ordering over[3], or equality between[4], situations.

Regression is defined in terms of a set of regression rules. Each rule describes how we can take a situation calculus formula ϕ, examine its structure, perform regression on its individual parts, and combine the results of those operations to determine if the formula ϕ is entailed by a basic action theory $\mathcal{D}$ (whose form is defined in Definition 5.1.1).

[2]The situation is of the form $do(a_n, do(a_{n-1}, \ldots do(a_1, S_0)))$ where each a_i is an action.

[3]$\sqsubset : situation \times situation$ denotes an ordering relation on situations where $s \sqsubset s'$ holds if and only if s is a proper subsequence of s'.

[4]The formula does not contain expressions of the form $s_1 = s_2$ where s_1 and s_2 are terms of sort situation.

Definition 5.1.5 Let ϕ_{S_0}, ϕ_1, and ϕ_2 denote regressable formulae, where ϕ_{S_0} is uniform in situation S_0. Let $F(\vec{x}, do(a,s)) \equiv \Phi(\vec{x}, a, s)$ denote a successor state axiom for a relational fluent $F(\vec{t}, do(\alpha,\sigma))$, $G(\vec{x}, do(a,s)) = y \equiv \Psi(\vec{x}, a, y, s)$ a successor state axiom for a functional fluent $G(\vec{t}, do(\alpha,\sigma))$, and $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$ a precondition axiom for action $A(\vec{t})$. Let ϕ_G denote an atomic regressable formula mentioning the functional fluent $G(\vec{t}, do(\alpha,\sigma))$, and $\phi|^{\psi}_{\psi'}$ the formula that results from replacing all instances of ψ in ϕ with ψ'.

The regression of a regressable formula ϕ given a basic action theory $\mathcal{D}$, denoted $R_{\mathcal{D}}[\phi]$, is defined by the following rules:

$$R_{\mathcal{D}}[\phi_{S_0}] \stackrel{def}{=} \phi_{S_0}$$

$$R_{\mathcal{D}}[Poss(A(\vec{t}),\sigma)] \stackrel{def}{=} R_{\mathcal{D}}[\Pi_A(\vec{t},\sigma)]$$

$$R_{\mathcal{D}}[F(\vec{t}, do(\alpha,\sigma))] \stackrel{def}{=} R_{\mathcal{D}}[\Phi(\vec{t},\alpha,\sigma)]$$

$$R_{\mathcal{D}}[\phi_G] \stackrel{def}{=} R_{\mathcal{D}}[\exists y.\Psi(\vec{t},\alpha,y,\sigma) \wedge \phi_G|^{G(\vec{t},do(\alpha,\sigma))}_{y}]$$

$$R_{\mathcal{D}}[\neg\phi_1] \stackrel{def}{=} \neg R_{\mathcal{D}}[\phi_1]$$

$$R_{\mathcal{D}}[\phi_1 \wedge \phi_2] \stackrel{def}{=} R_{\mathcal{D}}[\phi_1] \wedge R_{\mathcal{D}}[\phi_2]$$

$$R_{\mathcal{D}}[\exists v.\phi_1] \stackrel{def}{=} \exists v.\, R_{\mathcal{D}}[\phi_1]$$

I do not define regression rules for formulae involving the connectives $\vee$, $\forall$, and $\rightarrow$. These formulae can be expressed in terms of the primitives $\exists$, $\wedge$, and $\neg$. The regression of a formula ϕ produces an equivalent formula (relative to $\mathcal{D}$) that is uniform in the initial situation S_0, as shown in Definition 5.1.6 and proved by Pirri and Reiter (1999).

Definition 5.1.6 For a regressable formula ϕ and a basic action theory $\mathcal{D}$:

1. $R_{\mathcal{D}}[\phi]$ is uniform in the initial situation S_0; and,
2. $\mathcal{D} \models \phi$ if and only if $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models R_{\mathcal{D}}[\phi]$,

where $\mathcal{D}_{S_0}$, $\mathcal{D}_{una}$ are the set of initial state and unique names axioms (respectively) of $\mathcal{D}$.

The use of regression is demonstrated in Example 5.1.7. In this example, I show how regression is employed to determine if a window is open.

Example 5.1.7 Consider a domain which contains a window W and a box Box_0. In situation s, Box_0 has been picked up and window W has been opened. Let $a_1 = open(W)$, $a_2 = pickup(Box_0)$, and $s_1 = do(open(W), S_0)$. Then $s = do(a_2, s_1)$. Consider the relational fluent $Open(W, s)$, which holds if the window W is open in situation s. The basic action theory characterising this domain, denoted $\mathcal{D}_w$, contains the successor state axiom for the fluent $Open(W, s)$ shown in Example 5.1.4.

To determine if $Open(W, s)$ is entailed by $\mathcal{D}_w$, we can employ regression. The regression rule for a relational fluent, shown in Definition 5.1.5, is used.

$$
\begin{aligned}
\mathcal{D}_w \models\ & Open(W, s) \\
& \equiv R_{\mathcal{D}_w}[Open(W, do(a_2, s_1))] \\
& \equiv R_{\mathcal{D}_w}[a_2 = open(W) \vee Open(W, s_1) \wedge \neg(a_2 = close(W))] \\
& \equiv R_{\mathcal{D}_w}[a_2 = open(W)] \vee R_{\mathcal{D}_w}[Open(W, s_1) \wedge \neg(a_2 = close(W))] \\
& \equiv a_2 = open(W) \vee R_{\mathcal{D}_w}[Open(W, s_1)] \wedge \neg(a_2 = close(W)) \\
& \equiv false \vee R_{\mathcal{D}_w}[Open(W, s_1)] \wedge true \\
& \equiv R_{\mathcal{D}_w}[a_1 = open(W) \vee Open(W, S_0) \wedge \neg(a_1 = close(W))] \\
& \equiv a_1 = open(W) \vee R_{\mathcal{D}_w}[Open(W, S_0) \wedge \neg(a_1 = close(W))] \\
& \equiv true
\end{aligned}
$$

5.2 The Situation Calculus vs. STRIPS and ADL

Basic action theories (BATs) are more expressive than both the STRIPS [Fikes and Nilsson (1971)] and ADL [Pednault (1989)] action description formalisms. In this section I describe both action representations (Sections 5.2.1 and 5.2.2), and the restrictions that must be placed on a BAT to yield a theory no more expressive than ADL (Section 5.2.3).

These restrictions become relevant during my presentation of preference-guided GOLOG in Chapter 6. Preference-guided GOLOG interpreters have been developed that require the compilation of a program and a BAT into an ADL planning instance (Section 6.1.4). These ADL planning instances can then be solved, producing a most preferred execution of the original program, using efficient preference-based classical planners. I describe these compilation-based works in Chapter 6, and a range of preference-based

planners and search techniques in Chapter 8.

5.2.1 STRIPS

The STRIPS formalism[5] [Fikes and Nilsson (1971)] was developed to address typical problems faced by robotic problem solvers. STRIPS was introduced as a planning tool – its name, however, typically refers to the action description language it employs. Planning within the STRIPS formalism discovers a sequence of actions that transitions a planner from an initial to a goal state – a state in which a set of goal propositions hold. This thesis refers to the formalisation of Lifschitz (1986), presented in Definition 5.2.1 below.

Definition 5.2.1 A STRIPS planning problem is a triple $\mathcal{P} = \langle S, \mathcal{A}, \mathcal{G} \rangle$, where:

1. S is a set of propositional atoms denoting the facts that hold in the initial state of the planner. STRIPS operates under the closed world assumption – any fact that is not specified to hold in a state is assumed to be false in that state;
2. $\mathcal{A}$ is the set of actions (or operators) available to the planner, where each action $a \in \mathcal{A}$ has the form $a = \langle pre(a), del(a), add(a) \rangle$, and:
   (a) $pre(a)$ is a set of atoms denoting the preconditions of a – a set of facts that must hold in a state before the action a can be performed in it;
   (b) $del(a)$ is a set of delete-effects, where each $e_j \in del(a)$ is an atom that is made false in the next state by performing the action a;
   (c) $add(a)$ is a set of add-effects, where each $e_i \in add(a)$ is an atom that is made true in the next state by performing the action a.
3. $\mathcal{G}$ is the set of atoms to be achieved in the final (goal) state;
4. The state resulting from performing an action $a = \langle pre(a), del(a), add(a) \rangle$ in the state st is denoted $\phi(a, st)$, where $\phi(a, st) = (st \cup add(a)) \backslash del(a)$.

Example 5.2.1 Consider a state st in which the facts $st = \{lightOn, winOpen, doorClosed\}$ hold. In state st: the light is on, the window is open and the door is closed. The action of closing a window is expressed in the STRIPS formalism as follows:

[5] *ST*anford *R*esearch *I*nstitute *P*roblem *S*olver.

$$a_{close} = \langle winOpen, winOpen, winClosed \rangle$$

where its precondition requires the window to be open, and the *winClosed* atom is added in place of the deleted *winOpen*. The state that results after performing a_{close} in st is:

$$\phi(a_{close}, st) = \{lightOn, winClosed, doorClosed\}.$$

The STRIPS representation of actions, as originally defined by Fikes and Nilsson (1971), is more expressive than the formalism of Lifschitz (1986) in Definition 5.2.1. In its original form, world states, action preconditions, and effects were represented by sets of well-formed formulae in first order logic. These operator definitions were not sound, however. In its original incarnation, each formula that is provable in a state st but not provable in the state $\phi(a, st)$ must be mentioned in the delete list of action a. For arbitrary well-formed formulae this may only be possible if this delete list is infinite.

5.2.2 ADL – an Action Description Language

ADL was developed by Pednault (1989, 1994) in response to the limitations of STRIPS [Fikes and Nilsson (1971); Lifschitz (1986)] in representing many real world problems. In particular, STRIPS could not model the conditional effects of an action – additions or deletions of facts that are dependent on what holds when the action is performed.

In ADL, the set of possible states in a domain is described by the set of possible interpretations of a collection of n-ary relational predicates – the instantiations of their arguments for which they hold. In STRIPS, actions delete and add propositional atoms from (and to) a progressively maintained world state. In contrast, ADL state transitions involve the addition and deletion of relation interpretations (for example, the predicate $at(x)$ may have the interpretation $x = Library$ in one state while the interpretation $x = Shop$ in another). The process of planning with STRIPS and ADL is the same, however.

In the STRIPS representation [Fikes and Nilsson (1971); Lifschitz (1986)], world states, action preconditions, action effects, and goal states are represented by conjunctions of propositional atoms. In ADL, the formulae that represent preconditions, goals, and define which relation interpretations are added and removed from a world state, involve

disjunction, negation, and quantification. The reader is referred to the work of Pednault (1989, 1994) for a formal definition of the ADL action schema.

5.2.3 Restricted Basic Action Theories

Eyerich et al. (2006) describe a basic action theory fragment that is equivalent in expressiveness to ADL. The restricted basic action theories (RBAT) in this fragment can be compiled into ADL without an exponential blowup in the size of the theory and the plans induced from it. Röger and Nebel (2007) and Röger et al. (2008) extend this work, expanding the class of basic action theories (BAT) that can be considered expressively equivalent to ADL. In this section, I define the required constraints on a BAT (outlined by Röger and Nebel (2007) and Röger et al. (2008)) to achieve this equivalence.

Röger et al. (2008) define constraints on the initial situation axioms in a BAT to ensure its compilability into an ADL theory – these constraints are shown in Definition 5.2.2.

Definition 5.2.2 To ensure compilability of a BAT into an ADL theory, the initial situation axioms of the BAT must satisfy the following conditions [Röger et al. (2008)]:

1. For each relational fluent F there exists one of the following expressions:

$$\begin{aligned} &\neg F(x_1, \ldots, x_n, S_0) \\ &F(x_1, \ldots, x_n, S_0) \quad \equiv \quad (x_1 = D_{11} \wedge \ldots \wedge x_n = D_{1n}) \vee \\ &\qquad\qquad\qquad\qquad \ldots \vee (x_1 = D_{m1} \wedge \ldots \wedge x_n = D_{mn}) \end{aligned}$$

2. An analogous expression to those in Condition (1) must exist for all situation independent (rigid) and functional fluents;
3. For each distinct constant pair C_i and C_j there is a unique names axiom $C_i \neq C_j$.

Definition 5.2.2 requires the value of each relational and functional fluent in the initial situation to be solely characterised by the values of domain objects (denoted D_{ij} in Definition 5.2.2). The relative expressiveness of BATs and ADL has been considered in response to a growing body of work on the combination of situation calculus based action languages and efficient ADL-based planners [Claßen et al. (2007); Baier et al. (2007b)].

CONSTRUCT	DESCRIPTION
nil	An empty program.
α	Primitive action – the action α is performed.
$\phi?$	Test that condition ϕ holds in the current situation before proceeding.
$(\delta_1 : \delta_2)$	Sequence – perform program δ_1 and then perform program δ_2.
$(\delta_1 \mid \delta_2)$	Branch – choose to perform either δ_1 or δ_2.
$\pi v.\delta$	Choose a value for argument v to be used where v appears in the program δ.
δ^*	Iteration – perform the program δ zero, one, two, up to an infinite number of times.
if ϕ **then** δ_1 **else** δ_2	Conditional – if ϕ holds in the current situation then perform δ_1, otherwise perform δ_2.
while ϕ **do** δ **end**	Loop – while ϕ holds in the current situation, perform program δ.
proc $\beta(\vec{x})$ δ **end**	Procedure – a collection of constructs connected together forming a unit callable by name.

Table 5.2: The constructs of the GOLOG language [de Giacomo et al. (2000)].

5.3 GOLOG (alGOl in LOGic)

A GOLOG program is described using axioms of the situation calculus (Section 5.1), together with operators and constructs to encode procedures, actions, iteration, and nondeterministic choice. The constructs of the GOLOG programming language [de Giacomo et al. (2000); Levesque et al. (1997)] are presented in Table 5.2 alongside their meaning. The constructs in Table 5.2 are combined to produce programs for agent control.

Example 5.3.1 Consider the following GOLOG program instructing an agent to remove all blocks from a table [de Giacomo et al. (2000)]. This single-line program makes a call to the RemoveABlock procedure shown below.

RemoveABlock* : $(\neg\exists\ b.\ \text{OnTable}(b))?$

proc RemoveABlock

$\pi\ b.$ [OnTable(b)? : grab(b) : store(b)]

end

The RemoveABlock procedure shown in Example 5.3.1 instructs an agent to clear away a single block from the table. The agent must select a block ($\pi\ b$) that is on the table

(OnTable(b)?), pick it up (grab(b)) and put it away (store(b)). The single-line program tells the agent to continually perform the RemoveABlock procedure (RemoveABlock*) until no blocks remain on the table [($\neg\exists$ b. OnTable(b))?]. The relational fluent OnTable has its situation argument suppressed in the above program – when evaluated in a situation s during program interpretation, this situation argument will be restored (OnTable(b, s)).

5.3.1 A GOLOG Interpreter

A GOLOG interpreter is designed to find an execution of a GOLOG program given a basic action theory (BAT) (as defined in Definition 5.1.1). This interpreter is characterised using an evaluation [Levesque et al. (1997)] or a transition semantics [de Giacomo et al. (2000)]. A GOLOG interpreter designed using an evaluation semantics [Levesque et al. (1997)] consists of a series of axioms that characterise the complete execution of a program. A transition semantics characterises individual steps in the execution of a program. In this thesis, I consider and employ the transition semantics. The reader is referred to the work of Levesque et al. (1997) for an exploration of the evaluation semantics.

A transition semantics describes when an agent can transition from one situation s to another s' by performing one step in a program δ. Central to the semantics is the concept of a configuration – a program-situation pair representing a program to be executed in the given situation. For each of the GOLOG constructs presented in Table 5.2, a *Trans* and *Final* axiom is defined. The interpreter employs these axioms to determine which configurations (δ', s') can be transitioned to from an initial configuration (δ, s) by performing a single step in its program δ. A final axiom determines when a configuration is final. The program in a final configuration has no remaining steps to be executed.

Definitions 5.3.1 and 5.3.2 define the transition and final axioms associated with each of the GOLOG constructs of Table 5.2 (taken from de Giacomo et al. (2000)).

Definition 5.3.1 Each construct δ_c of Table 5.2 is associated with a transition axiom as follows, where: (δ', s') denotes the configuration that is transitioned to by performing one step of δ_c in a situation s; $\alpha[s]$ and $\phi[s]$ denote the result of restoring situation s as the final argument in the fluents within formula ϕ and action α; δ_1 and δ_2 denote programs; and δ_x^v denotes the program that results when all instances of v in δ are replaced with x.

$$Trans(nil, s, \delta', s') \equiv False$$

$$Trans(\alpha, s, \delta', s') \equiv Poss(\alpha[s], s) \wedge \delta' = nil \wedge s' = do(\alpha[s], s)$$

$$Trans(\phi?, s, \delta', s') \equiv \phi[s] \wedge \delta' = nil \wedge s' = s$$

$$Trans(\delta_1 : \delta_2, s, \delta', s') \equiv \exists\gamma.\delta' = (\gamma : \delta_2) \wedge Trans(\delta_1, s, \gamma, s') \vee Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta', s')$$

$$Trans(\delta_1|\delta_2, s, \delta', s') \equiv Trans(\delta_1, s, \delta', s') \vee Trans(\delta_2, s, \delta', s')$$

$$Trans(\pi v.\delta, s, \delta', s') \equiv \exists x.Trans(\delta_x^v, s, \delta', s')$$

$$Trans(\delta^*, s, \delta', s') \equiv \exists\gamma.(\delta' = \gamma : \delta^*) \wedge Trans(\delta, s, \gamma, s')$$

$$Trans(\textit{if } \phi \textit{ then } \delta_1 \textit{ else } \delta_2, s, \delta', s') \equiv \phi[s] \wedge Trans(\delta_1, s, \delta', s') \vee \neg\phi[s] \wedge Trans(\delta_2, s, \delta', s')$$

$$Trans(\textit{while } \phi \textit{ do } \delta \textit{ end}, s, \delta', s') \equiv \exists\gamma.(\delta' = \gamma : \textit{while } \phi \textit{ do } \delta \textit{ end}) \wedge \phi[s] \wedge Trans(\delta, s, \gamma, s')$$

A procedure call $\beta(\vec{x})$ of **proc** $\beta(\vec{x})$ δ **end** is transitioned by replacing it with δ and transitioning δ using the axioms above. I refer the reader to the work of de Giacomo et al. (2000) for a listing of the *Trans* axiom for procedure calls in second (and first) order logic.

Definition 5.3.2 Each construct δ_c of Table 5.2 is associated with a final axiom – describing whether (δ_c, s) is final for a given situation s – as follows, where: $\phi[s]$, δ_1, δ_2, and δ_x^v are defined as in Definition 5.3.1 and α denotes an action.

$$Final(nil, s) \equiv True$$

$$Final(\alpha, s) \equiv False$$

$$Final(\delta_1 : \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$

$$Final(\delta_1|\delta_2, s) \equiv Final(\delta_1, s) \vee Final(\delta_2, s)$$

$$Final(\pi v.\delta, s) \equiv \exists x.Final(\delta_x^v, s)$$

$$Final(\delta^*, s) \equiv True$$

$$Final(\textit{if } \phi \textit{ then } \delta_1 \textit{ else } \delta_2, s) \equiv \phi[s] \wedge Final(\delta_1, s) \vee \neg\phi[s] \wedge Final(\delta_2, s)$$

$$Final(\textit{while } \phi \textit{ do } \delta \textit{ end}, s) \equiv \neg\phi[s] \vee Final(\delta, s)$$

A procedure call $\beta(\vec{x})$ of **proc** $\beta(\vec{x})$ δ **end** is final if and only if δ instantiated with arguments $\vec{x}$ is final by the above axioms. I refer the reader to the work of de Giacomo et al.

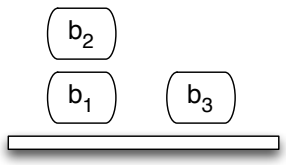


Figure 5.1: Table of blocks in Example 5.3.2.

(2000) for a listing of the *Final* axiom for procedures in second (and first) order logic.

A legal execution of a program δ in a situation s is found by repeatedly applying the transition relation *Trans* starting at (δ, s) until a final configuration (δ_f, s_f) is reached. The resulting situation s_f is an execution of δ. The set of all possible configurations (δ', s') that may be reached by successive application of *Trans* is denoted by $Trans^*(\delta, s, \delta', s')$.[6]

Definition 5.3.3 An execution s_f of a program δ in a situation s given a BAT $\mathcal{D}$ (Definition 5.1.1) is found as a by-product of proving the logical query in Equation 5.4.

$$\mathcal{D} \models \exists s_f . Do(\delta, s, s_f) \tag{5.4}$$

$$Do(\delta, s, s_f) \stackrel{def}{=} \exists \delta_f . Trans^*(\delta, s, \delta_f, s_f) \wedge Final(\delta_f, s_f) \tag{5.5}$$

In applying the transition axioms of Definition 5.3.1, a GOLOG interpreter will make choices (such as the choice of argument in the axiom for the construct $\pi v.\delta$ and the choice of branch in the axiom for the construct $\delta_1|\delta_2$). These choices may not lead to successful executions of a program. For example, the interpreter may arrive at a primitive action α to be performed in a situation s where $\neg Poss(\alpha[s], s)$. In these cases, the interpreter backtracks to the last point at which a choice was made and makes an alternative choice in its place. If all combinations of choices have been made and have led to failure, no legal execution of the program in the given situation and with the given BAT exists.

[6]For the interest of the reader, $Trans^*(\delta, s, \delta', s')$ is defined as in the work of de Giacomo et al. (2000) – by the second order formula: $Trans^*(\delta, s, \delta', s') \stackrel{def}{=} \forall T. [\ldots \supset T(\delta, s, \delta', s')]$ where $\ldots$ denotes the conjunction of the universal closure of the following two implications:

$$True \supset T(\delta, s, \delta, s)$$
$$Trans(\delta, s, \delta'', s'') \wedge T(\delta'', s'', \delta', s') \supset T(\delta, s, \delta', s')$$

Example 5.3.2 Consider the blocksworld program of Example 5.3.1 and the table of blocks shown in Figure 5.1. A GOLOG interpreter finds three executions of the program[7]:

1. $\text{grab}(b_2) \rightarrow \text{store}(b_2) \rightarrow \text{grab}(b_1) \rightarrow \text{store}(b_1) \rightarrow \text{grab}(b_3) \rightarrow \text{store}(b_3)$
2. $\text{grab}(b_2) \rightarrow \text{store}(b_2) \rightarrow \text{grab}(b_3) \rightarrow \text{store}(b_3) \rightarrow \text{grab}(b_1) \rightarrow \text{store}(b_1)$
3. $\text{grab}(b_3) \rightarrow \text{store}(b_3) \rightarrow \text{grab}(b_2) \rightarrow \text{store}(b_2) \rightarrow \text{grab}(b_1) \rightarrow \text{store}(b_1)$

5.4 CONGOLOG

CONGOLOG extends GOLOG by incorporating additional programming constructs to manage the concurrent execution of multiple programs [de Giacomo et al. (2000)]. Concurrency is achieved by interleaving the execution of steps in these programs. The constructs present in CONGOLOG, excluding those of GOLOG, are shown in Table 5.3.

CONSTRUCT	DESCRIPTION
$(\delta_1 \| \| \delta_2)$	Execute program δ_1 concurrently with δ_2.
$(\delta_1 >> \delta_2)$	Execute program δ_1 concurrently with δ_2 with priority given to δ_1. Transitions are only made in δ_2 if no transition is possible in δ_1 in the current situation.
$\delta^{\|\|}$	Iterate over the program δ with each iteration performed concurrently.
$\langle \phi \rightarrow \delta \rangle$	An interrupt – repeat the program δ until the condition ϕ no longer holds.

Table 5.3: The constructs of the CONGOLOG language [de Giacomo et al. (2000)].

A CONGOLOG interpreter is constructed in the same manner as the GOLOG interpreter of Section 5.3.1 – using a transition semantics. *Trans* and *Final* axioms are defined for the additional constructs in Table 5.3. In this thesis I consider and use only the GOLOG formalism of Section 5.3, and as such I do not describe these additional axioms. The reader is referred to the work of de Giacomo et al. (2000) for their characterisation.

5.5 INDIGOLOG

INDIGOLOG extends CONGOLOG with the capability of conducting a mixture of offline planning and online execution [de Giacomo and Levesque (1999)]. Within the GOLOG

[7] For ease of representation, each execution is presented as an ordered sequence of actions: $\alpha_1 \rightarrow \ldots \rightarrow \alpha_n$ denotes the execution $do(\alpha_n, \ldots do(\alpha_1, S_0))$ given an initial situation S_0.

and CONGOLOG formalisms, an entire legal execution of a program must be found before any part of it is executed. This approach is an example of offline planning. INDIGOLOG introduces a search operator ($\sum$) to indicate which portions of a program are to be planned offline and then executed. Outside of this search operator, each primitive action encountered is performed (if legal to do so) prior to processing the remainder of the program. Online execution avoids the delays induced while waiting for an execution to be found before taking action, and supports the integration of sensing during program interpretation. In Chapter 6, I discuss offline versus online interpretation within the context of the heuristic, preference-guided GOLOG interpreter I present.

In this thesis I consider and use only the GOLOG formalism of Section 5.3, and as such I do not describe the axioms defining an INDIGOLOG interpreter. The reader is referred to the work of de Giacomo and Levesque (1999) for their characterisation.

5.6 Concluding Remarks

In this chapter, I have explored the construction of situation calculus basic action theories (BATs) – from the use of successor state axioms (Section 5.1.4) in characterising fluent change, to the description of action preconditions (Section 5.1.3). I have shown how the process of regression (Section 5.1.6) is used to determine whether queries and formulae in the situation calculus are entailed by a BAT. I have demonstrated the rules of regression in a simple domain. The use of the situation calculus in planning has been discussed in the context of the GOLOG family of agent programming languages (Section 5.3).

Classical planning tools typically employ the STRIPS (Section 5.2.1) or ADL (Section 5.2.2) action representations – less expressive than a BAT in the situation calculus (Section 5.2.3). Within the GOLOG family of programming languages (Section 5.3), a domain is characterised by a BAT and planning proceeds through the interpretation of a GOLOG program. This program describes a high level plan – interpreted using a transition semantics describing single steps of program execution. Extensions of the basic GOLOG formalism increase its power in the modelling of concurrent processes (CONGOLOG, see Section 5.4) and support for online execution (INDIGOLOG, see Section 5.5).

In the next chapter, I present an alternative GOLOG interpreter to that described

in the preceding sections. This alternative aims to find a most preferred execution of a GOLOG program – in place of one in which executions are constructed from random choice selections. It shall be discovered in the following chapter that the consideration of preferences in planning by GOLOG is useful in a range of its applications.

Chapter 6
Heuristic Search for GOLOG

THE aim of a GOLOG interpreter [Levesque et al. (1997)], as described in Section 5.3, is to find an execution of a GOLOG program – a sequence of actions for an agent to perform. In this chapter I explore the importance of incorporating a notion of preference into the interpretation of a GOLOG program. I consider existing approaches that aim to tackle this task, their limitations, and how these problems can be overcome to develop a GOLOG interpreter that discovers the most desirable executions of a program.

As I have described in the introduction to Part II of this thesis, the GOLOG language has garnered a wide range of applications. These include the development of a robot for mail delivery [Lespérance et al. (1999)], a robotic museum tour guide [Burgard et al. (1998)], and the composition of web-services within a travel planning domain [Sohrabi et al. (2006)]. The ability to find a most preferred execution of a controlling GOLOG program within these domains is important. A travel planner that respects a traveller's preferences over airlines, car models, and hotel chains is more useful than one that does not. A museum tour guide that customises its route on the basis of group interest will provide a more satisfying visit than one that does not. To increase the applicability of GOLOG as a tool for programming agents, we do not want an interpreter to discover just any execution of a program, but one that satisfies desirable properties.

A GOLOG interpreter, as described in Section 5.3.1, is faced with a choice each time it comes across a non-deterministic construct (such as a branch). A typical implementation of a standard interpreter selects the first available choice (the first program in a branch,

for example). The axioms characterising the semantics of this traditional interpreter select choices at random. These standard interpreters do not consider the impact their choices have on the quality of the executions they find. Several existing techniques consider the combination of preferences and GOLOG to mitigate this issue. These approaches use preferences, expressed in a particular format, to evaluate these available choices – selecting those that are likely to lead to the most desirable or preferred executions.

Members of this existing family of tools include: DTGOLOG [Boutilier et al. (2000)], a GOLOG interpreter designed to discover an optimal policy for program execution within stochastic domains; qualitative DTGOLOG [Fritz and McIlraith (2005)], designed to search for the executions of a program that maximise a quantitative measure amongst those that satisfy a range of qualitative preferences; GOLOGPREF [Sohrabi et al. (2006)], a best-first search interpreter with an optimistic heuristic; and a range of reformulation-based approaches [Baier et al. (2007b, 2008); Fritz et al. (2008)], which compile a GOLOG program into an ADL planning instance [Pednault (1989)] to find a most preferred execution of the program with an appropriate ADL compliant planning tool[1].

This existing work, discussed further in Section 6.1, has its limitations. DTGOLOG [Boutilier et al. (2000)] and its descendants search the entire execution space of a program to discover an optimal policy for its interpretation – a mapping between each reachable situation and the best action that can be performed in that situation. The heuristic evaluation strategies employed in the work of Sohrabi et al. (2006) do not consider any form of lookahead (consideration of the transitions that could be made beyond available choices) when deciding upon paths to pursue in the search for executions. When considered as an alternative means of preference-based GOLOG interpretation, the work of Baier et al. (2007b, 2008) and Fritz et al. (2008) requires the basic action theory (Section 5.1.2) associated with a program to be formulated in ADL (limiting its expressiveness).

I develop in this chapter a heuristic interpreter, designed to address the limitations of this existing work. This interpreter draws inspiration from the use of relaxation-based heuristics – heuristics that have been used in classical planning to find optimal or near-

[1]The work of Baier et al. (2007b, 2008) and Fritz et al. (2008) is primarily designed to provide procedural control knowledge, in the form of a GOLOG or CONGOLOG program, to ADL compliant planners (a range of which are discussed in Chapter 8). As described in Section 6.1, this work can be viewed as an alternative to traditional interpretation when it is possible to express the relevant basic action theory in ADL.

optimal plans (with respect to plan length) with great success [Bonet and Geffner (2001); Hoffman and Nebel (2001)]. The available choices in the search for such a plan are evaluated by solving a relaxed version of the planning problem. This relaxed problem typically ignores the negative effects of actions (such as the depletion of a resource). For each choice, a relaxed plan is formed representing what an actual (legal) plan might look like if that choice is made. The properties of this relaxed plan (such as length) are used to select the most promising choices for further exploration. In Section 6.2, I explore and explain the use of relaxation-based heuristics within the classical planning domain.

I consider the relaxation-based heuristics of Bonet and Geffner (2001) and Hoffman and Nebel (2001) and adapt them for use in the interpretation of a GOLOG program. The resulting heuristic interpreter evaluates each choice it is faced with – a program-situation pair – by finding a relaxed execution of its associated program. I develop a relaxed GOLOG interpreter for this purpose, relying on a relaxed form of (situation calculus) regression to optimistically determine whether an action is possible in any given situation. This relaxed execution is designed to represent a desirable future that may exist if this particular choice in the interpretation of the program is made. The choice that is associated with the most desirable relaxed execution is selected for exploration.

This heuristic interpreter must have recourse to a definition of 'most preferred' when evaluating the desirability of these relaxed executions. A goal of this chapter is to explore the benefit of a relaxation-based lookahead in the interpretation of a GOLOG program – not the capacity for or benefit of supporting general preference expression[2]. Hence, I select a preference format that is employed in a majority of existing preference-based GOLOG interpreters (and preference-based planners in general). I assume that an agent using this interpreter has a numeric evaluation function f_e to assess the quality of an execution, and that the aim of the interpreter is to find an execution that minimises it.

This numeric function need not represent a utility function (as in the DTGOLOG [Boutilier et al. (2000)] range of approaches), but can represent the degree to which an execution satisfies a range of basic desire formulae (as in the work of Sohrabi et al. (2006) and Bienvenu et al. (2006)). This evaluation function f_e when applied to each relaxed

[2]The development of a GOLOG interpreter capable of discovering a most preferred execution on the basis of such general preferences is considered in Chapter 8.

execution (generated from the set of available choices) assigns a heuristic value to its associated choice. The presented heuristic interpreter explores the choice with the best (smallest) evaluation, and continues to do so until an execution is found.

The remainder of this chapter is structured as follows. I begin, in Section 6.1, with an analysis of existing work. In Section 6.2, I describe the application of relaxation-based heuristics in classical planning, and consequently present my theory of relaxed regression in Section 6.3. This theory is applied in the development of a relaxed GOLOG interpreter in Section 6.5. This relaxed interpreter then plays a vital role in the heuristic GOLOG interpreter presented in Section 6.6. I present these interpreters within the context of offline planning. I conclude by considering the use of this interpreter in an online setting. The performance of the heuristic interpreter I present in this chapter, relative to a range of alternative approaches, is assessed in Chapter 7 (on three different domains).

6.1 The Beginning of the Road

A number of existing works consider the search for a most preferred execution of a GOLOG program. In Section 6.1.1, I discuss DTGOLOG [Boutilier et al. (2000)] and its extensions [Finzi and Lukasiewicz (2004); Fritz and McIlraith (2005)]. I consider the integration of a rational search operator within the INDIGOLOG (Section 5.5) language in Section 6.1.2. Section 6.1.3 describes the application of GOLOG within the domain of web services, describing two techniques for the integration of preference. To conclude, I consider a range of works designed to fuse GOLOG with classical planning techniques.

6.1.1 DTGOLOG

DTGOLOG [Boutilier et al. (2000)] combines the theory of Markov Decision Processes (MDPs) [Howard (1960)] with the GOLOG language. A Markov Decision Process (MDP) is a structure composed of: a set of states; a set of actions that cause state transitions with a given probability; and a numeric reward assigned to each transition. The solution of an MDP is a policy that maps each of these states to the action that should be performed in that state. A policy is optimal if, over a horizon of interest – an upper bound on the

number of actions performed – the expected reward accumulated by the succession of state transitions is maximised. DTGOLOG represents an MDP with a situation calculus BAT (Definition 5.1.1 in Section 5.1.2) augmented with stochastic actions and a numeric reward function. The outcome of a stochastic action is determined by nature, where each outcome occurs with a given probability. The optimal policy found by a DTGOLOG interpreter informs an agent of the best next step to take in any situation that can arise during its execution (to ensure that its expected accumulated reward is maximised).

DTGOLOG has garnered a number of extensions and generalisations since its conception. Finzi and Lukasiewicz (2004) describe a game-theoretic (multi-agent) extension (denoted GTGOLOG), generalised by Finzi and Lukasiewicz (2005) to operate in a partially observable setting (POGTGOLOG). GTGOLOG views the computation of an optimal policy for a multi-agent program (of joint actions) as the solution of a Markov game [van der Wal (1981)] – a Nash equilibrium [Nash (1950)] – a policy assigned to each agent such that no agent stands to increase their reward by deviating from it.

Fritz and McIlraith (2005) consider the integration of qualitative and quantitative preferences within DTGOLOG. Qualitative preferences are expressed in terms of basic desire formulae (BDFs) and temporal logic. These preferences are compiled into a GOLOG program that is interpreted synchronously with the focus agent program, pruning from search any configuration that does not satisfy these qualitative preferences. The practices of DTGOLOG are applied among those that do. In the event that all reachable configurations are pruned, DTGOLOG is employed to find an optimal policy for the program in question. This interpreter discovers a policy for program execution that maximises expected reward while ensuring that qualitative preferences are respected.

The optimal interpreters of Boutilier et al. (2000), Finzi and Lukasiewicz (2004, 2005), and Fritz and McIlraith (2005), are required to search the entire execution (or configuration) space of a program before selecting an optimal execution or policy. This is the price of optimality, particularly within the stochastic environments they consider. This price, however, is too expensive as the complexity of the domain, program, and length of the action horizon employed, increases (as shall be discovered in Chapter 7).

In a non-stochastic domain, such extensive exploration can be avoided. The inter-

preter I present in this chapter employs heuristic hill-climbing search to discover a most preferred program execution without traversing the entire execution space.

6.1.2 Rational Search and GOLOG

Sardiña and Shapiro (2003) introduce a rational search operator (Δ) to the INDIGOLOG language (Section 5.5). This operator finds a rational execution of a program – an execution that is at least as good as all others with respect to a set of prioritised goals (x_G). The highest priority goal not satisfied by this execution is equal or lower in priority to that of its alternatives. $\Delta(\delta : x_G)$ transitions the program δ to a configuration (δ', s') in situation s if and only if it is the result of performing a single step in a rational execution of δ.

The formulation of $\Delta(\delta : x_G)$ involves the comparison of induced strategies (action selection functions). When given a situation s, an action selection function a_s defines an infinite sequence of situations – each situation the result of applying a_s to the situation before it in the sequence starting at s. Such a function is induced by a program in a situation s if that function is able to generate a sequence of actions that is equivalent to an execution of the program. A strategy σ_1 is at least as good as another σ_2 in a situation s if the highest priority goal not achieved in an action sequence defined by σ_1 in s is equal or lower in priority to that of σ_2. A configuration $(\Delta(\delta : x_G), s)$ is transitioned to (δ', s') if there exists a strategy that is: induced by δ in s; is consistent with this transition; and is at least as good as all alternative strategies induced by δ with respect to x_G.

A limitation of rational search, as it is presented by Sardiña and Shapiro (2003), is that the determination of which strategies are induced by a program will inevitably require an exploration of the entire execution space.

6.1.3 GOLOG and Web Service Composition

McIlraith and Son (2002) employ GOLOG in the control of web-service composition for human users. A composition of web services designed to complete a task is created by finding an execution of a GOLOG program. The preferences of a user over the properties of these compositions is incorporated through the use of a desirability predicate. McIlraith and Son (2002) present a GOLOG interpreter (employing a transition semantics,

as described in Section 5.3.1) whose transition axiom for actions allows an action a to be performed in a situation s only if it is both possible, and desirable ($desirable(a, s)$).

Example 6.1.1 McIlraith and Son (2002) consider web-service composition in a travel domain. The GOLOG procedure below calls on a number of service providers, where: o, d, d_1, and d_2 denote the origin, destination, departure date, and return date of the traveller.

proc Travel(o, d, d_1, d_2)
[(bookFlight(o, d, d_1, d_2) : bookCar(d, d_1, d_2)) | bookCar(o, d_1, d_2)] :
bookHotel(d, d_1, d_2) : sendEmail : updateExpenseClaim
end

Within the above program: bookFlight(o, d, d_1, d_2) books a flight for the traveller from its origin (o) to destination (d) with a departure and return date of d_1 and d_2; bookCar(d, d_1, d_2) arranges for a rental car, obtained at location d to be used between d_1 and d_2; bookHotel(d, d_1, d_2) reserves a hotel room in location d between d_1 and d_2; while sendEmail and updateExpenseClaim notify the traveller of their itinerary and expenses.

In Example 6.1.1, the action of booking a flight may be desirable if the driving distance between the traveller's origin and destination is significant, while arranging car travel between these two locations might be desirable when this is not the case.

Another approach designed to construct web-service compositions, given the preferences of a human user, is the work of Sohrabi et al. (2006). Sohrabi et al. (2006) develop GOLOGPREF, a GOLOG interpreter designed to discover a most preferred web service composition using the best-first search strategy of PPLAN [Bienvenu et al. (2006)].

In the PPLAN system [Bienvenu et al. (2006)] preferences are expressed in a language that extends $\mathcal{PP}$ [Son and Pontelli (2004)], described in Section 2.2.3 of Chapter 2. PPLAN introduces additional mechanisms of preference combination to those defined within $\mathcal{PP}$ and assigns weights (denoting degrees of importance) to basic desires in atomic preference formulae. These weights are used to compute a value for each partial plan on a search frontier, where preferences not already violated are assumed to be satisfied at a future point on the trajectory of a partial plan (an optimistic heuristic). PPLAN selects the most promising partial plan for exploration at each stage in search.

The work I have described in this section differs from the interpreter I present in a number of ways. The heuristics employed in the works of Sohrabi et al. (2006) and Bienvenu et al. (2006) make assumptions about what can happen in the future at a given point in the interpretation of a program, without conducting any further transitions to make a more informed gauge. As shall be discovered in Chapter 7, in some domains an agent must sometimes make choices that do not at first appear to be desirable, but which lead to the most desirable executions. In the work of McIlraith and Son (2002), it is only the desirability of individual actions that are considered, and not the desirability of an execution as a whole. The interpreter I present in this chapter employs a lookahead heuristic to guide search toward a (near) optimal execution of a GOLOG program.

6.1.4 The Fusion of GOLOG and Classical Planning

McIlraith and Fadel (2002) consider the compilation of complex actions expressed as GOLOG programs into primitive actions with preconditions and effects within ADL[3] [Pednault (1989)]. ADL compliant planners (a range of which are described in Chapter 8) can consequently be exploited to plan with complex actions (in place of low-level primitive activities) – a requirement in the automated composition of web-services.

In a similar vein, Baier et al. (2007b) and Fritz et al. (2008) integrate procedural control knowledge in the form of a GOLOG (or CONGOLOG) program with an existing ADL planning instance. The preconditions and effects of this planning instance are transformed such that any solution of the instance, discovered by an ADL compliant planner, represents an execution of the GOLOG program. Baier et al. (2008) extend this work to consider not only the compilation of a GOLOG program into ADL, but the transformation of temporal logic preferences (expressed in $\mathcal{LLP}$ [Bienvenu et al. (2006)], an extension of the linear temporal logic described within Section 2.2.3) into non-temporal soft goals (preferences over the final state of a plan). These preferences are compatible with, for example, PDDL3 [Gerevini and Long (2005)] compliant planners. A PDDL3 compliant classical planner can be employed to discover a most preferred execution of a GOLOG program given a problem description (action theory) formulated in ADL.

[3] Refer to Section 5.2.2 for a description of the ADL formalism.

In contrast to the transformation-based approaches of Baier et al. (2007b, 2008) and Fritz et al. (2008), the work of Claßen et al. (2007) embeds classical planning within a GOLOG program. This work looks at problems that in part are suited for representation in GOLOG, but have stages in which general planning is required (such as the navigation of an entity across a grid). Such planning problems are not suited for representation in GOLOG as a GOLOG interpreter does not cope well with loops that continually select any action to perform (at random) until a goal is satisfied. A GOLOG interpreter (typically) makes decisions without guidance, and in these circumstances behaves (performance-wise) like a classical planner that makes random action selections.

A GOLOG interpreter within the approach of Claßen et al. (2007) operates given a basic action theory (BAT, Section 5.1.2), translated from an ADL problem description. In this work, a GOLOG program contains calls to the Fast Forward [FF, Hoffman and Nebel (2001)] classical planner, allowing an interpreter to transition to a situation in which a given collection of formulae hold (without describing how to arrive at it). Upon reaching these calls, the ADL theory that formed this BAT is progressed in accordance with the actions present in the current situation. This progressed theory represents what holds in the current situation, and is provided as input to the FF planner. FF discovers a sequence of actions that when applied to the current situation, achieve the desired goals. The current situation is updated with this sequence, and interpretation continues.

The approach of Claßen et al. (2007) and my heuristic interpreter both combine GOLOG and classical planning. My interpreter, however, applies classical planning ideas to the entire interpretation process (rather than individual steps) to find an optimal or near optimal program execution (while the other does not). My interpreter is designed to work with problems that are suited for representation in GOLOG, while still involving combinatorial optimisation, and not general planning. In each of the approaches I have discussed in this section, their use as a GOLOG interpreter requires a BAT that is expressible in ADL. The interpreter I present in this chapter allows a domain to be characterised by a BAT[4] that is not restricted in this manner. Section 5.2.3 describes the restrictions that are placed on a BAT to ensure equivalent expressiveness with an ADL theory.

[4]The relative expressiveness of BATs and ADL is analysed by Eyerich et al. (2006) and Röger et al. (2008).

In the next section, I describe the inspiration behind the development of the heuristic interpreter I present in this thesis – the use of relaxation in classical planning.

6.2 Relaxation in Classical Planning

Relaxation-based heuristics have been used in a number of classical planners to efficiently discover solutions to STRIPS (Section 5.2.1) planning instances [Bonet and Geffner (2001); Hoffman and Nebel (2001)]. Their goal is to find solution plans while minimising their length and the time taken to find them. Bonet and Geffner (2001) present a heuristic search planner (HSP) in which choices available during the search for a plan are evaluated by solving a relaxed STRIPS planning instance. This planner is described in Section 6.2.1. A similar tool, the FF (Fast Forward) planner is discussed in Section 6.2.2.

Recall from Definition 5.2.1 in Section 5.2.1 the details of the STRIPS representation. In the STRIPS formalism, a solution to a planning problem is a sequence of actions that transition the planner from an initial state S to a goal state – a state in which a set of goal propositions $\mathcal{G}$ hold. How the planner transitions from one state to another, by performing an action, is outlined in Definition 5.2.1. This kind of state-space search, without the use of heuristics to guide search toward a goal state, is inefficient.

Heuristics provide mechanisms for estimating the distance (or cost of travel) between a state and a goal state. A solution to a planning problem is found by transitioning to states that, according to these estimations, are closest to the goal. Employing cost as a heuristic measure enables the search for low-cost (rather than short-length) solutions. A^* is a classic example of heuristic search designed to discover low-cost paths through a graph [Hart et al. (1968)]. Dechter and Pearl (1985) describe a generalised A^* (generalised best-first search) in which each node is assigned a numeric value denoting its merit. Merit is not restricted to be informed only by cost, but by any numeric quality measure.

6.2.1 HSP – A Heuristic Search Planner

In the HSP planner of Bonet and Geffner (2001), these distance estimates are computed by using the concept of relaxation. Relaxation is employed to transform a problem whose

Listing 6.2.1 The heuristic estimation algorithm used by HSP. This algorithm computes a heuristic value for a state S ($h(S)$) given an available action set $\mathcal{A}$, and a goal set $\mathcal{G}$.

$\forall p \in S, v(p) \leftarrow 0$
$Curr \leftarrow S$
while $\mathcal{G} \not\subseteq Curr$ **do**
  $OP \leftarrow \{a \mid a \in \mathcal{A}, pre(a) \subseteq Curr\}$
  $ADD \leftarrow \{p' \mid a \in OP, p' \in add(a), p' \notin Curr\}$
  $Curr \leftarrow Curr \cup ADD$
  $\forall p' \in ADD, v(p') \leftarrow 1 + min_{a \in OP, p' \in add(a)}[\sum_{r \in pre(a)} v(r)]$
end while
$h(S) \leftarrow \sum_{g \in \mathcal{G}} v(g)$

solutions are difficult to find into one whose solutions are relatively easy to discover – achieved by removing constraints or conditions that must be adhered to in the search for a problem solution. In a STRIPS planning problem, relaxation removes the delete lists of actions ($del(a)$ for an action a, as per Definition 5.2.1) from the problem representation.

HSP employs a hill-climbing algorithm in the search for a solution to a planning problem. Starting at an initial state, search proceeds by iteratively transitioning to states until a goal state is discovered. HSP selects states to visit in the state-space of a problem as follows. Each state S that the planner can transition to at a given point in search (the successors of its current state) is assigned a heuristic value. This value denotes how many more actions or operator applications it expects will be needed to achieve its goals.

The heuristic value assigned to a state S, $h(S)$, is computed as shown in Listing 6.2.1. Let: $v(p)$ denote the expected number of actions required, from state S, to achieve the proposition p; $\mathcal{G}$ the set of goals to be achieved by the planner; and $\mathcal{A}$ its set of available actions. Listing 6.2.1 expands the state S, adding to it all propositional add-effects of each action $a \in \mathcal{A}$ that can be performed within it ($pre(a) \subseteq S$). This process of expansion is repeated until a state S' results that contains each required goal ($\mathcal{G} \subseteq S'$).

Each proposition p that is added to this expanding state is assigned a value. This value is computed by searching for the action $a \in \mathcal{A}$ that adds it whose preconditions are the easiest (require the least number of actions) to achieve. The value assigned to p is the sum of the values of the preconditions of a ($\sum_{r \in pre(a)} v(r)$). The heuristic value assigned to state S is the sum of the values assigned to each goal within $\mathcal{G}$ ($\sum_{g \in \mathcal{G}} v(g)$). This heuristic

is inadmissible, and assumes independent achievement of each goal in $\mathcal{G}$.

HSP selects a state to transition to that minimises this heuristic. This process is repeated on the children of this state (and their children) until a goal state is reached[5].

6.2.2 The Fast Forward Planner

A planner similar in design to HSP [Bonet and Geffner (2001)] is FF (Fast Forward), developed by Hoffman and Nebel (2001). Both are based on the STRIPS representation and employ relaxation by delete list removal. As described in Section 6.2.1, HSP heuristic estimates for a state S are obtained by assigning values to propositions as they are added to an expanding state (initially containing the propositions of S). FF, in contrast, employs a plan graph structure [Blum and Furst (1997)] to find an actual solution (a plan) of a relaxed STRIPS instance in a given state. The length of this plan is used to heuristically evaluate the state in question, allowing the planner to avoid the assumption of independence amongst goal propositions. I refer the reader to the work of Blum and Furst (1997) and Hoffman and Nebel (2001) for a definition and exploration of planning graphs.

The work of Bonet and Geffner (2001) and Hoffman and Nebel (2001) has been built upon and employed in a number of works [Baier et al. (2007a); Yoon et al. (2006); Vidal (2004); Baier et al. (2009)]. Baier et al. (2007a, 2009) describe a planner designed to find solutions to an ADL planning instance that satisfy a series of temporally extended preferences – desirable properties that hold over a period of time (for example, *eventually* I will travel to Mexico and *sometime* after travelling to Mexico I will travel to Spain). Baier et al. (2007a) devise heuristics that use the relaxation techniques of Hoffman and Nebel (2001) to estimate the difficulty of achieving these preferences by transitioning to one state over others. This is achieved by transforming these formulae into final state preferences – goals that hold in the final state. The estimated 'distance' of these goals, found using these relaxation-based techniques, represents how difficult they are to achieve.

I use this concept of relaxation, as employed by Bonet and Geffner (2001) and Hoffman and Nebel (2001), to develop a relaxed GOLOG interpreter. This interpreter finds

[5]Note that HSP is not complete, as it does not backtrack to change past choices in the event that a dead-end state (a state in which no action is possible) is reached.

an execution of a GOLOG program while relaxing tests for the truth of action preconditions. To test whether an action precondition holds in a given situation, regression is employed. I have described the use of regression for the evaluation of situation properties in Section 5.1.6. I develop a relaxed form of regression – called relaxed regression – in Section 6.3. Relaxed regression is used to relax action precondition tests during the relaxed interpretation of a GOLOG program. I describe this relaxed interpreter in Section 6.5, and its use in the development of a heuristic GOLOG interpreter in Section 6.6.

6.3 Regression and Relaxed Regression

I have described in Section 5.1.6 of Chapter 5 the process of regression and its use in the querying of properties of situations in the situation calculus. Regression transforms these queries into formulae that refer only to an initial situation. These transformed queries can then be answered with respect to a set of initial situation axioms using a first order theorem prover. In the search for an execution of a GOLOG program (Section 5.3), regression determines: if an action a is possible in a situation s ($Poss(a[s], s)$); and whether a condition within a test, while, or if ... else construct holds in a given situation[6].

An operator that conducts regression in a relaxed manner can be employed to transform the problem of finding an execution of a GOLOG program into a relaxed variant. Such transformations will permit the use of relaxation-based search techniques within the GOLOG domain. This operator, developed in this thesis, is described in Section 6.3.1.

6.3.1 RR – A Relaxed Regression Operator

In this section I define a relaxed regression operator RR. To do so I combine the concept of relaxation (as used in a number of classical planners [Bonet and Geffner (2001); Hoffman and Nebel (2001)]) with the regression rules of standard regression[7].

In classical planning relaxation is typically conducted by ignoring the negative effects of actions (the effects that cause certain propositions – aspects within the environment of the planner – to be false). Propositions that are true in the initial state, or that become true

[6] I refer the reader to Table 5.2 in Section 5.3 for a description of the constructs of the GOLOG language.
[7] The rules of standard regression are shown in Definition 5.1.5.

by performing an action, persist – they remain true irrespective of any action performed that may make them false. I have used this idea in my formulation of relaxed regression. Consider the regression rule for a relational fluent in Definition 5.1.5. Under relaxed regression, a relational fluent $F(\vec{t}, do(\alpha, s))$ is said to hold in a situation $do(\alpha, s)$ if it holds in situation s or the action α has made it hold. I do not consider whether action α causes the fluent to be false. I describe the rules of relaxed regression in Definition 6.3.2. First, the required properties of the RR operator are outlined.

Definition 6.3.1 The relaxed regression of a regressable formula ψ given a basic action theory $\mathcal{D}$[8], denoted $RR_{\mathcal{D}}[\psi]$, is defined such that:

1. $RR_{\mathcal{D}}[\psi]$ is uniform in the initial situation S_0 (Definition 5.1.3 in Section 5.1.2); and,
2. If $\mathcal{D} \models \psi$ then $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models RR_{\mathcal{D}}[\psi]$,

where $\mathcal{D}_{S_0}$, $\mathcal{D}_{una}$ are the set of initial state and unique names axioms (respectively) of $\mathcal{D}$.

The second property states that if a regressable formula ψ holds given $\mathcal{D}$, then the transformation of ψ under relaxed regression holds given $\mathcal{D}_{S_0} \cup \mathcal{D}_{una}$. The converse will not necessarily be the case. If $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models RR_{\mathcal{D}}[\psi]$ it is inferred that ψ might hold, not that it does hold. However, if $RR_{\mathcal{D}}[\psi]$ does not hold, it is inferred that ψ does not hold (relative to $\mathcal{D}$). Contrast these properties with those of standard regression R [Pirri and Reiter (1999)] – R is sound and complete as the property $\mathcal{D} \models \psi$ if and only if $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models R_{\mathcal{D}}[\psi]$ holds (see Definition 5.1.6). RR is designed to be complete but not sound.

Definition 6.3.2 defines the rules of RR. These rules can be contrasted with their R counterparts in Definition 5.1.5. Before presenting these rules, I must explain an important observation. Negated and non-negated formulae must be regressed with different perspectives under relaxed regression. $RR_{\mathcal{D}}[\neg\psi]$ is an optimistic determination of whether ψ does not hold. It is not the case that $RR_{\mathcal{D}}[\neg\psi] \stackrel{def}{=} \neg RR_{\mathcal{D}}[\psi]$ as in traditional regression. Distinct RR rules are devised for atomic formulae and their negation.

To avoid the addition of a new rule for the negation of each type of compound formula, a formula is converted into negation normal form (NNF) prior to the application of

[8]Refer to Definition 5.1.4 in Section 5.1.6 for a definition of regressable formulae in the situation calculus, and Definition 5.1.1 of Section 5.1.2 for a definition of basic action theories.

RR. In this form: implications ($p \rightarrow q$) are eliminated by replacing them with their equivalent representation ($\neg p \vee q$); negation operators are moved inward using De Morgan's laws; and double negations are eliminated. The result is a logically equivalent formula in which the negation operator appears only before atomic sub-formulae. Let $\phi_n(\psi)$ denote the result of converting a formula ψ into NNF by the aforementioned process.

Remark 6.3.1 *For a situation calculus formula ψ, $\psi \equiv \phi_n(\psi)$. Moreover, $\phi_n(\psi)$ is uniform in a situation s (by Definition 5.1.3) if and only if ψ is uniform in s.*

Definition 6.3.2 In the following: σ and s are situations; α and a are actions; $F(\vec{t}, do(\alpha, s))$ is a relational fluent whose successor state axiom is defined in Equation 5.2 of Section 5.1.4; $G(\vec{t}, do(\alpha, s))$ is a functional fluent whose successor state axiom is defined in Equation 5.3; ψ_{S_0}, ψ_G, ψ_1, and ψ_2 denote regressable formulae in NNF, where ψ_{S_0} is uniform in situation S_0 and ψ_G is a formula involving the functional fluent $G(\vec{t}, do(\alpha, s))$; $A(\vec{t})$ is an action whose precondition axiom is shown in Equation 5.1 of Section 5.1.3; and $\psi|^{\phi}_{\phi'}$ is the formula that results from replacing all instances of ϕ in ψ with ϕ'.

The relaxed regression of a regressable formula ψ given a basic action theory $\mathcal{D}$, denoted $RR_{\mathcal{D}}[\psi]$, is defined by the following rules:

$$RR_{\mathcal{D}}[\psi_{S_0}] \stackrel{def}{=} \psi_{S_0}$$

$$RR_{\mathcal{D}}[Poss(A(\vec{t}), s)] \stackrel{def}{=} RR_{\mathcal{D}}[\phi_n(\Pi_A(\vec{t}, s))]$$

$$RR_{\mathcal{D}}[F(\vec{t}, do(\alpha, s))] \stackrel{def}{=} RR_{\mathcal{D}}[\phi_n(F(\vec{t}, s) \vee \gamma^+_F(\vec{t}, \alpha, s))]$$

$$RR_{\mathcal{D}}[\psi_G] \stackrel{def}{=} RR_{\mathcal{D}}[\phi_n((\exists y).\, (\delta_G(\vec{t}, y, \alpha, s) \vee G(\vec{t}, s) = y) \wedge \psi_G|^{G(\vec{t}, do(\alpha, s))}_{y})]$$

$$RR_{\mathcal{D}}[\neg Poss(A(\vec{t}), s)] \stackrel{def}{=} RR_{\mathcal{D}}[\phi_n(\neg\Pi_A(\vec{t}, s))]$$

$$RR_{\mathcal{D}}[\neg F(\vec{t}, do(\alpha, s))] \stackrel{def}{=} RR_{\mathcal{D}}[\phi_n(\neg F(\vec{t}, s) \vee \gamma^-_F(\vec{t}, \alpha, s))]$$

$$RR_{\mathcal{D}}[\neg\psi_G] \stackrel{def}{=} RR_{\mathcal{D}}[\phi_n((\exists y).\, (\delta_G(\vec{t}, y, \alpha, s) \vee G(\vec{t}, s) = y) \wedge \neg\psi_G|^{G(\vec{t}, do(\alpha, s))}_{y})]$$

$$RR_{\mathcal{D}}[\psi_1 \wedge \psi_2] \stackrel{def}{=} RR_{\mathcal{D}}[\psi_1] \wedge RR_{\mathcal{D}}[\psi_2]$$

$$RR_{\mathcal{D}}[(\exists v).\psi_1] \stackrel{def}{=} (\exists v).\, RR_{\mathcal{D}}[\psi_1]$$

To demonstrate the difference between the rules of relaxed regression RR and those of standard regression R, shown in Definition 5.1.5 of Section 5.1.6, consider the RR rule for a relational fluent $F(\vec{t}, do(\alpha, s))$ in Definition 6.3.2. The R rule for this fluent is:

$$R_{\mathcal{D}}[F(\vec{t}, do(\alpha, s))] \stackrel{def}{=} R_{\mathcal{D}}[\gamma_F^+(\vec{t}, \alpha, s) \vee F(\vec{t}, s) \wedge \neg\gamma_F^-(\vec{t}, \alpha, s)]$$

In the RR rule for the relational fluent $F(\vec{t}, do(\alpha, s))$, I do not consider whether an action causes it to be false ($\gamma_F^-(\vec{t}, \alpha, s)$) – only whether the action α causes it to hold ($\gamma_F^+(\vec{t}, \alpha, s)$) or it held in the previous situation ($F(\vec{t}, s)$). Similarly, consider the R rule for an atomic formula ψ_G involving a functional fluent $G(\vec{t}, do(\alpha, s))$ in Definition 5.1.5 of Section 5.1.6:

$$R_{\mathcal{D}}[\psi_G] \stackrel{def}{=} R_{\mathcal{D}}[(\exists y).(\delta_G(\vec{t}, y, \alpha, s) \vee G(\vec{t}, s) = y \wedge \neg(\exists y').\delta_G(\vec{t}, y', \alpha, s)) \wedge \psi_G|_y^{G(\vec{t}, do(\alpha, s))}]$$

In the RR rule for the formula ψ_G, I do not consider whether the action α changes the evaluation of the fluent $G(\vec{t}, s)$ in such a way as to invalidate ψ_G – only whether ψ_G held in the situation s or if the value of $G(\vec{t}, s)$ has changed such that ψ_G holds in $do(\alpha, s)$.

Example 6.3.1 Let us recall Example 5.1.7 in which I consider a domain with a box and a window. Box_0 has been picked up and window W has been opened. Let $a_1 = open(W)$, $a_2 = pickup(Box_0)$, and $s_1 = do(open(W), S_0)$. Then $s = do(a_2, s_1)$. Consider the relational fluent $Open(W, s)$, which holds if the window W is open in situation s. The basic action theory characterising this domain, denoted $\mathcal{D}_w$, contains the successor state axiom for the fluent $Open(W, s)$ shown in Example 5.1.4.

To determine if $Open(W, s)$ holds under relaxed regression, the RR rule for a relational fluent, shown in Definition 6.3.2, is employed.

$$\begin{aligned}
\mathcal{D}_w \models\ & Open(W, s) \\
& \approx RR_{\mathcal{D}_w}[Open(W, do(a_2, s_1))] \\
& \stackrel{def}{=} RR_{\mathcal{D}_w}[a_2 = open(W) \vee Open(W, s_1)] \\
& \stackrel{def}{=} RR_{\mathcal{D}_w}[a_2 = open(W)] \vee RR_{\mathcal{D}_w}[Open(W, s_1)] \\
& \stackrel{def}{=} a_2 = open(W) \vee RR_{\mathcal{D}_w}[a_1 = open(W) \vee Open(W, S_0)] \\
& \stackrel{def}{=} false \vee RR_{\mathcal{D}_w}[a_1 = open(W)] \vee RR_{\mathcal{D}_w}[Open(W, S_0)] \\
& \stackrel{def}{=} a_1 = open(W) \vee Open(W, S_0) \\
& \stackrel{def}{=} true
\end{aligned}$$

The following lemma is extracted from the work of Pirri and Reiter (1999) and is employed in my proof of the completeness of relaxed regression (Theorem 6.3.1).

Lemma 6.3.1 *Let $\mathcal{D}$ be a basic action theory (BAT) and ψ_{S_0} a formula that is uniform in the initial situation S_0. Then, $\mathcal{D} \models \psi_{S_0}$ if and only if $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \psi_{S_0}$.*

The following two definitions are also extracted from the work of Pirri and Reiter (1999). These definitions describe an ordering over regressable formulae, and allow us to prove the completeness of relaxed regression using the principles of induction.

Definition 6.3.3 Each regressable formula ψ is assigned an index of the form:

$$index(\psi) = [(\mathcal{C}, \lambda_1, \lambda_2, \ldots), \mathcal{P}]$$

where: $\mathcal{C}$ is the number of logical connectives and quantifiers in ψ; λ_i is the number of situation terms of length i in ψ (the length of a situation term is the number of actions performed in its history); and $\mathcal{P}$ is the number of *Poss* atoms in ψ.

An ordering over indices of the form shown in Definition 6.3.3 is described in Definition 6.3.4. Indices are first compared on the basis of the number of *Poss* atoms they contain, and if this number is equal, they are then compared by their first element.

Definition 6.3.4 Let $I_1 = [(\mathcal{C}', \lambda_1', \lambda_2', \ldots), \mathcal{P}']$ and $I_2 = [(\mathcal{C}, \lambda_1, \lambda_2, \ldots), \mathcal{P}]$.

1. I_1 precedes I_2 on the first element:

   $$(\mathcal{C}', \lambda_1', \lambda_2', \ldots) \prec_1 (\mathcal{C}, \lambda_1, \lambda_2, \ldots)$$

   if and only if (i) there exists an m such that $\lambda_m' < \lambda_m$ and for all n such that $n > m$, $\lambda_n' = \lambda_n$, or (ii) for all n, $\lambda_n' = \lambda_n$ and $\mathcal{C}' < \mathcal{C}$.

2. I_1 precedes I_2 in an ordering $\succ_T$ over whole indicies:

   $$[(\mathcal{C}', \lambda_1', \lambda_2', \ldots), \mathcal{P}'] \prec_T [(\mathcal{C}, \lambda_1, \lambda_2, \ldots), \mathcal{P}]$$

   if and only if: (i) $\mathcal{P}' < \mathcal{P}$, or (ii) $\mathcal{P}' = \mathcal{P}$ and $(\mathcal{C}', \lambda_1', \lambda_2', \ldots) \prec_1 (\mathcal{C}, \lambda_1, \lambda_2, \ldots)$. This ordering has a minimal element – the index $\mathbf{0} = [(0, 0, 0, \ldots), 0]$. A formula with this index contains no *Poss* atoms, and is uniform in the initial situation S_0.

Theorem 6.3.1 ***(Completeness of RR)***

Let ψ denote a regressable situation calculus formula (as per Definition 5.1.4) and $\mathcal{D}$ a BAT (as per Definition 5.1.1). Then, $RR_{\mathcal{D}}[\psi]$ is a formula uniform in the initial situation S_0, and:

$$\mathcal{D} \models (\forall).(\psi \rightarrow RR_{\mathcal{D}}[\psi])^9$$

Consequently, if $\mathcal{D} \models \psi$ then $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models RR_{\mathcal{D}}[\psi]$.

Proof: In this proof, I follow the basic steps of the completeness and soundness proof for standard regression *R* [Pirri and Reiter (1999)] and apply the principles of induction. The proof proceeds as follows. I first show that the theorem holds for all regressable formulae with the minimal index **0** (by Definition 6.3.4). I then consider regressable situation calculus formulae with an index $N \succ_T \mathbf{0}$. If it is the case that the theorem holds for each such formula provided it is assumed that it holds for each regressable situation calculus formula with an index $M \prec_T N$, then the theorem holds for all such formulae.

Base Case: Formulae ψ with $index(\psi) = \mathbf{0}$

All regressable situation calculus formulae ψ with index **0** are uniform in the initial situation S_0. For each such ψ, $RR_{\mathcal{D}}[\psi] \stackrel{def}{=} \psi$ (by Definition 6.3.2). It is clear that: (i) $\mathcal{D} \models (\forall).(\psi \rightarrow RR_{\mathcal{D}}[\psi])$, and that (ii) $RR_{\mathcal{D}}[\psi]$ is uniform in S_0. If $\mathcal{D} \models \psi$, then $\mathcal{D} \models RR_{\mathcal{D}}[\psi]$, and by Lemma 6.3.1 $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models RR_{\mathcal{D}}[\psi]$. Theorem 6.3.1 holds for all ψ with index **0**.

Case N: Formulae ψ with $index(\psi) = N \succ_T \mathbf{0}$

I now consider each form that ψ can assume, and make the assumption that the theorem holds for all regressable situation calculus formulae ψ' with $index(\psi') \prec_T N$. This assumption is referred to as ASSUMP in the remainder of this proof. I apply the following reasoning process when considering most of these forms (Remark 6.3.2).

Remark 6.3.2 *Given a regressable situation calculus formula ψ, let ψ' be a regressable formula such that by the definition of relaxed regression, Definition 6.3.2, $RR_{\mathcal{D}}[\psi] \stackrel{def}{=} RR_{\mathcal{D}}[\psi']$. If it can*

[9] As in Pirri and Reiter (1999), $(\forall).\psi$ denotes the universal closure of ψ with respect to its free variables.

be proved that $index(\psi') \prec_T index(\psi)$, then it is assumed by ASSUMP that Theorem 6.3.1 holds for ψ'. Consequently, $RR_{\mathcal{D}}[\psi']$ is uniform in S_0 and $\mathcal{D} \models (\forall).(\psi' \rightarrow RR_{\mathcal{D}}[\psi'])$. Moreover, if it can be proved that $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$, then by transitivity of implication it is the case that:

$$\mathcal{D} \models (\forall).(\psi \rightarrow RR_{\mathcal{D}}[\psi']) \tag{6.1}$$

$$\therefore \mathcal{D} \models (\forall).(\psi \rightarrow RR_{\mathcal{D}}[\psi]) \tag{6.2}$$

This, in conjunction with Lemma 6.3.1, proves Theorem 6.3.1 for ψ on the basis of ASSUMP.

I now proceed to prove the theorem for each form that the formula ψ can take. These are: a compound formula; and a (negated and non-negated) possibility atom, relational fluent, and atomic formula involving a functional fluent.

1. *ψ is a possibility atom $Poss(A(\vec{t}), s)$ where $A(\vec{t})$ is an action. The precondition axiom for action $A(\vec{t})$ is defined as $Poss(A(\vec{x}), \sigma) \equiv \Pi_A(\vec{x}, \sigma)$ (Equation 5.1 of Section 5.1.3), and the relaxed regression rule for this ψ is shown in Definition 6.3.2.*

   By Remark 6.3.1 regarding ϕ_n, $\psi = Poss(A(\vec{t}), s) \equiv \phi_n(\Pi_A(\vec{t}, s)) = \psi'$. Consequently, $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$. ψ' is uniform in s as $\Pi_A(\vec{t}, s)$ is uniform in s (and so contains no *Poss* atoms). It follows that $index(\psi') \prec_T index(\psi)$. By Remark 6.3.2, Theorem 6.3.1 holds for ψ on the basis of ASSUMP.

2. *ψ is a negated possibility atom $Poss(A(\vec{t}), s)$. The precondition axiom for action $A(\vec{t})$ is defined as $Poss(A(\vec{x}), \sigma) \equiv \Pi_A(\vec{x}, \sigma)$ (Equation 5.1 in Section 5.1.3), and the relaxed regression rule for this ψ is shown in Definition 6.3.2.*

   By Remark 6.3.1 regarding ϕ_n, $\psi = \neg Poss(A(\vec{t}), s) \equiv \phi_n(\neg\Pi_A(\vec{t}, s)) = \psi'$. Equivalent reasoning to that of Case 1 is applied from this point onward and I conclude that Theorem 6.3.1 holds for ψ, by Remark 6.3.2 on the basis of ASSUMP.

3. *ψ is a relational fluent $F(\vec{t}, do(\alpha, s))$. The successor state axiom for this fluent is $F(\vec{x}, do(a, \sigma)) \equiv \gamma_F^+(\vec{x}, a, \sigma) \vee F(\vec{x}, \sigma) \wedge \neg\gamma_F^-(\vec{x}, a, \sigma)$ (Equation 5.2 in Section 5.1.4), and*

the relaxed regression rule for this ψ is shown in Definition 6.3.2.

Let $\psi' = \phi_n(F(\vec{t}, s) \vee \gamma_F^+(\vec{t}, \alpha, s))$. By Remark 6.3.1, ψ' is uniform in situation s. Which of ψ or ψ' has a higher index is based on the length of the situation terms appearing in each formula. It is clear that ψ involves the situation term $do(\alpha, s)$, and that ψ' mentions only s. It follows that $index(\psi') \prec_T index(\psi)$.

It must now be determined if $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$. The two formulae are not equivalent as they have been in the previous two cases. ψ holds (relative to $\mathcal{D}$) if the expression $\gamma_F^+(\vec{t}, \alpha, s) \vee F(\vec{t}, s) \wedge \neg\gamma_F^-(\vec{t}, \alpha, s)$ holds. Consequently, ψ holds only when: (i) $\gamma_F^+(\vec{t}, \alpha, s)$ holds; or (ii) $F(\vec{t}, s) \wedge \neg\gamma_F^-(\vec{t}, \alpha, s)$ holds. If either (i) or (ii) is the case, it is clear that ψ' holds. It has now been established that $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$. By Remark 6.3.2, Theorem 6.3.1 holds for ψ on the basis of ASSUMP.

4. *ψ is an atomic formula mentioning a functional fluent $G(\vec{t}, do(\alpha, s))$. The successor state axiom for this fluent is defined in Equation 5.3 of Section 5.1.4, and has the form $G(\vec{x}, do(a, \sigma)) = y \equiv \delta_G(\vec{x}, y, a, \sigma) \vee G(\vec{x}, \sigma) = y \wedge \neg(\exists y').\delta_G(\vec{x}, y', a, \sigma)$. The relaxed regression rule for this ψ is shown in Definition 6.3.2.*

Let $\psi|_{\phi'}^{\phi}$ be defined as in Definition 6.3.2 and $\psi^* = \psi|_y^{G(\vec{t}, do(\alpha, s))}$. Let $\psi' = \phi_n((\exists y).(\delta_G(\vec{t}, y, \alpha, s) \vee G(\vec{t}, s) = y) \wedge \psi^*)$. By Remark 6.3.1, ψ' is uniform in situation s. To determine which of ψ or ψ' has a higher index, I apply the same reasoning process used in Case 3. It follows that $index(\psi') \prec_T index(\psi)$.

It must now be determined if it is that case that $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$. ψ holds (relative to $\mathcal{D}$) only if the expression $(\exists y).(\delta_G(\vec{t}, y, \alpha, s) \vee G(\vec{t}, s) = y \wedge \neg(\exists y').\delta_G(\vec{t}, y', \alpha, s)) \wedge \psi^*$ holds. So, ψ holds only when there exists a y such that (i) $\delta_G(\vec{t}, y, \alpha, s) \wedge \psi^*$ or (ii) $G(\vec{t}, s) = y \wedge \neg(\exists y').\delta_G(\vec{t}, y', \alpha, s) \wedge \psi^*$ holds. If either (i) or (ii) is the case, it is clear that ψ' holds. It has now been established that $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$. By Remark 6.3.2, Theorem 6.3.1 holds for ψ on the basis of ASSUMP.

5. *ψ is the negation of a relational fluent $F(\vec{t}, do(\alpha, s))$. The successor state axiom for this*

fluent is $F(\vec{x}, do(a,\sigma)) \equiv \gamma_F^+(\vec{x},a,\sigma) \vee F(\vec{x},\sigma) \wedge \neg\gamma_F^-(\vec{x},a,\sigma)$ *(Equation 5.2 in Section 5.1.4), and the relaxed regression rule for this* ψ *is shown in Definition 6.3.2.*

Let $\psi' = \phi_n(\neg F(\vec{t},s) \vee \gamma_F^-(\vec{t},\alpha,s))$. By Remark 6.3.1, ψ' is uniform in situation s. To determine which of ψ or ψ' has a higher index, I apply the same reasoning process used in Case 3. It follows that $index(\psi') \prec_T index(\psi)$.

It must now be determined if $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$ holds. ψ holds only when the expression $\neg[\gamma_F^+(\vec{t},\alpha,s) \vee F(\vec{t},s) \wedge \neg\gamma_F^-(\vec{t},\alpha,s)]$ holds. Hence, ψ holds only when (i) $\gamma_F^-(\vec{t},\alpha,s) \wedge \neg\gamma_F^+(\vec{t},\alpha,s)$ or (ii) $\neg F(\vec{t},s) \wedge \neg\gamma_F^+(\vec{t},\alpha,s)$ holds. If either (i) or (ii) is the case, it is clear that ψ' holds. It has now been established that $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$. By Remark 6.3.2, Theorem 6.3.1 holds for ψ on the basis of ASSUMP.

6. ψ *is the negation of an atomic formula* ψ_G *mentioning a functional fluent* $G(\vec{t}, do(\alpha,s))$. *The successor state axiom for this fluent is outlined in Equation 5.3 of Section 5.1.4:* $G(\vec{x}, do(a,\sigma)) = y \equiv \delta_G(\vec{x},y,a,\sigma) \vee G(\vec{x},\sigma) = y \wedge \neg(\exists y').\delta_G(\vec{x},y',a,\sigma)$. *The relaxed regression rule for this* ψ *is shown in Definition 6.3.2.*

As in Case 4, let $\psi|_{\phi'}^{\phi}$ be defined as in Definition 6.3.2 and $\psi_G^* = \psi_G|_y^{G(\vec{t},do(\alpha,s))}$. Let $\psi' = \phi_n((\exists y).(\delta_G(\vec{t},y,\alpha,s) \vee G(\vec{t},s) = y) \wedge \neg\psi_G^*)$. By Remark 6.3.1, ψ' is uniform in situation s. To determine which of ψ or ψ' has a higher index, I apply the same reasoning process used in Case 3. It follows that $index(\psi') \prec_T index(\psi)$.

It must now be determined if $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$ holds. ψ holds (relative to $\mathcal{D}$) only when $(\exists y).(\delta_G(\vec{t},y,\alpha,s) \vee G(\vec{t},s) = y \wedge \neg(\exists y').\delta_G(\vec{t},y',\alpha,s)) \wedge \neg\psi_G^*$ holds. Hence, ψ holds only when either (i) $(\exists y).\delta_G(\vec{t},y,\alpha,s) \wedge \neg\psi_G^*$, or (ii) $(\exists y).(G(\vec{t},s) = y \wedge \neg(\exists y').\delta_G(\vec{t},y',\alpha,s)) \wedge \neg\psi_G^*$ holds. If either of (i) or (ii) hold, it must be the case that ψ' holds. It has now been established that $\mathcal{D} \models (\forall).(\psi \rightarrow \psi')$. By Remark 6.3.2, Theorem 6.3.1 for ψ on the basis of ASSUMP.

7. ψ is a compound formula. Let ψ_1 and ψ_2 denote regressable formulae. I consider the primitive connectives $\wedge$ and $\exists$. Alternative compound formulae, constructed

using $\forall$, $\vee$, and $\rightarrow$, can be expressed in terms of these primitives (plus $\neg$). I do not consider the $\neg$ connective as each ψ is in NNF, where the $\neg$ operator appears only before atomic sub-formulae. I have already considered these cases.

(a) *Let* $\psi = \psi_1 \wedge \psi_2$.

Definition 6.3.2 presents the relaxed regression rule for ψ. It is evident that $index(\psi_1) \prec_T index(\psi)$ and $index(\psi_2) \prec_T index(\psi)$. By ASSUMP, Theorem 6.3.1 holds for ψ_1 and ψ_2 separately. Consequently, both $RR_{\mathcal{D}}[\psi_1]$ and $RR_{\mathcal{D}}[\psi_2]$ are uniform in S_0 (hence so is $RR_{\mathcal{D}}[\psi_1 \wedge \psi_2]$), and:

$$\mathcal{D} \models (\forall).(\psi_1 \rightarrow RR_{\mathcal{D}}[\psi_1]), \mathcal{D} \models (\forall).(\psi_2 \rightarrow RR_{\mathcal{D}}[\psi_2]) \quad (6.3)$$

$$\therefore \mathcal{D} \models (\forall).(\psi_1 \wedge \psi_2 \rightarrow RR_{\mathcal{D}}[\psi_1] \wedge RR_{\mathcal{D}}[\psi_2]) \quad (6.4)$$

$$\therefore \mathcal{D} \models (\forall).(\psi_1 \wedge \psi_2 \rightarrow RR_{\mathcal{D}}[\psi_1 \wedge \psi_2]) \quad (6.5)$$

By Lemma 6.3.1, Theorem 6.3.1 holds for ψ on the basis of ASSUMP.

(b) *Let* $\psi = (\exists v)\psi_1$.

Definition 6.3.2 presents the relaxed regression rule for ψ. It is clear that $index(\psi_1) \prec_T index((\exists v)\psi_1)$ as ψ_1 involves one less quantifier. By ASSUMP, the theorem holds for ψ_1. $RR_{\mathcal{D}}[\psi_1]$ is uniform in the initial situation S_0, and $\mathcal{D} \models (\forall).(\psi_1 \rightarrow RR_{\mathcal{D}}[\psi_1])$. Let us denote this entailment as E.

Assume that it is *not* the case that $\mathcal{D} \models (\forall)((\exists v).\psi_1 \rightarrow (\exists v).RR_{\mathcal{D}}[\psi_1])$ holds. It must be the case that ψ_1 is satisfiable given $\mathcal{D}$ (otherwise the entailment would trivially hold). Hence, it must be the case that (i) there exists a v such that ψ_1 holds given $\mathcal{D}$, but (ii) there exists no v such that $RR_{\mathcal{D}}[\psi_1]$ holds. By E above, (i) indicates that there does exist a v such that $RR_{\mathcal{D}}[\psi_1]$ holds. A contradiction has been arrived at. By ASSUMP, it must be the case that $\mathcal{D} \models (\forall).((\exists v).\psi_1 \rightarrow (\exists v).RR_{\mathcal{D}}[\psi_1])$. $RR_{\mathcal{D}}[\psi_1]$ is uniform in S_0 and so is $(\exists v)RR_{\mathcal{D}}[\psi_1]$. By Lemma 6.3.1, Theorem 6.3.1 holds for $\psi = (\exists v)\psi_1$.

Theorem 6.3.1 holds for all regressable situation calculus formulae of index $\mathbf{0}$. For any regressable situation calculus formula ψ of index $N \succ_T \mathbf{0}$, assuming the theorem holds

for all formula with index $M \prec_T N$ allows us to conclude that the theorem holds for ψ. By induction, the theorem holds for all regressable situation calculus formulae.

□

Armed with a theory of relaxed regression, I can now develop a relaxed GOLOG interpreter and, in doing so, piece together a heuristic preference-guided interpreter for GOLOG programs. I describe this relaxed interpreter in Section 6.5, and an implementation of this heuristic interpreter in Section 6.6. In the following section I revisit the goals of, and motivation behind, the development of this heuristic interpreter. This next section describes the algorithm that underpins heuristic GOLOG, presenting the context within which my relaxed GOLOG interpreter is employed.

6.4 A Heuristic GOLOG Interpreter – A Preamble

A GOLOG interpreter makes a sequence of choices in the search for a program execution. These choices are: selecting a path to pursue given a branch construct $(\delta_1|\delta_2)$; how to instantiate a variable v in $\pi v.\delta$; and whether to execute a program δ zero times or at least once in δ^*. When faced with a choice, a standard GOLOG interpreter selects the first available option (eg. δ_1 given the branch $\delta_1|\delta_2$). When search reaches a dead-end – a non-final configuration in which no further transitions can be made – the interpreter backtracks to the last choice made and selects a different choice in its place[10].

6.4.1 Goals and Motivation

A standard interpreter does not consider the impact of its choices on the quality of the executions it finds. I develop a heuristic interpreter that evaluates each available choice when faced with a non-deterministic construct. I assume: that there exists an evaluation function f_e that returns the numeric quality of an execution (Definition 6.4.1); and that the aim of my interpreter is to find a program execution that minimises this function.

Definition 6.4.1 $f_e : \delta \times s \to \mathbb{R}$ computes the desirability of a partial or complete execution s of a program δ' with program δ left to perform. A program-execution pair (δ_1, s_1)

[10]The backtracking referred to here is simply the process that underpins the logical programming languages in which a GOLOG interpreter is implemented (such as Prolog).

is more desirable than another (δ_2, s_2) if and only if $f_e(\delta_1, s_1) < f_e(\delta_2, s_2)$[11].

Example 6.4.1 Consider a mail delivery robot, controlled by a GOLOG program. The robot must navigate through a series of offices, delivering mail to their inhabitants. In this domain, if our preference is for the robot to minimise distance travelled, the evaluation function $f_e(\delta, s)$ will describe the distance travelled by the robot in execution s.

6.4.2 The Interpretation Process

Like a standard interpreter, my heuristic interpreter employs a transition semantics in its implementation, and operates as follows. In the presence of a deterministic construct, there is only one choice of transition and that transition is made. When faced with a non-deterministic construct, this interpreter has the option of transitioning to one of a number of configurations $[(\delta_{c_1}, s_{c_1}), \ldots, (\delta_{c_n}, s_{c_n})]$. For each of these choices, a relaxed version of the problem is solved – a relaxed interpreter (defined in Section 6.5) is used to find a relaxed execution (δ_{r_i}, s_{r_i}) of δ_{c_i} in s_{c_i}. Each (δ_{c_i}, s_{c_i}) is then assigned a numeric evaluation given by $f_e(\delta_{r_i}, s_{r_i})$ – as defined in Definition 6.4.1 and demonstrated in Example 6.4.1.

The choices available to this heuristic interpreter are ordered from best (smallest) evaluation to worst (largest). The best configuration (at the head of this list) is transitioned to, and if that choice leads to a dead-end (where there are no available choices that lead to a legal execution of the program), the next best configuration is considered upon backtracking. Upon selection of a final configuration, its situation is returned as the program execution. Section 6.6 presents an implementation of my heuristic interpreter. For this heuristic interpretation to be effective, these relaxed executions must give this interpreter an insight into the potential of the choice they have been generated from – representing a 'desirable future' that may arise by pursuing it (as described in Section 6.5.1).

In some respects, my interpreter is similar to the search algorithms of HSP [Bonet and Geffner (2001)] and FF [Hoffman and Nebel (2001)]. Once a choice of transition is made, it is typically not reconsidered – this interpreter will (if possible) return an execution that this choice leads to. One difference, however, is that upon reaching a dead-end the in-

[11] I have chosen minimisation as the convention in the interpreter I present – the heuristic interpreter could have been defined to use maximisation as its convention.

terpreter does not result in failure. As in a standard GOLOG interpreter, it backtracks to the last choice made and selects an alternative in its place. The heuristic interpreter of Sohrabi et al. (2006), described in Section 6.1.3, maintains unexplored choices on a search frontier and, in contrast, explores these choices if they are evaluated more favourably than the current path being pursued. I discover in Chapter 7 that this maintenance approach becomes infeasible in the moderately complex test domains I consider.

6.5 A Relaxed GOLOG Interpreter

Relaxed regression can be used to approximate the interpretation of a GOLOG program. A relaxed interpreter finds a program execution while optimistically deciding if an action is possible or not. The result is a relaxed execution of a program in a given situation.

6.5.1 Desirable Futures

The purpose of this relaxed interpreter is to determine what desirable future may result from making a particular choice in the interpretation of a program. It is the desirability of these relaxed executions that determine the heuristic values assigned to choices during heuristic interpretation. This relaxed interpreter should not make blind choices in the same way that a standard interpreter does – it must itself be guided by a heuristic.

To create this heuristic, I assume that an evaluation function f_p that assesses the quality of a configuration exists – which may or may not be the same as f_e (the evaluation function for executions). When faced with a range of configurations to transition to, my relaxed interpreter selects the configuration that minimises this function. If this configuration leads to a dead-end, where no further (relaxed) transitions can be made, the next best configuration is selected upon backtracking. The relaxed interpreter, whose implementation is shown in Definition 6.5.3, is designed to represent a heuristic interpreter with a greedy non-lookahead-based heuristic. I define f_p as follows.

Definition 6.5.1 $f_p : \delta \times s \rightarrow \mathbb{R}$ computes the desirability of a partial execution s of a program δ' with program δ left to perform. A program-situation pair (δ_1, s_1) is more desirable than another (δ_2, s_2) if and only if $f_p(\delta_1, s_1) < f_p(\delta_2, s_2)$.

Example 6.5.1 Recall the mail delivery robot of Example 6.4.1. The robot is at a configuration (δ, s). If in the next step the robot selects a package to deliver, the robot can select as its next choice one of a number of available packages. The evaluation function f_p prefers to select a package whose address is the closest office to its current location.

The relationship between f_e and f_p

As mentioned above, the evaluation functions f_e and f_p may be the same, or defined differently. Consider the evaluation of executions with respect to cost – f_e is designed such that an execution involving less cost is preferred to one that is more costly. Instantiating f_p to exhibit the same behaviour as f_e in this instance ($f_e \equiv f_p$) is sensible – selecting actions that form the least cost partial execution is a reasonable strategy for the discovery of a least cost execution (without performing any form of lookahead).

In contrast, consider a scenario in which a spacecraft (controlled by a GOLOG program) is designed to observe celestial targets within a limited time window (this scenario forms one of the domains used to evaluate my heuristic interpreter in Chapter 7). In this domain, to make an observation the spacecraft must rotate to, and focus its camera on, a target – each rotation consuming an amount of time dependent on the magnitude of its turn. If the spacecraft is required to maximise the number of observations made, the evaluation function f_e compares executions on the basis of observation count. When faced with a range of targets to next observe, irrespective of which is chosen the spacecraft will transition to a configuration in which the same number of observations have been made. Hence, it is not useful to define f_p in the same manner as f_e. Instead, f_p may be defined to select a target that is closest to the current focal point of the spacecraft.

6.5.2 Termination of Relaxed Interpretation

Relaxed interpretation must terminate if heuristic values are to be extracted from the executions it discovers. To ensure termination it must be assumed that: the domain in question is finite (ie. v in $\pi v.\delta$ ranges over a finite number of values); there is a finite horizon of actions h over which the relaxed interpreter searches for an execution[12]; and

[12]This finite action horizon is an upper bound on the number of actions added by the interpreter to a situation s when finding an execution of a program δ in situation s.

all paths through a while or iteration construct involve an action being performed. If this last condition is not satisfied, such loops can become infinite even with such a h.

In classical planning, selecting a horizon of actions that we expect a plan to be defined within can be difficult. As shall be seen in the test domains of Chapter 7, the structure of a GOLOG program provides a basis from which h can be derived. Given the intended use of the relaxed interpreter as a tool for generating heuristic values, the appropriate selection of h need not be agonised over. A h value that is too low will only yield a (potentially) less informed heuristic. As I shall discuss in Section 6.7, the use of this lookahead bound allows the presented heuristic interpreter to be used in an online setting.

A finite action horizon is used in several other GOLOG interpreters, including: DTGOLOG [Boutilier et al. (2000)]; DTGOLOG with qualitative preferences [Fritz and McIlraith (2005)]; and other DTGOLOG or MDP-based interpreters. Beck and Lakemeyer (2009) prevent the occurrence of infinite execution traces of a GOLOG program by replacing loops with a finite number of nested conditionals, and iteration statements with a choice operator over a finite sequence of programs – each program a finite number of repetitions of the iteration. My use of an action horizon has an equivalent impact.

6.5.3 A Transition Semantics

The presented relaxed interpreter is defined in terms of a transition semantics – employing the same *Trans* and *Final* axioms as the standard interpreter[13] of Definitions 5.3.1 and 5.3.2 in Section 5.3.1 with the exception of the *Trans* axiom for actions, branches, argument selection, and iteration. In this relaxed interpreter, these *Trans* and *Final* axioms are named $Trans_R$ and $Final_R$. Definition 6.5.3 defines these transition axioms – a subset of which are complemented with their implementation in SWI-Prolog. In these clauses, I deviate from the convention of Chapter 5 where lower case letters denote variables, and upper case letters constants – words starting with a capital letter denote variables.

The $Final_R$ axioms of the relaxed interpreter are the same as those of a standard interpreter (Definition 5.3.2 in Section 5.3.1). Hence, I do not present them in this section. Before listing the transition axioms of the relaxed interpreter, I describe (in Definition 6.5.2) how these axioms are used to find a relaxed execution of a GOLOG program.

[13] With the addition of bookkeeping related to the finite action horizon h.

Definition 6.5.2 The $Trans_R$ axioms presented in Definition 6.5.3 are used to find a relaxed execution s' of a program δ in situation s with a finite action horizon h and a BAT $\mathcal{D}$ (Definition 5.1.1) as follows. Execution s' is found as a by-product of solving the logical query in Equation 6.6 (where $Trans_R^*$ is the reflexive transitive closure of $Trans_R$[14]).

$$\mathcal{D} \models \exists s'.Do_R(\delta, s, s', h) \tag{6.6}$$
$$Do_R(\delta, s, s', h) \stackrel{def}{=} \exists \delta', h'.Trans_R^*(h, \delta, s, \delta', s', h') \wedge (h' = 0 \vee Final_R(\delta', s'))$$

Definition 6.5.3 Each construct δ_c of Table 5.2 in Section 5.3 is associated with a transition ($Trans_R$) axiom as follows, where: (δ', s') denotes the configuration that is transitioned to by performing one step of δ_c in a situation s; $\alpha[s]$ and $\phi[s]$ denote the result of restoring situation s as the final argument in the fluents within formula ϕ and action α; δ_1 and δ_2 denote programs; h and h' denote finite action horizons (h the horizon prior to the transition being made, and h' the horizon afterward)[15]; δ_x^v denotes the program that results when all instances of v in δ are replaced with x; and $Poss_R(\alpha[s], s)$ is true if and only if action α is possible in situation s under relaxed regression (Section 6.3.1).

$$Trans_R(h, nil, s, \delta', s', h') \equiv False$$

$$Trans_R(h, \alpha, s, \delta', s', h') \equiv h > 0 \wedge Poss_R(\alpha[s], s) \wedge \delta' = nil \wedge s' = do(\alpha[s], s) \wedge h' = h - 1$$

$$Trans_R(h, \phi?, s, \delta', s', h') \equiv \phi[s] \wedge \delta' = nil \wedge s' = s \wedge h' = h$$

$$Trans_R(h, \delta_1 : \delta_2, s, \delta', s', h') \equiv \exists \gamma.\delta' = (\gamma : \delta_2) \wedge Trans_R(h, \delta_1, s, \gamma, s', h') \vee$$
$$Final_R(\delta_1, s) \ \wedge \ Trans_R(h, \delta_2, s, \delta', s', h')$$

$$Trans_R(h, \text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2, s, \delta', s', h') \equiv \phi[s] \wedge Trans_R(h, \delta_1, s, \delta', s', h') \vee$$
$$\neg\phi[s] \wedge Trans_R(h, \delta_2, s, \delta', s', h')$$

$$Trans_R(h, \text{while } \phi \text{ do } \delta \text{ end}, s, \delta', s', h') \equiv \exists \gamma.(\delta' = \gamma : \text{while } \phi \text{ do } \delta \text{ end}) \wedge$$
$$\phi[s] \ \wedge \ Trans_R(h, \delta, s, \gamma, s', h')$$

[14] $Trans_R^*(h, \delta, s, \delta', s', h')$ is defined as $Trans_R^*(h, \delta, s, \delta', s', h') \stackrel{def}{=} \forall T.\,[\ldots \supset T(h, \delta, s, \delta', s', h')]$, as per de Giacomo et al. (2000), where $\ldots$ denotes the conjunction of the universal closure of the implications:

$$True \supset T(h, \delta, s, \delta, s, h)$$
$$Trans_R(h, \delta, s, \delta'', s'', h'') \wedge T(h'', \delta'', s'', \delta', s', h') \supset T(h, \delta, s, \delta', s', h')$$

[15] h is an upper bound on the number of actions the interpreter can add to s in the search for an execution of the program δ_c in situation s. Horizon h' is passed as input h in the next transition.

The transition axioms for the branch $(\delta_1|\delta_2)$, iteration (δ^*), and argument selection $(\pi v.\delta)$ constructs share a common form. Let δ_p denote a branch, iteration, or argument selection statement. $Trans_R$ for δ_p employs the predicate $Successor(h, \delta_p, s, \delta'_p, s', h')$ which holds if and only if (δ'_p, s') is a possible transition from (δ_p, s) given an action horizon h.

$$Successor(h, \delta_1|\delta_2, s, \delta', s', h') \equiv Trans_R(h, \delta_1, s, \delta', s', h') \vee Trans_R(h, \delta_2, s, \delta', s', h')$$
$$Successor(h, \delta^*, s, \delta', s', h') \equiv \exists\gamma.(\delta' = \gamma : \delta^*) \wedge Trans_R(h, \delta, s, \gamma, s', h')$$
$$Successor(h, \pi v.\delta, s, \delta', s', h') \equiv \exists x.Trans_R(h, \delta^v_x, s, \delta', s', h')$$

Recall from Definition 6.5.2 that $Do_R(\delta, s, s', h)$ finds a relaxed execution s' of (δ, s), up to h actions long, using $Trans_R$. Recall also that f_p is an evaluation function that determines the desirability of a partial execution of a program (Definition 6.5.1).

$$Trans_R(h, \delta_p, s, \delta'_p, s', h') \equiv$$
$$Successor(h, \delta_p, s, \delta'_p, s', h') \wedge \forall\delta''_p, s'', h''.[Successor(h, \delta_p, s, \delta''_p, s'', h'') \wedge$$
$$(\delta''_p, s'', h'') \neq (\delta'_p, s', h') \wedge f_p(\delta''_p, s'') < f_p(\delta'_p, s') \rightarrow \neg\exists s_f.Do_R(\delta''_p, s'', s_f, h'')]$$

This axiom states that (δ_p, s) transitions to (δ'_p, s'), with horizon h' remaining, only if each alternative transition (δ''_p, s'') is less than or equally desirable, $f_p(\delta'_p, s') \leq f_p(\delta''_p, s'')$, or there does not exist an execution of δ''_p in s'' by Do_R given the horizon available. A direct translation of this axiom into SWI-Prolog would be inefficient. The SWI-Prolog clause that appears in my implementation of this relaxed interpreter is shown below, where: $Trans_R$, h, δ_p, s, s', δ'_p, and h' are denoted *transR*, *H*, *DP*, *S*, *SD*, *DPD*, and *HD*.

```
transR(H, DP, S, DPD, SD, HD) :-
    successors(H, DP, S, SUCC), trans_sort(SUCC, SUCCS),
    member((DPD, SD, HD), SUCCS).
```

In this clause, *successors(H, DP, S, SUCC)* finds the set *SUCC* of transitions possible from (*DP*, *S*) given horizon *H* (using SWI-Prolog translations of the *Successor* axioms shown above). The predicate *trans_sort(SUCC, SUCCS)* produces the list *SUCCS* – a sorted version of *SUCC* in which transitions are sorted by their f_p evaluation (in ascending order). The last line of this clause selects the first element of this list as the next transition. If

this selection does not lead to a relaxed execution of the program DP in S (using a SWI-Prolog implementation of Do_R), SWI-Prolog will backtrack to this choice point and select the next transition in the list *SUCCS* in its place. By selecting transitions in ascending order of their f_p evaluation, the semantics of $Trans_R$ (above) are captured.

A procedure call $\beta(\vec{x})$ of **proc** $\beta(\vec{x})$ δ **end** is transitioned by replacing it with δ and transitioning δ using the axioms above.

Relaxed regression is not applied to assess the truth of test, while loop, and if . . . else conditions. It seems reasonable that relaxing all such tests should result in a faster interpretation. It is important to strike a balance, however, between how much is relaxed and how representative the resulting relaxed execution is of a legal execution of a program.

Test constructs and while loop conditions serve an important purpose in controlling when to stop performing a series of steps – for example, when a particular goal has been achieved. The danger in relaxing these conditions is that this collection of steps may never be performed (as under relaxation a test condition may hold quite often), or repeatedly performed until the action horizon is reached (as under relaxation a while loop condition may always hold). The resulting relaxed executions and the heuristics they derive are not useful in guiding the interpretation of a program. If the condition in an if . . . else construct is relaxed, the interpreter exhibits a bias toward following the first path of the construct – again limiting the representativeness of the resulting execution.

I demonstrate in Chapter 7 the impact that relaxing all tests during the course of relaxed interpretation has on the performance of the heuristic interpreter that employs it.

6.6 An Implementation of Heuristic GOLOG

Recall from Section 6.4 that the heuristic interpreter I present is designed to find an execution of a GOLOG program that minimises a numeric evaluation function f_e (Definition 6.4.1). The presented heuristic interpreter finds an execution of a GOLOG program in the same manner as a traditional interpreter – with a set of transition axioms. This heuristic interpreter uses the same *Final* axioms as the traditional interpreter (see Definition 5.3.2 in Section 5.3.1) but defines new *Trans* axioms for most of the GOLOG constructs,

denoted $Trans_H$ and $Final_H$. Definition 6.6.2 defines these transition axioms.

Before listing the transition axioms of the heuristic interpreter, I describe (in Definition 6.6.1) how these axioms are used to find an execution of a GOLOG program.

Definition 6.6.1 The $Trans_H$ axioms presented in Definition 6.6.2 are used to find an execution s' of a program δ in situation s with a finite action horizon h[16] and a BAT $\mathcal{D}$ (Definition 5.1.1) as follows. Execution s' is found as a by-product of solving the logical query in Equation 6.7 (where $Trans_H^*$ is the reflexive transitive closure of $Trans_H$[17]).

$$\mathcal{D} \models \exists s'.Do_H(\delta, s, s', h) \tag{6.7}$$
$$Do_H(\delta, s, s', h) \stackrel{def}{=} \exists \delta'.Trans_H^*(\delta, s, \delta', s', h) \wedge Final_H(\delta', s')$$

Definition 6.6.2 Each GOLOG construct δ_c of Table 5.2 in Section 5.3 is associated with a $Trans_H(\delta_c, s, \delta', s', h)$ axiom in my heuristic interpreter. Let: (δ', s') denote the configuration transitioned to by this interpreter after performing a single step of δ_c in a situation s; h an action horizon for the relaxed interpretation (see Definition 6.5.3) of a program; $\alpha[s]$ and $\phi[s]$ denote the result of restoring situation s as the final argument in the fluents within formula ϕ and action α; δ_1 and δ_2 denote programs; and $Poss(\alpha[s], s)$ hold if and only if action α is possible in situation s under standard regression (see Definition 5.1.5).

$$Trans_H(nil, s, \delta', s', h) \equiv False$$

$$Trans_H(\alpha, s, \delta', s', h) \equiv Poss(\alpha[s], s) \wedge \delta' = nil \wedge s' = do(\alpha[s], s)$$

$$Trans_H(\phi?, s, \delta', s', h) \equiv \phi[s] \wedge \delta' = nil \wedge s' = s$$

$$Trans_H(\textit{if } \phi \textit{ then } \delta_1 \textit{ else } \delta_2, s, \delta', s', h) \equiv \phi[s] \wedge Trans_H(\delta_1, s, \delta', s', h) \vee$$
$$\neg\phi[s] \wedge Trans_H(\delta_2, s, \delta', s', h)$$

Transition axioms for the branch $(\delta_1|\delta_2)$, iteration (δ^*), argument selection $(\pi v.\delta)$, sequence $(\delta_1 : \delta_2)$, and while loop $(\textbf{while } \phi \textbf{ do } \delta \textbf{ end})$ constructs share a common form. Let

[16] This action horizon is used only for the purposes of relaxed interpretation during the heuristic estimation of choices. It does not restrict the length of executions found by the heuristic interpreter in any way.

[17] $Trans_H^*(\delta, s, \delta', s', h)$ is defined as $Trans_H^*(\delta, s, \delta', s', h) \stackrel{def}{=} \forall T.\ [\ldots \supset T(\delta, s, \delta', s', h)]$, as per de Giacomo et al. (2000), where $\ldots$ denotes the conjunction of the universal closure of the implications:

$$True \supset T(\delta, s, \delta, s, h)$$
$$Trans_H(\delta, s, \delta'', s'', h) \wedge T(\delta'', s'', \delta', s', h) \supset T(\delta, s, \delta', s', h)$$

δ_p denote a branch, iteration, argument selection, sequence, or while loop statement. Recall from Definition 6.6.1 that: $Do_H(\delta, s, s', h)$ finds an execution s' of (δ, s), using $Trans_H$, while attempting to minimise an evaluation function f_e (Definition 6.4.1); and Do_R finds a relaxed execution of a program given an action horizon (Definition 6.5.2). $Trans_H$ for the δ_p constructs employs the *Trans* axioms of a standard GOLOG interpreter (Section 5.3.1), and selects a transition (of those possible) whose relaxed execution minimises f_e.

$$\begin{aligned} Trans_H(\delta_p, s, \delta'_p, s', h) \equiv Trans(\delta_p, s, \delta'_p, s') \wedge \forall \delta''_p, s''.[Trans(\delta_p, s, \delta''_p, s'') \wedge \\ (\delta''_p, s'') \neq (\delta'_p, s') \wedge \exists s'_r, s''_r.Do_R(\delta'_p, s', s'_r, h) \wedge Do_R(\delta''_p, s'', s''_r, h) \\ \wedge f_e(nil, s''_r) < f_e(nil, s'_r) \rightarrow \neg \exists s_f.Do_H(\delta''_p, s'', s_f, h)] \end{aligned}$$

This axiom states that (δ_p, s) transitions to (δ'_p, s') only if each alternative transition (δ''_p, s'') has a less than or equally desirable relaxed execution by Do_R $[f_e(nil, s'_r) \leq f_e(nil, s''_r)]$ or there does not exist an execution of δ''_p in s'' by Do_H $[\neg \exists s_f.Do_H(\delta''_p, s'', s_f, h)]$. A direct translation of this axiom into SWI-Prolog would be inefficient. The SWI-Prolog clause that appears in my implementation of this heuristic interpreter is shown below, where: $Trans_H$, h, δ_p, s, s', and δ'_p are denoted *transH*, *H*, *DP*, *S*, *SD*, and *DPD*.

```
transH(DP, S, DPD, SD, H) :-
        successors_h(DP, S, SUCC), trans_sort_h(H, SUCC, SUCCS),
        member((DPD, SD), SUCCS).
```

In this clause, *successors_h(DP, S, SUCC)* finds the set *SUCC* of transitions possible from (*DP*, *S*) using SWI-Prolog translations of the *Trans* axioms of Definition 5.3.1. The predicate *trans_sort_h(H, SUCC, SUCCS)* finds for each transition (configuration) in *SUCC* a relaxed execution using a SWI-Prolog implementation of Do_R (Definition 6.5.2) and horizon H, and sorts *SUCC* in ascending order of the f_e evaluation of their relaxed execution.

In the final line, the first element of *SUCCS* is selected as the next transition. If this selection does not lead to an execution of the program *DP* in *S* (using a SWI-Prolog implementation of Do_H), SWI-Prolog will backtrack to this choice point and select the next transition in the list *SUCCS* in its place. By selecting transitions in ascending order of the f_e evaluation of their relaxed executions, the semantics of $Trans_H$ (above) are captured.

A procedure call $\beta(\vec{x})$ of **proc** $\beta(\vec{x})$ δ **end** is transitioned by replacing it with δ and transitioning δ using the axioms above.

Remark 6.6.1 *The heuristic interpreter characterised in this section is not optimal.*

Proof: The reason is two fold. Executions found by my relaxed interpreter are not necessarily the best or even possible. The heuristic values that are assigned to the configurations that my heuristic interpreter can choose from do not necessarily represent which choices will definitely lead to the best legal executions. Once a choice is made, this choice is not altered (the SWI-Prolog implementation of the interpreter backtracks and alters choice selections only if it arrives at a dead-end – a configuration from which no transitions are possible) even if an unexplored choice previously discovered has a more promising heuristic value that those the interpreter currently has to choose from.

□

I have conducted a series of experiments to assess the performance of this heuristic interpreter across a range of domains. I present a detailed description of these domain examples and an analysis of the obtained results in Chapter 7.

6.7 Concluding Remarks and Future Work

In this chapter I have described a heuristic GOLOG interpreter that is able to find (near) optimal executions of a GOLOG program given a numeric evaluation function. For this purpose I have employed relaxation-based heuristics – heuristics that have been successfully used in classical planning [Bonet and Geffner (2001); Hoffman and Nebel (2001)].

These classical planners have inspired the development of a relaxed GOLOG interpreter, used to simulate the interpretation of a GOLOG program as a means of evaluating available choices in the search for an execution. This simulation relies on a relaxed variant of regression (relaxed regression) that I have presented in Section 6.3. This relaxed interpreter aims to discover a good execution of a program by employing a simple greedy heuristic when deciding which transitions to enact. My heuristic interpreter applies its evaluation function to the relaxed executions resulting from each available choice, associating with them a heuristic value. These values allow this heuristic interpreter to pursue paths that have been identified as leading to the best relaxed executions.

6.7.1 Online versus Offline Interpretation

In this chapter I have presented my heuristic interpreter in an offline setting. A GOLOG interpreter finds an execution of a program offline if it suspends action until a complete execution of the program has been found. In Section 5.5, I describe the INDIGOLOG interpreter, an extension of GOLOG that supports both online and offline interpretation. In the online interpretation of a GOLOG program, an agent performs actions as they are added to its execution – allowing agents to perform sensing actions, the results of which are used to guide interpretation of the remainder of the program. In this section I consider how my heuristic interpreter could be extended for use in an online setting.

Whether an agent performs actions as they are suggested by this heuristic interpreter or waits until a complete execution of a program is found before performing actions is of no consequence. There is no reason why this heuristic interpreter could not be used in the former sense. If, however, the actions being performed by an agent are sensing actions, the heuristic interpreter must be extended with the ability to handle them. The relaxed interpretation of a program – used to determine which configurations to transition to in the search for a program execution – is conducted offline. The relaxed interpreter does not have access to the results of a sensing action if it has not been performed.

Within the presented relaxed interpreter, we can view sensing actions as being choice constructs, where each possible sensing result is a choice. From this view, the interpreter can then select the sensing result it views as the most beneficial in terms of finding a desirable execution - it assumes the best case scenario - just as it would a choice between actions. As a finite action horizon h is used during relaxed interpretation, the heuristic interpreter can be used even with non-terminating programs (in an online setting). The selection of a path to pursue is made on the basis of which path, h steps in the future, leads to the most attractive partial execution (according to an evaluation function f_e).

I have not implemented an online variant of my heuristic interpreter, but leave it as a subject of future work. In the next chapter, I present an experimental evaluation of the heuristic GOLOG interpreter whose syntax and semantics have been described in this chapter. I discuss at the conclusion of this next chapter a range of ideas for the future improvement and development of this heuristic GOLOG interpreter.

Chapter 7
Case Studies

IN this chapter, I conduct a series of experiments, across a range of domains, to assess the performance of the heuristic interpreter I have developed and presented in Sections 6.4 and 6.6. These experiments assess the merit of my interpreter – the benefit of its lookahead-based heuristic when compared to the unguided standard interpreter of Section 5.3.1, and an equivalent heuristic interpreter without the lookahead. These experiments determine if this heuristic interpreter discovers executions of a higher quality than one unguided by a heuristic, and one guided by a heuristic without a lookahead.

Three domains, each based on a real-world application, are presented in this chapter. I first consider an example presented by Smith et al. (2000) in which a spacecraft is required to make scientific observations of celestial targets during a limited time window (while maximising the number of targets that are observed, as described in Section 7.1). My second domain considers a GOLOG program for the control of an iron ore mining agent (Section 7.2). The goal of this agent is to select blocks of ore at a mine site to excavate and place on a stockpile (while maintaining a desired stockpile quality). In my third domain, GOLOG is employed to control a task scheduling agent (Section 7.3). A small set of workers, with a varying array of skills, is required to complete a series of jobs – each job consisting of a set of tasks to be performed. Jobs must be scheduled, and tasks assigned to workers, while maximising worker utilisation (task-to-worker assignments).

Within this chapter, I characterise these domains as follows. I first describe the controlling GOLOG program that requires interpretation. The evaluation functions f_e and f_p

(Definitions 6.4.1 and 6.5.1) that describe which executions are more desirable than others are then presented. I then express the dynamics of the domain in terms of action precondition and successor state axioms. In addition, I describe which computations are relaxed during the relaxed regression of action preconditions. A range of test cases are then provided, each test case varying the configurable properties of the domain. The outcomes of the heuristic, standard, and non-lookahead-based interpreters, on this collection of tests, are then analysed. Each interpreter under examination is given 15 minutes to find an execution in each test – if it is unable to do so, it is said to be unable to solve the test.

The non-lookahead-based interpreter that I consider in this chapter has the same semantics as the relaxed interpreter presented in Section 6.5 of Chapter 6. The exception is that no action preconditions are relaxed during interpretation. In all other respects, the two interpreters have the same implementation. An evaluation function f_p (Definition 6.5.1 in Section 6.5) is used to select which configurations to transition to in the search for an execution, just as is done within my relaxed interpreter. I call this interpreter the 'greedy interpreter' throughout this chapter (in light of its greedy heuristic).

The lookahead heuristic I employ in the heuristic interpreter of Chapter 6 is relaxation-based. I investigate, in this chapter, whether performing an actual lookahead (heuristically evaluating choices on the basis of legal rather than relaxed executions) is practical. Moreover, I consider the time-savings achieved (if any) through the use of relaxation, and whether a non-relaxed lookahead results in the discovery of higher quality executions. In addition, I determine if my decision to relax only action precondition tests during interpretation yields a better performance than relaxing all tests (within test, if ... else, and while loop constructs). To conclude this evaluation, the feasibility of features present in the existing body of work on preference-based GOLOG is examined. To this end, I determine if the maintenance of unexplored choices on an explicit search frontier (a feature of the best-first search interpreter of Sohrabi et al. (2006), described in Section 6.1.3) and the exploration of entire execution spaces (characteristic of the DTGOLOG-based approaches of Section 6.1.1) is practical across the test domains described in this chapter.

Each of the test domains I consider in this chapter is designed to represent a scenario in which the decisions made by a GOLOG interpreter have an impact on the quality of

the executions it discovers. I conclude this chapter by discussing the limitations of my heuristic interpreter. I consider the range of scenarios in which I expect it to perform no better than a standard interpreter. In addition, I look at the time overhead required in the computation of its lookahead-based heuristic, and suggest a range of strategies by which this overhead can be minimised.

I begin with a description of the spacecraft control domain, and an analysis of the experimentation conducted within it.

7.1 Spacecraft Control

Consider a spacecraft designed to observe celestial targets (objects in space – located in clusters within a 3D space), during a limited time window [Smith et al. (2000)]. This spacecraft has a camera capable of taking images of targets – each observation requiring the craft to turn its camera to focus on the location of the target (consuming an amount of fuel and time dependent on the magnitude of the turn), and taking an image (consuming power). Fuel is limited, while power is renewable at a constant rate[1]. The craft's camera must be calibrated prior to its first observation. Calibration requires the craft to turn to, and focus its camera on, a calibration target (a celestial target with known properties). An execution of the GOLOG program SPACE forms a plan for the spacecraft.

Listing 7.1.1 Controlling GOLOG program for a spacecraft.

```
proc SPACE
    switchon :
    π t1. [ CalTarget(t1)? : turnto(t1) : calibrate(t1) ] :
    while ∃t. (Target(t) ∧ ¬Observed(t)) ∧ TimeLeft do
        (π tg. [(Target(tg) ∧ ¬Observed(tg))? : turnto(tg) : takeimage(tg) ] | wait)
    end :
    switchoff
end
```

In the above procedure, the craft selects and observes targets until the available time has elapsed. If it is not possible to observe a target (due to limited resources) the craft waits for one unit of time before checking whether a new target can be observed. The

[1] This is a simplification of the example presented by Smith et al. (2000) in which the power renewal rate is dependent on the orientation of the craft with respect to the sun.

fluents and primitive actions that appear within this procedure are defined in Table 7.1. The fluents within the above procedure have their situation argument suppressed. This argument is restored in Table 7.1 for each situation dependent fluent.

Target(t)	Holds if t is a celestial target.
Observed(t, s)	Holds if target t has been observed in situation s.
TimeLeft(s)	Holds if there is ≥ 1 unit of time left in s.
CalTarget(t)	Holds if t is a calibration target.
switchon	Turn the camera instrument of the spacecraft on.
calibrate(t)	Calibrate this instrument with respect to the target t.
turnto(t)	Turn the spacecraft instrument to the target t.
takeimage(t)	Take an image of the target t.
wait	Wait for one unit of time.
switchoff	Turn the spacecraft instrument off.

Table 7.1: Fluents and primitive actions within the spacecraft control domain.

In the successor state and action precondition axioms of this domain, described in Sections 7.1.2 and 7.1.3, a range of additional fluents are used, shown in Table 7.2.

TimeOk(tm, s)	At least tm amount of time remains in situation s.
PowerOk(p, s)	At least p amount of power remains in situation s.
FuelOk(f, s)	At least f amount of fuel remains in situation s.
TimeUsed(a, t_u, s)	Action a consumes t_u units of time in situation s.
FuelUsed(a, f_u, s)	Action a consumes f_u units of fuel in situation s.
PowerUsed(a, p_u, s)	Action a consumes p_u units of power in situation s.
On(s)	The craft's camera is on in situation s.
Pointing(lt, s)	This camera is focused on target lt in situation s.
TimeSpent(tm, s)	The actions in situation s have consumed tm time.
FuelSpent(f, s)	The actions in situation s have consumed f fuel.
PowerSpent(p, s)	The actions in situation s have consumed p power.

Table 7.2: Additional fluents within the spacecraft control domain.

The executions of Listing 7.1.1 that observe more targets are preferred to those that observe less. Finding a sequence of celestial targets to observe that allows the most observations to be made within a limited time window is a difficult problem – as the number of possible observation sequences will typically be too abundant to enumerate.

7.1.1 Evaluation Function

The evaluation function that the heuristic interpreter of Chapter 6 employs to find an execution of the program SPACE in Listing 7.1.1 is $f_e(\delta, s) = -num\_obs(s)$, where $num\_obs(s)$ denotes the number of observations made by the spacecraft in execution s. As a result, one execution is preferred to another if it makes a greater number of target observations.

The evaluation function employed by my relaxed and greedy interpreters encourages the selection of targets for observation that require the least degree of movement of the craft's instrument (in the hope of allowing more targets to be observed). This function is defined as $f_p(\delta, s) = distance(next(\delta, s), s)$, where $next(\delta, s)$ is the target next selected for observation in configuration (δ, s) and $distance(t, s)$ denotes the degree of movement required to move to target t from the craft's current focus point. If, in configuration (δ, s), the next action is not to move to a target, the function $distance(next(\delta, s), s)$ returns 360° – the maximum degree of movement possible by the camera of the craft. Recall from Section 6.5.1 the distinction between the functions f_e and f_p.

7.1.2 Action Precondition Axioms

The precondition axioms for the actions of this domain, shown in Table 7.1, are outlined in Definition 7.1.1. The fluents used in these axioms are defined in Tables 7.1 and 7.2.

Definition 7.1.1 In the following collection of axioms lt, ct, s, t_u, f_u, and p_u, respectively denote: a celestial target; a calibration target; a situation; and an amount of time, fuel, and power. Recall the form of an action precondition axiom from Equation 5.1 of Section 5.1.3. The right hand side of each of these axioms must be uniform in the situation s.

$$Poss(switchon, s) \equiv \neg On(s) \tag{7.1}$$

$$Poss(switchoff, s) \equiv On(s) \tag{7.2}$$

$$\begin{aligned} Poss(turnto(lt), s) \equiv \neg Pointing(lt, s) \wedge \exists t_u.TimeUsed(turnto(lt), t_u, s) \\ \wedge\ TimeOk(t_u, s) \wedge \exists f_u.FuelUsed(turnto(lt), f_u, s) \wedge FuelOk(f_u, s) \end{aligned} \tag{7.3}$$

$$\begin{aligned} Poss(takeimage(lt), s) \equiv Pointing(lt, s) \wedge \exists t_u.TimeUsed(takeimage(lt), t_u, s) \\ \wedge\ TimeOk(t_u, s) \wedge \exists p_u.PowerUsed(takeimage(lt), p_u, s) \wedge PowerOk(p_u, s) \end{aligned} \tag{7.4}$$

$$Poss(calibrate(ct), s) \equiv Pointing(ct, s) \wedge \exists t_u.TimeUsed(calibrate(ct), t_u, s)$$
$$\wedge\ TimeOk(t_u, s) \wedge \exists p_u.PowerUsed(calibrate(ct), p_u, s) \wedge PowerOk(p_u, s) \quad (7.5)$$

$$Poss(wait, s) \equiv \exists t_u.TimeUsed(wait, t_u, s) \wedge TimeOk(t_u, s) \quad (7.6)$$

Consider the precondition axiom shown in Equation 7.3 of Definition 7.1.1. This axiom describes the preconditions of turning the craft's instrument to focus on a target lt. It can only do so if it is not already focused upon it, and it has sufficient resources (time and fuel). To determine if it has sufficient time, it computes the time required to make the turn (t_u) and then checks whether there is at least t_u units of time left in the given situation s. A number of other axioms in Definition 7.1.1 follow this same pattern. This pattern is used to ensure that the right hand side of each axiom is uniform in situation s.

7.1.3 Successor State Axioms

The successor state axioms for the fluents within this domain (Table 7.1 and 7.2) are shown in Definition 7.1.2. In the initial situation, no targets have been observed, the craft's instrument is not on, and the craft is focused on a target as defined in each test case generated within the domain (Section 7.1.4). In Definition 7.1.2, I provide the successor state axiom for the fluent $TimeSpent$, but refrain from presenting those for $FuelSpent$ and $PowerSpent$ (these are defined similarly). In addition, I do not provide axioms for the fluents $TimeUsed$, $FuelUsed$ and $PowerUsed$. The time, power, and fuel used by each action is defined in Section 7.1.4 in which the tests of the domain are described. The power used by an action is offset by the amount renewed through the solar panels of the craft.

Definition 7.1.2 In the following collection of axioms, the predicates $Time(q)$, $Fuel(q)$ and $Power(q)$, denote the time, fuel, and power allowances available to the spacecraft. Recall the definition of a successor state axiom for a relational fluent in Equation 5.2 of Section 5.1.4. The right hand side of each axiom must be uniform in the situation s.

$$Observed(lt, do(a, s)) \equiv a = takeimage(lt) \vee Observed(lt, s) \quad (7.7)$$

$$Pointing(lt, do(a, s)) \equiv a = turnto(lt) \vee Pointing(lt, s)$$
$$\wedge \neg\exists t.\ t \neq lt \wedge a = turnto(t) \quad (7.8)$$

$$TimeLeft(do(a,s)) \equiv \exists t, t_u, q.\ TimeSpent(t,s) \wedge TimeUsed(a,t_u,s) \wedge Time(q) \wedge (q-(t+t_u)) \geq 1 \quad (7.9)$$

$$On(do(a,s)) \equiv a = switchon \vee On(s) \wedge a \neq switchoff \quad (7.10)$$

$$TimeOk(t', do(a,s)) \equiv \neg[\exists t, t_u, q.\ TimeSpent(t,s) \wedge TimeUsed(a,t_u,s) \wedge Time(q) \ \wedge \ q < t + t_u + t'] \quad (7.11)$$

$$PowerOk(p', do(a,s)) \equiv \neg[\exists p, p_u, q.\ PowerSpent(p,s) \wedge PowerUsed(a,p_u,s) \wedge Power(q) \ \wedge \ q < p + p_u + p'] \quad (7.12)$$

$$FuelOk(f', do(a,s)) \equiv \neg[\exists f, f_u, q.\ FuelSpent(f,s) \wedge FuelUsed(a,f_u,s) \wedge Fuel(q) \ \wedge \ q < f + f_u + f'] \quad (7.13)$$

$$TimeSpent(t, do(a,s)) \equiv \exists t_u, t'.\ TimeSpent(t',s) \wedge TimeUsed(a,t_u,s) \wedge t = t' + t_u \quad (7.14)$$

Consider the successor state axiom shown in Equation 7.11 of Definition 7.1.2. There are at least t' units of time available in the situation $do(a,s)$ if it is not the case that the sum of: the time used by action a in s (t_u), the cumulative time spent by prior actions in situation s (t), and the time t', exceeds the time quota available. These successor state axioms have been designed so that the relaxation of action preconditions removes the need to check for adequate resources and time to turn to and take images of targets. Recall from Definition 6.3.2 of Section 6.3.1 that the relaxed regression rule for a relational fluent $F(\vec{t}, do(\alpha,s))$ considers its successor state axiom (Equation 5.2 in Section 5.1.4):

$$F(\vec{x}, do(a,\sigma)) \equiv \gamma^+(\vec{x},a,\sigma) \vee (F(\vec{x},\sigma) \wedge \neg\gamma^-(\vec{x},a,\sigma))$$

and ignores the $\neg\gamma^-(\vec{x},a,\sigma)$ component. The reliance of my heuristic interpreter on an appropriate domain design to ensure that the relaxed interpretation of a program requires less time than an actual interpretation is discussed in Section 7.5.

7.1.4 Test Case Formulation and Results

In this domain, a set of test cases have been generated with a specific number of celestial and calibration targets, fuel, power, and time limits (Table 7.3 and A.1). Table A.1 in

N_T	TIME	FUEL	POWER	O_S	O_G	O_H
10	150	500	200	2	2	2
10	150	500	200	0	0	2
10	150	500	200	6	6	6
10	150	500	200	2	2	2
10	150	500	200	2	2	3
10	150	500	200	2	2	2
10	150	500	200	4	4	4
10	150	500	200	2	2	2
10	150	500	200	2	2	2
10	150	500	200	0	0	4
20	250	600	300	3	3	9
20	250	600	300	2	8	8
20	250	600	300	4	6	12
20	250	600	300	5	10	11
20	250	600	300	3	3	8
20	250	600	300	4	7	10
20	250	600	300	2	7	8
20	250	600	300	3	4	6
20	250	600	300	3	4	8
20	250	600	300	2	5	11
30	350	700	400	5	8	18
30	350	700	400	5	13	19

Table 7.3: Observations made in executions found by the standard (O_S), greedy (O_G), and heuristic (O_H) interpreters. In each test, N_T denotes the number of celestial targets.

Appendix A provides the results and descriptions of the full set of test cases, while Table 7.3 shows a subset (the first 22/50 test cases). The locations of the target clusters, and initial focus point of the spacecraft instrument, have been randomly generated in each test – each coordinate (x, y, and z) assigned a random integer between -100 and 100. The position of each target within a cluster has been randomly generated to lie within a small deviation of the cluster location. The action horizon in each test has been set to the number of time steps available. In each of the tests of Table 7.3 and A.1, the rate of power renewal is 2% of its starting power/unit of time, while 1 unit of time and 2 units of fuel are used per degree turned, 5 units of power are used during the takeimage(t) and calibrate(t) actions, and there are 5 calibration targets (with randomly defined locations).

Table 7.3 and A.1 demonstrate that in 43/50 of the tests, the heuristic interpreter found executions that made more observations than those found by the standard interpreter.

N_T	$\overline{O}_S$	$\overline{O}_G$	$\overline{O}_H$	$\overline{T}_S$ (s)	$\overline{T}_G$ (s)	$\overline{T}_H$ (s)
10	2.2 (1.75)	2.2 (1.75)	2.9 (1.37)	0.05 (0.05)	0.50 (0.49)	0.36 (0.36)
20	3.1 (0.99)	5.7 (2.31)	9.1 (1.85)	0.14 (0.12)	1.59 (1.31)	1.38 (0.58)
30	6.2 (1.87)	11 (2.75)	16.7 (1.70)	0.08 (0.09)	4.54 (3.40)	7.35 (1.58)
40	10.5 (4.35)	19.8 (3.16)	22.9 (1.60)	0.14 (0.08)	5.97 (4.45)	24.99 (3.67)
50	10 (2.71)	25.7 (4.50)	31.7 (1.83)	0.92 (0.29)	13.71 (6.76)	80.89 (8.02)

Table 7.4: Mean number of observations made in executions found by the standard ($\overline{O}_S$), greedy ($\overline{O}_G$), and heuristic ($\overline{O}_H$) interpreters (sample standard deviation in brackets) for tests with varying N_T (number of targets). $\overline{T}_S$, $\overline{T}_G$, and $\overline{T}_H$ denote the mean time used by the interpreters in finding a solution (sample standard deviation in brackets).

On the remaining 7 tests, the two interpreters found executions with equal observation counts. In 37/50 of the tests, the heuristic interpreter found executions that made more observations than those discovered by the greedy interpreter. On the remaining 13 tests, the two interpreters found executions with equal observation counts.

Each experiment conducted within this thesis has been run on a Windows XP (Service Pack 3) machine with a 2.40 GHZ Intel(R) Core(TM)2 CPU and 2GB of RAM. The average time required by the standard, greedy, and heuristic interpreters to find a solution in the collection of tests created for each number of targets (N_T) is highlighted in Table 7.4. Also stated is the average number of targets observed within each collection of tests. In brackets following each average is the (sample) standard deviation of the associated data set. As the number of targets increases, it is clear that the heuristic interpreter consumes more time than both the greedy and standard interpreters. The heuristic interpreter, however, consistently finds executions that observe, on average, more targets.

7.2 Mine Operations

Consider a mine that consists of a series of iron ore blocks (geometric regions) located underneath the surface of the mine landscape. A mining robot is designed to travel to a block, blast the block, collect the ore, and transfer the ore to a stockpile. Each block has a percentage amount of iron, silica, alumina, and phosphorus. Some blocks are dependent on others, and cannot be mined until these other blocks are mined first. This may be because they are not exposed until certain blocks are excavated. A mining robot can only travel between a block and the stockpile if there is a safe (non-hazardous) path between

the two. Hazards are defined as regions that the robot cannot pass through, and are assumed, for the purposes of this example, to be static (fixed). A plan for the robot miner is generated by finding an execution of the GOLOG program MINEOP, shown below.

Listing 7.2.1 Controlling GOLOG program for a mining robot.

```
proc MINEOP
    powerup :
    while StockRequired do
        π block. [ Available(block)? :
            moveto(block) : blast(block) : mine(block) :
            moveto(stockpile) : combine(block) ]
    end :
    powerdown
end
```

This mining robot is designed to build a stockpile of a desired tonnage and quality target, by mining an appropriate sequence of blocks. This target defines a bound on the percentages of iron, silica, alumina, and phosphorus in the stockpile. The fluents and primitive actions that appear within this procedure are defined in Table 7.5, with the situation argument of each situation dependent fluent in Listing 7.2.1 restored.

In the successor state and action precondition axioms of this domain, described in Sections 7.2.2 and 7.2.3, a range of additional fluents are used, shown in Table 7.6.

StockRequired(*s*)	Holds if the stockpile requires more ore in situation *s*.
Available(*block*, *s*)	Holds if *block* is a block that is available for mining (it has not been previously mined or blasted, and all blocks it depends on have been mined) in situation *s*.
powerup	Readies the robot for action.
powerdown	The robot enters a sleep mode.
moveto(*object*)	The robot travels to the given *object*.
blast(*block*)	The robot blasts the given *block*.
mine(*block*)	The robot collects the ore in the given *block*.
combine(*block*)	The robot adds the given ore *block* to the stockpile.

Table 7.5: Fluents and primitive actions within the mine operations domain.

The executions of Listing 7.2.1 that create stockpiles closer in composition to the desired target are preferred. The discovery of a block sequence for mining that creates a stockpile whose composition is close to the desired target is, like the problem of max-

Mined(*block*, *s*)	Holds if the given *block* has been mined in situation *s*.
Down(*s*)	Holds if the robot is in a rest state in situation *s*.
OnBody(*block*, *s*)	Holds if the robot is carrying the ore mined from the given *block* in situation *s*.
SafePath(o_1, o_2, *s*)	Holds if no hazards lie between any two points on a path between the objects o_1 and o_2.
Collected(*c*, *s*)	Defines the stockpile tonnage *c* in situation *s*.
Added(*am*, *a*, *s*)	Defines the amount of ore *am* added to the stockpile by action *a* in situation *s*.
At(*object*, *s*)	Holds if the robot is at the given *object* in situation *s*.
Blasted(*block*, *s*)	Holds if the given *block* has been blasted in *s*.

Table 7.6: Additional fluents within the mine operations domain.

imising target observations in the spacecraft control domain of Section 7.1, a difficult problem – with many different subsets of blocks available for selection.

7.2.1 Evaluation Function

The evaluation function my heuristic interpreter uses to find an execution of the program MINEOP in Listing 7.2.1 is denoted $f_e(\delta, s) = distance(s, target)$, where $distance(s, target)$ defines how far away from the desired *target* the stockpile being built by the robot is in situation s. One execution of the program MINEOP is preferred to another if the former results in a closer-to-target stockpile. The function $distance(s, target)$ is defined as:

$$distance(s, target) = \sum_i \frac{|c(i, s) - t(i, target)|}{e(i)} \tag{7.15}$$

where: i ranges over the components iron, silica, alumina and phosphorus; $c(i, s)$ is the percentage amount of component i in the mine stockpile in situation s; $t(i, target)$ is the target percentage of the component i; and $e(i)$ the acceptable $\pm$ deviation from $t(i, target)$.

The relaxed and greedy interpreters encourage the mining of blocks that are closest in composition to this target. Let $f_p(\delta, s) = deviation(next(\delta, s), target)$, where $next(\delta, s)$ is the next block the robot will move to in configuration (δ, s), and *deviation* is the distance between its composition and the target of the stockpile (defined as in Equation 7.15). If, in configuration (δ, s), the robot has not selected a block to move to, $f_p(\delta, s) = 0$.

7.2.2 Action Precondition Axioms

The precondition axioms for the actions of this domain, shown in Table 7.5, are outlined in Definition 7.2.1. The fluents used in these axioms are defined in Tables 7.5 and 7.6.

Definition 7.2.1 In the following: *block* is a block of iron ore; s is a situation; and *object* is a block or the stockpile. Recall the form of an action precondition axiom from Equation 5.1 of Section 5.1.3. The right hand side of each axiom is uniform in the situation s.

$$Poss(powerup, s) \equiv Down(s) \tag{7.16}$$

$$Poss(powerdown, s) \equiv \neg Down(s) \tag{7.17}$$

$$Poss(blast(block), s) \equiv At(block, s) \wedge Available(block, s) \tag{7.18}$$

$$Poss(combine(block), s) \equiv At(stockpile, s) \wedge OnBody(block, s) \tag{7.19}$$

$$Poss(mine(block), s) \equiv At(block, s) \wedge \neg Mined(block, s) \wedge Blasted(block, s) \tag{7.20}$$

$$Poss(moveto(object), s) \equiv \exists o.\ At(o, s) \wedge SafePath(o, object, s) \tag{7.21}$$

Consider the precondition axiom shown in Equation 7.21 of Definition 7.2.1. This axiom describes the preconditions of moving the robot to a given object. It can only do so if there exists a safe path between its current location and the location of the object.

7.2.3 Successor State Axioms

The successor state axioms for the fluents within this domain (Tables 7.5 and 7.6) are shown in Definition 7.2.2. In the initial situation, the stockpile is empty (no blocks have been mined and no ore collected), the robot is at the stockpile and in a rest state. The blocks that are initially available and the locations of hazards are defined within the generated test cases of the domain (these tests cases are described in Section 7.2.4).

Definition 7.2.2 In the following, the rigid predicate: $Route(o_1, o_2, r)$ denotes that r is a route between objects o_1 and o_2; $Segment(p_1, p_2, r)$ that p_1 and p_2 are locations that define a straight-line segment of route r; $Danger(p_1, p_2)$ that the segment between locations p_1 and p_2 passes through a hazard; $Hazard(r)$ that there is a hazard on route r; $Required(ton)$

that *ton* tonnes of ore is required; $Amount(b,\ t)$ that t is the amount of ore in block b; $Dependent(b_1, b_2)$ that block b_1 can only be mined after b_2; and $Block(b)$ that b is a block.

Recall the definition of a successor state axiom for a relational fluent in Equation 5.2 of Section 5.1.4. The right hand side of each axiom must be uniform in the situation s.

$$\begin{aligned} &StockRequired(do(a,s)) \equiv \exists am, c.\ Added(am,a,s) \wedge Collected(c,s) \\ &\qquad \wedge \exists r.\ Required(r) \wedge\ c + am < r \end{aligned} \tag{7.22}$$

$$\begin{aligned} &Collected(c,do(a,s)) \equiv \exists am.\ Collected(am,s) \wedge \{[\exists b,t.\ a = combine(b) \\ &\qquad \wedge Amount(b,t) \wedge c = am + t] \vee [(\neg\exists b'.\ a = combine(b')) \wedge c = am]\} \end{aligned} \tag{7.23}$$

$$\begin{aligned} &Available(b,do(a,s)) \equiv Block(b) \wedge \neg(a = blast(b) \vee Blasted(b,s)) \\ &\qquad \wedge\ \neg[\exists b'.\ Dependent(b,b') \wedge \neg(a = mine(b') \vee Mined(b',s))] \end{aligned} \tag{7.24}$$

$$Mined(b,do(a,s)) \equiv a = mine(b) \vee Mined(b,s) \tag{7.25}$$

$$Blasted(b,do(a,s)) \equiv a = blast(b) \vee Blasted(b,s) \tag{7.26}$$

$$Down(do(a,s)) \equiv a = powerdown \vee (Down(s) \wedge a \neq powerup) \tag{7.27}$$

$$At(object,do(a,s)) \equiv a = moveto(object) \vee (At(object,s) \wedge \neg\exists o.\ a = moveto(o)) \tag{7.28}$$

$$OnBody(b,do(a,s)) \equiv a = mine(b) \vee (OnBody(b,s) \wedge a \neq combine(b)) \tag{7.29}$$

$$SafePath(o_1,o_2,s) \equiv \exists r.\ Route(o_1,o_2,r) \wedge \neg Hazard(r) \tag{7.30}$$

$$Hazard(route) \equiv \exists p_1,p_2.\ Segment(p_1,p_2,route) \wedge Danger(p_1,p_2) \tag{7.31}$$

$$\begin{aligned} &Added(am,a,s) \equiv [\exists b.a = combine(b) \wedge Amount(b,am)] \vee \\ &\qquad [\neg\exists b'.a = combine(b') \wedge am = 0] \end{aligned} \tag{7.32}$$

Consider the successor state axiom shown in Equation 7.23 of Definition 7.2.2. The amount of ore c the robot has collected in a situation $do(a,s)$ is the sum of the: the amount of ore collected in situation s (am), and any ore placed on the stockpile by action a (t). These successor state axioms have been designed so that the relaxation of action preconditions within this domain removes the need to check for the presence of hazards along the paths between blocks and the stockpile – a potentially costly operation.

N_B	R_T (tonnes)	N_H	D_S	D_G	D_H
5	1500	5	6.84	3.38	3.38
5	1500	5	4.32	7.16	4.32
5	1500	5	2.83	9.90	2.83
5	1500	5	6.97	6.12	2.55
5	1500	5	10.15	7.29	6.04
5	1500	5	7.20	7.20	7.20
10	2500	5	2.09	1.99	1.15
10	2500	5	3.27	5.90	0.75
10	2500	5	7.46	3.73	3.73
10	2500	5	6.14	5.51	3.89
10	2500	5	9.56	2.32	1.53
10	2500	5	5.85	6.22	4.01
15	3500	5	6.67	5.19	3.94
15	3500	5	2.75	2.03	0.76
15	3500	5	6.14	5.46	6.08
15	3500	5	4.29	1.78	1.70
15	3500	5	6.27	4.00	3.11
15	3500	5	7.21	8.72	3.38
20	4500	10	3.27	2.01	0.83
20	4500	10	1.85	1.43	2.11
20	4500	10	3.14	5.32	0.93
20	4500	10	3.13	4.66	1.03
20	4500	10	3.99	3.80	2.06
20	4500	10	1.83	2.62	0.53
25	5500	10	2.60	2.83	2.47
25	5500	10	1.19	1.92	0.83
25	5500	10	2.62	4.09	0.86

Table 7.7: Achieved stockpile distance (from target) in executions found by the standard (D_S), greedy (D_G), and heuristic (D_H) interpreters. N_B, R_T, and N_H denote the number of blocks, required stockpile tonnage, and number of hazards in each test.

7.2.4 Test Case Formulation and Results

In this domain, a set of test cases have been generated, and are shown in Tables 7.7 and A.2. Each test is associated with a specific number of blocks (of equal tonnage) and hazards, located across a mine landscape. Table A.2 in Appendix A provides the results of the full set of test cases, while Table 7.7 shows a subset (the first 27/60 of the tests).

Within each test, each block and hazard is located at a coordinate in 2D space whose components are assigned a random integer between 0 and 500. The location of the stockpile is also randomly generated in this fashion. The set of blocks in the mine is divided

into groups, each group defining a chain of (dependent) blocks that can only be mined in sequence. The stockpile target (desired percentages of iron, silica, alumina and phosphorus in the resulting stockpile) is randomly configured, while the composition of each block is determined by adding or subtracting a random deviation from the stockpile target. Each test has a specific number of hazards located across the mine landscape, their locations randomly generated in the same manner as the block and stockpile coordinates. Each hazard is a 5×5 region through which the mining robot cannot pass. Between each block in the landscape and the stockpile there are three distinct routes. These routes are composed of small segments that map out a path from a block to the stockpile. The action horizon chosen in each test is $5\times$ the number of blocks defined within it.

Tables 7.7 and A.2 demonstrate that in 56/60 of the generated tests, my heuristic interpreter found executions that led to a closer-to-target stockpile than those found by the standard interpreter. In 3/60 of the tests, the two interpreters found executions that led to stockpiles equidistant from the target. In 1/60 of the tests, the heuristic interpreter found an execution that led to a stockpile that was further away from the target than that found by the standard. In 52/60 of these tests, the heuristic interpreter discovered executions that led to a closer-to-target stockpile than those found by the greedy interpreter. In 5/60 of the tests, the heuristic and greedy interpreters found executions that led to stockpiles that were equidistant from the target. In 3/60 of the tests, the greedy interpreter found an execution that resulted in a higher quality stockpile than that found by the heuristic interpreter. Table A.2 defines the groups of dependent blocks in each test together with the time required for each test case to complete under each of the three interpreters.

As in the spacecraft control domain (refer to Section 7.1), my heuristic interpreter yields better performance than the standard and greedy interpreters in a majority of the test cases within this domain. In this mining domain, the heuristic interpreter produces a lower quality execution in a small number of the tests relative to the two alternative interpreters. This is not wholly unexpected as its heuristic estimates are derived from relaxed executions that may not necessarily represent legal action sequences.

The average time required by the standard, greedy, and heuristic interpreters to find a solution in the collection of tests created for each number of ore blocks (N_B) is highlighted

N_B	$\overline{D}_S$	$\overline{D}_G$	$\overline{D}_H$	$\overline{T}_S$ (s)	$\overline{T}_G$ (s)	$\overline{T}_H$ (s)
5	6.39 (2.54)	6.84 (2.11)	4.39 (1.87)	0.05 (0.02)	0.08 (0.01)	0.07 (0.02)
10	5.73 (2.73)	4.28 (1.86)	2.51 (1.52)	0.09 (0.03)	0.11 (0.03)	0.13 (0.03)
15	5.56 (1.69)	4.53 (2.57)	3.16 (1.85)	0.13 (0.01)	0.13 (0.03)	0.33 (0.03)
20	2.87 (0.86)	3.31 (1.54)	1.25 (0.67)	0.29 (0.08)	0.33 (0.09)	0.84 (0.10)
25	3.83 (2.00)	2.07 (1.28)	1.11 (0.68)	0.42 (0.15)	0.41 (0.17)	1.74 (0.13)
30	2.94 (1.06)	3.01 (1.26)	1.15 (0.66)	0.89 (0.26)	0.94 (0.26)	3.83 (0.32)
35	1.57 (0.49)	2.81 (0.78)	0.68 (0.33)	0.96 (0.23)	1.10 (0.26)	6.23 (0.42)
40	3.01 (1.56)	2.77 (1.65)	0.84 (0.46)	0.98 (0.34)	1.39 (0.31)	11.33 (0.83)
45	3.46 (0.96)	1.70 (0.75)	0.85 (0.48)	1.63 (0.30)	1.56 (0.45)	20.54 (0.95)
50	2.97 (1.44)	2.02 (0.97)	0.63 (0.24)	2.53 (0.75)	2.62 (0.83)	32.99 (1.38)

Table 7.8: Mean achieved distance between stockpile and target composition in executions found by the standard ($\overline{D}_S$), greedy ($\overline{D}_G$), and heuristic ($\overline{D}_H$) interpreters for test sets with varying N_B (number of ore blocks). $\overline{T}_S$, $\overline{T}_G$, and $\overline{T}_H$ denote the mean time used by each interpreter in finding a solution (sample standard deviation in brackets).

in Table 7.8. Also stated is the average achieved stockpile distance (from the desired target) for each collection of tests. In brackets following each average is the (sample) standard deviation of the associated data set. While it is clear that the heuristic interpreter requires, on average, more time to arrive at a solution – the average quality of its solutions is consistently higher than those of the alternative interpreters. Moreover, there is less variation in the results (stockpile-to-target composition distances) produced by the heuristic interpreter relative to those obtained by the standard and greedy interpreters.

7.3 Task Scheduling

Consider a set of workers W, where each worker $w_i \in W$ has a set of skills S_i. In this domain, a scheduling agent is designed to select a set of jobs for the workers to perform within a limited time window. Each job consumes one unit of time, and consists of a series of tasks to be performed by the workers. Each of these tasks requires a specific subset of skills in order complete it. Only one job can be scheduled at a time, and only if there is a unique worker able to perform each of its tasks (each worker can only perform one task per job). Jobs have dependencies such that some jobs can only be scheduled if other jobs have been scheduled to be performed before them. A plan for the scheduler is generated by finding an execution of the GOLOG program SCHEDULE, shown below.

Listing 7.3.1 Controlling GOLOG program for a task scheduler.

```
proc SCHEDULE
    while TimeLeft do
        π job. [ Available(job)? :
          { π task. [ (Task(task, job) ∧ ¬Assigned(task, job))? :
              π w. [Worker(w)? : assign(w, task, job)]] }* :
          Scheduled(job)? ]
    end
end
```

The fluents and primitive actions that appear within this procedure are defined in Table 7.9. The fluents within this procedure have their situation argument suppressed. This argument is restored in Table 7.9 for each situation dependent fluent. Fluents in the successor state and action precondition axioms of this domain are shown in Table 7.10.

TimeLeft(s)	Denotes that there is ≥ 1 unit of time left in situation s.
Available(job, s)	Holds if job is a job that is available for scheduling (no tasks within it have been assigned and all jobs it depends on have been scheduled) in situation s.
Task($task$, job)	Holds if the given $task$ is part of the given job.
Assigned($task$, job, s)	Holds if the given $task$ in the given job has been assigned to a worker in situation s.
Worker(w)	Holds if w is a worker.
Scheduled(job, s)	Holds if the given job has been scheduled in situation s. All of its tasks have been assigned to a worker.
assign(w, $task$, job)	The scheduler assigns the given $task$, from the given job, to the given worker w.

Table 7.9: Fluents and primitive actions within the task scheduling domain.

SkillOf($skill$, $task$)	Holds if the given $task$ requires the given $skill$.
HasSkill(w, $skill$)	Holds if the worker w has the given $skill$.
Unassigned(w, job, s)	Holds if the given worker w is not assigned to a task in the given job in situation s.
TimeUsed(a, t_u, s)	Action a consumes t_u units of time in situation s.
TimeSpent(tm, s)	The actions in situation s have consumed tm time.

Table 7.10: Additional fluents within the task scheduling domain.

The executions of Listing 7.3.1 that assign more tasks to workers, within the limited time window, are preferred to those that assign less. The discovery of a sequence of jobs

to schedule that assigns the most tasks to workers is a difficult problem – with many different scheduling sequences available for selection.

7.3.1 Evaluation Function

The evaluation function employed by my heuristic interpreter in this domain is defined as $f_e(\delta, s) = -tasks(s)$, where $tasks(s)$ denotes the number of tasks that have been assigned to workers in execution s. An execution of the program shown in Listing 7.3.1 is preferred to another if the former assigns more tasks to workers.

The relaxed and greedy interpreters select jobs to schedule that consist of the most tasks for assignment. Hence, $f_p(\delta, s) = -num\_tasks(next(\delta, s))$ where $next(\delta, s)$ denotes the job next selected for scheduling in configuration (δ, s), and $num\_tasks(job)$ denotes the number of tasks within the given job. If, in configuration (δ, s), a job has not just been selected for scheduling, the evaluation function $f_p(\delta, s) = 0$.

7.3.2 Action Precondition Axioms

The precondition axioms for the actions of this domain, shown in Table 7.9, are outlined in Definition 7.3.1. The fluents used in these axioms are defined in Tables 7.9 and 7.10.

Definition 7.3.1 The precondition axiom for the lone action in the task scheduling domain is shown below. Recall the form of an action precondition axiom from Equation 5.1 of Section 5.1.3. The right hand side of each of axiom is uniform in the situation s.

$$\begin{aligned} Poss(assign(w, task, job), s) \equiv\ & Unassigned(w, job, s) \\ & \wedge \neg\exists skill.[SkillOf(skill, task) \wedge \neg HasSkill(w, skill)] \end{aligned} \tag{7.33}$$

Consider the precondition axiom shown in Definition 7.3.1 for the action of task assignment. This axiom states that a worker w can be assigned to a task within a job only if w has not already be assigned to that job, and has the required skills to complete it.

7.3.3 Successor State Axioms

The successor state axioms for the fluents within this domain (Tables 7.9 and 7.10) are shown in Definition 7.3.2. I do not present a successor state axiom for the fluents *Time-*

$Spent(tm, s)$ and $TimeLeft(s)$ as they share the same form as their counterparts in the spacecraft control domain (see Equations 7.14 and 7.9 in Definition 7.1.2).

In the initial situation, no jobs have been scheduled and the set of available jobs (the number of tasks within them and the skills required to complete each) is configured differently within each test case presented in Section 7.3.4. The time used by each assignment action is 1 unit if it completes the scheduling of a job and 0 units otherwise.

Definition 7.3.2 In the following, the rigid predicate: $Job(j)$ denotes that j is a job; $Dependent(j_1, j_2)$ denotes that job j_1 is dependent on the scheduling of job j_2; and $Time(t)$ denotes that t is the time available (t number of jobs can be scheduled).

Recall the definition of a successor state axiom for a relational fluent in Equation 5.2 of Section 5.1.4. The right hand side of each axiom must be uniform in the situation s.

$$\begin{aligned} Available(job, do(a,s)) \equiv Job(job) \wedge \neg\exists w,t.(a = assign(w,t,job) \\ \vee Assigned(t,job,s)) \wedge \forall j,t'.\ [Dependent(job,j) \wedge Task(t',j) \rightarrow \\ \exists w'.\, a = assign(w',t',j) \vee Assigned(t',j,s)] \end{aligned} \tag{7.34}$$

$$\begin{aligned} Scheduled(job, do(a,s)) \equiv (\exists w,t.\, a = assign(w,t,job) \wedge \forall t'.[t \neq t' \\ \wedge Task(t',job) \rightarrow Assigned(t',job,s)]) \vee Scheduled(job,s) \end{aligned} \tag{7.35}$$

$$Assigned(t, job, do(a,s)) \equiv \exists w.\, a = assign(w,t,job) \vee Assigned(t,job,s) \tag{7.36}$$

$$Unassigned(w, job, do(a,s)) \equiv Unassigned(w,job,s) \wedge \neg\exists t.\ a = assign(w,t,job) \tag{7.37}$$

Consider the successor state axiom of Equation 7.34 in Definition 7.3.2. A job is available for scheduling in a situation $do(a, s)$ if action a does not schedule any task in the job, and no such tasks have been scheduled by situation s. Moreover, it must be the case that all jobs that the job in question is dependent on have been scheduled by s. These axioms have been designed so that the relaxation of action preconditions within this domain allows a worker to be assigned to a task without checking if they are assigned to another. In this domain, the testing of action preconditions is not time consuming, but it may be difficult to find a set of worker-to-task assignments such that: each worker has the skills to perform their task; and no worker is assigned to more than one task in a given job.

N_{JOBS}	N_W	T_{MAX}	TIME	NT_S	NT_G	NT_H
15	8	7	5	21	18	22
15	8	7	5	12	18	24
15	8	7	5	17	19	20
15	8	7	5	7	15	19
15	8	7	5	13	15	19
15	6	5	5	11	12	15
15	6	5	5	9	13	15
15	6	5	5	11	8	14
15	6	5	5	13	13	15
15	6	5	5	8	11	15
20	8	7	10	30	30	38
20	8	7	10	25	31	40
20	8	7	10	25	34	43
20	8	7	10	29	36	39
20	8	7	10	34	35	38
20	6	5	10	25	35	35
20	6	5	10	24	27	31
20	6	5	10	27	27	32
20	6	5	10	18	19	22
20	6	5	10	24	30	30
30	6	5	15	38	39	48
30	6	5	15	42	37	49

Table 7.11: Tasks assigned to workers in executions found by the standard (NT_S), greedy (NT_G), and heuristic (NT_H) interpreters. N_{JOBS}, N_W, T_{MAX}, and TIME are the number of jobs, workers, maximum number of tasks per job, and number of schedulable jobs.

7.3.4 Test Generation and Results

A set of test cases have been generated with a specific number of workers, jobs, and a maximum number of tasks per job (Tables 7.11 and A.3). Table A.3 in Appendix A provides the results of the full set of test cases, while Table 7.11 shows a subset (the first 22/40). The number of tasks in each job is a randomly generated number between 1 and the maximum number of tasks per job. The jobs are divided into dependent groups, each group defining a chain of jobs that can only be scheduled in sequence (with jobs later in the sequence more likely, but not necessarily, to have more tasks). Each worker is assigned a randomly selected subset of skills (from a set of 10), and each task is assigned a set of skills such that there are at least two workers able to complete it. The action horizon in each test is the number of jobs $\times$ the maximum number of tasks per job.

N_{JOBS}	N_W	T_{MAX}	$\overline{NT}_S$	$\overline{NT}_G$	$\overline{NT}_H$	$\overline{T}_S$ (s)	$\overline{T}_G$ (s)	$\overline{T}_H$ (s)
15	8	7	14 (5.29)	17 (1.87)	20.8 (2.17)	0.03 (0.04)	0.08 (0.13)	4.93 (4.40)
15	6	5	10.4 (1.95)	11.4 (2.07)	14.8 (0.45)	0.01 (0.01)	0.02 (0.00)	0.82 (0.40)
20	8	7	28.6 (3.78)	33.2 (2.59)	39.6 (2.07)	0.32 (0.63)	0.61 (1.17)	43.9 (72.07)
20	6	5	23.6 (3.36)	27.6 (5.81)	30 (4.85)	0.02 (0.01)	0.05 (0.01)	3.67 (0.59)
30	6	5	38.8 (1.92)	40.2 (2.39)	47 (2.00)	0.04 (0.03)	0.11 (0.06)	17.34 (3.62)
30	8	7	47 (11.22)	53.75 (7.59)	58.5 (8.58)	0.46 (0.87)	0.83 (1.15)	168.08 (236.01)
40	8	7	64 (8.29)	77 (3.74)	81.5 (5.20)	0.36 (0.59)	1.38 (1.49)	261.24 (304.54)
40	6	5	53.6 (2.88)	60.4 (3.85)	61.6 (3.46)	0.05 (0.03)	0.17 (0.05)	42.31 (13.74)

Table 7.12: Mean number of tasks assigned to workers in executions found by the standard ($\overline{NT}_S$), greedy ($\overline{NT}_G$), and heuristic ($\overline{NT}_H$) interpreters (sample standard deviation in brackets) for test collections with varying N_{JOBS}, N_W, and T_{MAX} (the number of jobs, workers, and maximum number of tasks per job). $\overline{T}_S$, $\overline{T}_G$, and $\overline{T}_H$ denote the mean time used by each interpreter in finding a solution (sample standard deviation in brackets).

In 38/40 of the tests, executions found by my heuristic interpreter resulted in more task assignments than those found by the standard interpreter. In 2/40 of the tests, the heuristic interpreter was not able to find an execution within the 15 minute time limit, while the standard interpreter was able to find an execution in all of the 40 tests. In 31/40 of the tests, executions found by the heuristic interpreter resulted in more task assignments than those found by the greedy interpreter. In 7/40 of the tests, executions found by both the heuristic and greedy interpreters resulted in equal numbers of task assignments. In 2/40 tests, the heuristic interpreter could not find an execution within 15 minutes, while the greedy interpreter was able to find an execution in all of the 40 tests. Table A.3 defines the groups of dependent jobs in each test together with the time required for each test case to complete under each of the three interpreters examined.

The average time required by the standard, greedy, and heuristic interpreters to find a solution in the collection of tests created for each number of jobs (N_{JOBS}), workers (N_W), and maximum number of tasks per job (T_{MAX}), is highlighted in Table 7.12. Also stated

is the average number of tasks scheduled for each collection of tests. In brackets following each average is the (sample) standard deviation of the associated data set. Table 7.12 demonstrates that my heuristic interpreter consistently finds executions that schedule more tasks, on average, than the two alternatives. The time it uses to do so varies significantly across the collections of tests (more so than the times used by the standard and greedy interpreters). In rows 6 and 7 of this table, only the test cases that all three interpreters could solve were included in the averages (the heuristic interpreter was unable to solve one of the tests in the collections described in each of these rows).

In each of the domains I have described in this chapter – the spacecraft control domain of Section 7.1, the mine operations domain of Section 7.2, and the scheduling domain of this section – my heuristic interpreter finds executions higher in quality than those found by the standard and greedy interpreters in a majority of their test cases. In the next section I consider this heuristic interpreter in context with the features of existing work.

7.4 Motivations

I now consider the relative performance of my heuristic interpreter with the prior work I have discussed in Section 6.1. I first determine if it is feasible in the three presented test domains to explore the entire execution space. Following this I consider the practicality of employing a traditional best-first search algorithm within a GOLOG interpreter.

7.4.1 Exploring Whole Execution Spaces

A SWI-Prolog program has been written that searches for an execution of the control program in each domain (in each of its test cases) for which an unsatisfiable condition holds (that the execution is the initial situation). The interpreter will consider each execution (without maintaining an execution list) of a program before it determines that no such execution exists. I examine how long this search to failure takes in each domain.

In the spacecraft control domain, it was not possible (in any of the tests of Table 7.3) to explore the entire execution space within the 15 minute time limit given to each test. This was also the case in the scheduling domain. In the mining domain, it was possible to discover all executions of the control program in some of the simpler test cases – those with only a few blocks scattered across the mine landscape. In 14/60 of the test cases in

N_B	R_T (tonnes)	GROUPS	N_H	T (s)
5	1500	1,1,3	5	0.48
5	1500	1,1,3	5	0.36
5	1500	2,1,2	5	0.75
5	1500	2,1,2	5	0.59
5	1500	1,2,2	5	0.38
5	1500	1,2,2	5	0.30
10	2500	1,3,2,3,1	5	27.98
10	2500	1,3,2,3,1	5	41.97
10	2500	2,1,4,2,1	5	29.91
10	2500	2,1,4,2,1	5	20.63
10	2500	2,1,3,2,2	5	54.03
10	2500	2,1,3,2,2	5	42.05
15	3500	5,2,3,2,3	5	206.09
15	3500	5,2,3,2,3	5	—
15	3500	5,2,3,1,1,3	5	306.48

Table 7.13: The time T taken to explore the entire execution space in the mining domain, where: N_B and N_H denote the number of blocks and hazards across the landscape, and R_T the required stockpile tonnage. GROUPS denotes the number of dependent block groups, and '–' denotes that all executions could not be found within 15 minutes.

the mining domain, it was possible to explore the entire execution space (Table 7.13).

Based on these results, it is clear that it is not feasible to explore the entire execution space in any of these test domains. The DTGOLOG-based approaches discussed in Section 6.1.1 can consequently be ruled out as appropriate tools for the discovery of preferred executions within the domains and problems I have considered in this chapter.

7.4.2 The Application of Best-First Search

I now consider whether it is feasible in the described test domains to maintain on a search frontier all unexplored configurations in the search for an execution. In the work of Sohrabi et al. (2006), a GOLOG interpreter is developed to find a most preferred web service composition. This interpreter employs the PPLAN search algorithm of Bienvenu et al. (2006). A search frontier of unexplored configurations is maintained, and is expanded each time a transition to a configuration is made. This configuration is removed from the frontier, and the set of configurations that can be transitioned to from it (under the transition semantics of Section 5.3.1) is added to the frontier. Each of these configurations is assigned a heuristic value (as described in Section 6.1.3). The frontier is ordered

N_B	R_T (tonnes)	N_H	D_H	T_H (s)	D_P	T_P (s)
5	1500	5	3.38	0.08	3.38	0.97
5	1500	5	4.32	0.05	4.32	0.78
5	1500	5	2.83	0.06	2.83	1.14
5	1500	5	2.55	0.08	2.55	1.02
5	1500	5	6.04	0.03	6.04	0.67
5	1500	5	7.20	0.09	7.20	1.61
10	2500	5	1.15	0.14	1.15	6.77
10	2500	5	0.75	0.13	0.75	222.94
10	2500	5	3.73	0.09	2.27	246.75
10	2500	5	3.89	0.09	3.89	2.00
10	2500	5	1.53	0.16	1.53	108.91
10	2500	5	4.01	0.14	1.23	187.50
15	3500	5	3.94	0.27	3.94	225.77
15	3500	5	0.76	0.36	0.76	41.89
15	3500	5	6.08	0.34	5.22	12.78

Table 7.14: Achieved stockpile distance (from target) in executions found by the heuristic (D_H), and best-first search (D_P) interpreters. N_B, R_T, and N_H denote the number of blocks, required stockpile tonnage, and number of hazards in each test. T_H and T_P denote the time required for interpretation by the heuristic and best-first search interpreters.

by desirability, and the configuration at the front of this list is transitioned to.

I have developed a GOLOG interpreter that uses the best-first search algorithm of Sohrabi et al. (2006) and Bienvenu et al. (2006), but which employs my relaxed lookahead method of heuristically evaluating available choices. In the spacecraft control and scheduling domains, this interpreter was unable to find a solution in any of the tests shown in Tables 7.3, A.1, 7.11 and A.3. In each test, the interpreter either ran out of memory before the 15 minute time limit given to each test elapsed or could not return a solution within this time. In the mining domain, this interpreter was able to solve the first 15/60 tests shown in Tables 7.7 and A.2. Table 7.14 shows the results of these test cases alongside those of my heuristic interpreter. In 3/15 of these tests, the best-first search interpreter was able to find executions higher in quality than those found by the heuristic interpreter. In the remaining 12, the two interpreters found equally good executions. The best-first search interpreter was (on average) 507× slower than the heuristic interpreter.

This experiment was repeated with the relaxed lookahead-based heuristic used in this best-first interpreter swapped by one that does not employ a lookahead. This modified

N_T	$\overline{O}_H$	$\overline{T}_H$ (s)	$\overline{O}_B$	$\overline{T}_B$ (s)
10	2.9 (1.37)	0.36 (0.36)	1.9 (0.32)	0.26 (0.22)
20	9.1 (1.85)	1.38 (0.58)	3.1 (1.45)	1.01 (0.63)
30	16.7 (1.70)	7.35 (1.58)	4.8 (2.82)	3.12 (2.67)
40	22.9 (1.60)	24.99 (3.67)	8.8 (2.35)	5.08 (0.97)
50	31.88 (2.03)	80.12 (8.91)	12.38 (1.85)	26.22 (5.48)

Table 7.15: Mean number of observations made in executions found by the heuristic ($\overline{O}_H$), and modified best-first ($\overline{O}_B$) interpreters (sample standard deviation in brackets) for tests with varying N_T (number of targets). $\overline{T}_H$ and $\overline{T}_B$ denote the mean time used by the examined interpreters to arrive at a solution (sample standard deviation in brackets).

interpreter evaluates available choices using the f_p function in each domain – choices are selected that minimise f_p. In the spacecraft control domain, this modified best-first interpreter was unable to solve 2/50 tests (my heuristic interpreter solved all). On the 48 it was able to solve, the two interpreters found equally good executions in 7, while on the remainder my heuristic interpreter was the victor. Table A.4 shows the results of both interpreters in this domain, including the time required by each to arrive at a solution. The best-first interpreter took less time (my heuristic interpreter was on average $3\times$ slower) but was unable to equal or better the heuristic interpreter in a majority of tests. Table 7.15 lists the average number of targets observed in executions found by the heuristic and best-first interpreters, in test collections with varying N_T (targets available), and the average time each interpreter used in finding a solution (sample standard deviation in brackets). Tests that both interpreters could not solve were excluded in these averages.

In the mining domain, the modified best-first interpreter could not solve 1/60 of the tests. On 28/60, my heuristic interpreter found higher quality executions, while in 27/60 the best-first search interpreter was the victor. The two interpreters found equally good executions in 4/60 tests. Table A.5 shows the results of the modified best-first interpreter on each test in the mining domain, including the time required for it to arrive at a solution. The times required by both interpreters were roughly similar. Table 7.16 lists the average stockpile-to-target composition distance in executions found by the heuristic and best-first interpreters, in test collections with varying N_B (number of ore blocks), and the average time each interpreter used in finding a solution (sample standard deviation in brackets). Tests that both interpreters could not solve were excluded in these averages.

N_B	$\overline{D}_H$	$\overline{T}_H$ (s)	$\overline{D}_B$	$\overline{T}_B$ (s)
5	4.39 (1.87)	0.07 (0.02)	5.13 (1.58)	0.16 (0.03)
10	2.51 (1.52)	0.13 (0.03)	2.87 (1.21)	0.34 (0.11)
15	3.16 (1.85)	0.33 (0.03)	2.82 (2.43)	0.57 (0.13)
20	1.25 (0.67)	0.84 (0.10)	1.70 (0.84)	1.99 (0.80)
25	1.11 (0.68)	1.74 (0.13)	1.24 (0.85)	2.90 (0.92)
30	1.15 (0.66)	3.83 (0.32)	1.19 (0.70)	6.52 (1.56)
35	0.68 (0.33)	6.23 (0.42)	0.77 (0.19)	10.31 (5.01)
40	0.67 (0.21)	11.51 (0.79)	0.77 (0.19)	12.34 (2.09)
45	0.85 (0.48)	20.54 (0.95)	0.52 (0.19)	18.53 (5.11)
50	0.63 (0.24)	32.99 (1.38)	0.37 (0.14)	41.23 (9.21)

Table 7.16: Mean achieved distance between stockpile and target compositions in executions found by the heuristic ($\overline{D}_H$) and modified best-first ($\overline{D}_B$) interpreters for test sets with varying N_B (number of ore blocks). $\overline{T}_H$ and $\overline{T}_B$ denote the mean time required by each interpreter to arrive at a solution (sample standard deviation in brackets).

In the task scheduling domain, this interpreter was able to solve 27/40 (my heuristic interpreter solved 38/40) of the available tests. The best-first search interpreter could solve 1 test that my heuristic interpreter could not. On the subset of tests in this domain that both interpreters could solve (26 tests), my heuristic interpreter found executions higher in quality on 25, while on 1/26 equally good executions were discovered. Table A.6 shows the results of both interpreters in the scheduling domain, including the time required by each to arrive at a solution. My heuristic interpreter was slower in general, but the best-first search approach was unable to solve a large subset of tests. Table 7.17 lists the average number of task-to-worker assignments made in executions found by the heuristic and best-first interpreters, in test collections with varying N_{JOBS}, N_W, and T_{MAX} (the number of jobs, workers, and maximum number of tasks per job). The average time each interpreter used in finding a solution (sample standard deviation in brackets) is also provided. Tests that both interpreters could not solve were excluded in these averages – there were no test cases in rows 6 and 7 that both interpreters could solve.

With these results it can be concluded that 'forgetting' unexplored choices is necessary (within the considered domains) in the development of an interpreter that uses my relaxed lookahead-based heuristic and can be practically applied. While in some instances the (un-modified) best-first search interpreter produced higher quality executions than my heuristic approach, these situations were in the minority. A best-first search inter-

N_{JOBS}	N_W	T_{MAX}	$\overline{NT}_H$	$\overline{T}_H$ (s)	$\overline{NT}_B$	$\overline{T}_B$ (s)
15	8	7	21.25 (2.22)	5.64 (4.74)	16 (3.37)	0.25 (0.26)
15	6	5	14.8 (0.45)	0.82 (0.40)	13.6 (0.89)	0.17 (0.16)
20	8	7	40.67 (2.08)	7.59 (1.23)	34 (2.65)	3.9 (6.08)
20	6	5	30 (4.85)	3.67 (0.59)	23.8 (4.27)	0.37 (0.19)
30	6	5	47 (2.00)	17.34 (3.62)	37.4 (2.07)	0.81 (0.32)
30	8	7	–	–	–	–
40	8	7	–	–	–	–
40	6	5	60.75 (2.87)	38.07 (11.50)	48.75 (4.79)	2.15 (1.16)

Table 7.17: Mean number of task-to-worker assignments made in executions found by the heuristic ($\overline{NT}_H$) and modified best-first ($\overline{NT}_B$) interpreters for test sets with varying N_{JOBS}, N_W, and T_{MAX} (the number of jobs, workers, and maximum number of tasks per job) with sample standard deviation in brackets. $\overline{T}_H$ and $\overline{T}_B$ denote the mean time used by each interpreter in finding a solution (sample standard deviation in brackets).

preter that uses a greedy non-lookahead-based heuristic did not consistently find better executions than those found by my heuristic interpreter across the three test domains.

7.4.3 GOLOG & Classical Planning

In Section 6.1.4 of Chapter 6, I discuss several approaches designed to combine classical planning and GOLOG. Baier et al. (2007b, 2008) and Fritz et al. (2008) consider the compilation of a GOLOG program and its underlying basic action theory (BAT) into the ADL subset of the plan description language PDDL [Gerevini and Long (2005)]. A most preferred execution of a GOLOG program can then be found by a classical planner. I view the experimental comparison of this work and my approach as an interesting topic of future work. There are, however, advantages of my work relative to these compilation-based approaches, irrespective of the results of any experimental comparison.

Another piece of related work discussed in Section 6.1.4 is that of Claßen et al. (2007), where a GOLOG interpreter calls a classical planner to solve general planning tasks embedded within a GOLOG program. This work and mine are quite different in purpose. For the reasons listed in Section 6.1.4, an experimental comparison is not appropriate.

7.5 Relaxation: What is it Good For?

In this section, I consider and analyse the relative performance of an alternative style of lookahead (for use within my heuristic interpreter) to the relaxation-based formulation.

N_T	$\overline{O}_H$	$\overline{T}_H$ (s)	$\overline{O}_{NR}$	$\overline{T}_{NR}$ (s)
10	2.9 (1.37)	0.36 (0.36)	3.2 (1.32)	1.20 (0.69)
20	9.1 (1.85)	1.38 (0.58)	9.1 (1.85)	13.20 (7.12)
30	16.7 (1.70)	7.35 (1.58)	16.7 (1.70)	58.08 (20.23)
40	22.9 (1.60)	24.99 (3.67)	22.9 (1.79)	173.78 (34.55)
50	31.7 (1.83)	80.89 (8.02)	31.6 (1.78)	474.60 (50.85)

Table 7.18: Mean number of observations made in executions found by the heuristic ($\overline{O}_H$), and non-relaxed ($\overline{O}_{NR}$) interpreters (sample standard deviation in brackets) for tests with varying N_T (number of targets). $\overline{T}_H$ and $\overline{T}_{NR}$ denote the mean time used by the examined interpreters to arrive at a solution (sample standard deviation in brackets).

To the best of my knowledge, in the existing body of literature on GOLOG, the situation calculus, and preference-based GOLOG, there are no alternative formulations of a relaxation-based heuristic that can be implemented as part of a GOLOG interpreter. Consequently, I compare my heuristic interpreter with one that employs a non-relaxed variation of its heuristic estimation method. The aim is to determine whether the use of relaxation is warranted or has benefits over looking-ahead in a more precise fashion.

In each domain, I have compared my heuristic interpreter to one that uses actual interpretation (not relaxed) to evaluate choices. The implementation of both interpreters is equivalent in all respects excepting the use of relaxation to conduct a lookahead. A lookahead execution is found using the greedy interpreter implementation (described in the introduction to this chapter). I have used this non-relaxed interpreter to find executions in each of the test cases of the three domains described in this chapter.

In the spacecraft control domain (described in Section 7.1), this non-relaxed interpreter was on average 7× slower than my heuristic interpreter on its collection of tests (see Table A.7 in Appendix A). In 4/50 of the test cases in this domain, the non-relaxed interpreter found executions that observed more targets than those found by the heuristic interpreter. In 2/50 of the tests, the heuristic interpreter was the victor – it found executions that observed more targets. In the remainder, both interpreters found equally good executions. The average number of observations made in executions found by the two interpreters (and the average time taken to find these executions) is shown in Table 7.18.

In the mining domain (Section 7.2), the non-relaxed interpreter was on average 10× slower than my heuristic interpreter (see Table A.8 in Appendix A). In 16/60 of the

N_B	$\overline{D}_H$	$\overline{T}_H$ (s)	$\overline{D}_{NR}$	$\overline{T}_{NR}$ (s)
5	4.39 (1.87)	0.07 (0.02)	4.39 (1.87)	0.38 (0.09)
10	2.51 (1.52)	0.13 (0.03)	2.51 (1.52)	1.28 (0.30)
15	3.16 (1.85)	0.33 (0.03)	3.06 (1.66)	2.74 (0.70)
20	1.25 (0.67)	0.84 (0.10)	1.14 (0.54)	9.61 (2.22)
25	1.11 (0.68)	1.74 (0.13)	0.99 (0.40)	17.36 (5.20)
30	1.15 (0.66)	3.83 (0.32)	0.90 (0.60)	56.00 (13.37)
35	0.68 (0.33)	6.23 (0.42)	0.73 (0.52)	77.84 (22.14)
40	0.84 (0.46)	11.33 (0.83)	0.91 (0.71)	130.52 (35.14)
45	0.85 (0.48)	20.54 (0.95)	0.45 (0.30)	200.62 (64.92)
50	0.63 (0.24)	32.99 (1.38)	0.64 (0.30)	355.47 (99.34)

Table 7.19: Mean achieved distance between stockpile and target compositions in executions found by the heuristic ($\overline{D}_H$) and non-relaxed ($\overline{D}_{NR}$) interpreters for test sets with varying N_B (number of ore blocks). $\overline{T}_H$ and $\overline{T}_{NR}$ denote the mean time required by each interpreter to arrive at a solution (sample standard deviation in brackets).

tests within this domain, the non-relaxed interpreter found executions that resulted in a closer-to-target stockpile than the heuristic interpreter. In 5/60 of the tests, the heuristic interpreter found executions that led to the highest quality stockpiles. In the remaining tests, the two interpreters found executions that formed equally good stockpiles. The average stockpile-to-target composition variations in executions found by the two interpreters (and the average time taken to find these executions) are shown in Table 7.19.

In the scheduling domain (Section 7.3), the non-relaxed interpreter was on average only 2× slower than my heuristic interpreter (see Table A.9 in Appendix A for the complete set of results). However, in 5/40 of the test cases, the non-relaxed interpreter could not find an execution within the 15 minute time limit. The heuristic interpreter could not find a solution within the time limit in only 2/40 of the test cases. In the subset of test cases that both interpreters could solve, equally good executions were discovered. The average number of task-to-worker assignments made in executions found by the two interpreters (and the average time taken to find these executions) is shown in Table 7.20 (including only test cases that both interpreters could solve in the averages) with the sample standard deviation of the data in brackets. In rows 6 and 7, the sample size was reduced to 2–3 test cases, amongst which the variability in time used was significant.

The results in this section reflect the expectation that the use of a relaxation-based lookahead is worthwhile provided the relaxed interpretation of a program is less time

N_{JOBS}	N_W	T_{MAX}	$\overline{NT}_H$	$\overline{T}_H$ (s)	$\overline{NT}_{NR}$	$\overline{T}_{NR}$ (s)
15	8	7	20.8 (2.17)	4.93 (4.40)	20.8 (2.17)	16.27 (18.44)
15	6	5	14.8 (0.45)	0.82 (0.40)	14.8 (0.45)	1.69 (1.42)
20	8	7	39.6 (2.07)	43.9 (72.07)	39.6 (2.07)	109.23 (133.27)
20	6	5	30 (4.85)	3.67 (0.59)	30 (4.85)	5.91 (3.93)
30	6	5	47 (2.00)	17.34 (3.62)	47 (2.00)	24.55 (15.82)
30	8	7	51.5 (3.54)	48.23 (48.16)	51.5 (3.54)	53.86 (41.28)
40	8	7	82.3 (6.03)	295.45 (363.45)	82.3 (6.03)	204.24 (131.91)
40	6	5	61.6 (3.13)	42.31 (13.74)	61.6 (3.13)	43.43 (17.94)

Table 7.20: Mean number of tasks assigned to workers in executions found by the heuristic ($\overline{NT}_H$) and non-relaxed ($\overline{NT}_{NR}$) interpreters (sample standard deviation in brackets) for test collections with varying N_{JOBS}, N_W, and T_{MAX} (the number of jobs, workers, and maximum number of tasks per job). $\overline{T}_H$ and $\overline{T}_{NR}$ denote the mean time used by each interpreter in finding a solution (sample standard deviation in brackets).

consuming than an actual interpretation. In the spacecraft control and mining domains, relaxation of action preconditions allows an interpreter to avoid costly operations – operations that check time and resource constraint violations. Hence, the non-relaxed interpreter was significantly slower than my heuristic interpreter. In the task scheduling domain, the process of evaluating action preconditions is not time consuming. Relaxation in this domain allows an interpreter to avoid finding an appropriate combination of worker-to-task assignments without assigning a worker to more than one task. In some instances this can be difficult, and in others it may be easy. This mix of difficult and simple provides a plausible explanation of the above results, where the non-relaxed interpreter was slower by varying degrees (sometimes great and sometimes small).

The successor state axioms of the domains presented in this chapter could have been expressed differently – so that relaxation does not ignore any of their components. Domain design is important in allowing my heuristic interpreter to perform at its best.

7.5.1 Relaxing All Tests

My decision to relax only action precondition tests within the relaxed interpreter of Section 6.5 of Chapter 6 is now analysed. I determine if relaxing all tests (within test, if ... else, and while loop constructs) during interpretation results in a better performance by my heuristic interpreter. I alter the relaxed interpreter of Section 6.5 to relax all tests, and create a version of my heuristic interpreter that employs this altered relaxed interpreter

N_T	TIME	FUEL	POWER	O_H	T_H (s)	O_{AR}	T_{AR} (s)
10	150	500	200	2	0.23	2	0.80
10	150	500	200	2	0.39	2	0.98
10	150	500	200	6	0.28	2	16.53
10	150	500	200	2	0.92	2	2.45
10	150	500	200	3	0.08	2	4.30
10	150	500	200	2	0.08	2	0.23
10	150	500	200	4	0.22	2	2.61
10	150	500	200	2	0.16	2	0.20
10	150	500	200	2	1.09	2	1.95
10	150	500	200	4	0.14	2	2.47
20	250	600	300	9	1.20	2	153.28
20	250	600	300	8	0.98	2	28.86
20	250	600	300	12	1.81	5	90.52
20	250	600	300	11	1.70	2	207.14
20	250	600	300	8	1.38	3	121.34
20	250	600	300	10	1.09	2	136.33
20	250	600	300	8	0.88	2	76.42
20	250	600	300	6	2.70	5	13.86
20	250	600	300	8	0.67	1	72.28
20	250	600	300	11	1.34	5	63.64
30	350	700	400	18	8.80	–	–
30	350	700	400	19	8.84	–	–

Table 7.21: Observations made in executions found by the heuristic (O_H) and the all-relaxed (O_{AR}) interpreter. T_H and T_{AR} denote the time required (in seconds) to find a solution to each test by the heuristic and 'all-relaxed' interpreters (respectively). N_T denotes the number of celestial targets, TIME (FUEL and POWER) denote the units of time (fuel and power) available to the craft, and '–' denotes a failure to find a solution.

during heuristic evaluation. I reconsider the spacecraft control domain of Section 7.1.

I have run this version of my heuristic interpreter on the first 22 tests of the spacecraft control domain, whose original results are shown in Table 7.3. Table 7.21 displays the results of this 'all-relaxed' interpreter – the number of targets its executions observe and the time it requires to solve each test. Table 7.21 shows that this altered interpreter performs quite poorly relative to the non-altered heuristic interpreter – both in terms of the number of observations made by its executions and the time it needs to find them.

As I have described in Section 6.5.3, test constructs and while loop conditions serve an important purpose in controlling when to stop performing a series of steps during the interpretation of a GOLOG program. Moreover, test constructs are often used to constrain

the range of values available for selection in the construct $\pi v.\delta$. Removing these constraints may result in values being selected that never would have been in a non-relaxed interpretation. The result is a relaxed execution of a program that is a poor approximation of what a legal execution would look like. Using such a poor approximation to guide the heuristic interpretation of a program is likely to lead to poor results.

In the spacecraft control domain, Listing 7.1.1 shows that unobserved targets are selected for observation until either there is no time left, or no targets remain to be observed. Consider the successor state axiom for the *Observed* fluent, shown in Definition 7.1.2:

$$Observed(lt, do(a, s)) \equiv a = takeimage(lt) \vee Observed(lt, s)$$

Consider the relaxed regression rule for the negation of this fluent, given a basic action theory $\mathcal{D}$, shown in Definition 6.3.2 of Section 6.3.1:

$$RR_{\mathcal{D}}[\neg F(\vec{t}, do(\alpha, s))] \stackrel{def}{=} RR_{\mathcal{D}}[\phi_n(\neg F(\vec{t}, s) \vee \gamma_F^-(\vec{t}, \alpha, s))]$$

In the relaxed regression of $\neg Observed(lt, do(a, s))$, all targets that are unobserved in the initial situation are considered to be unobserved in any future situation. The $\gamma_F^-(\vec{t}, \alpha, s)$ component is empty in the successor state axiom for *Observed*. In the relaxed interpretation of the program shown in Listing 7.1.1, the craft repeatedly selects the same target to turn to and observe until no time remains[2]. The result is an execution that says nothing about how many targets can actually be observed in the future by making a particular target selection. This explains the poor results shown in Table 7.21.

7.6 Limitations of Heuristic GOLOG

In the experiments conducted thus far, it is evident that both the standard and greedy interpreters are faster than my heuristic interpreter. In each of the tests within the spacecraft control domain (Tables 7.3 and A.1), standard interpretation required less than 1.4 seconds. In the most complex test of this example, the heuristic interpreter required 97.3 seconds to find a solution, while the greedy interpreter required at most 28.8 seconds. The times required by these interpreters to solve each test are shown in Tables A.1 to A.3.

[2]The relaxation of the action preconditions for the *turnto* action allows the craft to turn to a target it is already focused upon. Under relaxation, the craft is not pointing at a target ($\neg Pointing(t, do(a, s))$) if it was not pointing at that target in situation s. Any target not focused on in S_0 is considered not focused on in s.

In each of the test cases within the mine operations domain (Tables 7.7 and A.2), standard interpretation took less than 3.6 seconds. In contrast, the heuristic interpreter took at most 33.7 seconds while the greedy interpreter took at most 3.95 seconds.

In the scheduling domain, whose test results are shown in Tables 7.11 and A.3, standard interpretation required at most 6.91 seconds. Heuristic interpretation could not solve two of these test cases within 15 minutes, and amongst the test cases that could be solved within the time limit the maximum time consumed was 714.81 seconds. The greedy interpreter required at most 14.05 seconds. In Section 7.7, I consider a range of approaches (as future work) designed to reduce the time overhead associated with my heuristic interpreter. Some time overhead is unavoidable, however, as improving the quality of discovered executions (with a more informed heuristic) comes at a price.

The test domains presented in this chapter are designed to be amenable for heuristic interpretation. The choices made by an interpreter in these domains have an impact on the quality of executions found by influencing the future availability of choices. Without this property, my heuristic interpreter would perform no better than the standard – if it is not important which choices are selected, selecting the first available choice is adequate. Amongst the problems that do satisfy this property, and are suited for representation in GOLOG[3], the use of relaxation may sometimes guide an interpreter toward a poor solution – as the relaxed regression of an action precondition may ignore important caveats.

Consider an agent designed to plan an itinerary for a traveller. It must organise flights to a destination, arrange a number of sight seeing tours, and remain within a set budget. The traveller will prefer to visit one destination over another if the sight seeing tours they can have at the former location are more desirable. These tours can only be booked if the traveller has enough money to cover their cost. If money is treated as a resource, in a manner similar to the treatment of time, power and fuel within the spacecraft control domain, then the relaxed execution of the agent's control program will ignore money constraints in its selection of destination and tours. The heuristic estimation of choices within this domain may result in the selection of a destination that has no desirable tours within the budget of the traveller. This becomes an issue as my heuristic interpreter does

[3]See the discussion in Section 6.1.4 concerning the work of Claßen et al. (2007).

not alter its choices unless a selected choice does not lead to a legal execution[4].

7.7 Concluding Remarks and Future Work

In this chapter, I have evaluated the performance of my heuristic GOLOG interpreter in three domains. The first domain involves a spacecraft charged with making observations of celestial targets within a limited time window. The second of these domains sees a mining robot traverse the landscape of a mine while building a stockpile of a desired tonnage and composition. In the final domain, a task scheduling agent responsible for assigning tasks to workers while maximising worker utilisation is considered.

The presented heuristic interpreter is used to find an execution of the controlling program in each of these domains. I compare the quality of the executions it discovers with those found by the standard interpreter of Section 5.3.1 and a heuristic interpreter not using a lookahead. This latter interpreter has the same semantics as the relaxed interpreter of Section 6.5, but does not relax the testing of action preconditions. I have found that on a majority of test cases within each of the three domains described, my heuristic interpreter finds executions higher in quality than both of the alternatives I consider.

I have determined that the use of actual (unrelaxed) interpretation in the formulation of a lookahead-based heuristic is impractical (Section 7.5). I have evaluated the performance of my heuristic interpreter, altered so that its relaxed method of lookahead is replaced with an actual lookahead. This alternative version of my interpreter was not able to find executions higher in quality than those found by the unaltered version in a majority of test cases within each of the three described domains. The non-relaxed interpreter was found to be on average 7 and $10\times$ slower than the heuristic approach in the spacecraft control and mining domains, respectively. The relaxation of action preconditions within these domains removes the need to perform costly resource checking operations.

In the scheduling domain, however, less of a difference between the relaxed and non-relaxed interpreters in terms of time usage was discovered. While action preconditions are not difficult to evaluate in this domain, it can in some cases be difficult to find a worker-to-task assignment that matches workers to the tasks they are trained to perform,

[4]Usually, however, a problem domain can be formulated so that relaxation does not ignore important conditions.

while assigning to each worker only one task per job. In some test cases, the non-relaxed interpreter was significantly slower than my heuristic approach. In others, both interpreters required a similar amount of time. This is to be expected if relaxed interpretation is sometimes faster and sometimes no faster than a standard interpretation.

In Section 7.4.1, I highlight the impracticality of exploring entire execution spaces – an aspect of the related DTGOLOG series of approaches (described in Section 6.1.1) – in the test domains of this chapter. Moreover, I demonstrate that maintaining unexplored choices on an explicit search frontier is not feasible within these domains. I do not in this chapter experimentally compare my heuristic approach to the related work of Baier et al. (2007b, 2008), Fritz et al. (2008) and Claßen et al. (2007). I have described the advantages of my approach to those of Baier et al. (2007b, 2008) and Fritz et al. (2008) in Section 6.1.4, and leave the experimental comparison of my interpreter and these works (on domains they can support) as future work. My approach and that of Claßen et al. (2007) are geared toward solving different problems – the heuristic interpreter of Chapter 6 aims to discover preferred executions while the approach of Claßen et al. (2007) is designed to allow general planning tasks to be embedded within a GOLOG program (as discussed in Section 6.1.4). For this reason, I do not conduct an experimental comparison.

In Section 7.5.1, I consider whether relaxing all tests during the relaxed interpretation of a GOLOG program (in test, if . . . else, and while loop constructs) improves the performance of the heuristic interpreter I have developed. I have found that in the spacecraft control domain this modified interpreter performs poorly when compared to its unaltered form – both in terms of the quality of its executions, and the time used to find them. Relaxed executions in this scenario are not representative of their legal counterparts.

The limitations of my heuristic interpreter are discussed in Section 7.6. The experimentation conducted in this chapter has demonstrated that this heuristic interpreter is significantly slower than the standard and greedy alternatives considered. In the following section, I discuss how this time overhead can be reduced. It must be noted, however, that some overhead is unavoidable as the improvement in execution quality this interpreter achieves incurs a cost – the cost of more informed heuristic estimation.

In this and the preceding chapter, I have just scratched the surface of possibilities sur-

rounding heuristic search for the interpretation of a GOLOG program. I have identified a series of potentially fruitful avenues of further research on the topic of these chapters.

In Section 6.1.4, I have discussed the compilation-based approaches of Baier et al. (2007b, 2008) and Fritz et al. (2008). In this work, a GOLOG (or CONGOLOG) program is combined with an ADL [Pednault (1989), Section 5.2] action theory to form an ADL instance whose solutions are executions of the program. A classical (preference or non-preference-based planner) can consequently be used to find an execution of the program. To investigate more fully the advantages of remaining within the GOLOG and situation calculus space relative to compilation, it would be interesting to investigate the relative performance of my interpreter with that of Baier et al. (2007b, 2008) on the subset of GOLOG programs and basic action theories (BATs) suitable for compilation into ADL.

As I have discussed in Section 6.7.1, the hypothetical techniques I present for the use of my heuristic interpreter in an online setting have not been implemented. I would like to investigate the performance of such an online heuristic interpreter and its dependence on the depth of lookahead used during relaxed interpretation. In an online setting, an agent is often controlled by a non-terminating program. In these circumstances, it is not possible to select an action horizon h for the relaxed lookahead that allows complete executions to be found. Instead, the interpreter would look h steps ahead.

The test domains I have considered in this chapter, while being moderately complex, are far less complex than a real-world application. An important future step in the evaluation and development of the presented heuristic interpreter is to determine how well it scales when faced with larger and more complex domains. As a part of this evaluation, I would like to consider potential variations of my heuristic interpreter aimed at reducing its complexity. As the complexity of these domains increases, the branching factor – the number of new choices that become available upon making a decision or pursuing a path – can become a significant issue in the ability of this interpreter to find an execution within an appropriate timespan. Of likely benefit will be techniques aimed at judiciously selecting which choices to evaluate by relaxed interpretation and which to prune from consideration. Such techniques will not only improve the ability of this interpreter to scale, but reduce the time it requires to solve the problems described in this chapter.

Chapter 8
Argumentation-Based Search

An Application in GOLOG Interpretation

THE task of problem solving in the early days of artificial intelligence was the playing field of goal-directed search. Simon (1969) describes human problem solving as a selective search through a "maze of possibilities" – each possibility a direction in which to focus the investigative effort of the problem solver. The key to successful problem solving is the navigation of this space so that only directions that lead to a solution are explored, while fruitless tangents are avoided. This chapter is concerned with preference-based search – the search for a preferred solution amongst all possibilities.

Within the paradigm of goal-directed search, a problem is represented in the form of a state-space, a transition function, a control strategy, and a goal test [Gardner (1980)]. This transition function describes how the problem solver can move from one state-of-affairs to another. A problem solution is a sequence of such transitions that lead the problem solver to a goal state from its initial position. A control strategy determines which transitions to make in each state, while a goal test determines if a state is a goal.

These concepts are demonstrated in the following two examples. In the search for a solution to the n-puzzle, each state is an arrangement of numbered tiles in a puzzle frame. In its goal state, tiles are ordered by their number while transitions slide tiles into empty spaces creating new arrangements. In the search for a route between two destinations, each state describes the changing location of the traveller. The goal is its final destination,

while available transitions move the traveller from one location to another.

Preference-based search is the search for a path from an initial to a goal state (a solution to a planning problem, for example) that is the most preferred according to a set of preferences. In the early days of preference-based search, the dominant form of preference representation was by numeric merit or utilities (the favoured form of preference representation in economics, as described in Chapter 2). A^* [Hart et al. (1968)] and generalised A^* [Dechter and Pearl (1985)] represent two early state-space search techniques that rely upon such simple preference representations. These algorithms discover lowest cost and highest merit solutions, respectively, to path planning problems.

The early ideas of Hart et al. (1968) and Dechter and Pearl (1985) have seen a vast array of extensions and development in the past few decades. The developments of particular interest to us, however, are the variants of these approaches in which the preferences that guide search are no longer required to take the form of a numeric evaluation function. Perny and Spanjaard (2002) describe a state-space graph search that uses a preference relation over value multi-sets to guide search. These multi-sets characterise partial (not terminating at a goal state) or complete (terminating at a goal state) paths in this graph. Each arc within a path is associated with a value, and each path is characterised by the set of values attributed to its arcs. It is this preference relation, not an evaluation function applied to individual states on a search frontier, that guides search for a terminal or goal state. Multi-objective path evaluation is another form of non-conventional search guidance, used by Stewart and White (1991), in an extension of the A^* algorithm.

These methods guide the search of a state-space using preference expressions that defy convention. General principles of path comparison, however, do not fall within their scope, as explained in Section 8.1. In Chapter 4, an argumentation framework is presented, called the ADM, designed to automate decision-making over a collection of available choices. A choice is selected on the basis of a range of preferences expressed as general comparison schemes. These schemes are capable of representing each of the varieties of preference discussed in Chapter 2. In this chapter, the ADM of Chapter 4 is used in the development of a goal-directed search method guided by the general schemes just described. The goal of this chapter is to create a search algorithm that can be applied

irrespective of the means by which its solutions are compared by a user or agent.

In this chapter I present an argumentation-based approach to goal-directed search. The search algorithm presented in this chapter, denoted argumentation-based search (ABS), is designed to search a tree-structured state-space[1] for a most preferred path from its initial state to a goal state. Preference is assumed to be expressed in the form of comparison schemes (Section 4.3) – each scheme outlining the circumstances under which one solution path is preferred to another. In Chapter 4, the properties of an agent using an ADM are described, including: a set of comparison schemes SCH; an ability to construct arguments over a collection of choices on the basis of SCH; a deductive system Γ; an ability to determine a preference $\succeq$ and conflict $\mathcal{R}$ relation over the arguments it constructs; and a decision-rule RL. A path p in the set of all solution paths $\mathcal{O}$ is defined as most preferred if it lies within the most preferred subset of $\mathcal{O}$ by an ADM constructed by this agent. ABS is designed to search a tree-structured state-space for such a path.

The ABS algorithm is similar in style to the hill-climbing search of HSP [Bonet and Geffner (2001)] and FF [Hoffman and Nebel (2001)] – two planners upon which the heuristic GOLOG interpreter of Chapter 6 has been modelled. From an initial state, ABS makes repeated state transitions until it discovers a goal state. ABS selects states to visit in this state-space on the back of a process of argumentation – a process involving an application of the ADM framework of Chapter 4. A state is transitioned to in search if it is believed to lead to a most preferred path. In making this assessment, a sample of initial to goal state paths are generated for each transition – each path denoting a possible solution arising from its selection. These paths need not be legal paths (legal executions or plans, for example), but relaxed solutions to the search problem. An ADM is applied to this collection of solutions. Arguments are formed over these samples, and the most preferred subset this ADM discovers is used to guide the selection of a transition. A transition is selected if it leads to a solution that lies within this most preferred subset.

ABS is not an optimal (admissible) algorithm – it is not guaranteed to find a path that lies within the most preferred subset of all paths. As has been discovered in Chapter 7, however, the characteristics required in optimal algorithms (the maintenance of all

[1] Within this state-space, the root of the tree is an initial state, and each transition (arising from an operator or action application) forms a branch connecting one state to another.

unexplored choices on a search frontier, for example) tend to make them impractical.

In this chapter, ABS is implemented and applied to form an argumentation-based interpreter, denoted AR-GOLOG, for GOLOG programs [Levesque et al. (1997)]. GOLOG is a logic programming language for agents designed to represent complex actions and procedures in the situation calculus (Section 5.3). Each execution of a GOLOG program represents one way of achieving a particular goal. Existing tools developed to discover most preferred program executions, and indeed preferred solutions to planning problems in general, are limited in terms of the range of preference types that can be used to guide search for a solution. A number of these approaches are described in Chapter 6. The work presented in this chapter employs argumentation to compare available transitions during program interpretation and select the most promising for pursual. To the best of my knowledge, this is the first work to integrate argumentation and the interpretation of GOLOG programs and to use argumentation as a tool for goal-directed search. This chapter concludes by considering the degree to which this interpreter represents a decoupled approach to planning with preferences (as per Criterion 2.1.1).

This chapter first describes a range of related works. In Section 8.1, a range of existing formulations for preference-based search, preference-guided interpretation of GOLOG programs, and preference-based planning, are presented. This is followed with a formalisation of argumentation-based search (ABS). ABS is applied in the development of an argumentation-based GOLOG interpreter (AR-GOLOG) in Section 8.3. In Section 8.4, this interpreter is evaluated within a travel planning domain, and the spacecraft control domain of Chapter 7. As a result of this evaluation, how successful argumentation-based GOLOG is in the discovery of most preferred program executions is determined. I conclude with a discussion of plans for the future development of this technique.

8.1 Related Work

In this section, a snapshot of the current state-of-the-art in preference-based search and planning is presented. The term preference-based search is used in this thesis to encompass the wide variety of work that incorporates a notion of preference over solutions as

a guide toward their discovery[2]. First considered is a range of preference-based search techniques that support non-conventional forms of preference – preference that cannot be encapsulated in an evaluation function. Following this, the realm of preference-based planning is explored, including an analysis of a number of preference-guided interpreters for GOLOG programs. The GOLOG language is described in Chapter 5.

8.1.1 Preference-Based Search

Recall that the aim of goal-directed search is to discover a path (a sequence of state transitions) from an initial state to a goal state within a graph (or tree) structured state-space. To explore this space, a search method maintains a search frontier. This frontier is a set of unexplored states, each representing a partial path from an initial to a goal state. Within preference-guided search methods, the most promising states on this frontier – those most likely to lead to a preferred solution path – are selected for exploration. The selected state is replaced on the frontier by the set of states that can be transitioned to from it. Promising states are selected on this evolving frontier until a goal state is found.

Early preference-based search methods, of which A^* [Hart et al. (1968)] is an example, assign a value to states on a search frontier using an evaluation function. The aim of A^* is to find a lowest cost path from an initial to a goal state. A^* assigns a cost estimate to each unexplored frontier state – the estimated cost required in travelling through that state to reach a goal. Within A^*, this evaluation function assigns to a state n the value:

$$f(n) = g(n) + h(n) \tag{8.1}$$

where: $g(n)$ is the cost required to reach state n from the initial state of search, and $h(n)$ is the estimated cost required to reach a goal state from n. If the function $h(n)$ underestimates this cost, then the transition to states that minimise the function f discovers a lowest cost path to a goal state. A number of A^* extensions are described in this section.

Multi-Objective Optimisation

Stewart and White (1991) present an extension of the A^* algorithm, called MOA^*, for

[2]Junker (2000) presents an approach for the solution of constraint satisfaction problems called preference-based search. My use of the term does not refer to this approach specifically.

use in multi-objective optimisation. This algorithm operates by maintaining a set of lists: OPEN; CLOSED; SOLUTION; COSTS; GOALS; and LABEL. OPEN denotes the set of states that are available for exploration – the frontier of search – and initially contains only the initial state. CLOSED maintains the set of states that have been explored by the algorithm. SOLUTION stores the solution paths found so far at any stage during search, COSTS identifies the costs associated with each of these solutions, while GOALS defines the goal states that each of these paths lead to. The set LABEL maintains an accrued cost (a vector of numbers – each element denoting the path cost by a particular criteria) for each state visited during search – the cost required to reach that state from the initial.

Search proceeds in the work of Stewart and White (1991) by selecting a state in OPEN whose estimated cost (a combination of its accrued cost vector in LABEL and a heuristic estimation of the cost vector associated with reaching a goal from that state) is not dominated by[3] an alternative choice in OPEN, or a solution within the SOLUTION set. If a state n selected for exploration is a goal, the SOLUTION, COSTS and GOALS sets are updated. Otherwise, the states that can be transitioned to from n (that are not dominated by an already found solution) are added to OPEN and their accrued costs are maintained within LABEL. States, once explored, are added to the CLOSED set. The search process continues in this manner until no further states remain in the OPEN set. At this stage in the search process, all non-dominated solution paths have been discovered.

This paradigm of multi-objective search has been extended in a number of different ways. Dasgupta et al. (1996) consider the application of heuristic search as presented within A^* for the multi-objective search of AND/OR graphs (representing the search for a reduction of a problem into primitive sub-problems). Denoted $MObj^*$, a solution is not a path through a graph, but a sub-graph. Within these sub-graphs, one child of each OR node, and all children of each AND node, are included amongst its vertices. The cost vector of each graph is defined recursively, adding the cost of each transition to an OR node, and summing over the costs of the graphs rooted at each AND node. Search proceeds by exploring sub-graphs within an evolving AND/OR graph structure, whose heuristically estimated cost vectors are non-dominated (as per Stewart and White

[3] A choice α is dominated within the work of Stewart and White (1991) if there exists an alternative that is at least as good as α on each of their objective criteria, and is strictly better than α on one of those criteria.

(1991)), until a solution sub-graph is discovered (a problem reduction). Multi-objective search with scalar cost vectors is considered in a number of other works [Mandow and de la Cruz (2005, 2007, 2009, 2010); Galand and Perny (2006); Perny and Spanjaard (2008)].

In the multi-objective search of Mandow and de la Cruz (2001), solution paths are discovered that best satisfy a collection of goals grouped into priority levels. A path is evaluated with respect to a set of goals, within a priority level, by determining the degree to which it deviates from their satisfaction. The degree to which a path satisfies a goal is represented by a scalar cost. The deviation of a path is the sum of the difference between these costs and a target level of satisfaction. Each path is associated with a vector of deviations – each element representing a different collection of goals. The aim of search is to discover a solution path whose deviation vector is the lexicographic minimum when compared to the vectors of alternative solutions. Such a vector is a lexicographic minimum if and only if: its deviation on the highest priority collection of goals is less than or equal to that of each alternative; its deviation on the second highest priority collection is less than or equal that of all alternatives that satisfy the previous condition; and so on. In an algorithm similar to that of Stewart and White (1991), states whose estimated deviation vectors[4] are the lexicographic minimum are selected for exploration.

Preference-guided search has been applied to problems outside of the domain of state-space graph search. Junker (2000) describes a preference-based approach to the solution of combinatorial optimisation tasks expressed as constraint satisfaction problems. In these problems, a set of variables require value assignments in such a way that a set of constraints are satisfied. The search for a preferred solution proceeds by iteratively selecting the (locally) most preferred assignment that is consistent with the collection of assignments made so far. Junker (2002) adapts this preference-based search for application in multi-criteria optimisation. Assuming the presence of a preference relation over criteria – highlighting their importance – search proceeds by iteratively selecting the most preferred (yet unconsidered) criterion. An optimisation sub-problem is then solved with respect to the selected criterion, determining its optimal value with respect to the space of possible solutions. This space is the set of solutions that are consistent with the optimal

[4]The estimated deviation vector for a path that passes through the given state to reach a goal state.

values selected for each of the criteria considered thus far in search.

Generalised Graph Search – An Abstract Representation of Preference

Perny and Spanjaard (2002) consider a more generalised approach to the search for a preferred path through a state-space graph – each arc of which is associated with a value (of no particular form). Preference over solution paths from an initial to a goal state is defined in terms of a preference relation $\mathcal{P}_M$ over value multi-sets – each multi-set a set of values, possibly containing duplicates. States are evaluated during the search for a solution path by employing heuristics to estimate the value multi-sets that are (potentially) associated with paths from these states to a goal. Each state n on a search frontier is assigned a value multi-set $F(n)$ – the union of two multi-sets $G(n)$ and $H(n)$. The set $H(n)$ is the most preferred (by $\mathcal{P}_M$) of the estimated multi-sets associated with paths from n to a goal state. The set $G(n)$ is the most preferred multi-set associated with a path from the initial state to n. States with the most preferred multi-set $F(n)$, by the (quasi-transitive) preference relation $\mathcal{P}_M$[5] described above, are selected for further exploration.

A key difference between the work of Perny and Spanjaard (2002), and the alternative works described thus far in this section is that the nature of how preference is determined amongst paths (value multi-sets) remains abstract. To ensure termination, completeness, and admissibility, several assumptions are made of this preference relation. These are weak-preadditivity (given paths x, y and z, if x is preferred to y, then $x + z$ is preferred to $y + z$), monotonicity (a sub-path is always preferred to its parent, or alternatively, a cyclic path can never be most preferred or maximal), and that preference over multi-sets is quasi-transitive (the strict preference relation over paths is transitive).

In Section 8.2, I present an algorithm for the preference-based search of tree-structured state-spaces that does not require a predefined relation over paths (or path descriptors) in this manner, or these assumptions to hold of the preferences that are present. In this approach, preference for one path through a given state-space over another is determined on the basis of general comparison schemes. These schemes are capable of capturing each of the ways in which paths are compared within the approaches of this section.

[5] A quasi-transitive preference relation is transitive in its strict form, but not necessarily so in its non-strict.

A^* Throughout the Ages

The A^* search algorithm [Hart et al. (1968)] has amassed a vast number of variations, extensions, and similar algorithms, since its development, beyond those described in this section. These include, but are by no means limited to: IDA^* [Korf (1985)], in which successive depth-first searches (with increasing path costs constraining depth) are conducted until a minimal cost solution is discovered; RTA^* and $LRTA^*$ [Korf (1990, 1987)] describe real-time search algorithms in which search is interleaved with action execution; D^* [Stentz (1994)], Focused D^* [Stentz (1995)], and D^* Lite [Koenig and Likhachev (2002)] for search through directed graphs with dynamically changing arc costs; HDA^* [Kishimoto et al. (2009)], a parallelisation of A^*, distributing the work conducted during search across different processes; Lifelong Planning A^* [Koenig and Likhachev (2004)], in which parts of searches are reused in similar subsequent searches; and Theta* [Nash et al. (2007)], an any-angle path planning algorithm that finds true shortest paths across a grid (path segments are not constrained to grid edges as in A^*). These approaches, while varying the way in which search is conducted, maintain the same goal – to find a path from an initial to a goal state (through a directed graph) that minimises a cost function.

In the following section, a range of existing preference-guided GOLOG interpreters are presented. These interpreters are described in greater detail within Section 6.1.

8.1.2 Preferences and GOLOG

Section 6.1 of Chapter 6 describes a range of GOLOG interpreters that use preference to guide search for program executions. DTGOLOG-based approaches [Boutilier et al. (2000)] require preference to take the form of a utility or reward function that must be maximised. Qualitative DTGOLOG uses preference expressed in temporal logic to prune the space of situations reachable from an initial configuration (program-situation pair) within a DTGOLOG interpretation [Fritz and McIlraith (2005)]. Sardiña and Shapiro (2003) consider the use of prioritised goals in characterising preferred executions within INDIGOLOG, while McIlraith and Son (2002) employ a desirability predicate outlining which actions are desirable in any given situation. Sohrabi et al. (2006) develop a best-first search approach to the interpretation of GOLOG programs, employing the PPLAN

algorithm of Bienvenu et al. (2006). Preference is expressed in terms of temporal logic formulae that describe the desirable conditions to be satisfied by an execution. Baier et al. (2007b, 2008) and Fritz et al. (2008) consider the compilation of GOLOG programs into ADL (Section 5.2.2), allowing a classical planner to find their execution.

In the following section, the techniques by which these classical planners discover most preferred plans are described.

8.1.3 Planning with Preferences

Classical planning discovers a finite linearly ordered sequence of actions that transitions a planner from an initial state to a desired goal state in a domain that is modelled by a restricted state-transition system [Ghallab et al. (2004)]. Many of the planners discussed in this section operate in this classical space. Across the range of existing preference-based planners, preference is expressed in temporal logics [Baier et al. (2007a, 2009); Bienvenu et al. (2006)], as soft goals and constraints [Brafman and Chernyavsky (2005); Benton et al. (2009); Edelkamp and Kissmann (2009)], or metric functions and action costs [Borowsky and Edelkamp (2008); Benton et al. (2005); Keyder and Geffner (2008)].

Baier et al. (2007a, 2009) translate temporal logic preferences – conditions that are satisfied or violated over the course of a plan (Section 2.2.3 of Chapter 2) – into conditions that must hold in its final state. Preference-guided search through a state-space is then used to discover a solution to a planning problem. At each stage (choice point) of search, a state on the search frontier is selected for exploration if it will allow a set of final state preferences to be most easily satisfied. A number of heuristics are described that, in a given state, estimate the difficulty of satisfying a set of preferences over the final state of a plan. These typically involve the construction of a relaxed plan graph from these states, as described by Hoffman and Nebel (2001). A preference distance function, for example, sums the depths, within such a graph, at which each of the final state preferences are first satisfied. Baier et al. (2007a, 2009) devise a planner that conducts an iterative series of planning episodes – each episode searching for a solution using an available heuristic. The result of each episode allows inferior plans to be pruned from search in the next.

The final state preferences described above represent soft goals – goals that we would

like a plan to achieve if possible. Hard goals are those that must be achieved by any legal plan. A common approach to the incorporation of preference within planning, is to represent these preferences as soft goals and conduct over subscription planning [Smith (2004)]. Within over subscription planning, it is recognised that not all of the goals of the planner can be achieved. The aim is to discover a plan that achieves the most goals, or the most important goals. Partial satisfaction planning, a similar technique, aims to find a plan that achieves a balance between the cost of achieving goals and their utility [Benton et al. (2009)]. Keyder and Geffner (2009) present a similar idea by compiling soft goals into a planning-with-action-costs problem. The aim is to discover a plan that maximises utility – the difference between the overall utility of the achieved soft goals and plan cost.

As in the best-first search methods of Hart et al. (1968) and Dechter and Pearl (1985), preference within planning is often represented as a function to be minimised. Planning with action costs and such functions has a well-developed following. In the over subscription and partial satisfaction planners described above, planning aims to minimise action costs while maximising a measure of goal utility [Keyder and Geffner (2009); Benton et al. (2009); Smith (2004)]. Nguyen et al. (2009) consider a trade-off between multiple objective functions (such as plan cost, or plan makespan – the minimum time required to execute the plan) in the search for a solution. A range of alternative works consider planning with action costs, objective functions, and utilities [Borowsky and Edelkamp (2008); Benton et al. (2005); Keyder and Geffner (2008); Fabre et al. (2010); Karpas and Domshlak (2009)] – a collection so numerous only a small subset can be mentioned.

In this section, I have by no means described the full range of existing approaches that plan with a representation of preference. I have, however, described a spectrum of ways in which preference is used in planning that is representative of this work.

8.2 Argumentation-Based Search (ABS)

The ability to faithfully represent and capture the preferences of a human planner or agent within goal-directed search is of great importance. This is particularly the case in planning applications available over the web and in personal software – the ability to elicit and manage a wide variety of preference types increases the utility of these tools as

human decision-making aides. In this section, I present an argumentation-based search (ABS) algorithm for use in the solution of preference-based planning problems. This search method discovers a path through a tree-structured state-space, from an initial to a goal state, given a set of preferences expressed in the form of comparison schemes (as per Definition 4.3.1). This chapter, as described in its introduction, is interested in preference-guided search that supports the use of general preference expressions – each preference describing one way in which a pair of problem solutions can be compared.

The search problem considered in this chapter is shown in Definition 8.2.1 – attention restricted to tree-structured state-spaces. ABS has been developed to discover a preferred solution to this problem – the definition of optimality it uses shown in Definition 8.2.2.

Definition 8.2.1 A search problem is a tuple $\mathcal{P} = \langle \mathbf{S}, f_{SUCC}, S'_0, f_{GOAL} \rangle$, where:

1. $\mathbf{S}$ is a finite, discrete, tree-structured state-space and $S'_0 \in \mathbf{S}$ the initial state;
2. $f_{SUCC} : \mathbf{S} \rightarrow \vec{\mathbf{S}}$ is a successor function mapping a state $s \in \mathbf{S}$ to the set of states that can be transitioned to from it by an action or operator;
3. $f_{GOAL} : \mathbf{S} \rightarrow \{true, false\}$ returns *true* if and only if its input $s \in \mathbf{S}$ is a goal state;
4. A solution to the search problem $\mathcal{P}$ is a path (a sequence of state transitions) from the state S'_0 to a state $s \in \mathbf{S}$ such that $f_{GOAL}(s)$ returns *true*.

In Section 8.2.1, the definition of an optimal (most preferred) solution to the problem of Definition 8.2.1 is presented. This is followed with a description of the ABS algorithm, together with an example of its application. The use of ABS in the creation of an argumentation-based GOLOG interpreter (AR-GOLOG) is discussed in Section 8.3. An evaluation of AR-GOLOG is provided in Section 8.4. This evaluation considers the frequency with which AR-GOLOG finds a most preferred program execution within a travel planning domain. The results of the AR-GOLOG interpreter on a spacecraft control domain are compared with those of the heuristic interpreter of Chapter 6.

8.2.1 A Standard of Optimality

Let $\mathcal{O}$ denote the set of solutions to a search problem (as shown in Definition 8.2.1). Let us assume that an agent seeking to use ABS is of the type described in Definition 4.2.2 – able to apply the ADM framework of Chapter 4 to a set of choices. This agent has: a

set of comparison schemes SCH (Section 4.3) representing preference; an ability to create practical $\mathcal{A}_p$ and epistemic $\mathcal{A}_e$ arguments over those choices on the basis of SCH and a deductive system Γ (Section 4.4); an ability to determine conflict between and preference over these arguments; a decision-rule RL (Section 4.6); and a semantics Sem upon which argument acceptability is defined. Further assume that the agent's ability to determine conflict between and preference over arguments is embodied within two functions $f_{conflict}$ and f_{pref}. The function $f_{conflict}(a_1, a_2)$ (respectively $f_{pref}(a_1, a_2)$) returns true for arguments a_1 and a_2 if and only if a_1 conflicts with a_2 (respectively a_1 is preferred to a_2) and false otherwise. The optimal solutions of $\mathcal{O}$ lie within the most preferred subset found through an application of the ADM formed by this agent (Definition 8.2.2).

Definition 8.2.2 Let $\mathcal{P}$ be a search problem (Definition 8.2.1) and $\mathcal{O}$ its solutions. Let $AS_{ADM} = \langle \mathcal{O}, \mathcal{A}_e, \mathcal{A}_p, \mathcal{X}, RL \rangle$ be an ADM (Definition 4.2.1) applied over $\mathcal{O}$ by an agent AGT (Definition 4.2.2). The most preferred subset of $\mathcal{O}$ for AGT is the set $RL(GR)$ where GR is the dominance graph formed over the choices in $\mathcal{O}$ by AS_{ADM} (Definition 4.6.3).

The ABS algorithm, described in Section 8.2.2, is designed to discover a solution to a search problem $\mathcal{P}$ – exploring only part of the problem state-space – that is most preferred for an agent, as defined in Definition 8.2.2. In all but simple problems, it is not possible for an agent to enumerate the set of all solutions and then apply an ADM.

8.2.2 The Search Algorithm

The ABS algorithm (Listing 8.2.1) conducts a similar style of search to that of the heuristic search planners HSP [Bonet and Geffner (2001)] and FF [Hoffman and Nebel (2001)]. Its goal is to find a sequence of state transitions that guide search from an initial to a goal state. Given a collection of available states to transition to, at a given stage in search, a heuristic evaluation is performed to select the state most likely to lead to an optimal solution. This state is transitioned to, and the process repeated on the states that can be transitioned to from this new position. Search ends when a goal state is discovered, or when a state is reached for which no further transitions can be made. In contrast to search within HSP and FF, a process of argumentation is used within ABS to select transitions.

Listing 8.2.1 Argumentation-Based Search (:: denotes concatenation).

Input:
$\mathcal{P} = \langle \mathbf{S}, f_{SUCC}, S'_0, f_{GOAL} \rangle$
$AGT = \langle \Gamma, SCH, Sem, f_{conflict}, f_{pref}, RL \rangle$
$N =$ Sampling Index

Algorithm:

$Curr \leftarrow S'_0$
$Path \leftarrow \emptyset$
while $\neg f_{GOAL}(Curr)$ **do**
  $Next \leftarrow f_{SUCC}(Curr)$
  if $Next = \emptyset$ **then**
    return failure
  else
    $s' \leftarrow$ EVAL$(Next, \mathcal{P}, AGT, N)$
    $Path \leftarrow Path :: \{s'\}$
    $Curr \leftarrow s'$
  end if
end while
return $Path$

The ABS algorithm is shown in Listing 8.2.1. ABS operates on a given search problem $\mathcal{P}$ (of the form shown in Definition 8.2.1) for an agent AGT (whose properties are described in Definition 4.2.2) with a sampling index N. This sampling index is used in the evaluation of choices during search. Starting at the initial state of the given search problem $\mathcal{P}$, ABS selects a state to transition to at each stage in search, from an available collection of successor states $Next$, on the basis of an argumentation-based evaluation (through the use of an EVAL function). This function is described in Listing 8.2.2. Search repeats this process of selection until a goal state is discovered. As in FF and HSP, if a state is reached in which no transitions can be made, search fails to find a solution.

Consider the function EVAL$(Next, \mathcal{P}, AGT, N)$ in Listing 8.2.2, where: $Next$ is the set of states available to transition to from a current state; $\mathcal{P}$ characterises the problem being solved; AGT is a tuple that describes the agent solving the problem (as defined in Definition 4.2.2); and N is a sampling index ($N > 0$). EVAL considers each of the available states $s_i \in Next$ and finds a set of N distinct *relaxed* solutions to the search problem $\mathcal{P}$ with s_i as its initial state, placing them in a set called $Samples$. This set of samples is designed to

Listing 8.2.2 EVAL($Next$, $\mathcal{P}$, AGT, N): The selection of a most preferred transition.

Input:
$Next = [s_1, \ldots, s_n]$
$\mathcal{P} = \langle \mathbf{S}, f_{SUCC}, S'_0, f_{GOAL} \rangle$
$AGT = \langle \Gamma, SCH, Sem, f_{conflict}, f_{pref}, RL \rangle$
$N =$ Sampling Index

Algorithm:

$Samples \leftarrow \emptyset$
for each $s_i \in Next$ **do**
    $Solutions \leftarrow$ SAMPLE(s_i, $\mathcal{P}$, N)
    $Samples \leftarrow Samples \cup Solutions$
end for
$MPS \leftarrow$ APPLYADM($Samples$, AGT)
$s' \leftarrow$ SELECT(MPS, $Next$, f_{GOAL})
return s'

be representative of the range of paths possible from s_i to a goal state. These samples are found by the function SAMPLE(s_i, $\mathcal{P}$, N), whose definition is problem-dependent. This function will be implemented differently, for example, if the search problem is expressed in the STRIPS formalism [Fikes and Nilsson (1971)] or in the form of a GOLOG program (Section 5.3) requiring interpretation. In Section 8.3, argumentation-based search is applied to create a preference-guided GOLOG interpreter. In the process, an implementation of SAMPLE to find (relaxed) executions of a GOLOG program is presented.

Once a set of N samples is found for each available next state at the given search stage, the ADM system of Chapter 4 is applied to the solutions in this set. This set approximates the set of complete solutions $\mathcal{O}$ to the search problem $\mathcal{P}$. The agent AGT applies an ADM as described in Definition 8.2.2 to find a most preferred subset $MPS \subseteq \mathcal{O}^*$[6]. As per Definition 4.6.3, the set MPS found by the decision-rule RL is always non-empty for a non-empty $\mathcal{O}^*$. This step is performed by the APPLYADM($Samples$, AGT) function.

The function SELECT(MPS, $Next$, f_{GOAL}), as shown in Listing 8.2.3, selects a state within $Next$ (the available set of next states) to transition to in the search for a goal state to the problem being solved. This function selects a state $s_i \in Next$ that lies on a path p within the most preferred subset of sample solutions MPS. Preference is given to goal states within $Next$ that meet this criterion. The relation MAPS(s_i, p) in Listing 8.2.3 holds

[6] $\mathcal{O}^*$ denotes the approximation of $\mathcal{O}$.

Listing 8.2.3 SELECT(MPS, $Next$, f_{GOAL})

Input:
MPS = Set of most preferred sample solutions
$Next = [s_1, \ldots, s_n]$ = States available to transition to
f_{GOAL} = Goal test of the search problem

Algorithm:

if $\exists s_i \in Next.\ f_{GOAL}(s_i) \wedge \exists p \in MPS.$ MAPS(s_i, p) **then**
    return Such a s_i
else
    return s_i such that $s_i \in Next \wedge \exists p \in MPS.$ MAPS(s_i, p)
end if

if the state s_i leads to the solution path p. The state selected by this function is returned by EVAL (see Listing 8.2.1) and transitioned to by the ABS algorithm. If the selected state is a goal state, search ends and the accumulated solution path is returned as a solution. In Proposition 8.2.1, the properties of ABS as it is presented in Listing 8.2.1 are described.

Proposition 8.2.1 *The* ABS *algorithm of Listing 8.2.1 is not complete or optimal (admissible).*

Proof: The ABS algorithm is not complete as it does not maintain unexplored choices on a search frontier. It selects states to transition to in the search for a goal state and if a state is reached in which no transitions can be made, search ends in failure. Search is not guaranteed to find a solution even if one exists. The ABS algorithm is not optimal (admissible) for a number of reasons. States available during search are selected on the basis of a heuristic evaluation (albeit a non-traditional one). A select number of approximate (relaxed) solutions of the search problem are found for each state. A process of argumentation (the application of the ADM of Chapter 4) is applied to these approximations to determine which transitions *might* lead to a most preferred solution. Choices are made on the basis of approximate information, and are not retracted. Even if backtracking does take place upon reaching a dead-end[7], the approximate solutions that are used to guide search may not be actual solutions, and only a select number of these are generated.

□

[7]The argumentation-based interpreter presented in Section 8.3 supports backtracking as a result of its implementation in SWI Prolog.

8.2.3 An Example in Travel Planning

This section presents an example that shall be used to demonstrate the concepts described within this chapter. This example lies within a travel planning domain, characterised by a situation calculus basic action theory (BAT, Definition 5.1.1).

Example 8.2.1 A planner seeks to travel from one European city to another, while having sight seeing adventures in the cities it visits. The planning environment is a simplified map of Spain and Portugal, consisting of: 18 cities; 25 road connections between cities; 4 connections by sea; 3 flight points between which the planner can travel by plane; and up to 3 sight seeing adventures per city. Each adventure has up to 4 properties ranging from active and educational, to scenic and relaxing. The planner can drive between cities connected by road (consuming fuel); take a boat between cities connected by sea; fly between flight points; and have a sight seeing adventure when in a city (if one is available). A search problem within this domain is described in Example 8.2.3.

Figure 8.1 shows the section of Europe just described. The cities Lisbon, Madrid, Seville, and Barcelona, each have sight seeing adventures available to their visitors. These adventures each have a subset of the following attributes:

[*Outdoors, By the sea, Religious, Active, Educational, Indoors, Historical, Scenic*]

Table 8.1 defines the sight seeing adventures available to this planner, and their attributes.

The situation calculus BAT that characterises the travel domain presented in Example 8.2.1 is now described. The situation dependent and rigid fluents of this domain are shown in Table 8.2. In this domain, a planner has four available actions to choose from: $fly(a, b)$ allows the planner to fly from city a to b; $drive(a, b)$ allows the planner to drive from city a to b; $boat(a, b)$ allows the planner to sail from city a to b; and $sightsee(v, a)$ allows the planner to have the sight seeing adventure v in city a. The precondition axioms for each of these actions are shown in Definition B.1.1 of Appendix B.

These axioms state that the planner can drive from a city a to a city b in situation s if it is currently in city a, it has never been to city b, there is a road connection between cities a and b, and it has enough fuel to drive the distance (the fuel used by the drive is

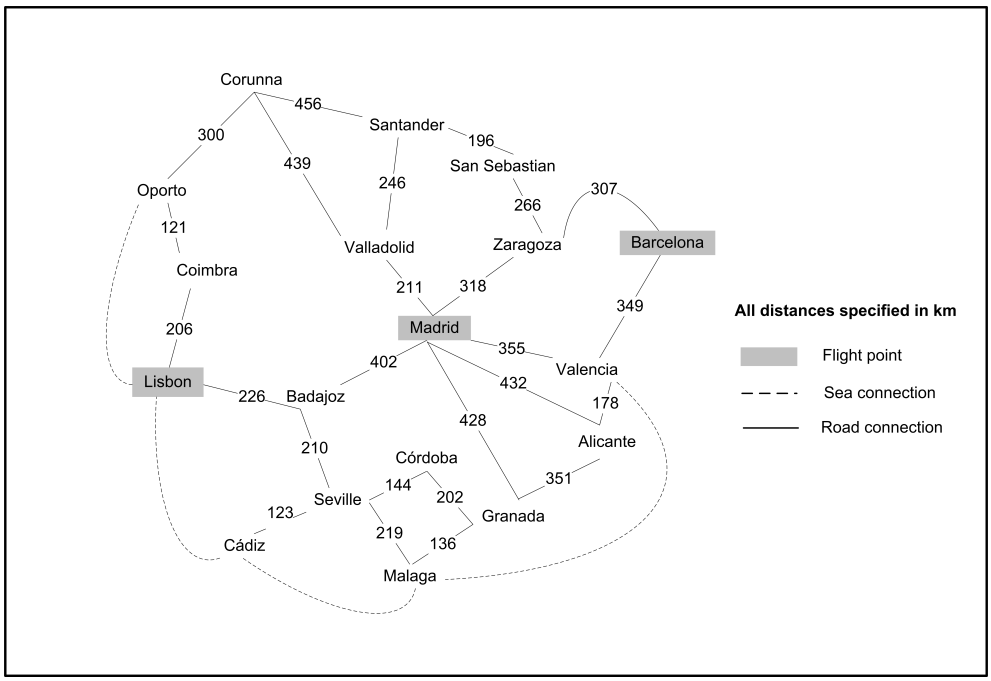


Figure 8.1: Section of a simplified map of Spain and Portugal, consisting of: 18 cities; 25 road connections between cities; 4 connections by sea; and 3 flight points.

CITY	ADVENTURE	PROPERTIES
Lisbon	Belém	Outdoors, Historical, Scenic
	Oceanarium	Indoors, Educational
	Churches Tour	Historical, Religious
Madrid	Prado Museum	Educational, Indoors
	Madrid Palace	Indoors, Historical
	Walking Tour	Outdoors, Scenic, Active
Seville	Port	Outdoors, By the sea, Scenic
	Walking Tour	Active, Outdoors, Scenic
Barcelona	Harbor	By the sea, Outdoors, Scenic
	Sagra Church	Historical, Religious

Table 8.1: Sight seeing adventures and their attributes.

available in situation s). Similarly, the planner can fly from city a to b if cities a and b are flight points, it has never been to city b, and it is currently in city a. The planner can sail from city a to b if it is currently in city a, it has never been to city b, and there is a sea connection between the two cities. The planner can have a sight seeing adventure v at city a if the adventure is available in that city, and it has not been experienced in the past.

At(a, s)	The planner is at city a in situation s.
Been(a, s)	The planner has been to city a by situation s.
Road(a, b)	There is a road connection between cities a and b.
Fpoint(a)	Holds if and only if city a is a flight point.
SeaPath(a, b)	There is a sea connection between cities a and b.
Adventure(v, a)	v is an available sight seeing adventure in city a.
Had(v, a, s)	Adventure v in city a has been had by situation s.
FuelOk(f, s)	At least f units of fuel remains in situation s.
FuelUsed(α, f_u, s)	Action α consumes f_u units of fuel in situation s.
FuelSpent(f, s)	f units of fuel have been consumed by situation s.
Capacity(f)	f units of fuel can be used over the course of travelling.
Distance(a, b, d)	d is the distance (in kilometres) between cities a and b.
Weather(w)	The weather in the planner's environment is w.
Num(att, n, s)	n denotes the number of adventures with attribute att had by situation s.
Order(o)	o is a strict, total ordering over the attributes of sight seeing adventures ($o = [a_1, a_2, \ldots, a_n]$) where each a_i is infinitely preferred to each a_j such that $j > i$.
Numbers(o, $list$, s)	$list$ is an array, where $list$[i] denotes the number of adventures with attribute o[i] had by situation s.
Less(n_1, n_2)	Holds if $n_1 < n_2$.
Carnival(c)	Holds if it is carnival time in city c.
OneAdvent(s)	Holds if at least one adventure has been had by s.
Attribute(att, v, a)	Holds if att is an attribute of adventure v in city a.

Table 8.2: Fluents and rigid predicates in the travel planning domain.

The successor state axioms for the non-rigid fluents of Table 8.2 are shown in Definition B.1.2 of Appendix B. A subset of these axioms are now (informally) described. The planner is at a city a in situation $do(\alpha, s)$ [At(a, $do(\alpha, s)$)] if the action α takes the planner to a (by car, for example), or the planner was at a in situation s and action α has not moved the planner on. The planner has had a sight seeing adventure v in a city a by $do(\alpha, s)$ [Had(v, a, $do(\alpha, s)$)] if the action α takes the planner on that adventure, or the adventure was had by situation s. The planner has been to a city a by $do(\alpha, s)$ [Been(a, $do(\alpha, s)$)] if the action α takes the planner to a, or the city has been visited by situation s.

To demonstrate ABS in this domain, an example set of initial situation axioms is provided in Example 8.2.2. These axioms refer to the fluents listed in Table 8.2.

Example 8.2.2 In the initial situation S_0, the planner is at Valladolid [At(Valladolid, S_0)], has been only to Valladolid [Been(Valladolid, S_0)], and has 2000 units of fuel available for any driving trips it takes during its journey [Capacity(2000)]. No adventures have been performed in S_0 – hence, $\forall att.$ Num(att, 0, S_0) and ¬OneAdvent(S_0). There is a predicate Attribute(att, v, a) for each attribute of each adventure v of each city a. The predicate Numbers(o, $list$, S_0) holds only if $list$ is an array of zeros, of the same length as the attribute order o. The attribute ordering of this planner is denoted Order(o), where:

$$o = [\textit{Outdoors, By the sea, Religious, Active, Educational, Indoors, Historical, Scenic}]$$

Listing 8.2.4 presents a GOLOG program whose execution shall be found through the use of ABS. In Section 8.3 a GOLOG interpreter that applies the principles of ABS within its transition axioms is presented. In Section 8.2.4, this interpretation is treated as a search through a space of configurations (program-situation pairs) from an initial to a final configuration – using the transition axioms of a standard interpreter as the successor function within a search problem defined as in Definition 8.2.1. In Listing 8.2.4, d is the destination city of the planner, and City(c) denotes that c is a city. The remaining fluents (shown in their situation restored form) are described in Table 8.2. TRAVEL requires the planner to repeatedly select a city to travel to by a chosen method of transport, or have a sight seeing adventure in its current city (if available), until it reaches its destination. Example 8.2.3 shows the search problem (as per Definition 8.2.1) describing the search for a program execution. Let us assume that the destination of the planner is Càdiz.

Example 8.2.3 Let $\mathcal{P}_{TRAVEL} = \langle \mathbf{S}, f_{SUCC}, S_0', f_{GOAL} \rangle$ be a search problem such that:

1. S_0' is an initial configuration – the program-situation pair (TRAVEL(Càdiz), S_0). The axioms characterising S_0 are defined in Example 8.2.2;
2. $f_{SUCC}(n)$ denotes the set of configurations that can be transitioned to from configuration n by the transition axioms of a standard interpreter (Section 5.3.1);
3. $f_{GOAL}(n)$ holds if n if a final configuration by the final axioms of this interpreter;

Listing 8.2.4 Controlling GOLOG program for a travel planner.

```
proc TRAVEL(d)
    while ¬At(d) do
        π c. [City(c)? : π cr. [At(cr)? : (fly(cr, c) | boat(cr, c) | drive(cr, c))]] |
         π c′. [At(c′)? : π adv. [Adventure(adv, c′)? : sightsee(adv, c′)]]
    end
end
```

4. **S** is the set of configurations reachable from S'_0 by f_{SUCC}.

The application of ABS to the search problem described in Example 8.2.3, within the travel domain, is demonstrated in the next section.

8.2.4 A Demonstration of ABS

This section demonstrates how ABS can be used to find a preferred solution to the problem described in Example 8.2.3. This example will also be used to experimentally evaluate the argumentation-based GOLOG interpreter of Section 8.3. Let BETTY denote an agent of the form described in Definition 4.2.2. Each comparison scheme of this planner (BETTY) – representing its preferences – is defined as in Definition 4.3.1 of Section 4.3. These schemes are presented in Definitions 8.2.5 to 8.2.9. I first describe how BETTY constructs epistemic arguments on the basis of a deductive system Γ. These arguments are used to dispute the practical arguments derived from BETTY's comparison schemes.

Epistemic arguments are constructed in support of beliefs – each argument containing a set of premises that infer a conclusion by the deductive system Γ of the planner. In this domain, beliefs are denoted Bel(f) (fluent f is believed) and NotBel(f) (f is not believed), and are inferred as described in Definition 8.2.3. I follow this with a definition of an epistemic argument within the travel domain, and BETTY's deductive system Γ.

Definition 8.2.3 Let $Support(b, pr)$ be a predicate that states that the belief b holds if and only if each premise within the set pr holds. A belief b holds if an only if $Sholds(b)$ holds.

$$Sholds(b) \equiv Support(b, []) \vee [\neg Support(b, []) \wedge \exists pr.Support(b, pr) \wedge \forall p \in pr.Sholds(p)]$$

In Section 4.4 of Chapter 4, epistemic arguments are defined as abstract entities – entities with an abstract set of premises that infer an abstract conclusion by the deductive

$Support(Bel(Weather(Hot)), [Bel(Summer)])$
$Support(Bel(Carnival(Barcelona)), [Bel(Summer)])$
$Support(Bel(Summer), [])$
$Support(NotBel(Weather(Hot)), [Bel(Summer), Bel(ColdSpell)])$
$Support(NotBel(Carnival(Barcelona)), [Bel(Summer), Bel(Cancel(Barcelona))])$
$Support(Bel(ColdSpell), [])$
$Support(Bel(Cancel(Barcelona)), [])$

Table 8.3: The Support predicates, where: Weather(w) and Carnival(c) are defined as in Table 8.2, Bel(Summer) holds if it is believed to be summer-time, Bel(Cancel(Barcelona)) holds if the Barcelona carnival is believed to be cancelled, and Bel(ColdSpell) holds if it is believed that the current weather is unseasonably cold.

system of the agent constructing the arguments. In this domain, BETTY instantiates these arguments as shown in Definition 8.2.4 – using the *Sholds* axiom of Definition 8.2.3.

Definition 8.2.4 An epistemic argument in the travel domain is a tuple $\langle H, h \rangle$ (as per Definition 4.4.2) where h is the predicate Bel(f) or NotBel(f), f is a fluent, and H is a set of beliefs (Bel(...) or NotBel(...)) required to infer h by *Sholds* of Definition 8.2.3, and a collection of Support predicates of the form shown in Definition 8.2.3. Moreover, it is the case that no subset $H' \subset H$ is able to infer h in the manner described above.

Given the collection of Support predicates shown in Table 8.3, BETTY is able to construct a number of epistemic arguments – shown in Example 8.2.4, below.

Example 8.2.4 The following epistemic arguments can be constructed, as per Definition 8.2.4, using the *Sholds* axiom of Definition 8.2.3 and the Support predicates of Table 8.3.

$$a_{e1} = \langle \{Bel(Summer)\}, Bel(Weather(Hot)) \rangle$$

$$a_{e2} = \langle \{Bel(Summer), Bel(ColdSpell)\}, NotBel(Weather(Hot)) \rangle$$

$$a_{e3} = \langle \{Bel(Summer)\}, Bel(Carnival(Barcelona)) \rangle$$

$$a_{e4} = \langle \{Bel(Summer), Bel(Cancel(Barcelona))\}, NotBel(Carnival(Barcelona)) \rangle$$

Example 8.2.5 Let Γ denote the deductive system of BETTY, containing: the BAT of Section 8.2.3, an *Sholds* axiom (Definition 8.2.3), and the collection of Support predicates in Table 8.3. Situation calculus formulae inferred by Γ are entailed by the axioms of this BAT. The *Sholds* axiom of Γ infers the belief (Bel(f)) or disbelief (NotBel(f)) of a fluent through the use of a collection of Support predicates outlining defeasible relationships.

BETTY's comparison scheme set SCH is now presented, consisting of the five schemes described below. In the first scheme, journeys (final configurations – each a program-situation pair – of the program shown in Listing 8.2.4) are compared lexicographically on the basis of the adventures had in them (using an ordering over adventure attributes).

Definition 8.2.5 Let Order(o) and Numbers(o, l_i, s_i) be defined as in Table 8.2, j_1 and j_2 be two journeys, and $j_i[s]$ the situation within j_i. Let l_1 and l_2 be defined such that Numbers(o, l_1, $j_1[s]$) and Numbers(o, l_2, $j_2[s]$). The scheme sc_{lexi} compares the journeys j_1 and j_2 by lexicographically comparing the arrays l_1 and l_2 (whose indices start at 1).

$$sc_{lexi} = \langle j_1, j_2, \{Order(o), Numbers(o, l_1, j_1[s]), Numbers(o, l_2, j_2[s]), LexiMax(l_1, l_2)\}, j_1 \succ_{sc_{lexi}} j_2 \rangle$$

where $LexiMax(l_1, l_2)$ holds if and only if $\exists i > 0.\ \forall j < i\ (l_1[j] = l_2[j])$ and $l_1[i] > l_2[i]$.

In the second scheme, one journey j_1 is preferred to another j_2 if the weather is believed to be hot and journey j_1 involves less outdoor sight seeing adventures.

Definition 8.2.6 Let j_1 and j_2 be journeys, and $j_i[s]$ the situation within journey j_i. Let the fluents Num(att, n, s_i), Less(n_1, n_2), and Weather(w) be defined as described in Table 8.2. The comparison scheme sc_{hot} is defined as follows.

$$sc_{hot} = \langle j_1, j_2, \{Bel(Weather(Hot)), Num(Outdoors, n_1, j_1[s]), \\ Num(Outdoors, n_2, j_2[s]), Less(n_1, n_2)\}, j_1 \succ_{sc_{hot}} j_2 \rangle$$

In the third scheme, one journey j_1 is preferred to another j_2 if, provided it is not believed to be carnival time in Barcelona, j_1 visits Lisbon (and not Barcelona), while j_2 visits Barcelona (and not Lisbon). BETTY prefers to visit Lisbon over Barcelona.

Definition 8.2.7 Let j_1 and j_2 be journeys, fluents Been(c, s_i) and Carnival(c) be defined as described in Table 8.2, and $j_i[s]$ the situation within journey j_i. The comparison scheme sc_{lisbon} is defined as follows, where L = Lisbon and B = Barcelona.

$$sc_{lisbon} = \langle j_1, j_2, \{NotBel(Carnival(B)), Been(L, j_1[s]), \neg Been(B, j_1[s]), \\ Been(B, j_2[s]), \neg Been(L, j_2[s])\}, j_1 \succ_{sc_{lisbon}} j_2\rangle$$

In the fourth scheme, one journey j_1 is preferred to another j_2 if, provided it is believed to be carnival time in Barcelona, j_1 visits Barcelona while j_2 does not.

Definition 8.2.8 Let j_1 and j_2 be journeys, fluents Been(c, s_i) and Carnival(c) be defined as described in Table 8.2, and $j_i[s]$ the situation within journey j_i. The comparison scheme sc_{barc} is defined as follows, where B = Barcelona.

$$sc_{barc} = \langle j_1, j_2, \{Bel(Carnival(B)), Been(B, j_1[s]), \neg Been(B, j_2[s])\}, j_1 \succ_{sc_{barc}} j_2\rangle$$

In the fifth, and final, of BETTY's comparison schemes, one journey j_1 is preferred to another j_2 if j_1 involves at least one sight seeing adventure while j_2 involves none.

Definition 8.2.9 Let j_1 and j_2 be journeys, fluent OneAdvent(s_i) be defined as in Table 8.2, and $j_i[s]$ the situation within journey j_i. The scheme sc_{advent} is defined as follows.

$$sc_{advent} = \langle j_1, j_2, \{OneAdvent(j_1[s]), \neg OneAdvent(j_2[s])\}, j_1 \succ_{sc_{advent}} j_2\rangle$$

The planner I have described within this section, BETTY, is now equipped with the necessary tools to construct practical $\mathcal{A}_p$ arguments (as per Definition 4.4.1) over a set of journeys from Valladolid to Càdiz, and epistemic arguments $\mathcal{A}_e$ (as per Definition 4.4.2) that dispute them. BETTY determines conflict and preference over arguments in a similar manner to the agent ALFRED of Chapter 4 in Examples 4.5.1 and 4.5.3 – through undercutting and rebuttal, and a preference ordering over schemes (see Example 8.2.6).

Let us assume that BETTY uses the Schwartz set (Example 4.6.3) as its decision-rule RL, and the preferred extension semantics (Definition 3.2.3) as Sem.

Example 8.2.6 Let $\mathcal{I}_{SCH}$ be an ordering over the schemes of Definitions 8.2.5 to 8.2.9, such that $\{sc_{hot}, sc_{lexi}, sc_{lisbon}, sc_{barc}, sc_{advent}\}$ are listed in order of strict priority (from highest to lowest). Let $s(a_i)$ denote the scheme used in argument $a_i \in \mathcal{A}_p$. An argument $a_1 \in \mathcal{A}_p$ is preferred to an argument $a_2 \in \mathcal{A}_p$ ($f_{pref}(a_1, a_2)$ returns true) only if $s(a_1)$ is preferred to $s(a_2)$ by $\mathcal{I}_{SCH}$. Conflict between practical arguments is defined as in Definition 4.5.1 (Chapter 4). An argument $a_1 \in \mathcal{A}_e$ is preferred to an argument $a_2 \in \mathcal{A}_e$ ($f_{pref}(a_1, a_2)$ returns true) if a_1 is more specific ($Premises(a_2) \subset Premises(a_1)$)[8]. An argument $a_1 \in \mathcal{A}_e$ conflicts with an argument $a_2 \in \mathcal{A}_e \cup \mathcal{A}_p$ ($f_{conflict}(a_1, a_2)$ returns true) only if the conclusion of a_1 is Bel(f) [NotBel(f)] and a premise or the conclusion of a_2 is NotBel(f) [Bel(f)].

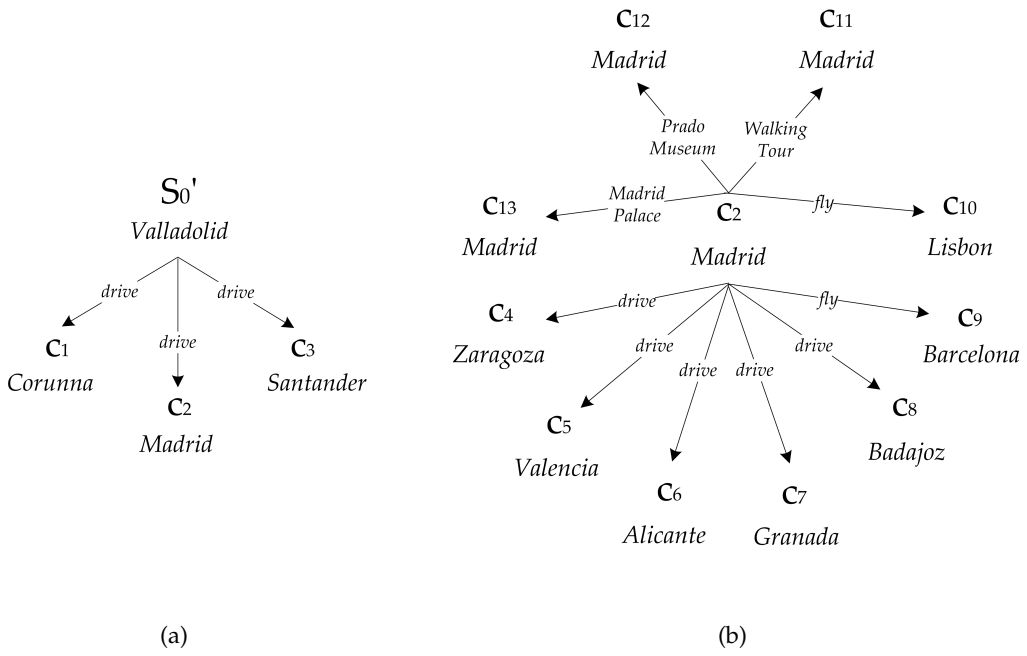
Recall the search problem in Example 8.2.3 for the travel planning domain ($\mathcal{P}_{TRAVEL}$). In this problem, for the planner described in this section (BETTY) and the domain description of Section 8.2.3, there are 1,680 executions of the program in Listing 8.2.4 that allow BETTY to travel from Valladolid to Càdiz (without the performance of any sight seeing adventures). If each of the adventures of Table 8.1 are available, it is not computationally feasible to determine how many solutions exist. The initial stages of an ABS search to discover an execution of the program of Listing 8.2.4 are shown below.

For the purposes of demonstration, a restriction is placed on the solutions of $\mathcal{P}_{TRAVEL}$ in Example 8.2.3. The execution that each solution describes must be (at most) 5 actions long. This restriction allows the generation of all solutions to $\mathcal{P}_{TRAVEL}$, upon which an ADM can be applied to the complete solution set. There are 82 distinct solutions to this problem, but only one is the most preferred (by Definition 8.2.2) for the planning agent BETTY. For brevity, the application of an ADM (the arguments generated, attack relation computed, dominance graph formed, and so on) to this set of 82 solutions is not described. Instead, the most preferred subset of solutions it discovers (as per Definition 8.2.2) is presented. This subset describes the following execution (presented informally):

$$\text{Valladolid} \overset{drive}{\rightarrow} \text{Madrid} \rightarrow \text{Walking Tour} \overset{fly}{\rightarrow} \text{Lisbon} \rightarrow \text{Visit Belém} \overset{boat}{\rightarrow} \text{Càdiz}$$

The sampling index N of search is set to $N = 1$. The first two stages in the search for a solution to $\mathcal{P}_{TRAVEL}$ with ABS are shown below. An implementation of the SAMPLE($Next$,

[8]Note that by the definition of preference and conflict within an ADM (Definition 4.5.1), epistemic arguments are preferred to practical arguments.



(a) (b)

Figure 8.2: (a) Stage I and (b) Stage II in the search for a preferred solution to $\mathcal{P}_{TRAVEL}$.

$\mathcal{P}$, N) function of Listing 8.2.1 is not defined. Demonstrated instead is how different samples produced from this function can both encourage search toward a most preferred solution, and steer it away. SAMPLE produces relaxed problem solutions – in this domain we assume that relaxation removes the need to check for adequate fuel when making a driving trip. In Section 8.3, an implementation of a GOLOG interpreter that uses ABS is presented – providing an instantiation of the SAMPLE function.

STAGE I: THE FIRST CHOICE

Figure 8.2(a) shows the first choice point of search – the planner BETTY can drive to Corunna (439 km), Madrid (211 km), or Santander (246 km)[9]. No sight seeing adventures are available in Valladolid. In Figure 8.2(a), each c_i is a configuration (a program-situation pair) and below each c_i is the location of BETTY (in the situation associated with c_i).

The function EVAL within Listing 8.2.2 finds one sample solution for each of the three available choices (with the SAMPLE function). The following three solutions may be generated (adhering to the restriction of 5 action plans) by this SAMPLE function ($N = 1$

[9]Recall from Example 8.2.2 that BETTY can drive a total of only 2000 km over the course of its plan.

solution for each choice, their situation components described informally):

$rs_1[s]$ = Valladolid $\overset{drive}{\rightarrow}$ Corunna $\overset{drive}{\rightarrow}$ Oporto $\overset{boat}{\rightarrow}$ Lisbon $\overset{boat}{\rightarrow}$ Càdiz
$rs_2[s]$ = Valladolid $\overset{drive}{\rightarrow}$ Madrid $\rightarrow$ Walking Tour $\overset{fly}{\rightarrow}$ Lisbon $\overset{boat}{\rightarrow}$ Càdiz
$rs_3[s]$ = Valladolid $\overset{drive}{\rightarrow}$ Santander $\overset{drive}{\rightarrow}$ Corunna $\overset{drive}{\rightarrow}$ Oporto $\overset{boat}{\rightarrow}$ Lisbon $\overset{boat}{\rightarrow}$ Càdiz

The function APPLYADM determines the most preferred subset of $Solutions = \{rs_1, rs_2, rs_3\}$ for the planning agent BETTY described in this section. An ADM is applied to these choices as described in Chapter 4. The epistemic arguments BETTY constructs in this domain are shown in Example 8.2.4 ($\mathcal{A}_e = \{a_{e1}, \ldots, a_{e4}\}$). The practical arguments constructed by BETTY over the set $Solutions$ with the comparison schemes $\{sc_{hot}, sc_{lexi}, sc_{lisbon}, sc_{barc}, sc_{advent}\}$ of Definitions 8.2.5 to 8.2.9 (as per Definition 4.4.1) are:

$a_{p1} = \langle sc_{lexi}, rs_2 \succ_{sc_{lexi}} rs_1 \rangle$ $a_{p3} = \langle sc_{hot}, rs_1 \succ_{sc_{hot}} rs_2 \rangle$ $a_{p5} = \langle sc_{advent}, rs_2 \succ_{sc_{advent}} rs_1 \rangle$
$a_{p2} = \langle sc_{lexi}, rs_2 \succ_{sc_{lexi}} rs_3 \rangle$ $a_{p4} = \langle sc_{hot}, rs_3 \succ_{sc_{hot}} rs_2 \rangle$ $a_{p6} = \langle sc_{advent}, rs_2 \succ_{sc_{advent}} rs_3 \rangle$

Argument a_{p1} reflects that journey rs_2 is preferred to rs_1 as the adventures had in $rs_2[s]$ are lexicographically preferred to those had in $rs_1[s]$ (in this case, $rs_1[s]$ involves no adventures). Argument a_{p2} expresses a similar statement between rs_2 and rs_3. Argument a_{p3} reflects that journey rs_1 is preferred to rs_2 as $rs_1[s]$ involves less outdoor adventures. A similar statement is expressed in argument a_{p4} between rs_3 and rs_2. Argument a_{p5} reflects that journey rs_2 is preferred to rs_1 as $rs_2[s]$ involves at least one adventure while $rs_1[s]$ does not. A similar statement is expressed in argument a_{p6} between rs_2 and rs_3.

On the basis of BETTY's definition of $f_{conflict}$ and f_{pref}, described in Example 8.2.6, it is found that: the arguments a_{p1} and a_{p5} rebut (are in conflict with) argument a_{p3}, while arguments a_{p2} and a_{p6} rebut argument a_{p4}. In addition, argument a_{e2} undercuts a_{p3} and a_{p4} (a_{e2} concludes that the weather is not hot, while the opposing premise is used in a_{p3} and a_{p4}), while arguments a_{e1} and a_{e2} rebut (one concluding that the weather is hot, and the other concluding that it is not), together with a_{e3} and a_{e4} (one concluding that it is carnival time in Barcelona and the other concluding that it is not).

On the basis of BETTY's ordering over comparison schemes (see Example 8.2.6), arguments a_{p3} and a_{p4} are preferred to a_{p1}, a_{p2}, a_{p5}, and a_{p6}, while arguments a_{p1} and a_{p2} are preferred to a_{p5}, and a_{p6}. Each epistemic argument is preferred to all $\{a_{p1}, \ldots, a_{p6}\}$,

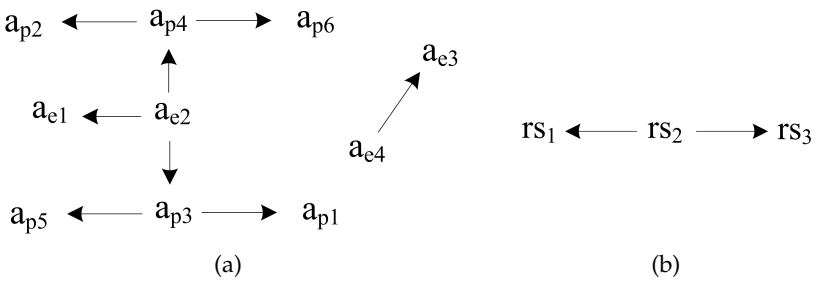


Figure 8.3: (a) Attack relation formed over the arguments in Stage I of search, (b) A dominance graph formed over the sample solutions rs_1, rs_2, and rs_3 in Stage I.

a_{e2} is preferred to a_{e1} (on the basis of specificity), and a_{e4} is preferred to a_{e3}. By Definition 4.6.1, Figure 8.3(a) displays the attack relation over this collection of arguments.

The arguments in Figure 8.3(a) sceptically accepted under the preferred semantics *Sem* are $\{a_{p2}, a_{p6}, a_{e4}, a_{e2}, a_{p5}, a_{p1}\}$. The dominance graph (Definition 4.6.2) formed over the sample solutions arising from choices $c_1 \ldots c_3$ is shown in Figure 8.3(b). By the Schwartz set decision-rule (Example 4.6.3), the most preferred solution is found to be rs_2, and hence the most preferred configuration c_2. The SELECT function selects the drive to Madrid at this choice point, resulting in the second choice point of Figure 8.2(b).

STAGE II: THE SECOND CHOICE

Figure 8.2(b) shows the choices available to BETTY in the second choice point of search. At this choice point, BETTY can drive to Zaragoza (318 km), Valencia (355 km), Alicante (432 km), Granada (428 km), and Badajoz (402 km). BETTY can also fly to Barcelona or Lisbon, or have a sight seeing adventure such as the Madrid Palace.

Let us now look at the relaxed sample executions that can be generated from each available choice at this stage in search (with the SAMPLE function).

$rs_4[s] = \text{Valladolid} \overset{drive}{\rightarrow} \text{Madrid} \overset{drive}{\rightarrow} \text{Zaragoza} \overset{drive}{\rightarrow} \text{Barcelona} \overset{fly}{\rightarrow} \text{Lisbon} \overset{boat}{\rightarrow} \text{Càdiz}$

$rs_5[s] = \text{Valladolid} \overset{drive}{\rightarrow} \text{Madrid} \overset{drive}{\rightarrow} \text{Valencia} \overset{drive}{\rightarrow} \text{Barcelona} \overset{fly}{\rightarrow} \text{Lisbon} \overset{boat}{\rightarrow} \text{Càdiz}$

$rs_6[s] = \text{Valladolid} \overset{drive}{\rightarrow} \text{Madrid} \overset{drive}{\rightarrow} \text{Alicante} \overset{drive}{\rightarrow} \text{Valencia} \overset{boat}{\rightarrow} \text{Malaga} \overset{boat}{\rightarrow} \text{Càdiz}$

$rs_7[s] = \text{Valladolid} \overset{drive}{\rightarrow} \text{Madrid} \overset{drive}{\rightarrow} \text{Granada} \overset{drive}{\rightarrow} \text{Malaga} \overset{boat}{\rightarrow} \text{Càdiz}$

$rs_8[s] = \text{Valladolid} \overset{drive}{\rightarrow} \text{Madrid} \overset{drive}{\rightarrow} \text{Badajoz} \overset{drive}{\rightarrow} \text{Lisbon} \rightarrow \text{Visit Belém} \overset{boat}{\rightarrow} \text{Càdiz}$

$rs_9[s]$ = Valladolid $\overset{drive}{\rightarrow}$ Madrid $\overset{fly}{\rightarrow}$ Barcelona $\overset{fly}{\rightarrow}$ Lisbon $\rightarrow$ Oceanarium $\overset{boat}{\rightarrow}$ Càdiz

$rs_{10}[s]$ = Valladolid $\overset{drive}{\rightarrow}$ Madrid $\overset{fly}{\rightarrow}$ Lisbon $\rightarrow$ Visit Belém $\rightarrow$ Churches Tour $\overset{boat}{\rightarrow}$ Càdiz

$rs_{11}[s]$ = Valladolid $\overset{drive}{\rightarrow}$ Madrid $\rightarrow$ Madrid Palace $\overset{fly}{\rightarrow}$ Lisbon $\rightarrow$ Churches Tour $\overset{boat}{\rightarrow}$ Càdiz

$rs_{12}[s]$ = Valladolid $\overset{drive}{\rightarrow}$ Madrid $\rightarrow$ Prado Museum $\overset{fly}{\rightarrow}$ Lisbon $\rightarrow$ Oceanarium $\overset{boat}{\rightarrow}$ Càdiz

$rs_{13}[s]$ = Valladolid $\overset{drive}{\rightarrow}$ Madrid $\rightarrow$ Walking Tour $\overset{fly}{\rightarrow}$ Lisbon $\rightarrow$ Churches Tour $\overset{boat}{\rightarrow}$ Càdiz

The function APPLYADM determines the most preferred subset of $Solutions = \{rs_4, \ldots, rs_{13}\}$ for the agent BETTY described in this section. An ADM is applied to these choices as described in Chapter 4 and demonstrated in the selection of the first choice in this example. The intermediate results discovered by this ADM are not presented. The most preferred subset of $Solutions$, denoted MPS, that it discovers is: $MPS = \{rs_{13}\}$. In the solution rs_{13}, BETTY opts to have a walking tour of Madrid before moving on to Lisbon by plane. The reasons for this selection are summarised in the following paragraphs.

The program execution within rs_{13} involves adventures that are outdoors, religious, active, historical and scenic. This solution is consequently preferred, by sc_{lexi} (Definition 8.2.5), to each of the other options (where BETTY's ordering over adventure attributes is shown in Example 8.2.2). Arguments constructed to rebut this use of the sc_{lexi} scheme, on the basis that the weather is hot (and hence, outdoor adventures should be minimised) are struck down by an (ultimately acceptable) argument in support of the belief that it is not hot (as was the case in the first stage of search). In addition, arguments are constructed at this stage in search preferring a visit to Barcelona based on the belief that it is carnival time – however, these are struck down by an accepted argument disputing this belief (as the carnival has been cancelled). The SELECT function selects to have a walking tour of Madrid at this choice point, leading to the third choice point of search.

At this third stage, the choices available to BETTY are similar to those available in the second – from having a sight seeing adventure, to moving on toward Zaragoza, Valencia, Alicante, Granada, Badajoz, or Barcelona. For brevity, the remaining stages of search are not described. Each stage follows the procedure outlined in the first two.

The ABS algorithm, in this instance, is on its way to finding the most preferred solution to $\mathcal{P}_{TRAVEL}$ – it has in the first two stages of search travelled to Madrid and experi-

enced a walking tour. These two actions form the beginning of the preferred solution:

$$\text{Valladolid} \overset{drive}{\rightarrow} \text{Madrid} \rightarrow \text{Walking Tour} \overset{fly}{\rightarrow} \text{Lisbon} \rightarrow \text{Visit Belém} \overset{boat}{\rightarrow} \text{Càdiz}$$

The relaxed solutions that are generated for each available choice at each choice point in search play an important role in guiding search toward a preferred solution. For example, consider the choice point in Stage I of search (Figure 8.2(a)) and the choice of travelling to Madrid by car. The relaxed solution generated for this choice involves a sight seeing adventure – having a walking tour of Madrid (rs_2). The presence of this adventure within this solution led to the selection of the first action – driving to Madrid. Imagine that instead of this solution, the following was generated in its place:

$$rs_2[s] = \text{Valladolid} \overset{drive}{\rightarrow} \text{Madrid} \overset{fly}{\rightarrow} \text{Lisbon} \overset{boat}{\rightarrow} \text{Càdiz}$$

The function APPLYADM would have determined that the entire set of generated solutions $\{rs_1, rs_2, rs_3\}$ was within the most prefered set MPS. No scheme would have distinguished either one of these choices as being preferred to another. In this case the SELECT function would have selected to travel to Corunna. As choices can not be retracted during search, ABS would not have found the most preferred solution to $\mathcal{P}_{TRAVEL}$.

8.2.5 An Implementation and Evaluation of ABS

In the next section, ABS is used in the development of a GOLOG interpreter designed to discover a most preferred execution of a program (given preference expressed in the form of comparison schemes over executions). This interpreter is evaluated within the travel domain of Section 8.2.3, for the agent BETTY of the previous section, to determine its success rate – the rate at which it discovers a most preferred solution. The definition of most preferred employed by ABS (Definition 8.2.2) partitions solutions into two sets – most preferred and the remainder. Hence, in the scenarios in which ABS does not find a preferred solution, it is not possible to analyse *how* good of a solution it does find. Moreover, in moderately complex domains, it is not possible to enumerate all possible solutions to determine which is the most preferred (to test the result obtained by search).

The argumentation-based interpreter AR-GOLOG is additionally evaluated on the spacecraft control domain of Section 7.1. In this domain, the quality of a solution –

the number of targets observed – is measurable. The quality of solutions found by AR-GOLOG are compared to those found by the three interpreters (including the heuristic interpreter developed in Chapter 6) described in Chapter 7, within this domain.

8.3 An Argumentation-Based GOLOG Interpreter

The GOLOG language, as described in Section 5.3, is a programming language for agents that provides high level instructions (a task specification) in the form of a program. These programs are interpreted, given a basic action theory (BAT, Definition 5.1.1) characterising the domain and environment of the agent, to find a program execution. This execution is a sequence of actions for the agent to perform to complete its desired task. I have presented a preference-guided GOLOG interpreter in Sections 6.4 and 6.6 – an interpreter designed to find a program execution that minimises a numeric evaluation function. In this chapter, I return to my goal of constructing preference-based planning techniques that are decoupled from the preferences they operate on. I develop an interpreter for GOLOG programs that: allows preference over executions to be expressed in the form of comparison schemes (Definition 4.3.1); and abstracts away from the nature of these preferences when selecting choices to make during interpretation. I instantiate and adapt the ABS algorithm of Section 8.2.2 in an implementation of this interpreter.

To create this interpreter, I adapt the transition semantics of the standard GOLOG interpreter of Section 5.3.1. A transition clause for each GOLOG construct is devised that follows the process of EVAL (Listing 8.2.2) when selecting one of a number of possible transitions to make during the interpretation of a program.

8.3.1 A Transition Semantics – An Overview

AR-GOLOG finds an execution of a GOLOG program in the same manner as a standard interpreter (Section 5.3.1) and the heuristic interpreter of Section 6.6 – with a set of transition axioms. The axioms that characterise AR-GOLOG, however, have been designed to reflect the principles of ABS. Recall from Section 5.3.1 that a transition semantics defines a transition and final axiom for each GOLOG construct of Table 5.2. The transition axioms

of a standard GOLOG interpreter are shown in Definition 5.3.1, and describe which configurations (δ', s') can be transitioned to by performing a single step of a program δ in a situation s. The situation s' is transitioned to, while program δ' remains to be performed. Each final axiom determines when a configuration is final – where no steps remain to be performed in its program – its situation defining an execution of the original program. The final axioms of the standard interpreter are shown in Definition 5.3.2.

AR-GOLOG is presented in terms of the SWI Prolog clauses that form its implementation. This argumentation-based interpreter uses the same final clauses as a traditional interpreter (see Definition 5.3.2 in Section 5.3.1) but defines new transition clauses for most of the GOLOG constructs of Table 5.2, denoted $transAR$ and $finalAR$ within AR-GOLOG. Definition 8.3.4 defines these transition clauses. Each $transAR$ clause for a non-deterministic construct (involving choice) shares a similar form. These clauses are an instantiation of the EVAL function of Listing 8.2.2. I first informally describe how these clauses operate, before formally defining their general form in Section 8.3.2.

Let us assume that the agent AGT employing AR-GOLOG is of the form described in Definition 4.2.2. The attributes of this agent form one of the inputs to the EVAL function of Listing 8.2.2. When faced with a non-deterministic construct (δ_c, s_c), an AR-GOLOG interpreter has the option of transitioning to one of a number of configurations [(δ_{c_1}, s_{c_1}), ..., (δ_{c_n}, s_{c_n})]. This set represents $Next$ in EVAL. For each (δ_{c_i}, s_{c_i}), ABS dictates that a set of N relaxed executions of the program δ_{c_i} in s_{c_i} be found, where N is a sampling index. These relaxed executions become choices in an application of the ADM of Chapter 4.

I implement the functionality of SAMPLE within EVAL by employing a variation of the relaxed GOLOG interpreter of Section 6.5. In this variation, the testing of action preconditions is determined by relaxed regression (Section 6.3.1) but no heuristic guidance is employed when making transitions. Let $RSols$ denote the set of N_R relaxed executions this (altered) relaxed interpreter finds of program δ_{c_i} in s_{c_i}. My implementation of SAMPLE is designed to find an even spread of N of these executions – N solutions spaced $\frac{N_R}{N}$ apart within the $RSols$ set (without needing to explicitly enumerate the entire $RSols$ solution set). This implementation of SAMPLE is presented in Section B.2 of Appendix B. An implementation of the APPLYADM function within EVAL – used to find a most preferred

subset of the relaxed solution set found by SAMPLE – is presented in Section 8.3.2.

Each *transAR* clause within the AR-GOLOG interpreter selects a transition to make following the procedure of SELECT, described in Listing 8.2.3. A transition to a configuration (δ_{c_i}, s_{c_i}) is made if it leads to a relaxed execution within the most preferred subset found by APPLYADM, with priority given to configurations that are final (for which $finalAR(\delta_{c_i}, s_{c_i})$ holds). AR-GOLOG finds an execution of a GOLOG program in the same manner as its traditional predecessor (Section 5.3.1). The *transAR* clauses of AR-GOLOG are repeatedly applied, starting with an initial configuration (δ, S_0), until a final configuration (δ_f, s_f) is found – the situation s_f denoting an execution of the program δ.

8.3.2 Transition Clauses of AR-GOLOG

Each *transAR* clause of AR-GOLOG has a general form, shown in Definition 8.3.4. This template refers to a number of predicates: *sample* (representing the SAMPLE function used in Listing 8.2.2); *applyADM* (representing APPLYADM used in Listing 8.2.2); and *select* (representing SELECT in Listing 8.2.3). Before defining each *transAR* clause of AR-GOLOG, an implementation of these predicates is presented. These implementations are described in either clause form or pseudocode[10]. The first of these predicates represents an implementation of SAMPLE, referred to within the EVAL function of Listing 8.2.2.

SAMPLE is designed to discover N relaxed solutions of a search problem – in this case, executions of a GOLOG program. Definition B.2.2 in Appendix B provides my implementation of SAMPLE, denoted $sample(N, \delta, S, H, Solutions)$ where: N is the number of relaxed executions of program δ in situation S required; H is a finite action horizon[11]; and *Solutions* the set of relaxed sample executions of δ in S. This predicate employs a variation of the relaxed interpreter of Section 6.5 (that does not use heuristic guidance when selecting transitions, but continues to relax the testing of action preconditions).

The APPLYADM function of Listing 8.2.2 takes the relaxed sample solutions found by *sample* and determines their most preferred subset given the preferences of the relevant

[10] In each of the presented clauses, greek letters must be replaced with an appropriate capital letter when implemented in SWI Prolog.

[11] Recall the purpose of this finite action horizon during the relaxed interpretation of a program δ in a situation S (Section 6.5.2). The horizon H is an upper bound on the number of actions that can be added to the situation S during the relaxed interpretation of δ.

agent (expressed as comparison schemes). This function is represented by the predicate $applyADM(Solutions, MPS)$, pseudocode for which is shown in Definition 8.3.1.

Definition 8.3.1 Let $Solutions$ denote a set of relaxed executions of a GOLOG program. The predicate $applyADM(Solutions, MPS)$ finds the subset $MPS \subseteq Solutions$ of most preferred executions (for a given agent), as shown below in pseudocode. Let us assume that this agent has an implementation of the predicates:

1. $scheme(SC, S_1, S_2)$ which holds if SC is a comparison scheme held by the agent and the scheme demonstrates a preference for the execution S_1 over S_2;
2. $conflict(A_1, A_2)$ which holds if argument A_1 conflicts with argument A_2;
3. $preferred(A_1, A_2)$ which holds if argument A_1 is strictly preferred to argument A_2;
4. $semantics(Sem)$ which holds if Sem is a Dung (1995) extension semantics (such as preferred, or grounded, see Section 3.2); and,
5. $rule(DG, MPS)$ which holds if MPS is the most preferred subset of the vertices in dominance graph DG (see Definition 4.6.2 for a description of a dominance graph).

The pseudocode for the predicate $applyADM$ is:

$Args \leftarrow \emptyset$
for each $S_1, S_2 \in Solutions$ where $S_1 \neq S_2$ **do**
  $For \leftarrow \{arg(SC, S_1, S_2) \mid scheme(SC, S_1, S_2)\}$
  $Args \leftarrow Args \cup For$
end for
$Args \leftarrow Args \cup createEpistemic(Args)$
$Attacks \leftarrow \{(A_1, A_2) \mid A_1, A_2 \in Args, conflict(A_1, A_2), \neg preferred(A_2, A_1)\}$
$Acceptable \leftarrow byDung(Args, Attacks)$
$MPS \leftarrow applyRule(Acceptable)$

Here, $byDung(Args, Attacks)$ finds the subset of acceptable practical arguments in $Args$ – those sceptically accepted under a semantics Sem ($semantics(Sem)$), as per Definition 4.6.1. The function $applyRule(Acceptable)$ transforms the conclusions of these acceptable arguments into a dominance graph DG over $Solutions$ (Definition 4.6.2), to which it applies a decision-rule ($rule(DG, MPS)$) to determine its most preferred subset, as per

Definition 4.6.3. Psuedocode for $createEpistemic(Args)$ is shown in Definition 8.3.2.

The function $createEpistemic(Args)$ used in Definition 8.3.1 returns the set of epistemic arguments relevant in determining the acceptability of the practical arguments in the set $Args$[12]. This set is formed as shown in the pseudocode of Definition 8.3.2.

Definition 8.3.2 Let us assume that an agent employing the AR-GOLOG interpreter maintains an implementation of the predicates: $support^*(PR, B)$ which holds if PR is a set of premises that support the inference of the belief B; and $disputes(B, A)$ which holds if the belief B disputes some aspect of the (practical or epistemic) argument A. Pseudocode of the function $createEpistemic(Args)$ is shown below.

$R \leftarrow \emptyset$
$EA \leftarrow \{arg(PR, B) \mid support^*(PR, B), \exists A \in Args.\, disputes(B, A)\}$
while $EA \neq \emptyset$ **do**
  $R \leftarrow R \cup EA$
  $EA \leftarrow \{arg(PR, B) \mid support^*(PR, B), \exists A \in EA.\, disputes(B, A), arg(PR, B) \notin R\}$
end while
return R

The predicates *scheme*, *conflict*, *preferred*, *semantics*, $support^*$, *disputes*, and *rule*, are agent-dependent. Each agent employing AR-GOLOG maintains their own implementation of these predicates. These predicates capture the attributes assumed to be present within an agent seeking to use the ABS algorithm of Listing 8.2.1 (as per Definition 4.2.2).

The predicate $select(Next_S, MPS, \delta', S')$ (representing SELECT of Listing 8.2.3) selects a configuration (δ', S') to transition to that leads to a relaxed execution $Sol \in MPS$ (a most preferred set of executions found by $applyADM$) – giving priority (in its first clause) to final configurations. The clauses of this *select* predicate are shown in Definition 8.3.3.

Definition 8.3.3 The predicate $maps(\delta', S', Sol)$ (representing MAPS in Listing 8.2.3) holds if Sol is a relaxed execution of δ' in situation S'. The clauses of $select(Next_S, MPS, \delta', S')$

[12]In the description of ABS in Section 8.2.2, all possible epistemic arguments are constructed with each ADM application. AR-GOLOG is implemented with greater efficiency – preventing epistemic arguments that have no bearing on the acceptability of practical arguments from being generated.

are shown below, the first appears on the left hand side and the second on the right.

$$
\begin{array}{l}
select(Next_S, MPS, \delta', S') \text{ :-} \\
\quad member(Sol, MPS), \\
\quad member((\delta', S'), Next_S), \\
\quad maps(\delta', S', Sol), finalAR(\delta', S').
\end{array}
\qquad
\begin{array}{l}
select(Next_S, MPS, \delta', S') \text{ :-} \\
\quad member(Sol, MPS), \\
\quad member((\delta', S'), Next_S), \\
\quad maps(\delta', S', Sol).
\end{array}
$$

On the back of Definitions B.2.2, and 8.3.1 to 8.3.3, the general form of a $transAR$ clause within AR-GOLOG can now be presented. The final clauses of AR-GOLOG ($finalAR$) are equivalent to those of the standard GOLOG interpreter of Section 5.3.1.

Definition 8.3.4 Each GOLOG construct δ_c of Table 5.2 in Section 5.3 is associated with a $transAR(H,\ N,\ \delta_c,\ S,\ \delta',\ S')$ clause within the argumentation-based interpreter AR-GOLOG. Let: (δ', S') denote the configuration transitioned to by performing a single step of δ_c in the situation S; H a finite action horizon for relaxed interpretation (used during the discovery of sample executions, see Definition B.2.2 in Appendix B); and N denote the sampling index (of search) used during interpretation. Each $transAR$ clause has a general form, shown below. In the following clause:

1. $next_S(\delta_c, S, Next_S)$ finds the set of configurations $Next_S$ that can be transitioned to from (δ_c, S) under the standard transition semantics of Section 5.3.1;
2. $findsamples(N,\ H,\ Next_S,\ Solutions)$ combines the result $List$ of $sample(N,\ \delta_i,\ S_i,\ H,\ List)$ (see Definition B.2.2) for each $(\delta_i, S_i) \in Next_S$, into a single set $Solutions$;
3. $applyADM$ and $select$ are defined as in Definition 8.3.1 and 8.3.3.

The general form of a $transAR$ clause is:

$$
\begin{array}{l}
transAR(H, N, \delta_c, S, \delta', S') \text{ :-} \\
\quad next_S(\delta_c, S, Next_S), findsamples(N, H, Next_S, Solutions), \\
\quad applyADM(Solutions, MPS), select(Next_S, MPS, \delta', S').
\end{array}
$$

Definition 8.3.5 The $transAR$ clauses whose form is shown in Definition 8.3.4 are used to find an execution S' of a program δ in situation S with finite action horizon H (for use

during relaxed interpretation[13]) and sampling index N as follows (where $transAR^*$ is the reflexive transitive closure of $transAR$ – a repeated application of $transAR$).

$$doAR(\delta, S, S', H, N) \text{ :- } transAR^*(H, N, \delta, S, \delta', S'), finalAR(\delta', S').$$

Recall from Proposition 8.2.1 that argumentation-based search (ABS) is neither complete or optimal (admissible). AR-GOLOG is also not optimal by Definition 8.2.2. It is complete, however, as once a non-final configuration is reached in which no further transitions can be made the interpreter backtracks to the last choice made and makes a different choice in its place. Backtracking is a process that underpins the logical programming languages in which a GOLOG interpreter is implemented (such as Prolog).

Proposition 8.3.1 AR-GOLOG *is complete, but not optimal by Definition 8.2.2.*

Proof: AR-GOLOG is complete – it will discover an execution of a GOLOG program if one exists – as the search for an execution backtracks (as described above) when a dead-end is reached (a non-final configuration from which no further transitions can be made). AR-GOLOG is sub-optimal for the same reasons that ABS and the heuristic interpreter presented in Chapter 6 are sub-optimal – search selects transitions to make on the basis of a heuristic evaluation using an approximation of what is possible after making those choices, and only retracts these decisions if a dead-end configuration is reached.

□

I now evaluate the AR-GOLOG interpreter within two domains – the travel planning domain of Section 8.2.3, and the spacecraft control domain of Section 7.1.

8.4 An Evaluation of AR-GOLOG

In Section 8.3.2, a number of predicates are identified that characterise an agent and its preferences: *scheme*, *conflict*, *preferred*, *semantics*, $support^*$, *disputes*, and *rule*. To evaluate AR-GOLOG, these predicates are instantiated within the travel and spacecraft control

[13]This horizon is used only for the purposes of relaxed interpretation during the argumentation-based evaluation of choices. It does not restrict the length of executions found by the AR-GOLOG interpreter.

domains of Sections 8.2.3 and 7.1. A number of tests are generated within the travel domain, in which the GOLOG program of Listing 8.2.4 must be interpreted. These tests vary the initial situation axioms of the domain and the preferences of the agent planner. AR-GOLOG is first restricted to finding executions of a small number of actions. In these tests, it is possible to enumerate all executions of the program and compare its optimal solution (by Definition 8.2.2) with the result found by the AR-GOLOG interpreter. This length restriction is then removed, and the ability of AR-GOLOG to find a program execution (where it is not possible to enumerate them all) is demonstrated.

8.4.1 Travel Planning

Recall from Sections 8.2.3 and 8.2.4 that the goal of the planning agent BETTY within the travel domain is to travel from one city in Europe (Valladolid) to another (Càdiz), while having sight seeing adventures on route (available adventures and their attributes are shown in Table 8.1). The GOLOG program to be interpreted is shown in Listing 8.2.4. In this procedure, the planner either chooses to select a city to travel to (by boat, car, or plane), or stays in its current city to partake in a sight seeing adventure (if available).

The predicates that characterise the planner (see Definitions 8.3.1 and 8.3.2), referenced by AR-GOLOG, are instantiated as follows. The predicate *scheme* describes the schemes of Definitions 8.2.5 to 8.2.9, while $support^*$ constructs epistemic arguments as defined in Definition 8.2.4 with the support predicates of Table 8.3 (the arguments constructed are shown in Section 8.2.4). The predicates *conflict*, *preferred*, and *disputes* determine conflict and preference amongst arguments as per Example 8.2.6 (following the ADM guidelines of Definition 4.5.1). The predicate *semantics* indicates that the preferred extension semantics are used when determining (the sceptical) acceptance of arguments, and *rule* that the planner uses the Schwartz set (Example 4.6.3) as its decision-rule.

A set of test cases have been generated in this domain, varying: the fuel available to the planner (Capacity(c), see Table 8.2 and Example 8.2.2); the ordering the planner maintains over its comparison schemes ($\mathcal{I}_{SCH}$, see Example 8.2.6); the ordering the planner maintains over the attributes of sight seeing adventures (Order(o), see Table 8.2 and Example 8.2.2); the sampling index N used in interpretation; and how the planner deter-

mines preference over epistemic arguments (with the *preferred* predicate). The planner either prefers a more specific argument a_1 to a less specific a_2 ($Premises(a_2) \subset Premises(a_1)$) or vice versa. In each test, the planner aims to travel from Valladolid to Cádiz.

The results of the generated test cases, in which the length of discovered program executions is restricted to a small number of actions, are shown in Table 8.4. The AR-GOLOG interpreter is applied to these tests to find an execution of the program in Listing 8.2.4. A program developed to enumerate all executions of the program and apply the ADM of Chapter 4 to find an optimal execution for the planner (by Definition 8.2.2) is used to test the success rate of AR-GOLOG. The time required (in seconds) for the AR-GOLOG interpreter to find an execution of the program in each test, and the time required to enumerate (and argue over) the complete execution set, are recorded. Table 8.4 identifies the tests in which AR-GOLOG was able to find an optimal solution.

The enumeration-based approach to finding an optimal solution in these test cases could not be applied (without running out of memory) to tests in which the restricted length (R_L) was greater than 5. Table 8.4 shows that AR-GOLOG was effective in finding an optimal solution where the executions were restricted to being 4 actions long. When increasing this limit to 5, and increasing N from 2 to 5, AR-GOLOG was less effective. Table 8.5 shows this same collection of 11 test cases (with $R_L = 5$) but with a sampling index of 7. AR-GOLOG has a better success rate using a sampling index of 7 on this latter set of test cases. In Table 8.6, the same set of test cases shown in Table 8.4 are solved, but with no length restriction on discovered executions. Table 8.6 demonstrates that AR-GOLOG is able to find an execution in each test within this setting.

The results presented in this section demonstrate that while it is possible for AR-GOLOG to find an optimal solution (the employed definition of optimality follows that of Definition 8.2.2), this may not happen in a majority of instances. Consider the results shown in Table 8.5. With a sampling index of 7, and a restricted execution length of 5, AR-GOLOG found an optimal solution in 5/11 of the tests. This evaluation reveals a significant limitation in the black and white definition of optimality used. Both AR-GOLOG and argumentation-based search (ABS) are sub-optimal, and hence cannot be expected to find optimal solutions. Without a graded definition of preference over execu-

R_L	CAPACITY	N	$Time_{AR}$ (s)	$Time_{CS}$ (s)	SUCCESS
4	700	2	0.33	0.20	Yes
4	700	2	0.34	0.19	No
4	700	2	0.34	0.20	Yes
4	700	2	0.27	0.19	Yes
4	700	2	0.34	0.19	Yes
4	700	2	0.36	0.19	Yes
4	700	2	0.22	0.19	Yes
4	700	2	0.33	0.19	Yes
4	700	2	0.27	0.20	Yes
4	700	2	0.38	0.19	Yes
4	700	2	0.25	0.19	Yes
5	2000	5	1.56	2.28	Yes
5	2000	5	1.64	2.34	Yes
5	2000	5	0.56	2.25	No
5	2000	5	1.63	2.30	No
5	2000	5	1.28	2.28	No
5	2000	5	1.72	2.34	No
5	2000	5	1.28	2.31	No
5	2000	5	1.27	2.31	No
5	2000	5	1.20	2.31	No
5	2000	5	1.11	2.25	Yes
5	2000	5	1.22	2.31	No

Table 8.4: The success rate of AR-GOLOG relative to a brute force enumeration, where: R_L denotes the restriction on execution length; CAPACITY is the available fuel (in units) for driving; N is the sampling index of search; $Time_{AR}$ and $Time_{CS}$ denote the time required (in seconds) for AR-GOLOG and the brute force algorithm to find a solution (respectively); and SUCCESS denotes whether AR-GOLOG has found an optimal solution.

tions (choices), it cannot be determined how good the executions found by AR-GOLOG are (if they are not optimal). An extension of the ADM of Chapter 4 to determine a preference ordering over (in place of a partitioning of) choices is a topic of future work.

In the next section, I apply AR-GOLOG to the spacecraft control domain of Section 7.1 in which the quality of executions is measurable. I compare the results of the AR-GOLOG interpreter in this domain to those obtained by the heuristic interpreter of Chapter 6. A goal of this comparison is to determine whether the manner in which search is guided within AR-GOLOG has promise in finding good (if not optimal) solutions in the presence of graded preference. Moreover, this comparison will highlight the time overhead incurred by AR-GOLOG relative to my heuristic interpreter.

R_L	CAPACITY	N	$Time_{AR}$ (s)	$Time_{CS}$ (s)	SUCCESS
5	2000	7	2.17	2.28	Yes
5	2000	7	2.16	2.34	Yes
5	2000	7	0.81	2.25	No
5	2000	7	1.72	2.30	Yes
5	2000	7	1.69	2.28	No
5	2000	7	1.80	2.34	Yes
5	2000	7	1.81	2.31	No
5	2000	7	1.78	2.31	No
5	2000	7	1.77	2.31	No
5	2000	7	1.63	2.25	Yes
5	2000	7	1.67	2.31	No

Table 8.5: The success rate of AR-GOLOG relative to a brute force enumeration, where: R_L denotes the restriction on execution length; CAPACITY is the available fuel (in units) for driving; N is the sampling index of search; $Time_{AR}$ and $Time_{CS}$ denote the time required (in seconds) for AR-GOLOG and the brute force algorithm to find a solution (respectively); and SUCCESS denotes whether AR-GOLOG has found an optimal solution.

8.4.2 Spacecraft Control

Recall the spacecraft control domain of Section 7.1. The AR-GOLOG interpreter of Section 8.3 is applied to the test cases created within this domain (Section 7.1.4). The fluents, actions, action preconditions, successor state axioms, and initial situation axioms of the domain are presented in Section 7.1. The program to be interpreted is provided in Listing 7.1.1 of Section 7.1. In this domain, the predicates that characterise the planner, referenced by the AR-GOLOG interpreter, are instantiated as follows. The predicate *scheme* is implemented using a single comparison scheme that compares executions on the basis of the number of targets they observe, described in Definition 8.4.1, below.

Definition 8.4.1 Let s_1 and s_2 be executions of the GOLOG program of Listing 7.1.1, and $num\_obs(s_i)$ a function that returns the number of celestial targets observed in execution s_i. The scheme sc_{obs} is defined below, where $Greater(n_1, n_2)$ holds if $n_1 > n_2$.

$$sc_{obs} = \langle s_1, s_2, \{Greater(num\_obs(s_1), num\_obs(s_2))\}, s_1 \succ_{sc_{obs}} s_2 \rangle$$

In this domain, arguments never conflict (as only one scheme is used to compare executions) and epistemic arguments are not created (the agent is unable to infer anything

CAPACITY	N	$Time_{AR}$ (s)	$Time_{CS}$ (s)	SUCCESS
700	2	3.28	–	?
700	2	3.17	–	?
700	2	3.23	–	?
700	2	3.28	–	?
700	2	3.20	–	?
700	2	1.88	–	?
700	2	3.17	–	?
700	2	1.98	–	?
700	2	3.19	–	?
700	2	3.17	–	?
700	2	3.25	–	?
2000	5	15.55	–	?
2000	5	15.39	–	?
2000	5	3.28	–	?
2000	5	15.23	–	?
2000	5	15.61	–	?
2000	5	15.16	–	?
2000	5	15.25	–	?
2000	5	15.81	–	?
2000	5	15.33	–	?
2000	5	3.25	–	?
2000	5	15.11	–	?

Table 8.6: AR-GOLOG vs. brute force enumeration, where: CAPACITY is the available fuel (in units) for driving; N is the sampling index of search; $Time_{AR}$ and $Time_{CS}$ denote the time required (in seconds) for AR-GOLOG and the brute force algorithm to find a solution (respectively) to each test; and SUCCESS denotes whether AR-GOLOG has found an optimal solution. Symbol '–' denotes that a solution could not be found within 15 minutes or available memory. Action horizon for relaxation $H = 50$ in each test.

with its *support** construct). Hence, how the predicates *conflict*, *disputes*, *preferred* and *semantics* are defined is of no consequence. For the purposes of implementation, it is assumed that they take the same form as in the travel planning domain (Section 8.4.1). As in the travel domain, the *rule* of the agent is the Schwartz set (Example 4.6.3).

This domain, with its single scheme, does not take advantage of the full capability of argumentation-based interpretation. The heuristic interpreter presented in Chapter 6 is tailored for use on these kinds of problems – problems in which preference can be represented as a numeric evaluation function. Applying AR-GOLOG to this example determines whether an approach that is not tailored to the use of any one kind of preference

N_T	TIME	FUEL	POWER	O_S	O_G	O_H	O_{AG}	T_H (s)	T_{AG} (s)
10	150	500	200	2	2	2	2	0.23	3.23
10	150	500	200	0	0	2	2	0.39	6.19
10	150	500	200	6	6	6	6	0.28	11.42
10	150	500	200	2	2	2	2	0.92	19.38
10	150	500	200	2	2	3	3	0.08	4.13
10	150	500	200	2	2	2	2	0.08	1.77
10	150	500	200	4	4	4	4	0.22	6.39
10	150	500	200	2	2	2	2	0.16	2.61
10	150	500	200	2	2	2	3	1.09	5.38
10	150	500	200	0	0	4	4	0.14	3.61
20	250	600	300	3	3	9	7	1.20	40.53
20	250	600	300	2	8	8	8	0.98	27.94
20	250	600	300	4	6	12	11	1.81	63.89
20	250	600	300	5	10	11	7	1.70	26.06
20	250	600	300	3	3	8	8	1.38	51.94
20	250	600	300	4	7	10	10	1.09	27.72
20	250	600	300	2	7	8	7	0.88	58.58
20	250	600	300	3	4	6	5	2.70	9.92
20	250	600	300	3	4	8	7	0.67	23.88
20	250	600	300	2	5	11	7	1.34	42.80
30	350	700	400	5	8	18	13	8.80	226.11
30	350	700	400	5	13	19	11	8.84	359.28

Table 8.7: Observations made in executions found by the standard (O_S), greedy (O_G), heuristic (O_H), and argumentation-based (O_{AG}) interpreters. In each test, N_T denotes the number of celestial targets, and T_H and T_{AG} denote the time (in seconds) required by the heuristic (Chapter 6) and AR-GOLOG interpreters to find a solution.

can yield a similar performance to one that does; and what time overhead AR-GOLOG suffers from relative to this tailored interpreter. It is expected that AR-GOLOG will be able to find executions that observe more targets than the standard and greedy[14] interpreters tested in Section 7.1, but that the heuristic interpreter (whose heuristic guidance is tailored to the use of numeric evaluation functions in representing preference) will ultimately yield the most superior results. It is also expected that AR-GOLOG will consume more time in its search for a solution to each test – a result of the need to perform pair-wise comparisons of available choices when selecting a transition.

Throughout the tests in this section, a sampling index of $N = 5$ is used. These tests have been generated in Section 7.1.4. The results of the AR-GOLOG interpreter on these

[14]The greedy interpreter is guided by a non-lookahead-based heuristic (see Chapter 7).

N_T	$\overline{O}_S$	$\overline{T}_S$ (s)	$\overline{O}_{AG}$	$\overline{T}_{AG}$ (s)
10	2.2 (1.75)	0.05 (0.05)	3 (1.33)	6.41 (5.31)
20	3.1 (0.99)	0.14 (0.12)	7.7 (1.70)	37.33 (17.16)
30	5 (0)	0.05 (0.02)	12 (1.41)	292.70 (94.17)
N_T	$\overline{O}_G$	$\overline{T}_G$ (s)	$\overline{O}_{AG}$	$\overline{T}_{AG}$ (s)
10	2.2 (1.75)	0.50 (0.49)	3 (1.33)	6.41 (5.31)
20	5.7 (2.31)	1.59 (1.31)	7.7 (1.70)	37.33 (17.16)
30	10.5 (3.54)	3.67 (0.31)	12 (1.41)	292.70 (94.17)
N_T	$\overline{O}_H$	$\overline{T}_H$ (s)	$\overline{O}_{AG}$	$\overline{T}_{AG}$ (s)
10	2.9 (1.37)	0.36 (0.36)	3 (1.33)	6.41 (5.31)
20	9.1 (1.85)	1.38 (0.58)	7.7 (1.70)	37.33 (17.16)
30	18.5 (0.71)	8.82 (0.03)	12 (1.41)	292.70 (94.17)

Table 8.8: Mean number of observations made in executions found by the standard ($\overline{O}_S$), greedy ($\overline{O}_G$), heuristic ($\overline{O}_H$), and argumentation-based ($\overline{O}_{AG}$) interpreters (sample standard deviation in brackets) for test collections with varying N_T (number of targets). $\overline{T}_S$, $\overline{T}_G$, $\overline{T}_H$, and $\overline{T}_{AG}$ denote the mean time used by each of these interpreters (respectively) to arrive at a solution (sample standard deviation in brackets).

tests are shown in Table 8.7. These results demonstrate that in terms of target observations, AR-GOLOG performs as well or better than the heuristic interpreter in 13/22 of the test cases shown in Table 8.7. In 20/22 of the tests, AR-GOLOG performs as well or better than the greedy interpreter of Chapter 7. On all 22 tests, AR-GOLOG finds an as good or higher quality execution than the standard interpreter of Section 5.3.1. These results demonstrate that AR-GOLOG is reasonably effective in finding executions that are as good (or nearly as good) as those found by the heuristic interpreter. The time required by AR-GOLOG, however, is much greater. Applying an ADM (Chapter 4) over a set of choices requires pair-wise comparisons – a more time consuming process than evaluating each choice on its own (as is done within the heuristic interpreter). The results of Table 8.7 highlight that strategies are required to ensure that AR-GOLOG can scale with problem complexity. These strategies are considered as future work in Section 8.5.

The average time required by the standard (Section 5.3.1), greedy (Chapter 7), heuristic (Section 6.6), and argumentation-based interpreters to find a solution in the collection of tests shown in Table 8.7, for each number of targets (N_T), is highlighted in Table 8.8. Also stated is the average number of targets observed within each collection of tests. In brackets following each average is the (sample) standard deviation of the associated data

set. As is evident in Table 8.7, only two tests were conducted with $N_T = 30$.

The evaluation of this section demonstrates that AR-GOLOG is a promising approach for planning with preferences expressed in the form of comparison schemes (albeit one that requires further development). In the travel domain, AR-GOLOG is able to find an optimal execution of a GOLOG program in a number of test cases. However, a process by which a preference ordering is computed over choices with an ADM (in place of a partitioning of choices into two sets, as per Definition 8.2.2) is required to effectively evaluate this interpreter. In the spacecraft control domain, the AR-GOLOG interpreter has been found to be successful in finding good solutions. The time it requires to discover these solutions is prohibitive, however. A more time consuming search, over the heuristic interpreter that forms the second key contribution of this thesis, is the price of increased generality. Future development of AR-GOLOG must focus on minimising this price.

Each experiment conducted within this chapter has been run on a Windows XP (Service Pack 3) machine with a 2.40 GHZ Intel(R) Core(TM)2 CPU and 2GB of RAM.

8.5 Concluding Remarks and Future Work

In this chapter, the ADM of Chapter 4 has been used in the creation of an algorithm for argumentation-based search (ABS). ABS is designed to find a most preferred solution to a search problem within a tree-structured state-space – a path from an initial to a goal state. Search operates by repeatedly selecting a state to transition to, amongst a number of available next states, until a goal state is reached. A transition is selected not on the basis of an individual evaluation of each possibility, but an argumentation-based comparison of each pair of choices (an application of an ADM). Choices are selected on the basis of a set of comparison schemes – each scheme describing one way of determining which of a choice pair is preferred. The goal of ABS is to try and find an optimal solution – a path in the most preferred set found by an ADM applied over the complete problem solution set.

At each choice point during search, ABS finds a set of relaxed problem solutions for each state that can be transitioned to. These relaxed solutions are designed to represent the future possibilities that my arise by pursuing that state. An ADM is applied to this collection of relaxed solutions, defining a most preferred subset. A transition is made

that leads to a solution within this set, with priority given to transitions that result in a goal state. The ABS algorithm is inspired, in part, by the hill-climbing heuristic search planners HSP [Bonet and Geffner (2001)] and FF [Hoffman and Nebel (2001)]. Once a transition is made, it is not retracted – search does not maintain a frontier of unexplored choices which it can revisit at will. ABS is consequently incomplete, and sub-optimal.

I have demonstrated the application of ABS within a travel planning domain – highlighting how the relaxed solutions found for available states have an impact on the discovery of an optimal solution. This example was followed by an instantiation (and adaptation) of ABS for the construction of an argumentation-based GOLOG interpreter. This interpreter, like my heuristic interpreter of Chapter 6, is characterised by a transition semantics – a set of clauses that define which program-situation pairs can be transitioned to by performing a single step of a program in a given situation. Each clause within this interpreter has a general form – applying the principles of ABS in selecting a next transition. The set of possible transitions is enumerated and a number of relaxed executions found for each (using an adaptation of the relaxed interpreter of Chapter 6). The ADM of Chapter 4 is applied, given a collection of schemes that compare pairs of program executions, to find a most preferred subset. A configuration is transitioned to that leads to an execution within this set, with priority given to configurations that are final.

Within this chapter, the developed argumentation-based GOLOG interpreter (AR-GOLOG) is evaluated within two domains – the travel planning domain used to demonstrate ABS, and the spacecraft control domain of Section 7.1. This evaluation demonstrates that while AR-GOLOG is capable of discovering an optimal solution within the travel planning domain, its classification of executions as either optimal or not is a disadvantage in its evaluation. AR-GOLOG is sub-optimal and a preference ordering over solutions will provide a means of gauging how good its solutions are. AR-GOLOG is shown to often yield executions that are comparable in quality to those found by the heuristic interpreter of Chapter 6 in the spacecraft control domain. The time it requires to find these solutions, however, is prohibitive. This evaluation, while demonstrating the promise of AR-GOLOG, identifies a number of important areas for further development.

The ADM of Chapter 4 defines a most preferred subset of a choice set – a partitioning

of this set into two classes, optimal and not optimal. If this framework is varied so that it finds an ordering over a set of choices, it can be used to find a preference ordering over solutions to a search problem or executions of a GOLOG program. With this ordering, the performance of AR-GOLOG in the travel domain (Tables 8.4 and 8.5) can be better judged – determining not whether it produces an optimal execution, but the position of this execution within this ordering. This modification of the ADM is task for future work. The second key area of future development is a reduction in the time overhead associated with AR-GOLOG. One strategy is to reduce the number of pair-wise comparisons required when applying an ADM to a collection of choices. This may be achieved by clustering similar choices, and treating each cluster as an individual option. Choices within the most preferred subset of clusters can then be organised into smaller clusters, repeating this process of ADM application until only a small number of choices remain. Argumentation over this small choice set determines a most preferred subset.

In this chapter, I have returned to my goal of developing automated decision-making and planning techniques that are decoupled from the preferences supplied to them. ABS and the AR-GOLOG interpreter satisfy the conditions of decoupling, expressed in Criterion 2.1.1. First, both approaches allow choices – solution paths and executions – to be distinguished by general (abstract) mechanisms of comparison. Second, the manner in which choices are selected during search and interpretation, leading to the ultimate recommendation of a solution, abstracts away from the details of this comparison.

As is the case in the ADM of Chapter 4, decision-making and search is provided with input that allows it achieve this feat. The implementation of the AR-GOLOG interpreter assumes the presence of agent-dependent predicates (or functions) that determine how: schemes are applied to a pair of choices; beliefs are inferred with a deductive system; conflict and preference is defined amongst arguments; and a decision-rule used to find a most preferred subset of choices given a dominance graph over them. The AR-GOLOG implementation remains the same irrespective of the manner in which an agent using it expresses preference or maintains its belief system. The properties of this agent are defined within predicates that are referenced by the interpreter – separated from the interpretation process just like the basic action theory that characterises its domain.

Chapter 9
Conclusion

THIS thesis has presented three key contributions within the domain of automated decision-making and planning with preferences. This thesis has demonstrated how to decouple preference from the algorithmic processes that occur within decision-making and planning tools. I have defined two conditions that must be satisfied by such decoupled approaches. First, and foremost, these methods must allow choices to be distinguished by general expressions of preference – general means by which two choices can be compared. Second, these approaches must abstract away from the representational detail of these preferences in the selection or recommendation of a choice.

This ability to support a wide range of mechanisms by which preference can be expressed, as I have stated in Chapter 1, imbues decoupled decision-making tools with a number of advantages. These algorithms can be reused across different domains – domains for which varying modes of preference are appropriate. Within the spacecraft control domain of Section 7.1, it was appropriate to express preference in the form of a numeric evaluation function. The travel domain of Section 8.2.3 supports more unconventional forms of preference, from soft goals to lexicographic comparisons and rules-of-thumb. These decoupled tools, moreover, are able to cater for a wide range of users – each user equipped their own preferred means of expressing their preferences.

This thesis has asked the questions: can decision-making and planning algorithms be developed that are decoupled from preference (as per Criterion 2.1.1), and, if so, how? I have presented the following three major contributions in this thesis:

1. A decoupled argumentation-based decision-making framework for the selection of a most preferred choice from a set of choices (denoted the ADM);

2. An integration of heuristic search within the interpretation of GOLOG programs, based on the application of a relaxed lookahead. This relaxed lookahead employs a relaxed variant of situation calculus regression – developed in this thesis;

3. A new approach to search – using an argumentation-based evaluation of available paths (choices) – demonstrated in the development of an argumentation-based GOLOG interpreter (denoted AR-GOLOG).

These contributions have demonstrated that it is possible to decouple preference from the algorithmic processes within decision-making and planning tools (as per Criterion 2.1.1). I have achieved this decoupling through the use of computational argumentation.

The first contribution of this thesis was an argumentation-based decision-making system (ADM) designed to abstract away from the human or agent preference it operates on. In this work, I adapt the abstract argumentation framework of Dung (1995) and unify a number of features present across different works in the argumentation-based decision-making community – the encapsulation of preference in the form of comparison schemes; the manipulation of abstract arguments in the process of decision-making; and the use of decision-rules in the ultimate selection of a choice. Given a collection of choices, these comparison schemes form the basis of arguments constructed in support of a preference for one choice over another. These arguments, in conjunction with those that dispute their premises, form part of a Dung (1995) argumentation framework. The acceptable of these arguments define which preference relationships have withstood scrutiny. A decision-rule (similar in nature to the social choice rules that permeate the study of voting) applied to these relationships discovers a most preferred subset of choices.

This system differs from the existing range of argumentation-based decision-making tools in that it satisfies the criteria for decoupling that I have defined. In Chapter 3, I have described this existing work and have analysed each approach with respect to these criteria – discovering that not one satisfies both conditions. The system I present is designed to form part of a larger system providing decision-making assistance to a user.

This system requires a number of inputs: a collection of choices; arguments constructed on the basis of a set of comparison schemes and a system of deduction; a conflict and preference relation over these arguments; and a decision-rule. With this information, the system is able to suggest a most preferred subset of these choices. The algorithm by which this occurs does not require a knowledge of what lies within these comparison schemes – abstracting away from the details of user preference in the formation of a most preferred subset of choices. The requirement that arguments support preference relationships over choices, in contrast with the majority of existing works discussed, allows the system to distinguish choices on the basis of general mechanisms of comparison. Each scheme is able to capture any type of preference that can be used to compare a pair of choices.

It is clear that while the presented argumentation-based decision-making system represents a decoupled algorithm (provided it receives its required input), many challenges remain in delivering it as part of a stand alone tool for decision recommendation. These include an ability to transform information provided by a user into the form of comparison schemes, and a knowledge of how to test the premises of these schemes when given a pair of choices. This knowledge involves the provision of a system of deduction (a means of deducing information about the environment of the decision-maker). Despite these challenges, I believe this first contribution is a promising first step toward the realisation of such systems – systems that communicate with a user to ascertain their preferences, described in a manner most comfortable and natural for them.

The second contribution of this thesis represents a departure from its theme of decision-making and planning with general preferences. I present an approach to planning with numeric evaluation functions in the GOLOG formalism [Levesque et al. (1997)]. I have developed a heuristic interpreter for GOLOG programs designed to discover program executions (sequences of actions for an agent or planner to perform) that minimise such a function. This interpreter is designed to reflect popular heuristic planning techniques within the classical realm. The relaxation-based heuristics of the HSP [Bonet and Geffner (2001)] and FF [Hoffman and Nebel (2001)] planners are adapted for use in guiding the interpretation of a GOLOG program. A GOLOG program, as I have described in Chapter 5, is composed of a collection of constructs (such as primitive actions,

branches, and argument selection). When faced with a non-deterministic construct (involving choice), a standard GOLOG interpreter does not consider the impact its choices have on the quality of the executions it finds. I develop an interpreter that heuristically evaluates available choices, determining which are likely to lead to most preferred executions. In doing so, I develop an interpreter that finds a relaxed execution of the program that remains to be performed in each of these choices. Choices are made by my heuristic interpreter that lead to the relaxed executions with the most promising evaluations.

As part of this second contribution, I have developed a relaxed interpreter for GOLOG programs, centered on the use of a relaxed form of situation calculus regression. Regression is a tool by which we can answer queries of entailment within the situation calculus [McCarthy and Hayes (1969); Reiter (2001)]. Given a basic action theory (BAT) characterising the dynamics of a domain – the available actions and their preconditions, and the changing of environment properties over time – entailment queries ask whether a formula holds with respect to such a theory. In the interpretation of a GOLOG program, these queries are answered in the testing of action preconditions, and in the evaluation of test conditions within test, if ... else, and while loop constructs. Regression is characterised by a series of rules that transform a formula into an expression uniform in an initial situation – an expression that can be proved with a first order theorem prover and a collection of initial situation axioms. Relaxed regression alters these rules to remove some of the tests that must be performed – for example, by considering only whether something has arisen to cause a fluent to hold, while neglecting to consider what may prevent it from holding. The presented relaxed interpreter discovers a program execution while using relaxed regression in the testing of action preconditions.

Through an analysis of existing preference-guided GOLOG interpreters, it was found that heuristic guidance within this work was limited. Heuristics were not lookahead-based, unlike the heuristic I have proposed. In addition, approaches designed to incorporate classical planning heuristics in the interpretation of a program required compilation of the program (and its associated BAT) into the ADL subset [Pednault (1989)] of PDDL [Fox and Long (2003); Gerevini and Long (2005)]. This compiled problem could then be solved by a classical planner. Such BATs must be expressible in ADL – limiting

their expressiveness. The heuristic interpreter I have presented in Chapter 6 does not restrict the relevant BAT in this manner. The presented heuristic interpreter was evaluated on three domains – spacecraft control, mine operations, and task scheduling. This interpreter, with its relaxation-based heuristic, discovered executions higher in quality (in a majority of generated tests) than those found by a standard interpreter (Section 5.3.1) and one guided by a non-lookahead-based heuristic (characteristic of existing work).

In the third contribution of this thesis, focus was returned to the decoupling of preference and planning through argumentation. I have developed a method of state-space search, using argumentation, for tree-structured state-spaces – geared toward the solution of planning problems. Classical planning is, after all, a search for a sequence of actions guiding a planner from an initial to a goal state through a state transition system. An argumentation-based search (ABS) algorithm has been designed to find a most preferred solution to such a problem given a set of preferences expressed in the form of comparison schemes. Optimality is defined in terms of the ADM system of Chapter 4. An optimal solution path is a member of the most preferred subset of paths found by an application of this ADM over the complete problem solution set. The goal of the ABS algorithm is to discover such a solution, without enumerating the entire set.

The ABS algorithm, like the presented heuristic GOLOG interpreter, involves a hill-climbing style search. Search proceeds, from an initial state, by making repeated state transitions until a goal state is reached. Given a collection of transitions to choose from, ABS finds a set of relaxed problem solutions for each next state – each relaxed solution outlining a future possibility that may arise by pursuing that state. An ADM system is applied to this combined set of relaxed solutions, determining which are the most preferred on the basis of a set of comparison schemes. A transition is made if it leads to a relaxed solution in this most preferred set, with priority given to those that transition search to a goal state. I have shown that ABS is not complete or optimal (admissible).

I have employed this ABS algorithm in the development of an argumentation-based GOLOG interpreter (AR-GOLOG). This interpreter is characterised by a transition semantics, in a similar manner to the characterisation of standard GOLOG and the heuristic interpreter of Chapter 6. This semantics defines a transition clause for each available

construct in the GOLOG language, describing which configurations to transition to by executing a single step of a program in a given situation. Each of these clauses has a general form – following the principles of state selection within the ABS algorithm.

Within each transition clause of AR-GOLOG, the interpreter has a choice of transitioning to one of a number of configurations. The principles of ABS are used in the selection of a transition. For each of these configurations (program-situation pairs), the AR-GOLOG interpreter finds a number of relaxed executions of their program in their associated situation. These relaxed executions are found through the use of a relaxed interpreter, as described above, that utilises relaxed regression in the testing of action preconditions. An ADM is applied to determine the most preferred of these executions on the basis of a set of comparison schemes. A transition is made to a configuration if it leads to an execution within this most preferred set, with priority given to those that are final (in which no steps remain to be performed in their program).

I have evaluated this AR-GOLOG interpreter within two domains – travel planning and spacecraft control. Within the travel planning domain, the ability of this interpreter to discover an optimal solution is analysed. Several simple instances of this domain are devised – simple enough to allow the enumeration of all executions of the controlling GOLOG program and an application of an ADM. I compare the execution obtained by AR-GOLOG with the most preferred subset discovered through this latter brute force approach. AR-GOLOG was found, in some of my tests but not a majority, to discover an optimal solution. The black and white picture an ADM paints of a collection of choices, separating them into two sets (optimal and the rest), prevented a quality assessment of the sub-optimal executions discovered. Within the domain of spacecraft control, I have applied AR-GOLOG to find a solution to each of the tests used in the evaluation of my heuristic interpreter – the second contribution of this thesis. I have discovered that the general AR-GOLOG is competitive with the narrow heuristic interpreter (tailored specifically for preference expressed as a numeric evaluation function) on these tests. The time it required to discover executions, however, was found to be prohibitive.

To the best of my knowledge, the work I have presented in the development of an argumentation-based GOLOG interpreter is the first work to integrate argumentation

and the interpretation of GOLOG programs, and to use argumentation as a tool for goal-directed search. These approaches, as I have highlighted in Section 8.5, satisfy my test for decoupling. The limitations of their current form – their black and white definition of optimality and their time complexity – provide ample scope for future development.

Future Research

Each of the three contributions I have presented in this thesis provide a firm basis for future development and research. These avenues of future work are described below:

1. An extension of the ADM of Chapter 4 to define a preference ordering over choices, rather than a partitioning, and a framework for multi-agent decision-making;
2. An online version of the presented heuristic interpreter of Chapter 6 and an evaluation of its performance relative to compilation-based approaches for preference-guided interpretation [Baier et al. (2007b, 2008); Fritz et al. (2008)] ;
3. Strategies to minimise the time overhead associated with both the heuristic interpreter of Chapter 6 and the argumentation-based interpreter of Chapter 8.

There are a variety of directions in which the argumentation-based decision-making framework (ADM) of Chapter 4 can be taken. I would like to consider whether this framework is a generalisation of existing decision-making techniques within the argumentation-based decision-making literature. The application of this framework for use within multi-agent decision-making presents a second prime area of future work. These potential avenues of further development are described in greater detail within Section 4.9 of Chapter 4. As I have highlighted in Chapter 8, and in the discussion above, the re-modelling of the presented ADM to define a preference ordering over a collection of choices (rather than a partitioning of choices into two sets) is a topic of future work designed to facilitate the evaluation of my argumentation-based interpreter (AR-GOLOG).

This thesis has only scratched the surface of possibilities surrounding the use of heuristic search within a GOLOG interpreter. A range of avenues for its further development are described in Sections 6.7 and 7.7. One key area of future development is to incor-

porate an ability to combine both offline and online interpretation within the heuristic interpreter I have presented. I describe a strategy for achieving this in Section 6.7. The implementation of this online interpreter presents another area of future work.

Within Section 6.1.4 a range of compilation-based approaches to the interpretation of GOLOG programs [Baier et al. (2007b, 2008); Fritz et al. (2008)] are described. These works compile a GOLOG program into the ADL subset [Pednault (1989)] of PDDL [Fox and Long (2003); Gerevini and Long (2005)] – the solution of which (an execution of the original program) can be found by a (preference or non-preference-based) classical planner. To investigate more fully the advantages of remaining within the GOLOG and situation calculus space, relative to compilation, I would like to consider the relative performance of this interpreter with these compilation-based works on the subset of GOLOG programs and basic action theories (BATs) that are suitable for compilation.

Plans for the further development of ABS and AR-GOLOG are outlined in Section 8.5 of Chapter 8. These ideas are focused on facilitating the evaluation of the AR-GOLOG interpreter and reducing the time cost incurred by its generality. One such strategy is described in Section 8.5, involving the use of choice clustering to minimise the number of pair-wise comparisons required in an application of an ADM. Similar choices are clustered, and treated as individual decisions during argumentation. Choices within the most preferred subset of clusters can then be organised into smaller clusters, repeating this process of ADM application until each cluster contains only one decision.

This thesis has explored the extent to which automated decision-making and planning techniques that are decoupled from the preferences supplied to them can be developed. I have discovered that such techniques can be created through the use of computational argumentation. In the process I have created two decoupled systems – an argumentation-based decision-making system (ADM) and an argumentation-based search (ABS) method (applied in the development of an argumentation-based GOLOG interpreter). These techniques represent a first step toward the creation of applications that interact with a human user, eliciting from them their preference in a form they are comfortable with, and suggesting to them a decision or plan on the basis of these preferences.

Appendix A
Results

Each experiment conducted within this thesis has been run on a Windows XP (Service Pack 3) machine with a 2.40 GHZ Intel(R) Core(TM)2 CPU and 2GB of RAM.

Table A.1: Observations made in executions found by the standard (O_S), greedy (O_G), and heuristic (O_H) interpreters. For each test, N_T and CLUSTERS denote the number of celestial targets, and the cluster configuration (size and number of clusters), while T_S, T_G, and T_H denote the time taken (in seconds) by the standard, greedy, and heuristic interpreters.

N_T	CLUSTERS	TIME	FUEL	POWER	O_S	O_G	O_H	T_S (s)	T_G (s)	T_H (s)
10	2,2,2,2,2	150	500	200	2	2	2	0.00	0.31	0.23
10	2,2,2,2,2	150	500	200	0	0	2	0.02	0.09	0.39
10	2,2,2,2,2	150	500	200	6	6	6	0.05	0.48	0.28
10	2,2,2,2,2	150	500	200	2	2	2	0.14	1.11	0.92
10	2,2,2,2,2	150	500	200	2	2	3	0.11	0.88	0.08
10	2,2,2,2,2	150	500	200	2	2	2	0.02	0.11	0.08
10	2,2,2,2,2	150	500	200	4	4	4	0.05	0.23	0.22
10	2,2,2,2,2	150	500	200	2	2	2	0.02	0.20	0.16
10	2,2,2,2,2	150	500	200	2	2	2	0.13	1.47	1.09
10	2,2,2,2,2	150	500	200	0	0	4	0.00	0.09	0.14
20	2,2,5,2,2,5,1,1	250	600	300	3	3	9	0.13	1.94	1.20
20	2,2,5,2,2,5,1,1	250	600	300	2	8	8	0.08	1.78	0.98
20	2,2,5,2,2,5,1,1	250	600	300	4	6	12	0.16	0.36	1.81
20	2,2,5,2,2,5,1,1	250	600	300	5	10	11	0.02	0.52	1.70
20	2,2,5,2,2,5,1,1	250	600	300	3	3	8	0.05	1.08	1.38

Continued on Next Page...

N_T	CLUSTERS	TIME	FUEL	POWER	O_S	O_G	O_H	T_S (s)	T_G (s)	T_H (s)
20	2,2,5,2,2,5,1,1	250	600	300	4	7	10	0.34	0.28	1.09
20	2,2,5,2,2,5,1,1	250	600	300	2	7	8	0.30	0.17	0.88
20	2,2,5,2,2,5,1,1	250	600	300	3	4	6	0.00	3.00	2.70
20	2,2,5,2,2,5,1,1	250	600	300	3	4	8	0.02	3.14	0.67
20	2,2,5,2,2,5,1,1	250	600	300	2	5	11	0.25	3.64	1.34
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	5	8	18	0.06	3.45	8.80
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	5	13	19	0.03	3.89	8.84
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	7	14	14	0.02	1.80	5.08
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	9	14	14	0.19	9.88	5.30
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	5	8	16	0.06	6.14	6.59
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	8	14	17	0.02	0.72	6.94
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	5	8	17	0.02	4.80	9.28
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	9	12	18	0.25	0.73	8.39
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	4	11	16	0.17	10.34	5.88
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	5	8	18	0.00	3.61	8.44
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	16	17	22	0.14	10.86	23.08
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	5	18	24	0.19	4.42	28.77
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	16	23	24	0.08	3.39	27.63

Continued on Next Page…

N_T	CLUSTERS	TIME	FUEL	POWER	O_S	O_G	O_H	T_S (s)	T_G (s)	T_H (s)
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	5	23	23	0.23	16.23	26.91
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	14	18	20	0.13	2.27	18.58
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	10	21	22	0.03	3.73	21.97
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	11	24	24	0.13	5.20	27.78
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	5	17	21	0.27	3.30	20.28
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	13	15	24	0.05	7.63	26.78
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	10	22	25	0.19	2.67	28.08
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	8	26	31	0.97	7.42	83.94
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	9	28	31	0.94	17.42	69.88
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	13	24	31	0.55	7.28	83.92
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	14	22	29	0.64	28.77	72.19
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	13	32	33	0.48	10.05	79.81
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	7	32	32	1.22	9.92	83.66
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	9	19	31	1.11	16.80	76.33
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	8	20	35	0.97	9.69	97.25
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	7	27	30	1.38	19.02	75.17
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	12	27	34	0.91	10.77	86.70

End of Table.

Table A.2: Deviation from desired stockpile target in executions found by the standard (D_S), greedy (D_G), and heuristic (D_H) interpreters. N_B, R_T, and N_H denote the number of blocks, required stockpile tonnage (in tonnes), and number of hazards in each test. GROUPS defines the size and number of dependent block groups. T_S, T_G and T_H denote the time (in seconds) required to find an execution by the standard, greedy and heuristic interpreters (respectively).

N_B	R_T (tonnes)	GROUPS	N_H	D_S	D_G	D_H	T_S (s)	T_G (s)	T_H (s)
5	1500	1,1,3	5	6.84	3.38	3.38	0.06	0.08	0.08
5	1500	1,1,3	5	4.32	7.16	4.32	0.05	0.06	0.05
5	1500	2,1,2	5	2.83	9.90	2.83	0.05	0.09	0.06
5	1500	2,1,2	5	6.97	6.12	2.55	0.03	0.08	0.08
5	1500	1,2,2	5	10.15	7.29	6.04	0.03	0.06	0.03
5	1500	1,2,2	5	7.20	7.20	7.20	0.09	0.09	0.09
10	2500	1,3,2,3,1	5	2.09	1.99	1.15	0.05	0.13	0.14
10	2500	1,3,2,3,1	5	3.27	5.90	0.75	0.11	0.13	0.13
10	2500	2,1,4,2,1	5	7.46	3.73	3.73	0.08	0.08	0.09
10	2500	2,1,4,2,1	5	6.14	5.51	3.89	0.06	0.08	0.09
10	2500	2,1,3,2,2	5	9.56	2.32	1.53	0.14	0.13	0.16
10	2500	2,1,3,2,2	5	5.85	6.22	4.01	0.08	0.08	0.14
15	3500	5,2,3,2,3	5	6.67	5.19	3.94	0.11	0.11	0.27
15	3500	5,2,3,2,3	5	2.75	2.03	0.76	0.14	0.14	0.36

Continued on Next Page...

N_B	R_T (tonnes)	GROUPS	N_H	D_S	D_G	D_H	T_S (s)	T_G (s)	T_H (s)
15	3500	5,2,3,1,1,3	5	6.14	5.46	6.08	0.11	0.11	0.34
15	3500	5,2,3,1,1,3	5	4.29	1.78	1.70	0.13	0.13	0.34
15	3500	4,2,3,1,2,3	5	6.27	4.00	3.11	0.13	0.17	0.34
15	3500	4,2,3,1,2,3	5	7.21	8.72	3.38	0.13	0.09	0.34
20	4500	6,2,2,3,2,5	10	3.27	2.01	0.83	0.33	0.25	0.75
20	4500	6,2,2,3,2,5	10	1.85	1.43	2.11	0.31	0.45	0.81
20	4500	4,2,2,3,2,5,2	10	3.14	5.32	0.93	0.23	0.25	0.84
20	4500	4,2,2,3,2,5,2	10	3.13	4.66	1.03	0.25	0.30	0.81
20	4500	4,2,3,3,2,4,2	10	3.99	3.80	2.06	0.41	0.42	1.02
20	4500	4,2,3,3,2,4,2	10	1.83	2.62	0.53	0.20	0.28	0.78
25	5500	4,2,3,3,5,2,4,2	10	2.60	2.83	2.47	0.39	0.48	1.69
25	5500	4,2,3,3,5,2,4,2	10	1.19	1.92	0.83	0.23	0.23	1.59
25	5500	4,2,3,3,6,1,4,2	10	2.62	4.09	0.86	0.67	0.70	1.94
25	5500	4,2,3,3,6,1,4,2	10	4.69	0.55	0.55	0.47	0.39	1.73
25	5500	4,2,3,3,4,2,4,2,1	10	6.40	0.97	0.97	0.44	0.38	1.84
25	5500	4,2,3,3,4,2,4,2,1	10	5.49	2.05	0.97	0.30	0.27	1.67
30	6500	2,3,4,5,3,3,2,4,2,1,1	15	4.69	2.46	2.01	0.55	0.95	4.19
30	6500	2,3,4,5,3,3,2,4,2,1,1	15	1.54	4.10	0.36	0.75	0.88	3.91

Continued on Next Page...

N_B	R_T (tonnes)	GROUPS	N_H	D_S	D_G	D_H	T_S (s)	T_G (s)	T_H (s)
30	6500	4,2,5,3,3,4,2,5,2	15	3.45	3.87	1.14	1.22	0.88	3.66
30	6500	4,2,5,3,3,4,2,5,2	15	2.55	3.43	1.88	0.94	0.64	3.28
30	6500	4,2,3,3,3,4,2,5,2,2	15	2.50	3.49	0.71	0.72	0.89	4.03
30	6500	4,2,3,3,3,4,2,5,2,2	15	2.90	0.72	0.82	1.14	1.42	3.89
35	7500	4,2,3,3,5,3,4,2,5,2,2	15	2.34	2.74	1.24	1.19	1.31	6.67
35	7500	4,2,3,3,5,3,4,2,5,2,2	15	1.38	2.52	0.53	0.53	0.69	6.06
35	7500	4,2,3,3,5,3,4,2,5,4	15	1.22	2.60	0.70	0.98	1.39	5.50
35	7500	4,2,3,3,5,3,4,2,5,4	15	1.23	2.58	0.34	1.02	1.19	6.36
35	7500	4,2,3,3,5,3,1,3,2,5,4	15	1.23	4.34	0.42	1.11	0.97	6.56
35	7500	4,2,3,3,5,3,1,3,2,5,4	15	2.02	2.08	0.82	0.92	1.03	6.22
40	8500	4,2,3,3,5,6,3,1,3,2,5,3	20	2.37	1.58	0.81	1.34	1.28	10.45
40	8500	4,2,3,3,5,6,3,1,3,2,5,3	20	3.23	4.16	1.70	1.34	1.48	10.44
40	8500	4,2,3,3,5,6,3,3,3,5,1,1,1	20	5.14	4.72	0.95	0.98	1.92	11.66
40	8500	4,2,3,3,5,6,3,3,3,5,1,1,1	20	3.92	0.69	0.58	0.44	1.02	11.06
40	8500	1,3,2,3,3,5,6,2,1,3,3,5,1,1,1	20	0.49	1.68	0.41	0.84	1.17	11.86
40	8500	1,3,2,3,3,5,6,2,1,3,3,5,1,1,1	20	2.91	3.76	0.59	0.95	1.45	12.53
45	9500	1,3,2,3,3,5,6,2,1,3,3,5,1,5,1,1	20	3.10	1.78	1.07	1.28	1.11	19.25
45	9500	1,3,2,3,3,5,6,2,1,3,3,5,1,5,1,1	20	3.63	2.66	1.30	1.39	1.33	19.59

Continued on Next Page...

N_B	R_T (tonnes)	GROUPS	N_H	D_S	D_G	D_H	T_S (s)	T_G (s)	T_H (s)
45	9500	5,2,3,3,4,5,4,1,3,1,2,2,1,1,6,1,1	20	4.78	2.38	1.40	1.88	2.39	21.61
45	9500	5,2,3,3,4,5,4,1,3,1,2,2,1,1,6,1,1	20	4.13	1.65	0.67	1.75	1.72	21.34
45	9500	1,5,2,2,3,4,5,5,3,3,2,1,1,4,2,1,1	20	2.02	1.00	0.38	2.03	1.41	20.91
45	9500	1,5,2,2,3,4,5,5,3,3,2,1,1,4,2,1,1	20	3.08	0.73	0.25	1.44	1.38	20.56
50	10500	1,3,2,3,2,5,1,5,2,4,2,1,3,3,5,1,6,1	25	3.03	0.56	0.38	1.88	2.14	30.42
50	10500	1,3,2,3,2,5,1,5,2,4,2,1,3,3,5,1,6,1	25	3.50	2.17	0.93	3.14	3.95	34.30
50	10500	1,3,2,3,2,2,3,1,2,2,1,2,4,2,1,3,3,5,1,6,1	25	2.07	2.55	0.31	1.56	1.97	33.70
50	10500	1,3,2,3,2,2,3,1,2,2,1,2,4,2,1,3,3,5,1,6,1	25	1.20	3.43	0.78	3.56	2.22	32.55
50	10500	1,3,5,2,2,3,1,2,2,1,2,4,2,1,6,5,1,6,1	25	5.43	1.47	0.63	2.38	2.05	33.44
50	10500	1,3,5,2,2,3,1,2,2,1,2,4,2,1,6,5,1,6,1	25	2.61	1.93	0.75	2.66	3.36	33.50

End of Table.

Table A.3: Number of task-to-worker assignments in executions found by the standard (NT_S), greedy (NT_G), and heuristic (NT_H) interpreters (in the task scheduling domain). N_{JOBS}, N_W, T_{MAX}, GROUPS, and TIME denote the number of jobs, workers, maximum number of tasks per job, groups of dependent jobs, and the number of jobs allowed to be scheduled. T_S, T_G, and T_H denote the time taken (in seconds) to solve each test by the standard, greedy, and heuristic interpreters. The symbol '–' denotes that the interpreter could not find an execution within the 15 minute time limit.

N_{JOBS}	N_W	T_{MAX}	GROUPS	TIME	NT_S	NT_G	NT_H	T_S (s)	T_G (s)	T_H (s)
15	8	7	1,1,3,3,2,2,3	5	21	18	22	0.00	0.03	1.55
15	8	7	1,1,3,3,2,2,3	5	12	18	24	0.03	0.06	4.34
15	8	7	1,1,3,3,2,2,3	5	17	19	20	0.09	0.31	4.17
15	8	7	1,1,3,3,2,2,3	5	7	15	19	0.00	0.00	12.48
15	8	7	1,1,3,3,2,2,3	5	13	15	19	0.05	0.02	2.09
15	6	5	1,1,3,3,2,2,3	5	11	12	15	0.02	0.02	0.80
15	6	5	1,1,3,3,2,2,3	5	9	13	15	0.00	0.02	0.81
15	6	5	1,1,3,3,2,2,3	5	11	8	14	0.00	0.02	0.47
15	6	5	1,1,3,3,2,2,3	5	13	13	15	0.02	0.02	0.55
15	6	5	1,1,3,3,2,2,3	5	8	11	15	0.02	0.03	1.48
20	8	7	1,1,1,3,3,3,4,3,1	10	30	30	38	0.11	0.19	24.59
20	8	7	1,1,1,3,3,3,4,3,1	10	25	31	40	0.03	0.05	8.92
20	8	7	1,1,1,3,3,3,4,3,1	10	25	34	43	0.00	0.05	7.38

Continued on Next Page...

N_{JOBS}	N_W	T_{MAX}	GROUPS	TIME	NT_S	NT_G	NT_H	T_S (s)	T_G (s)	T_H (s)
20	8	7	1,1,1,3,3,3,4,3,1	10	29	36	39	0.03	0.06	6.48
20	8	7	1,1,1,3,3,3,4,3,1	10	34	35	38	1.45	2.69	172.14
20	6	5	1,1,2,2,3,3,4,3,1	10	25	35	35	0.00	0.05	4.19
20	6	5	1,1,2,2,3,3,4,3,1	10	24	27	31	0.02	0.05	3.92
20	6	5	1,1,2,2,3,3,4,3,1	10	27	27	32	0.02	0.05	4.17
20	6	5	1,1,2,2,3,3,4,3,1	10	18	19	22	0.02	0.03	3.03
20	6	5	1,1,2,2,3,3,4,3,1	10	24	30	30	0.02	0.06	3.03
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	38	39	48	0.00	0.08	14.95
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	42	37	49	0.03	0.06	15.78
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	39	40	46	0.02	0.08	13.83
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	38	43	48	0.05	0.20	22.50
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	37	42	44	0.08	0.13	19.66
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	53	53	–	6.91	8.56	–
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	49	51	63	0.03	0.13	56.27
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	40	54	54	0.02	0.58	82.28
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	62	64	68	1.77	2.52	519.58
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	37	46	49	0.02	0.08	14.17
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	76	82	88	0.06	0.19	71.73

Continued on Next Page…

N_{JOBS}	N_W	T_{MAX}	GROUPS	TIME	NT_S	NT_G	NT_H	T_S (s)	T_G (s)	T_H (s)
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	62	77	79	0.05	3.28	158.61
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	57	73	83	0.09	0.19	99.80
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	75	85	–	2.89	14.05	–
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	61	76	76	1.25	1.86	714.81
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	56	63	63	0.08	0.16	36.13
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	50	55	57	0.02	0.11	29.56
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	56	65	65	0.06	0.19	59.25
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	55	59	63	0.08	0.25	54.83
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	51	60	60	0.02	0.14	31.77

End of Table.

Table A.4: Observations made in executions found by the heuristic (O_H), and (non-lookahead-based) best-first search (O_B) interpreters. For each test, N_T denotes the number of celestial targets, while T_H, and T_B denote the time taken (in seconds) by the heuristic and best-first search interpreters. Symbol '–' denotes that a solution could not be found within 15 minutes or available memory.

N_T	TIME	FUEL	POWER	O_H	T_H (s)	O_B	T_B (s)
10	150	500	200	2	0.23	2	0.08
10	150	500	200	2	0.39	2	0.14
10	150	500	200	6	0.28	2	0.20
10	150	500	200	2	0.92	2	0.30
10	150	500	200	3	0.08	2	0.38
10	150	500	200	2	0.08	2	0.23
10	150	500	200	4	0.22	1	0.11
10	150	500	200	2	0.16	2	0.20
10	150	500	200	2	1.09	2	0.11
10	150	500	200	4	0.14	2	0.81
20	250	600	300	9	1.20	1	2.19
20	250	600	300	8	0.98	4	0.77
20	250	600	300	12	1.81	4	0.50
20	250	600	300	11	1.70	2	0.53
20	250	600	300	8	1.38	2	0.50
20	250	600	300	10	1.09	2	0.42
20	250	600	300	8	0.88	3	1.56
20	250	600	300	6	2.70	6	1.86
20	250	600	300	8	0.67	4	0.81
20	250	600	300	11	1.34	3	0.98
30	350	700	400	18	8.80	3	0.91
30	350	700	400	19	8.84	3	0.98

Continued on Next Page...

N_T	TIME	FUEL	POWER	O_H	T_H (s)	O_B	T_B (s)
30	350	700	400	14	5.08	4	1.17
30	350	700	400	14	5.30	7	3.56
30	350	700	400	16	6.59	4	1.53
30	350	700	400	17	6.94	6	2.81
30	350	700	400	17	9.28	6	9.17
30	350	700	400	18	8.39	11	5.30
30	350	700	400	16	5.88	1	4.69
30	350	700	400	18	8.44	3	1.03
40	450	800	500	22	23.08	12	5.72
40	450	800	500	24	28.77	9	3.72
40	450	800	500	24	27.63	9	4.48
40	450	800	500	23	26.91	12	5.48
40	450	800	500	20	18.58	6	4.66
40	450	800	500	22	21.97	9	7.38
40	450	800	500	24	27.78	8	4.75
40	450	800	500	21	20.28	6	4.67
40	450	800	500	24	26.78	11	4.98
40	450	800	500	25	28.08	6	4.94
50	550	900	600	31	83.94	–	–
50	550	900	600	31	69.88	13	20.56
50	550	900	600	31	83.92	–	–
50	550	900	600	29	72.19	12	26.27
50	550	900	600	33	79.81	11	23.81
50	550	900	600	32	83.66	11	24.33
50	550	900	600	31	76.33	13	24.84
50	550	900	600	35	97.25	10	37.53
50	550	900	600	30	75.17	13	21.77

Continued on Next Page...

N_T	TIME	FUEL	POWER	O_H	T_H (s)	O_B	T_B (s)
50	550	900	600	34	86.70	16	30.64

End of Table.

Table A.5: Deviation from desired stockpile target in executions found by the heuristic (D_H), and (non-lookahead-based) best-first search (D_B) interpreters. N_B, R_T, and N_H denote the number of blocks, required stockpile tonnage, and number of hazards in each test. T_H and T_B denote the time required for interpretation by the heuristic and best-first search interpreters. The symbol '–' denotes that a solution could not be found within 15 minutes or available memory.

N_B	R_T (tonnes)	N_H	D_H	T_H (s)	D_B	T_B (s)
5	1500	5	3.38	0.08	4.35	0.19
5	1500	5	4.32	0.05	5.10	0.16
5	1500	5	2.83	0.06	5.55	0.19
5	1500	5	2.55	0.08	2.55	0.14
5	1500	5	6.04	0.03	6.04	0.11
5	1500	5	7.20	0.09	7.20	0.16
10	2500	5	1.15	0.14	1.99	0.30
10	2500	5	0.75	0.13	1.22	0.44
10	2500	5	3.73	0.09	2.25	0.33
10	2500	5	3.89	0.09	3.98	0.19
10	2500	5	1.53	0.16	4.18	0.48
10	2500	5	4.01	0.14	3.60	0.30
15	3500	5	3.94	0.27	4.58	0.38
15	3500	5	0.76	0.36	0.91	0.78
15	3500	5	6.08	0.34	6.08	0.53
15	3500	5	1.70	0.34	0.65	0.58
15	3500	5	3.11	0.34	0.48	0.53

Continued on Next Page...

N_B	R_T (tonnes)	N_H	D_H	T_H (s)	D_B	T_B (s)
15	3500	5	3.38	0.34	4.21	0.59
20	4500	10	0.83	0.75	1.78	1.70
20	4500	10	2.11	0.81	2.22	1.77
20	4500	10	0.93	0.84	0.41	1.31
20	4500	10	1.03	0.81	1.58	1.66
20	4500	10	2.06	1.02	2.90	3.56
20	4500	10	0.53	0.78	1.32	1.91
25	5500	10	2.47	1.69	2.91	2.05
25	5500	10	0.83	1.59	0.81	1.47
25	5500	10	0.86	1.94	0.79	3.77
25	5500	10	0.55	1.73	1.37	3.28
25	5500	10	0.97	1.84	0.72	3.48
25	5500	10	0.97	1.67	0.81	3.36
30	6500	15	2.01	4.19	1.25	5.95
30	6500	15	0.36	3.91	1.04	8.36
30	6500	15	1.14	3.66	2.55	7.53
30	6500	15	1.88	3.28	0.93	4.38
30	6500	15	0.71	4.03	0.70	7.63
30	6500	15	0.82	3.89	0.69	5.27
35	7500	15	1.24	6.67	0.64	11.50
35	7500	15	0.53	6.06	0.60	3.63
35	7500	15	0.70	5.50	1.11	8.64
35	7500	15	0.34	6.36	0.87	18.64
35	7500	15	0.42	6.56	0.73	11.55
35	7500	15	0.82	6.22	0.68	7.89
40	8500	20	0.81	10.45	0.54	13.02
40	8500	20	1.70	10.44	–	–

Continued on Next Page...

N_B	R_T (tonnes)	N_H	D_H	T_H (s)	D_B	T_B (s)
40	8500	20	0.95	11.66	1.03	14.97
40	8500	20	0.58	11.06	0.72	9.27
40	8500	20	0.41	11.86	0.88	11.64
40	8500	20	0.59	12.53	0.69	12.81
45	9500	20	1.07	19.25	0.59	13.80
45	9500	20	1.30	19.59	0.83	11.72
45	9500	20	1.40	21.61	0.48	23.39
45	9500	20	0.67	21.34	0.41	18.56
45	9500	20	0.38	20.91	0.25	24.67
45	9500	20	0.25	20.56	0.53	19.05
50	10500	25	0.38	30.42	0.27	28.64
50	10500	25	0.93	34.30	0.59	49.16
50	10500	25	0.31	33.70	0.22	33.81
50	10500	25	0.78	32.55	0.27	50.00
50	10500	25	0.63	33.44	0.43	37.17
50	10500	25	0.75	33.50	0.45	48.58

End of Table.

Table A.6: Number of task-to-worker assignments in executions found by the heuristic (NT_H) and (non-lookahead-based) best-first search (NT_B) interpreters. N_{JOBS}, N_W, and TIME denote the number of jobs, workers, and the number of jobs allowed to be scheduled. T_H and T_B denote the time taken (in seconds) to solve each test by the heuristic and best-first search interpreters. The symbol '–' denotes that a solution could not be found within 15 minutes or available memory.

N_{JOBS}	N_W	TIME	NT_H	T_H (s)	NT_B	T_B (s)
15	8	5	22	1.55	11	0.11
15	8	5	24	4.34	18	0.64

Continued on Next Page...

N_{JOBS}	N_W	TIME	NT_H	T_H (s)	NT_B	T_B (s)
15	8	5	20	4.17	17	0.11
15	8	5	19	12.48	18	0.14
15	8	5	19	2.09	–	–
15	6	5	15	0.80	15	0.41
15	6	5	15	0.81	13	0.06
15	6	5	14	0.47	13	0.25
15	6	5	15	0.55	14	0.06
15	6	5	15	1.48	13	0.05
20	8	10	38	24.59	–	–
20	8	10	40	8.92	37	10.92
20	8	10	43	7.38	32	0.42
20	8	10	39	6.48	33	0.36
20	8	10	38	172.14	–	–
20	6	10	35	4.19	31	0.23
20	6	10	31	3.92	24	0.41
20	6	10	32	4.17	22	0.61
20	6	10	22	3.03	20	0.13
20	6	10	30	3.03	22	0.47
30	6	15	48	14.95	38	0.55
30	6	15	49	15.78	39	1.31
30	6	15	46	13.83	34	0.86
30	6	15	48	22.50	39	0.52
30	6	15	44	19.66	37	0.81
30	8	15	–	–	40	1.69
30	8	15	63	56.27	–	–
30	8	15	54	82.28	–	–
30	8	15	68	519.58	–	–

Continued on Next Page...

N_{JOBS}	N_W	TIME	NT_H	T_H (s)	NT_B	T_B (s)
30	8	15	49	14.17	–	–
40	8	20	88	71.73	–	–
40	8	20	79	158.61	–	–
40	8	20	83	99.80	–	–
40	8	20	–	–	–	–
40	8	20	76	714.81	–	–
40	6	20	63	36.13	50	1.41
40	6	20	57	29.56	45	3.66
40	6	20	65	59.25	–	–
40	6	20	63	54.83	45	2.45
40	6	20	60	31.77	55	1.09

End of Table.

Table A.7: Observations made in executions found by the heuristic (O_H) and non-relaxed (O_{NR}) interpreters. For each test, N_T and CLUSTERS denote the number of celestial targets, and the cluster configuration (size and number of clusters), while T_H and T_{NR} denote the time taken (in seconds) by the heuristic and non-relaxed heuristic interpreters.

N_T	CLUSTERS	TIME	FUEL	POWER	O_H	T_H (s)	O_{NR}	T_{NR} (s)
10	2,2,2,2,2	150	500	200	2	0.23	2	0.95
10	2,2,2,2,2	150	500	200	2	0.39	2	1.89
10	2,2,2,2,2	150	500	200	6	0.28	6	2.52
10	2,2,2,2,2	150	500	200	2	0.92	3	1.45
10	2,2,2,2,2	150	500	200	3	0.08	4	0.73
10	2,2,2,2,2	150	500	200	2	0.08	2	0.25
10	2,2,2,2,2	150	500	200	4	0.22	4	1.47
10	2,2,2,2,2	150	500	200	2	0.16	2	0.55
10	2,2,2,2,2	150	500	200	2	1.09	3	1.45
10	2,2,2,2,2	150	500	200	4	0.14	4	0.73
20	2,2,5,2,2,5,1,1	250	600	300	9	1.20	9	10.16
20	2,2,5,2,2,5,1,1	250	600	300	8	0.98	8	7.27
20	2,2,5,2,2,5,1,1	250	600	300	12	1.81	12	19.50
20	2,2,5,2,2,5,1,1	250	600	300	11	1.70	11	26.75
20	2,2,5,2,2,5,1,1	250	600	300	8	1.38	8	10.06

Continued on Next Page...

N_T	CLUSTERS	TIME	FUEL	POWER	O_H	T_H (s)	O_{NR}	T_{NR} (s)
20	2,2,5,2,2,5,1,1	250	600	300	10	1.09	10	4.97
20	2,2,5,2,2,5,1,1	250	600	300	8	0.88	8	7.81
20	2,2,5,2,2,5,1,1	250	600	300	6	2.70	6	20.20
20	2,2,5,2,2,5,1,1	250	600	300	8	0.67	8	8.66
20	2,2,5,2,2,5,1,1	250	600	300	11	1.34	11	16.64
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	18	8.80	18	77.23
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	19	8.84	19	72.17
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	14	5.08	14	42.70
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	14	5.30	14	47.45
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	16	6.59	16	38.09
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	17	6.94	17	40.06
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	17	9.28	17	100.28
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	18	8.39	18	56.22
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	16	5.88	16	42.98
30	2,2,5,2,5,2,5,1,4,1,1	350	700	400	18	8.44	18	63.61
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	22	23.08	22	145.22
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	24	28.77	24	201.52
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	24	27.63	24	180.28

Continued on Next Page...

N_T	CLUSTERS	TIME	FUEL	POWER	O_H	T_H (s)	O_{NR}	T_{NR} (s)
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	23	26.91	24	185.98
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	20	18.58	20	143.59
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	22	21.97	22	165.14
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	24	27.78	24	200.45
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	21	20.28	20	114.75
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	24	26.78	24	166.02
40	2,2,5,2,5,2,5,1,4,1,5,2,2,2	450	800	500	25	28.08	25	234.81
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	31	83.94	31	500.64
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	31	69.88	31	436.52
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	31	83.92	31	445.06
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	29	72.19	29	444.27
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	33	79.81	32	503.92
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	32	83.66	32	486.19
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	31	76.33	31	443.27
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	35	97.25	35	480.09
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	30	75.17	30	414.34
50	3,2,2,5,2,5,2,3,5,1,4,1,5,2,2,6	550	900	600	34	86.70	34	591.72

End of Table.

Table A.8: Deviation from desired stockpile target in executions found by the heuristic (D_H) and non-relaxed (D_{NR}) interpreters. N_B, R_T, and N_H denote the number of blocks, required stockpile tonnage, and number of hazards in each test. GROUPS defines the size and number of dependent block groups. T_H and T_{NR} denote the time (in seconds) required to find an execution by the heuristic and non-relaxed interpreters (respectively).

N_B	R_T (tonnes)	GROUPS	N_H	D_H	T_H (s)	D_{NR}	T_{NR} (s)
5	1500	1,1,3	5	3.38	0.08	3.38	0.41
5	1500	1,1,3	5	4.32	0.05	4.32	0.30
5	1500	2,1,2	5	2.83	0.06	2.83	0.50
5	1500	2,1,2	5	2.55	0.08	2.55	0.44
5	1500	1,2,2	5	6.04	0.03	6.04	0.27
5	1500	1,2,2	5	7.20	0.09	7.20	0.33
10	2500	1,3,2,3,1	5	1.15	0.14	1.15	1.34
10	2500	1,3,2,3,1	5	0.75	0.13	0.75	1.53
10	2500	2,1,4,2,1	5	3.73	0.09	3.73	0.98
10	2500	2,1,4,2,1	5	3.89	0.09	3.89	0.88
10	2500	2,1,3,2,2	5	1.53	0.16	1.53	1.66
10	2500	2,1,3,2,2	5	4.01	0.14	4.01	1.27
15	3500	5,2,3,2,3	5	3.94	0.27	3.94	1.72
15	3500	5,2,3,2,3	5	0.76	0.36	0.76	3.86

Continued on Next Page...

N_B	R_T (tonnes)	GROUPS	N_H	D_H	T_H (s)	D_{NR}	T_{NR} (s)
15	3500	5,2,3,1,1,3	5	6.08	0.34	5.46	2.39
15	3500	5,2,3,1,1,3	5	1.70	0.34	1.70	2.78
15	3500	4,2,3,1,2,3	5	3.11	0.34	3.11	2.73
15	3500	4,2,3,1,2,3	5	3.38	0.34	3.38	2.94
20	4500	6,2,2,3,2,5	10	0.83	0.75	0.83	7.38
20	4500	6,2,2,3,2,5	10	2.11	0.81	1.43	10.45
20	4500	4,2,2,3,2,5,2	10	0.93	0.84	0.93	7.81
20	4500	4,2,2,3,2,5,2	10	1.03	0.81	1.03	9.30
20	4500	4,2,3,3,2,4,2	10	2.06	1.02	2.06	13.55
20	4500	4,2,3,3,2,4,2	10	0.53	0.78	0.53	9.17
25	5500	4,2,3,3,5,2,4,2	10	2.47	1.69	1.73	14.23
25	5500	4,2,3,3,5,2,4,2	10	0.83	1.59	0.83	10.89
25	5500	4,2,3,3,6,1,4,2	10	0.86	1.94	0.86	26.34
25	5500	4,2,3,3,6,1,4,2	10	0.55	1.73	0.55	18.91
25	5500	4,2,3,3,4,2,4,2,1	10	0.97	1.84	0.97	16.91
25	5500	4,2,3,3,4,2,4,2,1	10	0.97	1.67	0.97	16.86
30	6500	2,3,4,5,3,3,2,4,2,1,1	15	2.01	4.19	2.01	61.66
30	6500	2,3,4,5,3,3,2,4,2,1,1	15	0.36	3.91	0.36	48.08

Continued on Next Page...

N_B	R_T (tonnes)	GROUPS	N_H	D_H	T_H (s)	D_{NR}	T_{NR} (s)
30	6500	4,2,5,3,3,4,2,5,2	15	1.14	3.66	1.14	58.84
30	6500	4,2,5,3,3,4,2,5,2	15	1.88	3.28	0.50	32.88
30	6500	4,2,3,3,3,4,2,5,2,2	15	0.71	4.03	0.71	66.61
30	6500	4,2,3,3,3,4,2,5,2,2	15	0.82	3.89	0.69	67.98
35	7500	4,2,3,3,5,3,4,2,5,2,2	15	1.24	6.67	1.24	93.86
35	7500	4,2,3,3,5,3,4,2,5,2,2	15	0.53	6.06	0.62	35.92
35	7500	4,2,3,3,5,3,4,2,5,4	15	0.70	5.50	1.52	89.89
35	7500	4,2,3,3,5,3,4,2,5,4	15	0.34	6.36	0.34	88.16
35	7500	4,2,3,3,5,3,1,3,2,5,4	15	0.42	6.56	0.42	89.09
35	7500	4,2,3,3,5,3,1,3,2,5,4	15	0.82	6.22	0.26	70.13
40	8500	4,2,3,3,5,6,3,1,3,2,5,3	20	0.81	10.45	0.81	130.80
40	8500	4,2,3,3,5,6,3,1,3,2,5,3	20	1.70	10.44	2.28	77.64
40	8500	4,2,3,3,5,6,3,3,3,5,1,1,1	20	0.95	11.66	0.95	176.25
40	8500	4,2,3,3,5,6,3,3,3,5,1,1,1	20	0.58	11.06	0.58	125.02
40	8500	1,3,2,3,3,5,6,2,1,3,3,5,1,1,1	20	0.41	11.86	0.39	112.44
40	8500	1,3,2,3,3,5,6,2,1,3,3,5,1,1,1	20	0.59	12.53	0.44	160.98
45	9500	1,3,2,3,3,5,6,2,1,3,3,5,1,5,1,1	20	1.07	19.25	0.47	135.78
45	9500	1,3,2,3,3,5,6,2,1,3,3,5,1,5,1,1	20	1.30	19.59	1.04	121.28

Continued on Next Page...

N_B	R_T (tonnes)	GROUPS	N_H	D_H	T_H (s)	D_{NR}	T_{NR} (s)
45	9500	5,2,3,3,4,5,4,1,3,1,2,2,1,1,6,1,1	20	1.40	21.61	0.28	277.31
45	9500	5,2,3,3,4,5,4,1,3,1,2,2,1,1,6,1,1	20	0.67	21.34	0.28	268.66
45	9500	1,5,2,2,3,4,5,5,3,3,2,1,1,4,2,1,1	20	0.38	20.91	0.38	205.83
45	9500	1,5,2,2,3,4,5,5,3,3,2,1,1,4,2,1,1	20	0.25	20.56	0.25	194.88
50	10500	1,3,2,3,2,5,1,5,2,4,2,1,3,3,5,1,6,1	25	0.38	30.42	0.26	292.80
50	10500	1,3,2,3,2,5,1,5,2,4,2,1,3,3,5,1,6,1	25	0.93	34.30	0.85	528.69
50	10500	1,3,2,3,2,2,3,1,2,2,1,2,4,2,1,3,3,5,1,6,1	25	0.31	33.70	0.29	362.30
50	10500	1,3,2,3,2,2,3,1,2,2,1,2,4,2,1,3,3,5,1,6,1	25	0.78	32.55	0.81	263.33
50	10500	1,3,5,2,2,3,1,2,2,1,2,4,2,1,6,5,1,6,1	25	0.63	33.44	0.95	285.59
50	10500	1,3,5,2,2,3,1,2,2,1,2,4,2,1,6,5,1,6,1	25	0.75	33.50	0.68	400.08

End of Table.

Table A.9: Number of task-to-worker assignments made in executions found by the heuristic (NT_H) and non-relaxed (NT_{NR}) interpreters (within the task scheduling domain). N_{JOBS}, N_W, T_{MAX}, GROUPS, and TIME denote the number of jobs, workers, maximum number of tasks per job, groups of dependent jobs, and the number of jobs allowed to be scheduled. T_H and T_{NR} denote the time required (in seconds) by the heuristic and non-relaxed interpreters to solve each test. The symbol '–' denotes that the interpreter could not find an execution within the 15 minute time limit.

N_{JOBS}	N_W	T_{MAX}	GROUPS	TIME	NT_H	T_H (s)	NT_{NR}	T_{NR} (s)
15	8	7	1,1,3,3,2,2,3	5	22	1.55	22	2.84
15	8	7	1,1,3,3,2,2,3	5	24	4.34	24	16.75
15	8	7	1,1,3,3,2,2,3	5	20	4.17	20	47.91
15	8	7	1,1,3,3,2,2,3	5	19	12.48	19	8.44
15	8	7	1,1,3,3,2,2,3	5	19	2.09	19	5.42
15	6	5	1,1,3,3,2,2,3	5	15	0.80	15	3.89
15	6	5	1,1,3,3,2,2,3	5	15	0.81	15	0.97
15	6	5	1,1,3,3,2,2,3	5	14	0.47	14	0.70
15	6	5	1,1,3,3,2,2,3	5	15	0.55	15	0.55
15	6	5	1,1,3,3,2,2,3	5	15	1.48	15	2.33
20	8	7	1,1,1,3,3,3,4,3,1	10	38	24.59	38	215.16
20	8	7	1,1,1,3,3,3,4,3,1	10	40	8.92	40	17.59
20	8	7	1,1,1,3,3,3,4,3,1	10	43	7.38	43	15.55

Continued on Next Page...

N_{JOBS}	N_W	T_{MAX}	GROUPS	TIME	NT_H	T_H (s)	NT_{NR}	T_{NR} (s)
20	8	7	1,1,1,3,3,3,4,3,1	10	39	6.48	39	8.41
20	8	7	1,1,1,3,3,3,4,3,1	10	38	172.14	38	289.45
20	6	5	1,1,2,2,3,3,4,3,1	10	35	4.19	35	4.56
20	6	5	1,1,2,2,3,3,4,3,1	10	31	3.92	31	12.50
20	6	5	1,1,2,2,3,3,4,3,1	10	32	4.17	32	5.27
20	6	5	1,1,2,2,3,3,4,3,1	10	22	3.03	22	1.95
20	6	5	1,1,2,2,3,3,4,3,1	10	30	3.03	30	5.23
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	48	14.95	48	15.73
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	49	15.78	49	27.92
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	46	13.83	46	12.56
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	48	22.50	48	50.83
30	6	5	1,1,2,2,4,4,3,2,1,3,3,4	15	44	19.66	44	15.73
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	—	—	—	—
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	63	56.27	—	—
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	54	82.28	54	83.05
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	68	519.58	—	—
30	8	7	1,1,2,2,4,4,1,3,2,3,3,4	15	49	14.17	49	24.67
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	88	71.73	88	115.59

Continued on Next Page…

N_{JOBS}	N_W	T_{MAX}	GROUPS	TIME	NT_H	T_H (s)	NT_{NR}	T_{NR} (s)
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	79	158.61	—	—
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	83	99.80	83	141.31
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	—	—	—	—
40	8	7	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	76	714.81	76	355.83
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	63	36.13	63	33.17
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	57	29.56	57	30.20
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	65	59.25	65	47.61
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	63	54.83	63	73.13
40	6	5	1,1,3,3,4,4,3,2,2,3,4,5,2,2,1	20	60	31.77	60	33.06

End of Table.

Appendix B
Additional Material

In this chapter I provide a number of definitions that complement the material in Chapter 8. These definitions characterise the presented travel planning domain. In Section B.2, I provide the implementation of a number of predicates used by AR-GOLOG (Chapter 8).

B.1 A Travel Planning Domain

In the travel planning domain, a planner has four actions available to it: $fly(a, b)$ allows the planner to fly from city a to b; $drive(a, b)$ allows the planner to drive from city a to b; $boat(a, b)$ allows the planner to sail from city a to b; and $sightsee(v, a)$ allows the planner to have the sight seeing adventure v in city a (where Table 8.1 defines the adventures available in each city). The precondition axiom for each action is shown in Definition B.1.1, below. These axioms refer to the fluents listed in Table 8.2 in Chapter 8.

Definition B.1.1 The precondition axioms for the actions of the travel domain are shown below. Recall the definition of a precondition axiom in Equation 5.1 of Section 5.1.3. The right hand side of each axiom must be uniform in the situation s.

$$Poss(drive(a,b),s) \equiv At(a,s) \wedge Road(a,b) \wedge \neg Been(b,s) \wedge \\ \exists f_u.FuelUsed(drive(a,b),f_u,s) \wedge FuelOk(f_u,s) \tag{B.1}$$

$$Poss(fly(a,b),s) \equiv At(a,s) \wedge Fpoint(a) \wedge Fpoint(b) \wedge \neg Been(b,s) \tag{B.2}$$

$$Poss(boat(a,b),s) \equiv At(a,s) \wedge SeaPath(a,b) \wedge \neg Been(b,s) \tag{B.3}$$

$$Poss(sightsee(v,a),s) \equiv At(a,s) \wedge Adventure(v,a) \wedge \neg Had(v,a,s) \tag{B.4}$$

The successor state axioms for the non-rigid fluents of Table 8.2 in Chapter 8 are shown in Definition B.1.2, below.

Definition B.1.2 The successor state axioms for the non-rigid fluents of Table 8.2 are shown below. Recall the definition of a successor state axiom for a relational fluent in Equation 5.2 of Section 5.1.4. The right hand side of each axiom must be uniform in the situation s. In the following, Attribute(α, $o[i]$) holds only if α is a sight seeing adventure action with attribute $o[i]$. Array(arr, l) holds only if arr is an array of length l.

$$At(a, do(\alpha, s)) \equiv \exists b.[\alpha = drive(b, a) \vee \alpha = fly(b, a) \vee \alpha = boat(b, a)] \vee$$
$$At(a, s) \wedge \neg\exists b'.[\alpha = drive(a, b') \vee \alpha = fly(a, b') \vee \alpha = boat(a, b')] \quad \text{(B.5)}$$

$$FuelOk(f', do(\alpha, s)) \equiv \neg[\exists f, f_u, q.FuelSpent(f, s) \wedge FuelUsed(\alpha, f_u, s) \wedge$$
$$Capacity(q) \wedge q < f + f_u + f'] \quad \text{(B.6)}$$

$$FuelSpent(f, do(\alpha, s)) \equiv \exists f_u.FuelUsed(\alpha, f_u, s) \wedge \exists f'.FuelSpent(f', s) \wedge$$
$$f = f_u + f' \quad \text{(B.7)}$$

$$FuelUsed(\alpha, f_u, s) \equiv [\exists a, b.\alpha = drive(a, b) \wedge Distance(a, b, d) \wedge f_u = d] \vee$$
$$[f_u = 0 \wedge \neg\exists a', b'.\alpha = drive(a', b')] \quad \text{(B.8)}$$

$$Been(a, do(\alpha, s)) \equiv [\exists b.\alpha = drive(b, a) \vee \alpha = fly(b, a) \vee \alpha = boat(b, a)]$$
$$\vee Been(a, s) \quad \text{(B.9)}$$

$$Had(v, a, do(\alpha, s)) \equiv \alpha = sightsee(v, a) \vee Had(v, a, s) \quad \text{(B.10)}$$

$$OneAdvent(do(\alpha, s)) \equiv \exists v, a.\alpha = sightsee(v, a) \vee OneAdvent(s) \quad \text{(B.11)}$$

$$Num(att, n, do(\alpha, s)) \equiv Num(att, n', s) \wedge [(Attribute(\alpha, att) \wedge n = n' + 1) \vee$$
$$(\neg Attribute(\alpha, att) \wedge n = n')] \quad \text{(B.12)}$$

$$Numbers(o, list, do(\alpha, s)) \equiv \exists l.Array(o, l) \wedge Array(list, l) \wedge \forall i.(1 \leq i \wedge i \leq l$$
$$\rightarrow \exists n_i.Num(o[i], n_i, s) \wedge [(Attribute(\alpha, o[i]) \wedge list[i] = n_i + 1) \vee$$
$$(\neg Attribute(\alpha, o[i]) \wedge list[i] = n_i)] \quad \text{(B.13)}$$

B.2 AR-GOLOG Supplement

Within the following definitions, Do'_R denotes a variation of the relaxed interpreter of Section 6.5 that does not use heuristic guidance when selecting transitions (but continues to relax the testing of action preconditions). I present an implementation of the SAMPLE function referred to within the EVAL function of Listing 8.2.2 in Definition B.2.2.

Definition B.2.1 Let N denote the number of relaxed executions required of a program δ in situation S. Let $doRSampler(PRT, \delta, S, S', H)$ denote a relaxed GOLOG interpreter designed to find a relaxed execution S' of δ in S that is sufficiently different from those found by previous invocations. Here, H denotes a finite action horizon[1], while PRT denotes the percentage of required executions discovered thus far (if $1/2$ have been found $PRT = 0.50$). Each transition clause of *doRSampler* has the form shown below, where:

1. (δ', S') denotes the configuration selected to transition to;
2. F denotes the action horizon that remains after this transition is made;
3. $next_R(H, \delta, S, Next_R)$ finds the set of configurations $Next_R$ that can be transitioned to under the semantics of Do'_R (a relaxed interpreter without heuristic guidance);
4. *shuffle* takes this set $Next_R$ and reorders it such that the first $PRT \times 100$ percent of the set is moved to its rear – its first element then selected for transition.

$$
\begin{aligned}
&transRSampler(PRT, H, \delta, S, \delta', S', F) \text{ :-} \\
&\quad next_R(H, \delta, S, Next_R), length(Next_R, LN), \\
&\quad shuffle(Next_R, LN * PRT, Shuffled), member([\delta', S', F], Shuffled).
\end{aligned}
$$

The predicate *doRSampler* is defined as follows:

$$
\begin{aligned}
&doRSampler(PRT, \delta, S, S', H) \text{ :-} \\
&\quad transRSampler^*(PRT, H, \delta, S, \delta', S', F), \\
&\quad (F = 0; finalRSampler(\delta', S')).
\end{aligned}
$$

Predicate $transRSampler^*$ is the reflexive transitive closure of *transRSampler*, while predicate $finalRSampler$ is equivalent to *Final* within the standard interpreter of Section 5.3.1.

[1]Recall the purpose of this finite action horizon during the relaxed interpretation of a program δ in a situation S (Section 6.5.2). The horizon H is an upper bound on the number of actions that can be added to the situation S during the relaxed interpretation of δ.

Example B.2.1 Let us consider an example use of the *shuffle* predicate described in Definition B.2.1. Imagine that the following set of configurations are available to transition to within an application of a *transRSampler* clause: $Next_R = [(\delta_1, s_1), (\delta_2, s_2), (\delta_3, s_3), (\delta_4, s_4)]$, and that *doRSampler* is being used to find $N = 2$ executions, 1 of which has already been found through a prior invocation ($PRT = 0.50$). The set $Next_R$ is shuffled, with the first $4 \times 0.50 = 2$ elements placed at the back of the set, forming: $Shuffled = [(\delta_3, s_3), (\delta_4, s_4), (\delta_1, s_1), (\delta_2, s_2)]$. The idea is to encourage the *doRSampler* interpreter to make transitions that are different from those made in previous invocations, and find a set of relaxed executions of a program that are spread across the complete execution set.

The *doRSampler* interpreter of Definition B.2.1 is used in my implementation of the SAMPLE function referred to in Listing 8.2.2.

Definition B.2.2 Let $sample(N, \delta, S, H, Solutions)$ be a predicate that finds a set $Solutions$ of (up to) N relaxed executions of program δ in situation S, given a finite action horizon H. If the program has N_R relaxed executions, then no more than N_R executions are found. The predicate *sample* operates by invoking the *doRSampler* interpreter. Pseudocode demonstrating how this predicate arrives at the set $Solutions$ is shown below.

$Solutions \leftarrow \emptyset$
for $Index = 0 \ldots N - 1$ **do**
    $PRT \leftarrow \frac{Index}{N}$
    if $\exists S'.\ doRSampler(PRT, \delta, S, S', H)$ **then**
        $Solutions \leftarrow Solutions \cup S'$
    end if
end for
$Solutions \leftarrow filterDuplicates(Solutions)$

The function *filterDuplicates*($Solutions$) filters the duplicates from the collection of executions in $Solutions$ to produce the result of the predicate *sample*.

While the predicate *sample* is designed to find a set of N distinct solutions – this implementation is simple and approximate, finding some duplicates amongst this number.

References

V. Aleven. Using background knowledge in case-based legal reasoning: A computational model and an intelligent learning environment. *Artificial Intelligence*, 150:183–237, 2003.

M. Allais. Le Comportement de l'Homme Rationnel devant le Risque: Critique des Postulats et Axiomes de l'École Américaine. *Econometrica*, 21(4):503 – 546, 1953.

J. Ambite, G. Barish, C. Knoblock, M. Muslea, J. Oh, and S. Minton. Getting from Here to There: Interactive Planning and Agent Execution for Optimising Travel. In *Innovative Applications of Artificial Intelligence (IAAI)*, pages 862–869, 2002.

L. Amgoud. A formal framework for handling conflicting desires. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 552–563, 2003.

L. Amgoud and C. Cayrol. On the Acceptability of Arguments in Preference-Based Argumentation. In *Uncertainty in Artificial Intelligence (UAI)*, pages 1–7, 1998.

L. Amgoud and C. Cayrol. A Reasoning Model Based on the Production of Acceptable Arguments. *Annals of Math and Artificial Intelligence*, 34(1–3):197–215, 2002a.

L. Amgoud and C. Cayrol. Inferring from Inconsistency in Preference-Based Argumentation Frameworks. *Journal of Automated Reasoning*, 29(2):125–169, 2002b.

L. Amgoud and N. Hameurlain. A formal model for designing dialogue strategies. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 414–416, 2006.

L. Amgoud and S. Kaci. On the Generation of Bipolar Goals in Argumentation-Based Negotiation. In *Argumentation in Multi-Agent Systems (ArgMAS)*, pages 192–207, 2004.

L. Amgoud and S. Kaci. An argumentation framework for merging conflicting knowledge bases. *International Journal of Approximate Reasoning*, 45(2):321–340, 2007.

L. Amgoud and N. Maudet. Strategical considerations for argumentative agents. In *Non-Monotonic Reasoning (NMR)*, pages 399–407, 2002.

L. Amgoud and H. Prade. Using arguments for making decisions: A possibilistic logic approach. In *Uncertainty in Artificial Intelligence (UAI)*, pages 10–17, 2004a.

L. Amgoud and H. Prade. Reaching agreement through argumentation: A possibilistic approach. In *Knowledge Representation and Reasoning (KR)*, pages 175–182, 2004b.

L. Amgoud and H. Prade. Generation and evaluation of different types of arguments in negotiation. In *Non-Monotonic Reasoning (NMR)*, pages 10–15, 2004c.

L. Amgoud and H. Prade. Explaining qualitative decision under uncertainty by argumentation. In *AAAI Conference on Artificial Intelligence*, pages 219–224, 2006.

L. Amgoud, S. Parsons, and L. Perrussel. An argumentation framework based on contextual preferences. In *Formal and Applied Practical Reasoning (FAPR)*, pages 59–67, 2000.

L. Amgoud, N. Maudet, and S. Parsons. An argumentation-based semantics for agent communication languages. In *European Conference on Artificial Intelligence (ECAI)*, pages 38–42, 2002.

L. Amgoud, S. Belabbès, and H. Prade. Towards a formal framework for the search of a consensus between autonomous agents. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 537–543, 2005a.

L. Amgoud, J. Bonnefon, and H. Prade. An Argumentation-based Approach for Multiple Criteria Decision. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 269–280, 2005b.

L. Amgoud, S. Belabbès, and H. Prade. A Formal General Setting for Dialogue Protocols. In *Artificial Intelligence: Methodology, Systems, Applications (AIMSA)*, pages 13–23, 2006.

L. Amgoud, Y. Dimopoulos, and P. Moraitis. A unified and general framework for argumentation-based negotiation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 963–970, 2007.

L. Amgoud, C. Devred, and M. Lagasquie-Schiex. A constrained argumentation system for practical reasoning. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 429–436, 2008a.

L. Amgoud, Y. Dimopoulos, and P. Moraitis. Making decisions through preference-based argumentation. In *Knowledge Representation and Reasoning (KR)*, pages 113–123, 2008b.

K. Arrow, A. Sen, and K. Suzumura, editors. *Handbook of Social Choice and Welfare*. North-Holland, 2002.

K. Ashley. *Modeling Legal Argument*. MIT Press, 1990.

K. Atkinson and T. Bench-Capon. Practical reasoning as presumptive argumentation using action-based alternating transition systems. *Artificial Intelligence*, 171(10–15):855–874, 2007a.

K. Atkinson and T. Bench-Capon. Action-based alternating transition systems for arguments about action. In *AAAI Conference on Artificial Intelligence*, pages 24–29, 2007b.

K. Atkinson and T. Bench-Capon. Addressing moral problems through practical reasoning. *Journal of Applied Logic*, 6(2):135–151, 2008.

K. Atkinson, T. Bench-Capon, and P. McBurney. Generating intentions through argumentation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1261–1262, 2005a.

K. Atkinson, T. Bench-Capon, and P. McBurney. Arguing about cases as practical reasoning. In *International Conference on Artificial Intelligence and Law (ICAIL)*, pages 35–44, 2005b.

K. Atkinson, T. Bench-Capon, and P. McBurney. A dialogue-game protocol for multi-agent argument over proposals for action. *Journal of Autonomous Agents and Multi-Agent Systems*, 11:153–171, 2005c.

K. Atkinson, T. Bench-Capon, and P. McBurney. Computational Representation of Practical Argument. *Synthese*, 152(2):157–206, 2006.

K. Atkinson, R. Girle, P. McBurney, and S. Parsons. Command Dialogues. In *Argumentation in Multi-Agent Systems (ArgMAS)*, pages 93–106, 2008.

J. Baier, F. Bacchus, and S. McIlraith. A Heuristic Search Approach to Planning with Temporally Extended Preferences. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1808–1815, 2007a.

J. Baier, C. Fritz, and S. McIlraith. Exploiting procedural domain control knowledge in state-of-the-art planners. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 26–33, 2007b.

J. Baier, C. Fritz, M. Bienvenu, and S. McIlraith. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *AAAI Conference on Artificial Intelligence*, pages 1509–1512, 2008.

J. Baier, F. Bacchus, and S. McIlraith. A Heuristic Search Approach to Planning with Temporally Extended Preferences. *Artificial Intelligence*, 173:593–618, 2009.

D. Beck and G. Lakemeyer. Reinforcement learning for GOLOG programs. In *KI Conference on Artificial Intelligence*, pages 64–78, 2009.

A. Belesiotis, M. Rovatsos, and I. Rahwan. Agreeing on Plans Through Iterated Disputes. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 765–772, 2010.

D. Bell, H. Raiffa, and A. Tversky, editors. *Decision making: descriptive, normative, and prescriptive interactions*. Cambridge University Press, 1988.

T. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13(3):429–448, 2003a.

T. Bench-Capon. Agreeing to differ: modelling persuasive dialogue between parties with different values. *Informal Logic*, 22:231–245, 2003b.

T. Bench-Capon and P. Dunne. Argumentation and dialogue in artificial intelligence. In *Tutorial T12 of International Joint Conference on Artificial Intelligence*, 2005.

T. Bench-Capon and H. Prakken. Justifying actions by accruing arguments. In *Computational Models of Argument (COMMA)*, pages 247–258, 2006.

T. Bench-Capon, K. Atkinson, and P. McBurney. Altruism and agents: an argumentation based approach to designing agent decision mechanisms. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1073–1080, 2009.

S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade. Inconsistency management and prioritised syntax-based entailment. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 640–647, 1993.

S. Benferhat, D. Dubois, S. Kaci, and H. Prade. Bipolar representation and fusion of preferences in the possibilistic logic framework. In *Knowledge Representation and Reasoning (KR)*, pages 421–423, 2002.

S. Benferhat, D. Dubois, S. Kaci, and H. Prade. Bipolar possibility theory in preference modeling: Representation, fusion and optimal solutions. *Information Fusion*, 7(1):135–150, 2006.

J. Benton, M. Do, and S. Kambhampati. Over-subscription planning with numeric goals. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1207–1213, 2005.

J. Benton, M. Do, and S. Kambhampati. Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence*, 173(5–6):562 – 592, 2009.

T. Berners-Lee and M. Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. Harper, 1999.

M. Bienvenu, C. Fritz, and S. McIlraith. Planning with qualitative temporal preferences. In *Knowledge Representation and Reasoning (KR)*, pages 134–144, 2006.

M. Bienvenu, C. Fritz, and S. McIlraith. Specifying and computing preferred plans. *Artificial Intelligence (to appear)*, 2010a.

M. Bienvenu, J. Lang, and N. Wilson. From preference logics to preference languages, and back. In *Knowledge Representation and Reasoning (KR)*, pages 414–424, 2010b.

L. Birnbaum. Argument Molecules: A Functional Representation of Argument Structure. In *AAAI Conference on Artificial Intelligence*, pages 63–65, 1982.

L. Birnbaum, M. Flowers, and R. McGuire. Towards an AI Model of Argumentation. In *AAAI Conference on Artificial Intelligence*, pages 313–315, 1980.

M. Blom. Optimising the Interpretation of GOLOG Programs with Argumentation. In *European Workshop on Multi-Agent Systems (EUMAS)*, pages 597–611, 2008.

M. Blom and A. Pearce. An Argumentation-Based Interpreter for GOLOG Programs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 690–695, 2009.

M. Blom and A. Pearce. Relaxing Regression for a Heuristic GOLOG. In T. Ågnotes, editor, *Proceedings of the Fifth Starting AI Researchers' Symposium (STAIRS 2010)*, pages 37–49. IOS Press, 2010.

A. Blum and M. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90(1–2):281–300, 1997.

G. Boella, D. Gabbay, L. van der Torre, and S. Villata. Meta-Argumentation Modelling I: Methodology and Techniques. *Studia Logica*, 93:297–355, 2009.

A. Bondarenko, P. Dung, R. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93(1-2):63–101, 1997.

B. Bonet and H. Geffner. Arguing for Decisions: A Qualitative Model of Decision Making. In *Uncertainty in Artificial Intelligence (UAI)*, pages 98–105, 1996.

B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33, 2001.

B. Borowsky and S. Edelkamp. Optimal metric planning with state sets in automata representation. In *AAAI Conference on Artificial Intelligence*, pages 874–879, 2008.

J. Bourguet, L. Amgoud, and R. Thomopoulos. Towards a Unified Model of Preference-Based Argumentation. In *Foundations of Information and Knowledge Systems (FOIKS)*, pages 326–344, 2010.

C. Boutilier, R. Brafman, H. Hoos, and D. Poole. Reasoning with Conditional Ceteris Paribus Preference Statements. In *Uncertainty in Artificial Intelligence (UAI)*, pages 71–80, 1999.

C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In *AAAI Conference on Artificial Intelligence*, pages 355–362, 2000.

C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. CP-nets: A Tool for Representing and Reasoning about Conditional Ceteris Paribus Preference Statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

S. Bouveret, U. Endriss, and J. Lang. Conditional Importance Networks: A Graphical Language for Representing Ordinal, Monotonic Preferences over Sets of Goods. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 67–72, 2009.

R. Brafman and Y. Chernyavsky. Planning with goal preferences and constraints. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 182–191, 2005.

R. Brafman and C. Domshlak. Introducing Variable Importance Tradeoffs into CP-nets. In *Uncertainty in Artificial Intelligence (UAI)*, pages 69–76, 2002.

F. Brandt, F. Fischer, and P. Harrenstein. The Computational Complexity of Choice Sets. In *Theoretical Aspects of Rationality and Knowledge (TARK)*, pages 82–91, 2007.

J. Brans, B. Vincke, and B. Mareschal. How to select and how to rank projects: The PROMETHEE method. *European Journal of Operational Research*, 24(2):228–238, 1986.

L. Branting. A reduction-graph model of precedent in legal analysis. *Artificial Intelligence*, 150(1-2):59–95, 2003.

G. Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1043–1048, 1989.

G. Brewka. Dynamic argument systems: A formal model of argumentation processes based on the situation calculus. *Journal of Logic and Computation*, 11(2):257–282, 2001.

G. Brewka. A Rank Based Description Language for Qualitative Preferences. In *European Conference on Artificial Intelligence (ECAI)*, pages 303–308, 2004.

G. Brewka and T. Gordon. How to Buy a Porsche: An Approach to Defeasible Decision Making. In *AAAI Workshop on Computational Dialectics*, pages 28–38, 1994.

D. Brickley and R. Guha. RDF vocabulary description language 1.0: RDF schema, 2004. URL `www.w3.org/TR/rdf-schema/`.

S. Buckingham Shum. Cohere: Towards Web 2.0 Argumentation. In *Computational Models of Argument (COMMA)*, pages 97–108, 2008.

W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *AAAI Conference on Artificial Intelligence*, pages 11–18, 1998.

D. Camacho, D. Borrajo, and J. Molina. Intelligent Travel Planning: A Multi-Agent Planning System to Solve Web Problems in the e-Tourism Domain. *Autonomous Agents and Multi-Agent Systems*, 4(4):387–392, 2001.

M. Caminada and P. Dunne. Semi-stable semantics. In *Computational Models of Argument (COMMA)*, pages 216–227, 2006.

D. Carbogim, D. Robertson, and J. Lee. Argument-based applications to knowledge engineering. *Knowledge Engineering Review*, 15(2):119–149, 2000.

C. Cayrol, S. Doutre, and J. Mengin. Dialectical proof theories for the credulous preferred semantics of argumentation frameworks. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 668–679, 2001.

H. Chalupsky, Y. Gil, C. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. Russ, and M. Tambe. Electric elves: Applying agent technology to support human organizations. *AI Magazine*, 23(2):1–24, 2002.

C. Chesñevar, A. Maguitman, and R. Loui. Logical models of argument. *ACM Computing Surveys*, 32:337–383, 2000.

C. Chesñevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *Knowledge Engineering Review*, 21:293–316, 2006.

J. Claßen, P. Eyerich, G. Lakemeyer, and B. Nebel. Towards an integration of GOLOG and planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1846–1851, 2007.

C. Cloos. The utilibot project: An autonomous mobile robot based on utilitarianism. *AAAI Fall Symposium on Machine Ethics*, 2005.

D. Cook and S. Das. How smart are our environments? An updated look at the state of the art. *Pervasive and Mobile Computing*, 3(2):53 – 73, 2007.

P. Dasgupta, P. Chakrabarti, and S. DeSarkar. Multi-Objective Heuristic Search of AND/OR Graphs. *Journal of Algorithms*, 20(2):282–311, 1996.

G. de Giacomo and H. Levesque. An Incremental Interpreter for High-Level Programs with Sensing. In *Logical Foundation for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 86–102. 1999.

G. de Giacomo, Y. Lespérance, and H. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

R. Dechter and J. Pearl. Generalised Best-First Strategies and the Optimality of A*. *Journal of ACM*, 32(3):505–536, 1985.

J. Delgrande, T. Schaub, and H. Tompits. A General Framework for Expressing Preferences in Causal Reasoning and Planning. *Journal of Logic and Computation*, 17(5): 871–907, 2007.

C. Devred and S. Doutre. Dialectical proof theories for the credulous prudent preferred semantics of argumentation. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 271–282, 2007.

Y. Dimopoulos, P. Moraitis, and L. Amgoud. Theoretical and computational properties of preference-based argumentation. In *European Conference on Artificial Intelligence (ECAI)*, pages 463–467, 2008.

Y. Dimopoulos, P. Moraitis, and L. Amgoud. Extending argumentation to make good decisions. In *Algorithmic Decision Theory (ADT)*, pages 225–236, 2009.

C. Domshlak. A snapshot on reasoning with qualitative preference statements in AI. In *Preferences and Similarities*, pages 265–282. 2008.

S. Doutre, P. McBurney, M. Wooldridge, and W. Barden. Information-seeking agent dialogs with permissions and arguments. Technical Report ULCS-05-010, Department of Computer Science, University of Liverpool, 2005.

S. Doutre, P. McBurney, L. Perrussel, and J. Thévenin. Arguing for gaining access to information. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 85–87, 2007.

J. Doyle. Truth maintenance for problem solving. Technical Report TR-419, MIT AI lab, January 1978.

J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):251–272, 1979.

J. Doyle and M. Wellman. Representing Preferences as Ceteris Paribus Comparatives. In *AAAI Spring Symposium on Decision Theoretic Planning*, pages 69–75, 1994.

D. Dubois, H. Fargier, and J. Bonnefon. On the qualitative comparison of decisions having positive and negative features. *Journal of Artificial Intelligence Research*, 32:385–417, 2008.

P. Dung. On the acceptability of arguments and its fundamental role in non-monotonic reasoning and logic programming. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 852–857, 1993a.

P. Dung. An argumentation semantics for logic programming with explicit negation. In *International Conference on Logic Programming (ICLP)*, pages 616–630, 1993b.

P. Dung. On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming, and N-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

P. Dung, R. Kowalski, and F. Toni. Dialectic proof theories for assumption-based admissible argumentation. *Artificial Intelligence*, 170(2):114–159, 2006.

P. Dung, P. Mancarella, and F. Toni. Computing Ideal Sceptical Argumentation. *Artificial Intelligence*, 171(10–15):642–674, 2007.

P. Dung, P. Thang, and F. Toni. Towards argumentation-based contract negotiation. In *Computational Models of Argument (COMMA)*, pages 134–146, 2008.

P. Dunne and T. Bench-Capon. Two party immediate response disputes: properties and efficiency. *Artificial Intelligence*, 149(2):221–250, 2003.

P. Dunne and T. Bench-Capon. Complexity in Value-Based Argumentation Systems. In *European Conference on Logics in Artificial Intelligence (JELIA)*, pages 360–371, 2004.

C. e Costa and J. Vansnick. MACBETH: an interactive path towards the construction of cardinal value functions. *International Transactions in Operational Research*, 1(4):489–500, 1994.

S. Edelkamp and P. Kissmann. Optimal symbolic planning with action costs and preferences. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1690–1695, 2009.

M. Elvang-Gøransson, P. Krause, and J. Fox. Dialectic reasoning with inconsistent information. In *Uncertainty in Artificial Intelligence (UAI)*, pages 114–121, 1993a.

M. Elvang-Gøransson, P. Krause, and J. Fox. Acceptability of arguments as 'logical uncertainty'. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 85–90, 1993b.

P. Eyerich, B. Nebel, G. Lakemeyer, and J. Claßen. GOLOG and PDDL: What is the relative expressiveness? In *International Symposium on Practical Cognitive Agents and Robots (PCAR)*, pages 93–104, 2006.

E. Fabre, L. Jezequel, P. Haslum, and S. Thiébaux. Cost-optimal factored planning: Promises and pitfalls. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 65–72, 2010.

P. Farquhar. Utility Assessment Methods. *Management Science*, 30(11):1283–1300, 1984.

A. Ferrein, C. Fritz, and G. Lakemeyer. Using GOLOG for deliberation and team coordination in robotic soccer. In *KI Conference on Artificial Intelligence*, pages 24–31, 2005.

E. Ferretti, M. Errecalde, A. García, and G. Simari. Decision Rules and Arguments in Defeasible Decision Making. In *Computational Models of Argument (COMMA)*, pages 171–182, 2008.

J. Figueira, S. Greco, and M. Ehrgott, editors. *Multiple criteria decision analysis: state of the art surveys*. Springer, 2005.

R. Fikes and N. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 5(2):189–208, 1971.

A. Finzi and T. Lukasiewicz. Game-Theoretic Agent Programming in GOLOG. In *European Conference on Artificial Intelligence (ECAI)*, pages 23–27, 2004.

A. Finzi and T. Lukasiewicz. Game theoretic GOLOG under partial observability. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1301–1302, 2005.

P. Fishburn. Non-transitive measurable utility. *Journal of Mathematical Psychology*, 26(1): 31–67, 1982.

M. Flowers. On Being Contradictory. In *AAAI Conference on Artificial Intelligence*, pages 269–272, 1982.

M. Flowers, R. McGuire, and L. Birnbaum. Adversary arguments and the logic of personal attacks. In W. Lehnert and M. Ringle, editors, *Strategies for Natural Language Processing*. Lawrence Erlbaum, NJ, 1982.

J. Fox and S. Parsons. On using arguments for reasoning about actions and values. In *AAAI Spring Symposium on Qualitative Preferences in Deliberation and Practical Reasoning*, pages 55–63, 1997.

J. Fox, A. Glowinski, and M. O'Neil. The Oxford System of Medicine: A Prototype Information System for Primary Care. In *European Conference on Artificial Intelligence in Medicine (AIME)*, pages 58–60, 1987.

J. Fox, A. Glowinski, C. Gordon, S. Hajnal, and M. O'Neil. Logic engineering for knowledge engineering: design and implementation of the Oxford System of Medicine. *Artificial Intelligence in Medicine*, 2:323–339, 1990.

J. Fox, P. Krause, and S. Ambler. Arguments, Contradictions and Practical Reasoning. In *European Conference on Artificial Intelligence (ECAI)*, pages 623–627, 1992.

J. Fox, N. Johns, C. Lyons, A. Rahmanzadeh, R. Thomson, and P. Wilson. PROforma: a general technology for clinical decision support systems. *Computer Methods and Programs in Biomedicine*, 54:59–67, 1997.

M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

C. Fritz and S. McIlraith. Compiling Qualitative Preferences into Decision-Theoretic Golog Programs. In *Non-Monotonic Reasoning (NMR)*, 2005.

C. Fritz, J. Baier, and S. McIlraith. ConGolog, Sin Trans: Compiling ConGolog into basic action theories for planning and beyond. In *Knowledge Representation and Reasoning (KR)*, pages 600–610, 2008.

D. Gabbay. Semantics for Higher Level Attacks in Extended Argumentation Frames Part I: Overview. *Studia Logica*, 93(2-3):357–381, 2009.

L. Galand and P. Perny. Search for Compromise Solutions in Multi-Objective State Space Graphs. In *European Conference on Artificial Intelligence (ECAI)*, pages 93–97, 2006.

A. García and G. Simari. Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming*, 4(2):95–138, 2004.

D. García, A. García, and G. Simari. Planning and Defeasible Reasoning. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1–3, 2007.

D. García, A. García, and G. Simari. Defeasible Reasoning and Partial Order Planning. In *Foundations of Information and Knowledge Systems (FOIKS)*, pages 311–328, 2008.

A. Gardner. Search: An overview. *AI Magazine*, 2(1):2–6, 1980.

M. Gavenelli and M. Pini. FCP-nets: extending constrained CP-nets with objective functions. In *Constraint Solving and Constraint Logic Programming (ERCIM)*, 2008.

A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. Technical Report 2005-08-07, University of Brescia, Italy, 2005.

M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, Amsterdam, 2004.

G. Gigerenzer. *Rationality for Mortals*. Oxford University Press, 2008.

T. Gordon and N. Karacapilidis. The Zeno Argumentation Framework. In *International Conference on Artificial Intelligence and Law (ICAIL)*, pages 10–18, 1997.

M. Grabisch, S. Greco, and M. Pirlot. Bipolar and bivariate models in multi-criteria decision analysis: descriptive and constructive approaches. *Journal of Intelligent Systems*, 23 (9):930–969, 2008.

S. Greco, B. Matarazzo, and R. Slowinski. Rule-Based Decision Support in Multi Criteria Choice and Ranking. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 29–47, 2001.

J. Hage. Teleological reasoning in reason-based logic. In *International Conference on Artificial Intelligence and Law (ICAIL)*, pages 11–20, 1995.

J. Hage. Dialectical models in artificial intelligence and law. *Artificial Intelligence and Law*, 8(2–3):137–172, 2000.

S. Hansson. What is ceteris paribus preference? *Journal of Philosophical Logic*, 25(3):307–332, 1996.

S. Hansson. Preference Logic. *Handbook of Philosophical Logic*, 4:319–393, 2001.

B. Hardy-Vallèe. Decision-making in robotics and psychology: A distributed account. *New Ideas in Psychology*, 2009. doi: 10.1016/j.newideapsych.2009.07.009.

P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics*, 4(2):100–107, 1968.

J. Hoffman and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

J. Hulstijn and L. van der Torre. Combining goal generation and planning in an argumentation framework. In *Non-Monotonic Reasoning (NMR)*, pages 212–218, 2004.

H. Jakobovits and D. Vermeir. Dialectic semantics for argumentation frameworks. In *International Conference on Artificial Intelligence and Law (ICAIL)*, pages 53–62, 1999.

U. Junker. Preference-based search for scheduling. In *AAAI Conference on Artificial Intelligence*, pages 904–909, 2000.

U. Junker. Preference-based search and multi-criteria optimization. In *AAAI Conference on Artificial Intelligence*, pages 34–40, 2002.

S. Kaci and L. van der Torre. Preference-based argumentation: Arguments supporting multiple values. *International Journal of Approximate Reasoning*, 48(3):730 – 751, 2008.

S. Kaci, L. van der Torre, and E. Weydert. Acyclic argumentation: Attack = conflict + preference. In *Prestigious Applications of Intelligent Systems (PAIS)*, pages 725–726, 2006.

D. Kahneman and A. Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47:263–291, 1979.

A. Kakas and P. Moraitis. Argumentation-based decision making for autonomous agents. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 883–890, 2003.

A. Kakas and P. Moraitis. Adaptive agent negotiation via argumentation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 384–391, 2006.

A. Kakas, N. Maudet, and P. Moraitis. Flexible agent dialogue strategies and societal communication protocols. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1434–1435, 2004.

E. Karpas and C. Domshlak. Cost-optimal planning with landmarks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1728–1733, 2009.

N. Karunatillake, N. Jennings, I. Rahwan, and T. Norman. Argument-based Negotiation in a Social Context. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1331–1332, 2005.

N. Karunatillake, N. Jennings, I. Rahwan, and P. McBurney. Dialogue Games that Agents Play within a Society. *Artificial Intelligence*, 173(9-10):935–981, 2009.

R. Keeney. Decision Analysis: An Overview. *Operations Research*, 30(5):803–838, 1982.

R. Keeney and H. Raiffa, editors. *Decisions with Multiple Objectives: Preferences and Values Tradeoffs*. Wiley, New York, 1976.

E. Keyder and H. Geffner. Heuristics for planning with action costs revisited. In *European Conference on Artificial Intelligence (ECAI)*, pages 588–592, 2008.

E. Keyder and H. Geffner. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36(1):547–556, 2009.

H. Kim, W. Kim, and M. Lee. Semantic web constraint language and its application to an intelligent shopping agent. *Decision Support Systems*, 46(4):882 – 894, 2009.

A. Kishimoto, A. Fukunaga, and A. Botea. Scalable, Parallel Best-First Search for Optimal Sequential Planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 201–208, 2009.

C. Knoblock. Building Software Agents for Planning, Monitoring, and Optimizing Travel. In *Information and Communication Technologies in Tourism*, pages 1–15, 2004.

S. Koenig and M. Likhachev. D* Lite. In *AAAI Conference on Artificial Intelligence*, pages 476–483, 2002.

S. Koenig and M. Likhachev. Lifelong Planning A*. *Artificial Intelligence*, 155(1-2):93–146, 2004.

R. Korf. Iterative-Deepening-A*: an optimal admissible tree search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1034–1036, 1985.

R. Korf. Real-Time Heuristic Search: First results. In *AAAI Conference on Artificial Intelligence*, pages 133–138, 1987.

R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

S. Kraus, M. Nirkhe, and K. Sycara. Reaching Agreements Through Argumentation: A Logical Model. In *Workshop on Distributed Artificial Intelligence (DAI)*, pages 233–247, 1993.

S. Kraus, K. Sycara, and A. Evenchik. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104(1-2):1–69, 1998.

P. Krause, S. Ambler, M. Elvang-Goransson, and J. Fox. A logic of argumentation for reasoning under uncertainty. *Computational Intelligence*, 11:113–131, 1995a.

P. Krause, J. Fox, and P. Judson. Is There a Role for Qualitative Risk Assessment? In *Uncertainty in Artificial Intelligence (UAI)*, pages 386–439, 1995b.

A. Leslie. Pretense and representation: The origins of 'Theory of Mind'. *Psychological Review*, 94(4):412–426, 1987.

Y. Lespérance, H. Levesque, and S. Ruman. An experiment in using GOLOG to build a personal banking assistant. In *Workshop on Intelligent Agent Systems, Theoretical and Practical Issues at PRICAI*, pages 27–43, 1997.

Y. Lespérance, K. Tam, and M. Jenkin. Reactivity in a logic-based robot programming framework. In *Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages (ATAL)*, pages 173–187, 1999.

H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–83, 1997.

H. Levesque, F. Pirri, and R. Reiter. Foundations for the Situation Calculus. *Electronic Transactions on Artificial Intelligence*, 2(3–4):159–178, 1998.

V. Lifschitz. On the semantics of STRIPS. In *Reasoning about Actions and Plans*, pages 1–9. 1986.

F. Lin and Y. Shoham. Argument Systems: A Uniform Basis for Non-monotonic Reasoning. In *Knowledge Representation and Reasoning (KR)*, pages 245–255, 1989.

R. Loui. Defeat among arguments: a system of defeasible inference. *Computational Intelligence*, 2(1):100–106, 1987.

M. Lucero, C. Chesñevar, and G. Simari. On the Accrual of Arguments in Defeasible Logic Programming. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 804–809, 2009a.

M. Lucero, C. Chesñevar, and G. Simari. Modelling argument accrual in possibilistic defeasible logic programming. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 131–143, 2009b.

L. Mandow and J. de la Cruz. A heuristic search algorithm with lexicographic goals. *Engineering Applications of Artificial Intelligence*, 14(6):751–762, 2001.

L. Mandow and J. de la Cruz. A New Approach to Multi-Objective A* Search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 218–223, 2005.

L. Mandow and J. de la Cruz. A Multi-Objective Frontier Search Algorithm. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2340–2345, 2007.

L. Mandow and J. de la Cruz. A Memory-Efficient Search Strategy for Multi-Objective Shortest Path Problems. In *KI Conference on Artificial Intelligence*, pages 25–32, 2009.

L. Mandow and J. de la Cruz. Multi-objective A* search with consistent heuristics. *Journal of the ACM*, 57(5):1–25, 2010.

P. Matt and F. Toni. A Game-Theoretic Measure of Argument Strength for Abstract Argumentation. In *European Conference on Logics in Artificial Intelligence (JELIA)*, pages 285–297, 2008.

P. McBurney and S. Parsons. Risk agoras: Dialectical argumentation for scientific reasoning. In *Uncertainty in Artificial Intelligence (UAI)*, pages 371–379, 2000.

P. McBurney and S. Parsons. Representing epistemic uncertainty by means of dialectical argumentation. *Annals of Mathematics and Artificial Intelligence*, 32(1–4):125–169, 2001a.

P. McBurney and S. Parsons. Strawmen and eidolons: using argumentation to reason across scenarios. Technical Report ULCS-01-008, Department of Computer Science, University of Liverpool, 2001b.

P. McBurney and S. Parsons. Dialectical argumentation for reasoning about chemical carcinogenicity. *Logic Journal of the IGPL*, 9(2):191–203, 2001c.

P McBurney and S. Parsons. Games That Agents Play: A Formal Framework for Dialogues between Autonomous Agents. *Journal of Logic, Language and Information*, 11(3): 315–334, 2002.

P. McBurney and S. Parsons. Chance Discovery and Scenario Analysis. *New Generation Computing*, 21(1):13–22, 2003.

P. McBurney and S. Parsons. Locutions for argumentation in agent interaction protocols. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1240–1241, 2004.

P. McBurney, S. Parsons, and M. Wooldridge. Desiderata for agent argumentation protocols. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 402–409, 2002.

P. McBurney, R. Van Euk, S. Parsons, and L. Amgoud. A Dialogue Game Protocol for Agent Purchase Negotiations. *Autonomous Agents and Multi-Agent Systems*, 7(3):235–273, 2003.

J. McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

D. McGuinness and F. van Harmelen. OWL web ontology language overview. `www.w3.org/TR/owl-features/`, June 2010.

R. McGuire, L. Birnbaum, and M. Flowers. Opportunistic Processing in Arguments. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 58–60, 1981.

S. McIlraith and R. Fadel. Planning with complex actions. In *Non-Monotonic Reasoning (NMR)*, pages 356–364, 2002.

S. McIlraith and T. Son. Adapting GOLOG for Composition of Semantic Web Services. In *Knowledge Representation and Reasoning (KR)*, pages 482–496, 2002.

F. Menczer, W. Street, N. Vishwakarma, A. Monge, and M. Jakobsson. IntelliShopper: a proactive, personal, private shopping assistant. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1001–1008, 2002.

S. Modgil. Value-based argumentation in hierarchical argumentation frameworks. In *Computational Models of Argument (COMMA)*, pages 297–308, 2006a.

S. Modgil. Hierarchical Argumentation. In *European Conference on Logics in Artificial Intelligence (JELIA)*, pages 319–332, 2006b.

S. Modgil. An abstract theory of argumentation that accommodates defeasible reasoning about preferences. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 648–659, 2007.

S. Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173(9–10):901–934, 2009a.

S. Modgil. Labellings and games for extended argumentation frameworks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 873–878, 2009b.

S. Modgil and T. Bench-Capon. Integrating Object and Meta-Level Value Based Argumentation. In *Computational Models of Argument (COMMA)*, pages 240–251, 2008.

S. Modgil and T. Bench-Capon. Meta-level argumentation. *Journal of Logic and Computation*, 2010. doi: 10.1093/logcom/exq054.

R. Moore. Possible-world semantics for autoepistemic logic. In *AAAI Workshop on Non-Monotonic Reasoning*, pages 344–354, 1984.

M. Morge and K. Stathis. The agent architecture revisited. In *European Workshop on Multi-Agent Systems (EUMAS)*, pages 581–595, 2008.

A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-Angle Path Planning on Grids. In *AAAI Conference on Artificial Intelligence*, pages 1177–1183, 2007.

J. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36:48–49, 1950.

F. Nawab, T. Bench-Capon, and P. Dunne. A methodology for action-selection using value-based argumentation. In *Computational Models of Argument (COMMA)*, pages 264–275, 2008.

T. Nguyen, M. Do, S. Kambhampati, and B. Srivastava. Planning with partial preference models. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1772–1777, 2009.

N. Oren, T. Norman, and A. Preece. Subjective logic and arguing with evidence. *Artificial Intelligence*, 171(10–15):838–854, 2007a.

N. Oren, T. Norman, and A. Preece. Argumentation-based contract monitoring in uncertain domains. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1434–1439, 2007b.

W. Ouerdane, N. Maudet, and A. Tsoukiàs. Argument Schemes and Critical Questions for the Decision Aiding Process. In *Computational Models of Argument (COMMA)*, pages 285–296, 2008.

S. Parsons. A proof theoretic approach to qualitative probabilistic reasoning. *International Journal of Approximate Reasoning*, 19(3–4):265–297, 1998.

S. Parsons and S. Green. Argumentation and qualitative decision making. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 328–340, 1999.

S. Parsons and N. Jennings. Negotiation through argumentation – a preliminary report. In *International Conference on Multi-Agent Systems (ICMAS)*, pages 267–274, 1996.

S. Parsons and P. McBurney. Argumentation-based communication between agents. In *Communication in Multi-Agent Systems*, pages 164–178, 2003.

S. Parsons, C. Sierra, and N. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

S. Parsons, P. McBurney, E. Sklar, and M. Wooldridge. On the relevance of utterances in formal inter-agent dialogues. In *Argumentation and Multi-Agent Systems (ArgMAS)*, pages 47–62, 2007.

J. Pearl. *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

E. Pednault. ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus. In *Knowledge Representation and Reasoning (KR)*, pages 324–332, 1989.

E. Pednault. ADL and the State-Transition Model of Action. *Journal of Logic and Computation*, 4(5):467–512, 1994.

B. Peintner, P. Viappiani, and N. Yorke-Smith. Preferences in Interactive Systems: Technical Challenges and Case Studies. *AI Magazine*, 29(4):13–24, 2008.

C. Perelman. *Justice, Law and Argument*. Reidel, 1980.

P. Perny and O. Spanjaard. On preference-based search in state space graphs. In *AAAI Conference on Artificial Intelligence*, pages 751–756, 2002.

P. Perny and O. Spanjaard. Near admissible algorithms for multiobjective search. In *European Conference on Artificial Intelligence (ECAI)*, pages 490–494, 2008.

W. Pickard-Cambridge. The internet classics archive – topics by aristotle. `http://classics.mit.edu/Aristotle/topics.3.iii.html`, June 2010.

F. Pirri and R. Reiter. Some contributions to the meta-theory of the situation calculus. *Journal of the ACM*, 46(3):325–361, 1999.

J. Pollock. *Knowledge and Justification*. Princeton University Press, 1974.

J. Pollock. Defeasible reasoning. *Cognitive Science*, 11(4):481–518, 1987.

J. Pollock. OSCAR. In *Conference on Automated Deduction (CADE)*, pages 669–670, 1990.

J. Pollock. How to reason defeasibly. *Artificial Intelligence*, 57(1):1–42, 1992.

J. Pollock. Justification and defeat. *Artificial Intelligence*, 67(2):377–407, 1994.

J. Pollock. *Cognitive Carpentry: A Blueprint for how to Build a Person*. MIT Press, Cambridge, MA, 1995.

J. Pollock. Defeasible reasoning with variable degrees of justification. *Artificial Intelligence*, 133(1–2):233–282, 2001.

D. Poole. On the Comparison of Theories: Preferring the Most Specific Explanation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 144–147, 1985.

D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.

E. Poulton. *Behavioral decision theory: A new approach*. Cambridge University Press, 1994.

H. Prakken. Dialectical Proof Theory for Defeasible Argumentation with Defeasible Priorities. In *ESPRIT Workshop on Formal Models of Agents*, pages 202–215, 1999.

H. Prakken. A study of accrual of arguments, with applications to evidential reasoning. In *International Conference on Artificial Intelligence and Law (ICAIL)*, pages 85–94, 2005a.

H. Prakken. AI & Law, Logic and Argument Schemes. *Argumentation*, 19(3):303–320, 2005b.

H. Prakken and G. Sartor. A system for defeasible argumentation, with defeasible priorities. In *International Conference on Formal and Applied Practical Reasoning (FAPR)*, pages 510–524, 1996a.

H. Prakken and G. Sartor. A system for defeasible argumentation, with defeasible priorities. In *International Conference on Formal and Applied Practical Reasoning (FAPR)*, pages 510–524, 1996b.

H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 7, 1997.

H. Prakken and G. Sartor. Modelling reasoning with precedents in a formal dialogue game. *Artificial Intelligence and Law*, 6(2–4):231–287, 1998.

H. Prakken and G. Sartor. The Role of Logic in Computational Models of Legal Argument: A Critical Survey. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, pages 342–381, 2002.

A. Procaccia and J. Rosenchein. Extensive-form argumentation games. In *European Workshop on Multi-Agent Systems (EUMAS)*, pages 312–322, 2005.

I. Rahwan. Guest editorial: Argumentation in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 11(2):115–125, 2005.

I. Rahwan. Mass argumentation and the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):29–37, 2008.

I. Rahwan and L. Amgoud. An argumentation based approach for practical reasoning. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 347–354, 2006.

I. Rahwan and B. Banihashemi. Arguments in OWL: A Progress Report. In *Computational Models of Argument (COMMA)*, pages 297–310, 2008.

I. Rahwan and K. Larson. Mechanism design for abstract argumentation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1031–1038, 2008a.

I. Rahwan and K. Larson. Pareto optimality in abstract argumentation. In *AAAI Conference on Artificial intelligence*, pages 150–155, 2008b.

I. Rahwan and P. Sakeer. Towards representing and querying arguments on the semantic web. In *Computational Models of Argument (COMMA)*, pages 3–14, 2006.

I. Rahwan and F. Tohmé. Collective argument evaluation as judgement aggregation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 417–424, 2010.

I. Rahwan, L. Sonenberg, and F. Dignum. Towards Interest-Based Negotiation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 773 – 780, 2003.

I. Rahwan, L. Sonenberg, and P. McBurney. Bargaining and Argument-Based Negotiation: Some Preliminary Comparisons. In *Argumentation in Multi-Agent Systems (ArgMAS)*, pages 176–191, 2004.

I. Rahwan, P. Pasquier, L. Sonenberg, and F. Dignum. On the benefits of exploiting underlying goals in argument-based negotiation. In *AAAI Conference on Artificial Intelligence*, pages 116–121, 2007a.

I. Rahwan, F. Zablith, and C. Reed. Toward large scale argumentation support on the semantic web. In *AAAI Conference on Artificial Intelligence*, pages 1446–1451, 2007b.

I. Rahwan, K. Larson, and F. Tohmé. A characterisation of strategy-proofness for grounded argumentation semantics. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 251–256, 2009.

A. Rao and M. Georgeff. Modelling rational agents within a BDI-Architecture. In *Knowledge Representation and Reasoning (KR)*, pages 473–484, 1991.

C. Reed, S. Wells, J. Devereux, and G. Rowe. AIF+: Dialogue in the Argument Interchange Format. In *Computational Models of Argument (COMMA)*, pages 311–323, 2008.

R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1–2):81–132, 1980.

R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380, 1991.

R. Reiter. *Knowledge in Action*. MIT Press, 2001.

J. Renaud-Salis and P. Taylor. The Bordeaux Oncology Support System: Knowledge Representation and Prototype. Technical Report Project LEMMA, Imperial Research Cancer Fund, February 1990.

N. Rescher. *Dialectics: A controversy-oriented approach to the theory of knowledge*. State University of New York Press, Albany, 1977.

H. Rittel and M. Webber. Dilemmas in a general theory of planning. *Policy Sciences*, 4(2): 155–169, 1973.

R. Riveret, H. Prakken, A. Rotolo, and G. Sartor. Heuristics in Argumentation: A Game-Theoretical Investigation. In *Computational Models of Argument (COMMA)*, pages 324–335, 2008.

G. Röger and B. Nebel. Expressiveness of ADL and GOLOG: Functions make a difference. In *AAAI Conference on Artificial Intelligence*, pages 1051–1056, 2007.

G. Röger, M. Helmert, and B. Nebel. On the Relative Expressiveness of ADL and GOLOG: The Last Piece in the Puzzle. In *Knowledge Representation and Reasoning (KR)*, pages 544–550, 2008.

N. Rotstein, A. García, and G. Simari. Reasoning from desires to intentions: A dialectical framework. In *AAAI Conference on Artificial Intelligence*, pages 136–141, 2007.

B. Roy. The outranking approach and the foundations of ELECTRE methods. *Theory and Decision*, 31:49–73, 1991.

B. Roy and D. Vanderpooten. The European School of MCDA: Emergence, Basic Features and Current Works. *Journal of Multi-Criteria Decision Analysis*, 5:22–38, 1996.

T. Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. McGraw-Hill, New York, 1980.

T. Saaty. How to make a decision: The Analytic Hierarchy Process. *European Journal of Operational Research*, 48:9–26, 1990.

S. Sardiña and S. Shapiro. Rational action in agent programs with prioritized goals. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 417–424, 2003.

L. Savage. *The foundations of statistics*. John Wiley & Sons, 1954.

T. Schwartz. Rationality and the myth of the maximum. *Noûs*, 6:97–117, 1972.

A. Sen. Rationality and Social Choice. *The American Economic Review*, 85:1–24, 1995.

E. Shafir, I. Simonson, and A. Tversky. Reason-based choice. *Cognition*, 49:11–36, 1993.

C. Sierra and P. Noriega. Agent-Mediated Interaction: From Auctions to Negotiation and Argumentation. In *Foundations and Applications of Multi-Agent Systems*, pages 27–48, 2002.

G. Simari and R. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53(2-3):125–157, 1992.

G. Simari, C. Chesñevar, A. García, B. Pierce, and D. Turner. The Role of Dialectics in Defeasible Argumentation. 1994.

G. Simari, A. García, and M. Capobianco. Actions, planning and defeasible reasoning. In *Non-Monotonic Reasoning (NMR)*, pages 377–384, 2004.

H. Simon. *The sciences of the artificial*. MIT Press, 1969.

D. Smith. Choosing objectives in over-subscription planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 393–401, 2004.

D. Smith, J. Frank, and A. Jónsson. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.

M. Smith and R. Dimitrakopoulos. The influence of deposit uncertainty on mine production scheduling. *International Journal of Surface Mining, Reclamation and the Environment*, 13:173–178, 1999.

S. Sohrabi, N. Prokoshyna, and S. McIlraith. Web service composition via generic procedures and customizing user preferences. In *International Semantic Web Conference (ISWC)*, pages 597–611, 2006.

S. Sohrabi, J. Baier, and S. McIlraith. Diagnosis as Planning Revisted. In *Knowledge Representation and Reasoning (KR)*, pages 26–36, 2010.

T. Son and E. Pontelli. Planning with preferences using logic programming. In *International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR)*, pages 247–260, 2004.

C. Starmer. Explaining risky choices without assuming preferences. *Social Choice and Welfare*, 13(2):201–213, 1996.

A. Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation*, pages 3310–3317, 1994.

A. Stentz. The Focused D* Algorithm for Real-Time Replanning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1652–1659, 1995.

B. Stewart and C. White. Multi-Objective A*. *Journal of the ACM*, 38(4):775–814, 1991.

K. Sycara. Argumentation: Planning Other Agents' Plans. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 517–523, 1989.

K. Tam, J. Lloyd, Y. Lespérance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and M. Jenkin. Controlling autonomous robots with GOLOG. In *Australian Joint Conference on Artificial Intelligence (AI)*, pages 1–12, 1997.

M. Tambe and H. Jung. The Benefits of Arguing in a Team. *AI Magazine*, 20:85–92, 1999.

M. Tambe, E. Bowring, J. Pearce, P. Varakantham, P. Scerri, and D. Pynadath. Electric Elves: What Went Wrong and Why. *AI Magazine*, 29(2):23–31, 2008.

P. Thang, P. Dung, and N. Hung. Towards a common framework for dialectical proof procedures in abstract argumentation. *Journal of Logic and Computation*, 19(6):1071–1109, 2009.

P. Tolchinsky, K. Atkinson, P. McBurney, S. Modgil, and U. Cortés. Agents deliberating over action proposals using the *proclaim* model. In *International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS)*, pages 32–41, 2007.

S. Toulmin. *The Uses of Argument*. Cambridge University Press, 1958.

S. Toulmin, R. Reike, and A. Janik. *An Introduction to Reasoning*. Macmillan, New York, 1979.

A. Tsoukiàs. From decision theory to decision aiding methodology. *European Journal of Operational Research*, 187(1):138–161, 2008.

A. Tsoukiàs, P. Perny, and P. Vincke. On the generalisation of the concordance discordance principle in multiple criteria decision analysis. *Aiding Decision with Multiple Criteria*, pages 147–174, 2002.

A. Tversky. Intransitivity of Preferences. *Psychological Review*, 76:31–48, 1969.

A. Tversky and D. Kahneman. The Framing of Decisions and the Psychology of Choice. *Science*, 211:453–458, 1981.

J. van der Wal. Stochastic Dynamic Programming. *Mathematical Centre Tracts*, 139, 1981.

B. Verheij. Accrual of arguments in defeasible argumentation. In *Dutch/German Workshop on Non-Monotonic Reasoning*, pages 217–224, 1995.

B. Verheij. *Rules, reasons, arguments: formal studies of argumentation and defeat*. Dissertation, University of Tübingen, 1996a.

B. Verheij. Two approaches to dialectical argumentation: Admissible sets and argumentation stages. *International Conference on Formal and Applied Practical Reasoning (FAPR)*, pages 357–368, 1996b.

B. Verheij, J. Hage, and H. van der Herik. An integrated view on rules and principles. *Artificial Intelligence and Law*, 6(1):3–26, 1998.

V. Vidal. A Lookahead Strategy for Heuristic Search Planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 150–159, 2004.

J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behaviour*. Princeton University Press, 1944.

D. von Winterfeldt and W. Edwards. *Decision Analysis and Behavioural Research*. Cambridge University Press, 1986.

G. Vreeswijk. Abstract argumentation systems. *Artificial Intelligence*, 90:225–279, 1997.

P. Wakker and D. Deneffe. Eliciting von Neumann-Morgenstern utilities when probabilities are distorted or unknown. *Management Science*, 42(8):1131–1150, 1996.

D. Walton. *Argumentation Schemes for Presumptive Reasoning*. Lawrence Erlbaum Associates, Mahwah, NJ, 1996.

D. Walton. *The New Dialectic: Conversational Contexts of Argument*. University of Toronto Press, 1998.

D. Walton and E. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. University of New York Press, 1995.

R. Walton, C. Gierl, P. Yudkin, H. Mistry, M. Vessey, and J. Fox. Evaluation of Computer Support for Prescribing (CAPSULE) Using Simulated Cases. *British Medical Journal*, 315:791–795, 1997.

M. Wellman and J. Doyle. Preferential semantics for goals. In *AAAI Conference on Artificial Intelligence*, pages 698–703, 1991.

M. Wellman, A. Greenwald, P. Stone, and P. Wurman. The 2001 trading agent competition. In *Innovative Applications of Artificial Intelligence (IAAI)*, pages 935–941, 2002.

M. Witkowski and K. Stathis. A dialectic architecture for computational autonomy. *Agents and Computational Autonomy*, pages 261–273, 2004.

M. Wooldridge, P. McBurney, and S. Parsons. On the meta-logic of arguments. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 560–567, 2005.

A. Wyner, T. Bench-Capon, and K. Atkinson. Arguments, Values and Baseballs: Representation of Popov v. Hayashi. In *Legal Knowledge and Information Systems*, pages 151–160, 2007.

H. Yim, H. Ahn, J. Kim, and S. Park. Agent-based adaptive travel planning system in peak seasons. *Expert Systems with Applications*, 27(2):211 – 222, 2004.

S. Yoon, A. Fern, and R. Givan. Learning heuristic functions from relaxed plans. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 162–171, 2006.

G. Youngblood, E. Heierman III, L. Holder, and D. Cook. Automation Intelligence for the Smart Environment. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1513–1514, 2005.