ADAPTIVE AND DYNAMIC ARGUMENTATION

Evgenios Hadjisoteriou

University of Cyprus, 2016

People use argumentation to make decisions in daily life, such as which product to buy, which film to watch, or which hotel to stay in. In this thesis we study adaptive and dynamic argumentation for decision making where decisions need to adapt to personal preferences and dynamically changing problem environment. We study how logic-based reasoning about actions and change (RAC) with its problems of temporal projection and qualification can be formalized in terms of argumentation. In particular, the earlier work of translating the language $\mathcal{E}$ for RAC into a logic-based argumentation framework (AF) is extended by introducing new types of arguments for (i) backward persistence and (ii) persistence from observations. Our framework is always consistent and it maintains a representation of the world for any time point and time period. This framework enables adaptation over time and allows dynamic changes in the problem environment to be handled over time.

We illustrate this by formalizing the details of the decision problem of choosing a hotel to stay at according to users' preferences and purposes of going on a trip. A first implementation of the decision problem 'Hotel for ME' using the $Gorgias$ argumentation system is developed. This is used to study the approaches' adaptability and flexibility of recommendations under a dynamic environment.

A new way to formalizing and computing argumentation through matrices is also studied. Abstract AF is interpreted in terms of matrix multiplication and we present matrix operation algorithms that can answer whether a given set of arguments is part of an argumentation extension

under the various semantics of AF. This has been implemented in a program called ASSA that finds stable extensions and ASSA$_G$ that finds the grounded extension of any AF and allows the user to add or remove arguments or attacks while the application is running. The properties of dynamic argumentation are examined through empirical experiments using these systems.

ADAPTIVE AND DYNAMIC ARGUMENTATION

Evgenios Hadjisoteriou

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

May, 2016

APPROVAL PAGE

Doctor of Philosophy Dissertation

ADAPTIVE AND DYNAMIC ARGUMENTATION

Presented by

Evgenios Hadjisoteriou

Research Supervisor _____
Antonis C. Kakas

Committee Member _____
Constantinos Pattichis

Committee Member _____
Nikos Karacapilidis

Committee Member _____
Pavlos Moraitis

Committee Member _____
Yannis Dimopoulos

University of Cyprus

May, 2016

"If a man empties his purse into his head no man can take it from him. An investment in knowledge pays the best interest."

Benjamin Franklin

ACKNOWLEDGEMENTS

I would like to thank the following people who have helped me complete this thesis:

- **My Supervisor: Antonis C. Kakas**

  For encouraging me through the entire process and calling attention to all of the interesting tangents. Just like a pro athlete must have a training regiment, Dr. Kakas' ideas and self-assignments were an inspiration.

- **My Family: Clelia, Styliana, Artemis, and Giorgos**

  For providing loving support. Their patience and understanding was my motivation.

- **My Parents: Giorgos, Meropi**

  For being there for me and helping me find the strength to continue whenever I had reservations.

- **My Closest Friends and Fellow Students**

  For being there for me with helpful comments and suggestions, especially Michael A. Georgiou who helped me with the implementation of ASSA and ASSA$_G$.

CREDITS

Publications that lead to this thesis:[38] [40] [35] [39] [36] [37].

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I dedicate my thesis work to my wife Clelia who was always there for me. My children Styliana, Artemis, and Giorgo who helped me keep all in good measure.

Notation	Meaning
AF	**A**rgumentation **F**ramework
AI	**A**rtificial **I**ntelligence
cont.	**cont**inues
DeLP	**De**feasible **L**ogic **P**rogramming
et al.	**et al**ii
EC	**E**vent **C**alculus
e.g.	**e**xempli **g**ratia meaning 'for example'
etc.	**et c**etera
i.e.	**i**d **e**st meaning 'that is'
iff	**if** and only **if**
LPwNF	**L**ogic **P**rogramming **w**ithout **N**egation as **F**ailure
NA	**N**egative **A**ssumption
NBP	**N**egative **B**ackwards **P**ersistence
NFP	**N**egative **F**orward **P**ersistence
NG$_B$	**N**egative **B**ackwards **G**eneration
NG$_F$	**N**egative **F**orward **G**eneration
NO	**N**egative **O**bservation
PA	**P**ositive **A**ssumption
PBP	**P**ositive **B**ackwards **P**ersistence
PFP	**P**ositive **F**orward **P**ersistence
PG$_B$	**P**ositive **B**ackwards **G**eneration
PG$_F$	**P**ositive **F**orward **G**eneration

PO	**P**ositive **O**bservation
resp.	**resp**ectively
RAC	**R**easoning about **A**ctions and **C**hange
s.t.	**s**uch **t**hat
wrt	**w**ith **r**espect **t**o

Table 1: Table of Abbreviations

Chapter 1

Introduction

"Skill in photography is acquired by practice and not by purchase."

Percy W. Harris

Argumentation is all around us and a child's most frequent question is 'Why?'. Generally, humans are trying to answer questions as an attempt to find reasons behind behavior. Starting from ancient Greece, India, and China, argumentation reasoning invaded peoples's lives [68]. Aristotle, with his methodology and proof through mathematics, initiated this field. Philosophers and rhetoricians observed some errors-fallacies in reasoning when trying to use arguments, helping for the birth of this field. Under any given scenario, humans must make decisions that depend on the current state of the world, the current needs, and the final goal. Argumentation can be used to explain the choices that people make, or to evaluate potential choices.

Researches in the field changed their goals continuously indicating that this area started to evolve rapidly. One of the goals was to form premises to reach reasonable-correct conclusions. Leibniz, Boole, Turing and others tried to add computation as a new category in reasoning. Their vision was that artificial intelligence (AI) can be formalized with a machine and much effort was

distributed to finding a way to simulate rational behavior. To understand someone's behavior, one must put themselves in the other person's place. Initially, this is how scientists tried to approach this matter; they also looked for a way to formalize rational behavior through mathematics [68].

At that time, deductive logic did not help to solve these kinds of problems. New approaches were needed. Hamblin (1970) thought of argumentation more as a reasoning process since it involves interaction. Hamblin's idea was as follows:
Two parties discuss an issue. The first party is trying to convince the other party that their conclusion (which conflicts with the other party's conclusion) is correct. The way to do this is by using premises that may be acceptable by the other party but that end in conflict to the final statement of the other party. Many groups adapted this way of thinking and argumentation became a mechanism for interacting with others.

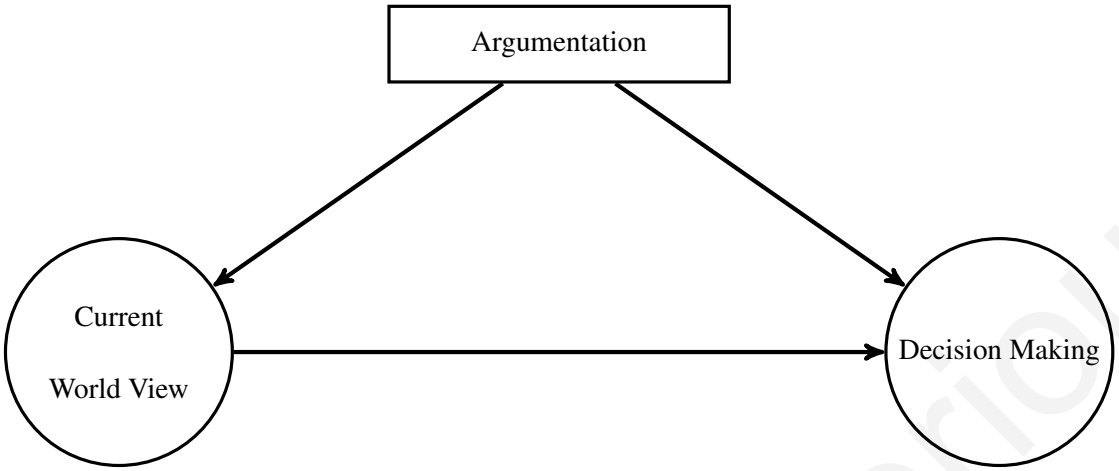
Recent developments combine argumentation techniques with AI. These fields (non-monotonic reasoning and multi agent systems) are currently expanding rapidly. Argumentation interacts through a notion called attacking relation between arguments. By understanding argumentation, individuals can learn more about themselves and can defend and express their thoughts and opinions in a conversation.

Decision making systems through argumentation have been studied in depth in recent years [29, 3, 30]. Given a set of alternative choices, arguments are constructed for and against [24]. By prioritizing these arguments through a technique (e.g. arithmetic evaluation), decision-making problems can be solved. As an evolving area, new systems obey rules in a better way than ever before and are heading into simulating human decision-making behavior. Argumentation-based agents [44] develop an architecture where hierarchies and preferences are given through argumentation. Modules exist that separated priorities into different high level schemas that can dynamically change under the agents operating environment. World view shows the outside world that helps the agent

adapt and decide what the next action will be. These decisions are based on the agent's motivations and needs.

Overall Goal of This Thesis: Adaptive systems can help people with their everyday tasks. Living in a constantly changing world, with incomplete information, people need to solve everyday problems. These problems, no matter how small or insignificant they are, can often be bothersome. Old systems that operate in a predefined way, despite the user or the operating conditions, are outdated. Agent software needs to be (1) autonomous, (2) able to operate in uncertain and dynamic environments, (3) adaptive and able to change dynamically when new changes appear in the world, and (4) consider the users' preferences and needs before deciding or recommending something useful to the user.

The aim of this thesis is to use argumentation to study adaptive and dynamic decision making. In this thesis by dynamic decision making and dynamic argumentation we mean how argumentation changes as the dynamics of the world changes. In a problem environment with changes happening over time, by dynamic argumentation we mean how the argumentation reasoning adapts to these changes or the environment. The thesis attempts to answer the question of whether real life problems can be mapped onto this abstract framework to create adaptive, personalized, and dynamic systems, and how arguments are parameterized and become valid or not in time in our working environment. Argumentation is used to keep track of the continually changing world, and to decide what option to follow given the current state of the world and the operating user. The world's view is required for decision making as a reference and a point to use for the argumentation process.



We aim to construct an adaptive, personalized, decision-making computational mechanism [44] that can operate in any environment. It should be able to adapt dynamically and present solutions to the user based on the current state of the world as well as the user's needs. Argumentation is the main functioning tool that is used in this study to construct recommendations, as it is closer to human behavior and can justify the conclusion.

We study how a dynamically changing world can be modeled using argumentation that can adapt to changes in the environment and to new information provided. We have re-examined the argumentation reformulation of language $\mathcal{E}$ and introduced backwards persistence as well as forward persistence arguments to deal uniformly with the frame and qualification problems. This has enabled us to extend in a meaningful way domains that the language $\mathcal{E}$ could not interpret when observations are included in the narrative. This argumentation interpretation corresponds to the unknown occurrences of events that could resolve potential inconsistencies between properties at different time points. Our extended argumentation framework comes closer to the original Event Calculus and shows how this could be extended to include observations. Using this we also provide a general parameterized argumentation based formulation of an adaptive and dynamic decision problem. To apply this theory, an application was created for hotel booking called 'Hotel for ME'.

Finally, although this was not the main intent, a matrix approach to computing argumentation is studied. The thesis describes how argumentation is computed with the help of graph theory and matrix multiplication, using theorems from these fields to optimize the results. We will develop algorithms and systems that compute the stable and grounded extension(s) of a given AF.

The overall focus of this work is divided into three parts: (1) Using abstract argumentation as a basis (Chapter 2), the problem is described in general and real-life problems for decision making are mapped from an argumentation point of view (Chapter 3). (2) By extending already existing work (see Chapter 4) adaptation over time is captured. (3) Finally, the theoretical part is applied by implementing a method for computing argumentation through matrices (Chapter 5) and trying to capture dynamic changes with different parameters under different time points. Chapter 6 concludes the thesis.

Chapter 2

A Review of Argumentation

"Notebook. No photographer should be without one."

Ansel Adams

2.1 Introduction

In AI, reasoning is done through a process of proof. For a proof to be acceptable, all premises should be valid in order to accept the conclusion. In courts, lawyers use facts to prove whether a person is guilty or not. In mathematics, proofs stand on theories that are proven to be correct; these theories stand on grounded premises that are assumed correct and acceptable to everyone. To justify the statements, arguments are used to directly imply and support the final statement or trigger another statement that will imply and support the final statement. This way of thinking is captured for example through the modus ponens rule $\frac{A \leftarrow B \quad B}{A}$. Unfortunately, in real-life scenarios where knowledge is uncertain, contradictory conclusions may be derived. Arguments may contradict the final statement or the premises that lead to the goal. Which premises are valid will always depend on the working environment and on the person who must make this decision.

For example, one would argue that no parent will deny water to his or her child. But what if the water is dirty and the child will become infected with an illness after drinking it? No person should yell at, or in an extreme case kill another person, but what if you are in a state of war? Another example is that in the context of reasoning about triangles by two mathematicians, argument a may be 'the sum of the three angles in a triangle is **less than** 180^o', while argument b is 'the sum of the three angles in **any** triangle is **equal to** 180^o'. These two statements are both correct; the first argument is correct in hyperbolic geometry, while the second argument is correct in Euclidian geometry. To make a decision, it is necessary to know as much information as possible and the decision will depend on the working environment and the personal needs and preferences relevant to this decision.

Argumentation is a process of reasoning, where given the current state of the world and, the available options, conclusions can be made. Conclusions are the result of 'winning' arguments that are time dependent, and a specific value is given depending on the persons involved with these arguments. For these situations, argumentation can help with the identification of information that may be missing in real-life problems, and thus help with the decision making task. If we associate argumentation with dialog then it can help us manage conflicts of opinion, conflicts of interest, and personal dilemmas.

Humans base decisions on criteria such as utility, cost, and references. There are two methods for making decisions: (a) optimization algorithms that evaluate criteria, and choose among alternative choices, and (b) argumentation based approaches that mimic human behavior of persuasion. The former procedure is formal and mainly suited to large problems such as how to optimize logistics, how to deliver items within a country, and how to deliver a company's merchandise by optimizing different variables. The latter is not as optimized as it is qualitative and mainly suitable for decisions that are dynamic, such as environments that are continuously changing. Case (b) is

mostly preferred for decisions and recommendations in environments that continuously change, as they can easily adapt. Case (a) is much harder, as adapting to changes means continuously changing the optimization algorithms.

Using argumentation theory to draw conclusions, one can: 1) Use a knowledge base to form arguments and determine how these arguments are in conflict and attack each other. The result is then an argumentation framework, that can be represented as a directed graph. 2) Based on this AF and some predefined rules, determine which arguments are accepted (these are called argumentation extensions). 3) Draw conclusions based on the set of accepted arguments.

Abstracting the structure that arguments have and their attacking relationship is introduced in Section 2.2, where certain notions that are introduced by Dung [24] are discussed.

2.2 Abstract Argumentation

"Laugh on laugh on my freind
Hee laugheth best that laugheth to the end."

Anonymous Jacobean student play, in 'The Christmas Prince'

Ceratin basic background notions that were introduced by Dung [24] are reviewed, such as acceptable, conflict-free, and complete extension. Every AF can be represented by a directed graph where nodes represent arguments and arrows represent attacks on arguments. Dungs AF is general, as no assumption is made about the way that arguments are built or the properties that the attacking relation has. Due to its generality, many researchers have adopted this way of thinking and use this representation as a base.

An abstract AF is defined as a pair $\langle \mathcal{A}, \mathcal{R} \rangle$, where $\mathcal{A}$ is a set of **arguments** and $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ is a binary relation on $\mathcal{A}$, called the **attack relation**. The set $\mathcal{A}$ may contain a finite or infinite

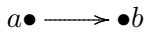


Figure 1: Attack

amount of arguments. Since our study is only interested in real-life problems, it is restricted to AF with finite arguments. Having a set of arguments where some arguments attack others, which of them can be accepted by a rational agent? To do this, the attackers should be examined, as well as the attackers' attackers. For this reason, an 'attacking relation' is applied to the AF to evaluate the 'winning' arguments under different attacking relations.

Let a, b be two arguments and $\mathcal{S}$ be a set of arguments. $attacks(a, b)$ (often written as $(a, b) \in \mathcal{R}$) means that the argument a attacks the argument b, or that b is attacked by a (see Figure 1) and $attacks(\mathcal{S}, b)$ (we will often write $(\mathcal{S}, b) \in \mathcal{R}$), which means that the argument $a \in \mathcal{S}$ s.t. $attacks(a, b)$ holds. In addition, a defeats b can be written instead of a attacks b. Arguments and attacking relations in an abstract AF do not have a specific structure. An AF can be represented as a directed graph where nodes represent the arguments and the edges represent the attacks. A simple example of an AF is shown in Figure 1.

The set $\mathcal{S}$ is said to be **conflict-free** if arguments $a, b \in \mathcal{S}$ s.t. $(a, b) \in \mathcal{R}$ do not exist (i.e. no self-attacking).

Definition 1 (Taken from Dung [24])

- An argument $a \in \mathcal{A}$ is **acceptable** wrt a set of arguments $\mathcal{S}$ iff for each argument $b \in \mathcal{A}$: if $(b, a) \in \mathcal{R}$ then $(\mathcal{S}, b) \in \mathcal{R}$.
- A conflict-free set of arguments $\mathcal{S}$ is **admissible** iff each argument in $\mathcal{S}$ is acceptable wrt $\mathcal{S}$, that is, $\mathcal{S}$ is conflict-free and defends itself.

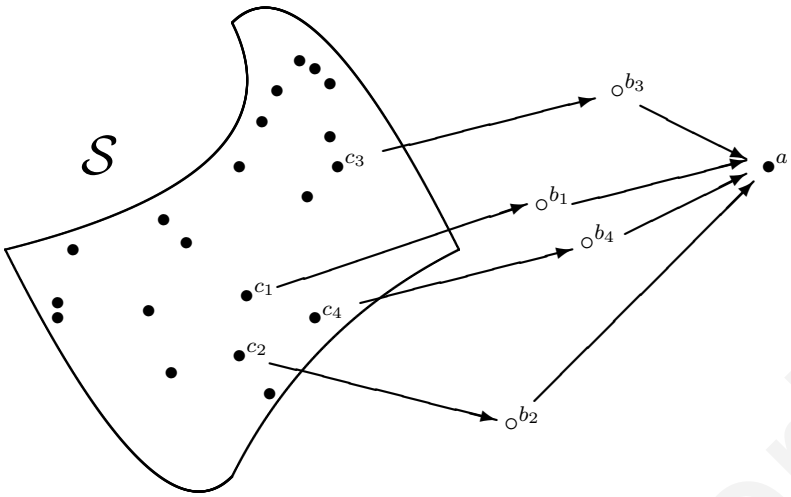


Figure 2: Argument a is Acceptable wrt $\mathcal{S}$

Informally speaking, an argument a is acceptable wrt set $\mathcal{S}$ if $\mathcal{S}$ can counterattack all attacks on a (see Figure 2). Additionally, set $\mathcal{S}$ is admissible if it is acceptable and no self-attacking occurs. From the above definition it follows that $\emptyset$ is admissible for any AF.

Example 1 Let $AF = \langle \mathcal{A}, \mathcal{R} \rangle$, where $\{a, b, c\} \in \mathcal{A}$ are three arguments s.t. $\{(a, b), (b, c)\} \subseteq \mathcal{R}$.

In Example 1, argument a is acceptable wrt $\mathcal{S} = \{a, c\}$ and $\{(a, c), (c, a)\} \notin \mathcal{R}$. Intuitively, when accepting a wrt $\mathcal{S}$, $\mathcal{S}$ should be not be self-attacking and must be acceptable.

Lemma 1 (Fundamental lemma) Let S be an admissible set of arguments, and a, a' be two acceptable arguments wrt $\mathcal{S}$. Then:
(1) $\mathcal{S}' = S \cup \{a\}$ is admissible, and
(2) a' is acceptable wrt $\mathcal{S}'$

If argument a is acceptable wrt $\mathcal{S}$, then $\mathcal{S}$ can counter attack all of the attacks on a. Therefore, the arguments $b_i \in \mathcal{S}, \forall i \in \{1, 2, \ldots, n\}$ exist, which counter attack any attack on a. Any other superset $\mathcal{S}'$, s.t. $\mathcal{S} \subset \mathcal{S}'$ still contains the b_i arguments $\forall i \in \{1, 2, \ldots, n\}$. Thus, a is still

Figure 3: Self Attacking

acceptable wrt $\mathcal{S}'$. Set $\mathcal{S}$ helps an argument to be acceptable wrt it, but it also has to fulfill the conflict free conditions (i.e., it cannot be self-attacking).

2.2.1 Semantics

Semantics in AF can be found in terms of extensions [24] or by labellings [17]. Extensions are discussed first.

Dung [24] studied the mathematics of three types arguments sets: preferred extensions, stable extensions, and grounded extensions. A set of arguments is called a **preferred extension** if it is a maximal admissible set of arguments that attacks all arguments that are attacking the set. Furthermore, a set of arguments $\mathcal{S}$ that attacks all arguments that do not belong to $\mathcal{S}$ is called a **stable extension**. It follows that every stable extension is a preferred extension but every preferred extension is not a stable extension. For example, the AF shown in Figure 3 has the empty set as a preferred extension but the empty set is not stable.

The function $\mathcal{F}_{AF} : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}$ of an AF $\langle \mathcal{A}, \mathcal{R} \rangle$ is called the **characteristic function**. The domain of $\mathcal{F}_{AF}$ is a set of arguments $\mathcal{S} \subseteq \mathcal{A}$ and the range is the set of all acceptable arguments wrt $\mathcal{S}$. The least fixed point of the characteristic function is called a **grounded extension**. There is only one grounded extension. To construct a grounded extension, one must start from the empty set and arguments that are only attacked by arguments that were already defended against are added.

Definition 2

- A conflict-free set of arguments $\mathcal{S}$ is called a **stable extension** iff every argument not in $\mathcal{S}$ is attacked by an argument in $\mathcal{S}$.
- An admissible set $\mathcal{S}$ of arguments is a **complete extension** iff $\mathcal{S}$ contains all of the arguments that it defends.

Definition 3 Let $AF = \langle \mathcal{A}, \mathcal{R} \rangle$ be an AF. The **grounded extension** is the minimal fix point of the characteristic function $\mathcal{F}_{AF}$.

By definition of the grounded extension, the grounded extension is the smallest complete extension. Additionally, it holds that the grounded extension is equal to the intersection of all complete extensions. Preferred extensions are admissible extensions. Given an AF, all admissible extensions are found and the extension with the maximal arguments (wrt set-inclusion) is the preferred extension. A preferred extension is a maximal complete extension.

Another way to find the semantics of an AF is with the **labelling approach** [17, 84]. Both the labelling approach and the extension-based approach achieve the same goal, which is to find the semantics of an AF. The thesis introduces both approaches, as the researches believe that in different instances, each approach may have an advantage of clearly illustrating a point. Both approaches are related, as shown in [53, 5] where a mapping relation is described. Given an AF and its labell, the extension set is equivalent to the sets of arguments that are labelled in and vice versa.

For the labelling approach, each argument receives a labell, which is either in, out, or $undec$, for arguments that are acceptable, not acceptable, and undecidable, respectively, based on the following definition:

Definition 4 Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an AF and L be a function s.t.

$L : \mathcal{A} \mapsto \{in, out, undec\}$. We say that L is a complete labelling iff it satisfies:

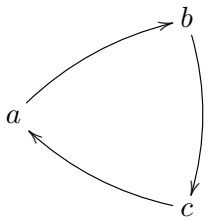


Figure 4: Undecidable is Necessary

- $\forall a \in \mathcal{A}$, $L(a) = out$ iff $\exists\, b \in \mathcal{A}$ s.t. $((b, a) \in \mathcal{R}$ and $L(b) = in)$
- $\forall a \in \mathcal{A}$, $L(a) = in$ iff $\forall\, b \in \mathcal{A}$ s.t. $((b, a) \in \mathcal{R}$ then $L(b) = out)$
- $\forall a \in \mathcal{A}$, $L(a) = undec$ iff $\exists\, b \in \mathcal{A}$ s.t. $((b, a) \in \mathcal{R}$ and $L(b) \neq out)$ and $\nexists\, c \in \mathcal{A}$ s.t. $((c, a) \in \mathcal{R}$ and $L(c) = in)$

Informally, labelling an argument in means the argument is accepted, labelling an argument out means that the argument has not been accepted (i.e., rejected) and labelling the argument $undec$ means that the argument cannot be accepted or rejected. Each argument then receives exactly one labell. Thus:

- an argument is labelled in iff all its attackers are labelled out
- an argument is labelled out iff it has at least one attacker that is labelled in
- an argument is labelled $undec$ iff none of the attackers is labelled in and not all of the attackers are labelled out

Example 2 This example shows that the labell $undec$ is needed. Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an AF, where $\mathcal{A} = \{a, b, c\}$ and $\mathcal{R} = \{(a, b), (b, c), (c, a)\}$ as illustrated in Figure 4. Here, only one labelling function exists, L, with $L(a) = undec$, $L(b) = undec$ and $L(c) = undec$.

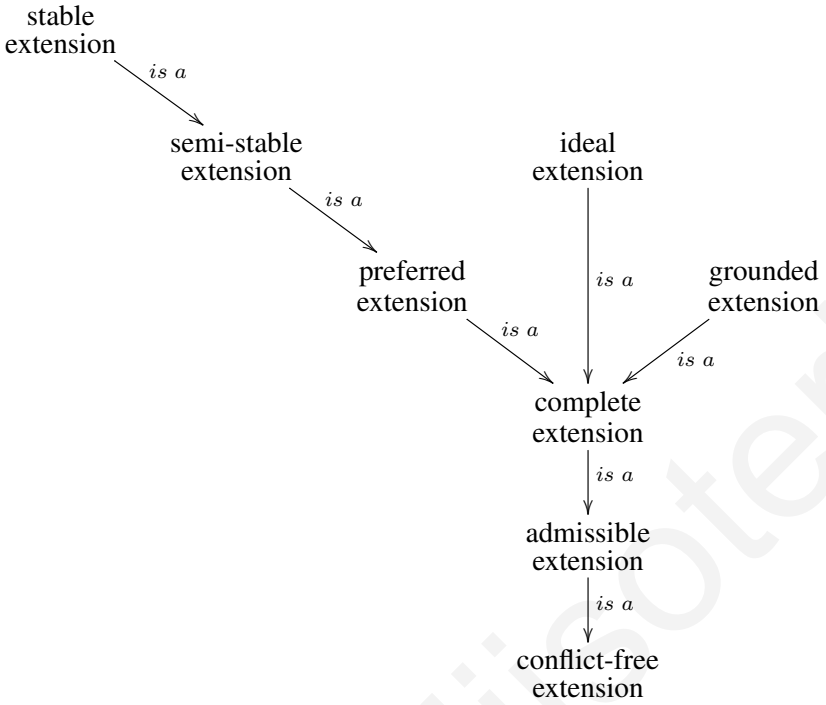


Figure 5: Extensions Overview

The **grounded labellings** that are considered a **sceptical approach** are examined. For every AF, there is **always** exactly **one** grounded extension. To find the grounded extension, all of the complete labellings must be selected. The grounded extension is the set with the minimal (wrt set-inclusion) arguments labelled in. The grounded extension is also conflict-free, which means that the grounded extension is actually the smallest complete extension. As an attempt to minimize arguments labelled in, arguments labelled out are also minimized and therefore, arguments labelled $undec$ are maximized. Thus, grounded semantics selects from the complete extensions the one extension with minimal in and out and maximal $undec$.

Many semantics have been established, such as grounded (yield exactly one unique extension), complete, preferred, stable, semi-stable [16], and ideal [25], and all of them come with certain properties [24, 5, 25]. An overall view of all of the extensions is shown in Figure 5. Table 2, taken from [8] shows the complexity of certain problems relating to AF.

Description	Complexity
Is $\mathcal{S}$ admissible?	P
Is $\mathcal{S}$ preferred?	CO-NP-complete
Is $\mathcal{S}$ stable?	P
Does $\langle \mathcal{A}, \mathcal{R} \rangle$ have a stable ext?	NP-complete
Is argument a accepted credulously?	NP-complete
Is argument a accepted sceptically?	Π_2^p-complete

Table 2: Complexity of Problems Relating to AFs

2.3 Preference-based Argumentation

In preference-based argumentation abstract AFs are realized with rules and priorities and preferences on rules. Here is an example to illustrate this statement:

Example 3 (Hotel booking) Let John be a doctor who wants to book a hotel for a medical conference talk. John wants to book a hotel that is near the conference and has good sports facilities. Available options for this trip include all of the hotels near the conference area with good sport facilities. Arguments for and against each exist that can filter out available options. If a hotel is closer to our user's profile, it will be preferred to other hotels that are farther away. Suppose that John likes hotel A better than hotel B or hotel C, and hotel B better than hotel C. Under this structure, John should choose hotel A.

In this simple example, argumentation can be used to express the relation between the three hotels. If a preferred rule such as the relation 'better than' is introduced then:

Hotel A is 'better than' hotel B,

Hotel B is 'better than' hotel C, and

Hotel A is 'better than' hotel C.

Since hotel A is 'better than' the other hotels, hotel A should be chosen.

There are other ways to express this preference. Suppose that values are assigned to each hotel, $A(100)$, $B(75)$ and $C(50)$. Through this arithmetic method, hotel A is still chosen since it has a higher score. More complex systems can assign negative values to options as well.

Under any dilemma, humans create arguments that will help them reach a final conclusion; these arguments may conflict. For instance, in the context of reasoning about hotel booking, argument a may be associated with the statement: 'Hotel A costs \$150 per night because it says that on their website', while argument b with the statement 'Hotel A costs \$50 per night if you enter the discount code Black-Friday-Rocks'. Here argument b attacks argument a based on the price and therefore argument b is preferred.

A well-known form of argumentation is **Logic Programming without Negation as Failure** ($LPwNF$) where, as its name implies, it does not use negation as failure but only explicit negation [46, 22, 43]. For this reason, in order to prove a literal a, a proof for a must exist. Additionally, if a counter attack on a exists, then it must be counterattacked. Argumentation theory in $LPwNF$ consists of a set of rules $L \leftarrow l_1, l_2, \ldots, l_n$ (in the background monotonic logic $(\mathcal{L}, \vdash)$), where L is the head of the rule and $l_1, l_2, \ldots, l_n$ is the body of the rule that may be positive or negative literals. $\vdash$ is obtained by the repeated application of the modus ponens rule s.t. $\frac{L \leftarrow l_1, l_2, \ldots, l_n \quad l_1, l_2, \ldots, l_n}{L}$.

Let G, W be two non empty sets of argument rules and T_0 be the background theory. G attacks W iff there exists a literal l and sets $G_1 \subseteq G$ and $W_1 \subseteq W$ s.t:

(i) $T_0 \cup G_1 \vdash_{min}$ [1] l and $T_0 \cup W_1 \vdash_{min} \neg l$

(ii) if there exist $r^{'} \in G_1, r \in W_1$ s.t $r^{'} < r$ then there exist $r^{'} \in G_1, r \in W_1$ s.t $r < r^{'}$.

[1] $T_0 \cup X \vdash_{min} a$ iff $T_0 \cup X \vdash a$ and there does not exist $X^{'} \subset X$ s.t. $T_0 \cup X^{'} \vdash a$

Let G be a set of arguments derived only by $\vdash$, i.e. modus ponens rule. G is admissible iff:

(i) $T_0 \cup S$ does not derive a literal l and its complement $\neg l$ and

(ii) for any $G^{'} \subseteq A$ if $G^{'}$ attacks G then G attack $G^{'}$.

This attacking relation is defined s.t. a set of arguments would attack another if it derives a contrary conclusion and its argument rules in doing so are not weaker than the opposing argument rules. One of the main benefits of $LPwNF$ is that it supports credulous or sceptical reasoning and has an argumentation-theoretic characterization.

$Gorgias$ is a system that implements $LPwNF$ combining also and abduction. It is a logic programming language that uses argumentation without negation as failure. Default reasoning is modeled by rules and a priority relation among rules. It can reason based on any given information that in many cases may be incomplete, based on preferences and policies defined by the developer. The output for a given query is followed by a set of rules called Delta that works as traced back user rules. Thus, we can justify any final decision by showing the path that was followed to reach the final conclusion. Additionally, $Gorgias$ system is an admissible system where it produces only admissible answers. If such an admissible answer does not exist, then it does not return an answer (i.e., acts in a sceptical way).

Defeasible logic programming ($DeLP$) [15] is a non-monotonic knowledge representation language that is based on facts, and strict and defeasible rules. The language has two different negations: 1) classical negation, which is used for representing contradictory knowledge and 2) negation as failure, which is used for representing incomplete information.

Decisions can be made in an environment with incomplete or missing information. $DeLP$ is applicable in a real-world environment; thus, it can operate under contradictory rules. A priority binary relation shows the stronger rule and this is where conclusions are drawn. As new information arrives, new rules are created, and priorities may change as rule strength is altered.

Informally, a defeasible logic theory is a collection of rules that allows us to reason about a set of facts or known truths, and reach a set of defeasible conclusions. Because multiple conflicting rules may be applicable to any given situation, a defeasible logic theory also includes a relation for resolving these conflicts.

There are three kinds of rules: 1) strict rules, where if the body becomes true then the head also becomes true, 2) defeasible rules, which represent conclusions that hold in most cases, and 3) defeaters, which are strict rules having as a head its negation. These rules have the following priority among them: strict rules are stronger than defeasible rules, which are stronger than defeaters.

A computational framework that provides recommendations based on argumentation techniques through the use of qualitative analysis is presented in [20]. The argumentation based tool that they have used is $DeLP$ [32], adding to their framework the advantage of handling dynamic domains with incomplete information and still retrieving sound conclusions. Users' preferences are constructed/modeled by the use of facts, strict rules, and defeasible rules. Based on existing knowledge and through a priority relation, they managed to hierarchically place preferences of the user with an AF. Thus, the recommendations to the users are found through an argumentation-based technique.

In [20], the authors found that 1) "existing recommender systems are incapable of dealing formally with the defeasible nature of users' preferences in complex environments" and 2) "The quantitative approaches (measures of similarity between objects or users) adopted by most existing recommender systems do not have a clean underlying model. This makes it hard to provide users with a clear explanation of the factors and procedures that led the system to come up with certain recommendations. As a result, serious trustworthiness issues may arise". Furthermore,

they continue by stating that a fundamental problem found in recommender systems techniques is determining which items are relevant to the users' needs.

Belief revision is a process of changing beliefs as new information arrives. It is related to AI for the design of rational agents. When an agent operates in a changing environment with uncertainty, new information can arrive through observations or interaction with other agents. After the new information becomes available, agents have to revise their beliefs. Argumentation deals with strategies that agents employ for their own reasoning, as well as strategies for how to change the belief of other agents. Thus, argumentation and belief revision are perceived as two sides of the same coin.

Frameworks in belief revision consist of a language L. Logical formulas $a_i \in L$ are used to represent knowledge. A complete set of connectives on formulas can be used to construct more complex formulas. The theory in [2] is abstract. The ways that belief revision can be applied in argumentation, as described in [[68], Chapter 17] are as follows: (1) changing by adding or deleting an argument(s) (or a set of arguments), (2) changing the attack or defeat relation among arguments, (3) changing the status beliefs as conclusions of arguments, and (4) changing the type of argument from strict to defeasible or vice versa. After observing what the result would be when some changes occur, the system can be better understood, including how it operates, and can ultimately be used for a beneficial purpose.

Another relevant work is the work of Bench-Capon [9]. They extend the AF that was introduced by Dung by allowing values to be associated with each argument. In this way, some arguments can be 'stronger' than other arguments wrt a certain value that the argument is assigned, and so the success of an attack between conflicting arguments will depend on these values. Their value based AF assigns a value to each argument through a function. This function takes the audience into consideration, and as a result prioritizes arguments based on an attack relation

and on a notion of relative strength, that is assigned by the value function. Therefore, it may not be sufficient to say that an argument attacks another argument if the value that is assigned to the attacker is less than the value that is assigned to the argument under attack. A value-based AF often written as VAF = $\langle \mathcal{A}, \mathcal{R}, \mathcal{V}, val, pref \rangle$, where $\mathcal{A}$ is a set of arguments and $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ is a binary relation on $\mathcal{A}$, $\mathcal{V}$ is a non empty set of values, val is a function that associates each element from $\mathcal{A}$ to an element from $\mathcal{V}$, and $pref \subseteq \mathcal{V} \times \mathcal{V}$. In the value-based AF, $a \in \mathcal{A}$ attacks $b \in \mathcal{A}$ iff: 1) $(a, b) \in \mathcal{R}$ and 2) $(val(b), val(a)) \notin pref$. As discussed later, parametric argumentation is used in the thesis, which has similar properties to the value based AF.

When an agent is trying to overcome a difficult situation, its sensors receive information, from which s/he tries to reach a judgment. Neither classical logic, deductive logic, inductive logic, or abduction are sufficient tools to help the agent reach a conclusion, as even the most detailed information is incomplete. In cases that are similar to this one, the agent must overcome these limitations. S/he needs a combination of logic and tools that are able to reason despite incomplete information and uncertainty.

Argumentation has been proven to be an effective approach to complex problems. Argumentation constructs arguments and evaluates them based on priorities. This process can help an agent make a decision in dynamic environments where missing information or uncertainty exists. Thus, argumentation is a process of creating strategies to change other agents' goals and protect their own. Argumentation can be used in a *negotiation*, (where an agent tries trough a dialectical process to make the best deal possible through a dialectical process), *inquiry* (when agents are not familiar with a subject but they have to find proof or destroy one), *deliberation* (when agents are trying to find a common ground on which to act), and *information seeking* (where agents seek knowledge from other agents through dialectical process).

The researches believe that argumentation is still evolving and that many areas need more exploration. There will always be room for another AF as a new way of reasoning with incomplete information. This way can either be constructed from scratch or it can be a hybrid approach from the already existing frameworks.

AI combines logic, computation, mathematics, argumentation, and human behavior. Modern applications [29, 31] use AI in order to decide which action to choose. These decision problems must update their knowledge base continually to provide legitimate information. With the rapid growth of the internet, it became interesting to study problems from a human perspective. After approaching each user individually by observing their needs, preferences, and habits, targeted, adaptive, and personalized decision-making suggestions became necessary.

2.4 Dynamic Argumentation

We have seen that an abstract AF is defined as a pair $\langle \mathcal{A}, \mathcal{R} \rangle$, where $\mathcal{A}$ is a set of arguments and $\mathcal{R}$ is a binary relation on $\mathcal{A}$, called the attack relation. Many semantics have been established, and all of them come with certain properties [24, 5, 25]. With dynamic argumentation, arguments and the attack relation for each AF are studied in depth, and properties are then found that can evaluate semantics faster based on these properties. By knowing all arguments and their attacking relation, we can evaluate and predict the system's behavior. In a problem environment with changes happening over time, by dynamic argumentation we mean how the argumentation reasoning adapts to these changes or the environment.

Dynamic argumentation has been receiving much attention as an agent work in dynamic environments and changes happen continually. If a change happens then the world changes and some arguments and attacks may be influenced. How the acceptability of an argument affects other arguments is studied in [12]. On an abstract AF, since arguments and attacks on arguments are the

only components, one of the following steps can be taken:

1) Add an argument or a set of arguments,

2) Remove an argument or a set of arguments,

3) Add an attack or a set of attacks,

4) Remove an attack or a set of attacks, and

5) Combination of the above.

Having an AF and its extensions, when a change takes place and alters the framework; it is not necessary to evaluate from scratch the new extensions. For example, an isolated set of arguments that is not affected by the change, all of its arguments that were a part of an extension will still be a part of that extension after the change [6]. In additions, even when the set is affected, dynamic argumentation uses the existing extension set to evaluate the new extensions faster.

Other questions that may arise are: 1) Can we find the acceptability of arguments without computing all extensions [61, 11]? This question was asked in the ICCMA'15 competition. 2) What is the minimal change that can happen so that an argument a still belongs to the extensions [[68], Chapter 17]?

In Section 5.3 we present a matrix-based theory that takes under consideration existing evaluations of the AF. When the AF is updated after a change to a new AF, the old knowledge is considered to find faster the new extensions.

Chapter 3

Time-based Argumentation Frameworks for Decision Making

"A person who won't read has no advantage over one who can't read."

Mark Twain

3.1 Introduction

Decision theory is a theory about choosing options according to some criteria. Having a set of options, which one should be chosen to solve a task? Is there more than one acceptable solution? The process of choosing the best options among a set of alternatives under the current working environment was used by philosophers and rhetoricians [73], economists (how to maximize profit or minimize loss), psychologists (how to mimic human behavior [59]), doctors (what is the best treatment or how to make a medical diagnosis [34]), and computer scientists (how to optimize a solution [30]). Humans follow rules to find valid alternative paths in order to reach their goals. By prioritizing these rules, the 'best' option(s) will eventually stand out and by backtracking to critical points of the procedure, the end results can be justified or other solutions to the problem can be found that will be optimal or sufficient.

Many problems have more than one solution and can be solved by a decision theory process: (1) Shall I wear my coat today? This decision depends on future knowledge I do not have the answer to right now. (2) I am looking for a car to buy. Shall I buy this one? This offer I found looks intriguing but if I search a little bit more I might find a better offer. When will the searching procedure stop? (3) Nobody likes being bold. Taking one hair out of my head will not make me bold. If I continue doing this, a point in time will come where I will have no hair. (4) The judge has to decide if this person is guilty or not. (5) Three examiners have to decide if this person should retake the exams. What if they have different opinions? There should be a process that provides a solution to this problem when followed.

People experience dilemmas every day, mainly because of missing information. These decision problems take the working environment into consideration and choose the best alternative option(s) [29, 55]. When a solution already exists, decision theory problems can help us optimize this solution [30], where some other systems can provide the reasoning behind each decision [44, 31].

Each user has a unique personal profile consisting of the user's preferences and needs at the time point where the decision has to be made. Therefore, under different time points, the user will make decisions that may differ. Each decision problem is stigmatized and driven by the parameter spaces that define what holds in the current snapshot of the world; it can qualify options based on pragmatic considerations. Additionally, qualified options must also be filtered by the user's needs. In this way, by prioritizing and filtering options that best suit the user can be identified. Newly developed methods and systems should be adaptive as choices are feasible, (i.e., under different circumstances, different choices may qualify) and should also be personalized as the recommended choices should be suited to the user.

Shall I wear a coat today? This question is driven by the goal that I do not want to be cold at any point of the day, and is based on each person's preferences, (i.e., one person may want to wear the

coat when the temperature is less than 17^oC and another person when it is windy). The decision is also based on the current state of the world, whether it is spring, summer, autumn, or winter. Newly developed decision theory systems must be able to suggest options under any working environment and adapt dynamically to a continually changing world, recommend option(s) despite uncertainty and missing information, and justify the end decision at any time point [44, 31].

The following example is adapted from [34, 30] and summarizes these notions and captures the challenge of trying to solve these types of problems.

Example 4 (Medical. Adapted from [34, 30]) Doctor J has patients $p_1, \ldots, p_n$. Each patient has symptoms placing an illness higher in a hierarchy over another. Possible illnesses form the general parametric space. Based on the patients' medical history, this taxonomy is now restructured as it might fit the patient best. The new hierarchy is the special personal profile of the patient. What test should the doctor perform next in order to recommend the best medicine? Tests are enabled by the physical world (i.e., we do not have this expensive machinery in this clinic (argument against) or this test does not take long to perform; argument for). When new tests are performed and new information arrives, the knowledge base is extended. New preconditions may then be satisfied supporting medicine A or disfavoring it. Clearly, Doctor J has to make a decision in the end and recommend an action.

Argumentation with parameters were assigned to each option. This approach takes us closer to the value based AF [9]. Our attacking relation is assigned based on the priorities of each parameter, where the attacking relation on the value based AF is assigned by combining and analyzing the values that each option has.

3.1.1 Related Work

This subsection concentrates on existing work for decision making and the rest of this chapter will study how we extend this work by introducing time-based argumentation.

Recommender systems [71, 69] are software systems that help users make decisions based on (1) past behavior and (2) relevant history of other similar users by suggesting available alternatives [70]. Information over the internet expands rapidly and users find it hard to make decisions over the plethora of information available. A filtering of information technique can help users overcome the information overload problem.

A recommendation is considered good when the user decides to follow it. Many techniques exist that perform different types of filtering. Filtering is based on probabilities, machine learning combined with probability algorithms, and filtering based on keywords [72]. Other techniques analyze user's likes and dislikes [66] while others place users in similar groups [77]. Mainly, filtering techniques are separated into the following groups: (a) **collaborative filtering** that infers the preferences of an individual user based on the behavior and preferences of similar users, (b) **content-based filtering** that infers that users' preferences persist through time and recommendations are built on similar items that the user liked in the past, and (c) **hybrid filtering** is a combination of technique (a) and (b) [4].

Recommender systems are techniques where a specific user and a set of options will provide suggestions. These suggestions are qualified after a decision-making process, such as deciding what goods to buy, film to watch, or music to listen. Two types of recommendation exist: (1) Those that are non-personal and disregard whether the user is a man, woman, child or adult; the system will suggest the most popular options. (2) Personalized recommendation systems stand on

previous actions, analyze user's habits and needs, and suggest targeted recommendations throughout a procedure. Early recommender systems [55] placed users into groups based on past behavior. If a user in a group of similar users liked a product either because they rated it with a high score or left a positive comment, the system would have recommended this product to all other users under that group. Extending this procedure, if user A agreed with user B in the past, then some of users B's preferences would interest user A.

The internet evolves rapidly and new products emerge quickly, making them difficult to track. In addition, existing web-sites have grown and new web-sites have created that have made this tracking procedure difficult for the users to follow all of the suggested options. Therefore, increasing the amount of choices is not the best strategy and targeted options may be the best solution. The users should be studied and all of their relevant actions should be taken into consideration in order to find possible recommendations. More advanced filtering methods for more targeted recommendations were then created. If the users searched for a new product, this new information was blended with previous information and a new profile of the user was constructed, which resulted in new, more focused recommendations.

Recommender systems are used over the web, and include web-sites such as amazon, YouTube, ebay, IMDb. These online applications are implemented in many ways including through argumentation; Example 6 illustrates this statement. Similar problems on decision making can also be formalized through argumentation. Any system that can construct admissible semantics, taking into consideration the users profile and special needs, can produce a solution to our problem. Recommender systems follow a procedure to recommend an action. If this action can be easily 'digested' by the user, it is more likely to be appreciated. Companies want users to follow options that they provide, making these systems a powerful tool in their hands.

Recommender systems have many capabilities and advantages and serve a variety of applications, but still require further improvements as these techniques are not optimal. At his time point, many limitations exist, because recommender systems process is complex and multi-variable. These systems must consider much information and analyze many variables, even for the simplest recommendation. Many recommender systems' recommendations rely on users with similar interests or the overall user ranking system. As stated in [20], recommender systems find it difficult to perform qualitative inference on the final recommendations and to deal with the defeasible nature of users preferences. Many recommendations are based on preferences that similar users may have, resulting in weak statements that rationally justify why the new user should buy or consider the new recommended product [42], and trustworthiness issues may arise. One more disadvantage of content-based recommender systems is that if two different items are represented by the same set of features (i.e., are represented by the same list as they are similar) then these items are not distinguishable.

On the other hand, these problems have been addressed through defeasible AFs. For a recommendation to be made, information about each user is needed. Usually older products are preferred as a ranking value likely exists. Therefore, a lack of data, which is common in real-world environment, may reflect ineffective recommendations. Furthermore, keeping track of new products is a difficult procedure. Another disadvantage is the change in user preferences, as a user may be looking for a book to read today, but tomorrow the same user may be looking for a present.

The advantage of the approach that is presented here is that it is flexible and can evolve as time passes and factors become more or less important. With this modularity, a mechanism for handling relatively new knowledge can easily be accommodated. New priority relations can then be applied to obtain new results whenever needed. Moreover, argumentation handles recommendations in a

qualitative way. Therefore, recommendations can change from day-to-day because a new special offer may occur that reflects the active user.

3.2 Parameterized Argumentation - Theoretical Framework

The aim of this sections is to formalize a decision problem using standard notions from argumentation in AI [24, 68]. Decision making is explored in terms of argumentation. This section shows the link that each option has to different parameter spaces (e.g., parameters based on the working environment and the users' mental conditions). Arguments can then be constructed to prioritize these options and as a result, find the 'winning' options. Since the state of the world continually changes and the knowledge base is updated fast, the need for dynamic systems that can handle these updates and deliver a good representation of the world is introduced. Technologies that can support this type of reasoning and adjust dynamically based on the operating user and the pragmatic truths of the world are desired. Allowing these technologies to be part of the user's decision process will help with overcoming difficult situations. Old technologies that operate in the same way no matter who the user is or do not take the working environment into consideration need to be developed further.

The last decade's argumentation has been widely used for single agents and for multi-agent systems, and manages to play an important role in the implementation of these technologies. Currently, agent software is autonomous and uses argumentation to make decisions in uncertain and constantly changing environments in order to present only relevant results to the user. Furthermore, these agents can interact with other agents to exchange information and in activities such as negotiation, cooperation, and dialogue.

The thesis describes the development of an adaptive and personalized system that can help the user throughout the decision process. This is done by selecting admissible choices in an AF [24].

An abstract AF that adapts dynamically to the changes of the world and the users' preferences is described. In this AF, each option is time dependent and each user has different needs that may change as time passes. Therefore, the users' preferences are also time dependent.

Using argumentation, an action is computed, recommended and explained, or the best option is chosen from a set of exclusive alternatives. Our argumentative decision-making process is influenced by the working environment and the user's preferences. To preserve persistence, this theory is base on the AF of language $\mathcal{E}$ [48], and is extended [35, 38, 40, 39] to accommodate arguments based on property observations, and arguments for backwards persistence. Given a time line, priorities for arguments and observations are introduced to manage and maintain a good representation of the world. Ultimately, the system should suggest options under the extended working conditions, followed by a report that justifies how the system reached that conclusion.

To compute and explain the selected decisions, *Gorgias*[1] argumentative-based system was used to produce the 'winning' options with an explanation-report that was generated on the working process. A time model agent was used that applies argumentation on more than one level to decide and recommend new options to the user. As time passes, new knowledge arrives. This knowledge must be understood to be represented in the AF and extract conclusions. Therefore, our system should be able to adapt dynamically to a continually changing environment. To accomplish this, when the world changes arguments, for and against options are enabled. Thus, there are arguments that may support or attack an option as time changes.

Definition 5 (Options) In a decision problem there is a set of alternatives, $\mathcal{O}$, called the options of the problem.

Options are parameterized based on ontological problem parameters and pragmatic parameters.

[1] http://www.cs.ucy.ac.cy/~nkd/gorgias/

Definition 6 (General Parameter Space of an Option) Let $\mathcal{P}_j$ be ontological hierarchies for option o, and let $\mathcal{P}^p$ be the pragmatic set characterizing o. The general parameter set of an option is the set, $\mathcal{P} = \bigcup_{\forall j} \mathcal{P}_j \cup \mathcal{P}^p$.

Users' have preferences and needs. For any parameter in any parametric space, a labelling function can be applied, such that parameters are mapped to the set $\mathcal{S} = \{1, 0, -1\}$ based on user preferences. Parameters that a user is interested in are labelled as 1, -1 for the parameters that the user shows no interest in, and 0 are those parameters that have not yet been specified by the user (i.e., that are unknown). A total order on this labelling is then applied to reveal user's special preferences.

Definition 7 (labelling function) Given user U, let o be an option and $\mathcal{P}$ be its general parameter space. For each $\mathcal{P}_i \subseteq \mathcal{P}$ there is a labelling function:

- $f_U : \mathcal{P}_i \to 1$ iff user is interested in parameter p_i, also denoted by $p_i^U(1)$
- $f_U : \mathcal{P}_i \to 0$ iff parameter p_i is unspecified, also denoted by $p_i^U(0)$
- $f_U : \mathcal{P}_i \to -1$ iff user is specifically not interested in parameter p_i, also denoted by $p_i^U(-1)$

Labelling parameters is not an easy task, but it can be retrieved either by explicitly asking the user or implicitly through the user's actions. The labelling function f can be applied to construct three subsets of parameters for the user since each parameter can be mapped to one of the parameters of $\mathcal{S} = \{-1, 0, 1\}$. The thesis is primarily interested in the set containing parameters $p_i^U(1)$.

To construct priorities, the constructed sets are considered by the labelling function. Set $p_i^U(1)$ is preferred over the sets $p_i^U(-1)$ and $p_i^U(0)$. Priorities are also allowed among the set $p_i^U(1)$,

which means that if a user likes both parameters both are labelled 1, but amongst two of these, this priority relation stimulates a preferred option. The formal definition of this is presented below.

Building options can conflict with each other. An argument for one option may attack arguments for other conflicting option; this depends on the strength of the arguments. Throughout this conflicting relation, a stronger argument can attack any other argument that is not as strong and can counter attack any potential attack from weaker arguments. As a result, any decision-making process may end when the winning argument stands out.

Arguments can be built for each option. Based on the user's special personal profile, arguments are built for or against an option.

Definition 8 (Arguments) Let U be a user. We denote by $arg(o; Sup(o); Par(o))$ an argument built to support option o, where $Sup(o)$ is a set of conditions $\{c_1, c_2, \ldots, c_k\}$ that can be evaluated in the external environment of the application, and these take the value of true or false and are time dependent. $Par(o)$ is a set of constraints on the options that are time dependent. There is a labelling function that picks o and maps it to two different sets: a) Par ontologically these are the characteristics of the option and b) Sup pragmatically must hold in the world for this option to be enabled.

Definition 9 (Priority over arguments for user U) Let U be a user with profile $Prof_U = \{[p_i^U(1)], [p_i^U(0)], [p_i^U(-1)]\}$ and $arg_1(o_1; Sup_1; Par_1)$, $arg_2(o_2; Sup_2; Par_2)$ two arguments where o_1 and o_2 are in conflict. If both sets Sup_1 and Sup_2 hold in the decision environment, then $arg_1 \geq_u arg_2$ if $Par_1 \sqsupseteq_u Par_2$, where $\sqsupseteq_u$ is defined under the labelling function and the personal priorities of the user as a partial order priority on $p_i^U(1)$ and it is time dependet.

To explain Definition 9, let Klelia be a user that tries to find a hotel to stay in, with profile: $Prof_{Klelia} = \{[\text{pool}], [], []\}$. Klelia specified that she is interested in hotels with a pool, and she has given no information on what she dislikes. Let hotel_1 and hotel_2 be two hotels with a pool that are in conflict and $arg_1(\text{hotel}_1; []; [\text{pool under construction}])$, where $arg_2(\text{hotel}_2; []; [])$. Even though both hotels support the need of Klelia, which is a pool, $arg_2 \geq_u arg_1$ since $Par_1 \sqsupseteq_u Par_2$ as the pool in hotel_1 is under construction.

Arguments may support an option or be against it. Direct attacks come from the user's special personal profile, as it consists of contradicting sets: the 1 and the -1 labelled set. Indirect attack exists as more than one alternatives exist, and the support of one argument under the scope of an option is an attack under the scope of a different option. In addition, under the working conditions that are included in each argument through the Sup set, an option may cease to qualify. Therefore, prioritizing arguments is an important step, and through an admissibility process, these qualified options are focused and an option is recommended to the user.

Definition 10 (Attacks) Let $arg_1(o_1; Sup_1; Par_1)$ and $arg_2(o_2; Sup_2; Par_2)$ be two arguments. arg_1 attacks arg_2 if o_1 and o_2 are in conflict and $arg_1 \geq_u arg_2$.

Depending on the problem and the number of the admissible solutions, attacks may be strict in order to focus on the winning option. Each user is treated as a special and unique entity, and for this reason different users may receive a different recommendation. As each user has their own parametric space where options are prioritized, even when two options for different users may be admissible, one option may be suggested to one user over another user, as these parametric spaces may prioritize one option over the other in different ways. This adds one last step of hierarchy to the options, which results in a personalized recommendation. The qualified options are the solutions, and the recommended options of the system.

Definition 11 (Solution) Let U be a user with profile $Prof_U$. Let $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ be the set of options that are supported by at least one argument where the arguments Sup holds in the decision environment. We say that o_i is a solution if it is admissible.

3.3 An Example Application: 'Hotel for ME'

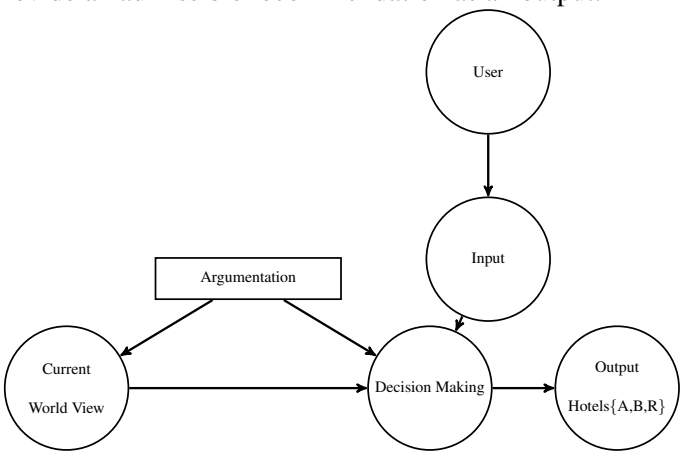
People stay in hotels for many reasons, including work or pleasure, with family or alone; hotels exist to serve their needs. Each user is different and at a specific time point the needs of the user may vary. Every hotel offers services differently from other hotels. As time passes, the support of arguments for the various options of hotels may change and new information may override old information. As a result, the options' strengths and weaknesses change.

In this study, argumentation and priority relation among arguments were used to allow dynamic manipulation over the changing world. The applications became personalized since the user profiles were analyzed, including needs and habits by extracting them through a set of queries. As implemented on $Gorgias$, all of the rules and priorities that were used to reach the final conclusion were managed in an attempt to convince the user to follow a recommended option.

A concrete way to understand the problem was studied through an application that helps the user decided which hotel to stay in. This system to recommend the option that best suit the user from the available options. The decision-making process and maintaining good representation of the world that keeps track of all changes, described in Chapter 4, both use argumentation. Therefore, argumentation is used to keep a track of the world and to decide what option to follow given the current state of the world.

As shown in the following Figure, for this system to provide a recommendation, the user and their needs that are given as an input must be known. The decision-making process uses

argumentation with the restrictions and the constraints that the working environment has in order to provide an admissible recommendation as an output.



Example 5 (Hotel Booking) Artemis needs to book a hotel next month. She has to choose among several hotels. Which one should she choose? What if some hotels have special offers? Suppose that hotel A offers special rates during the weekend. Our system should be able to update its knowledge base and preserve this new information for the predefined weekend. Also when the weekend ends it should be able to check if the offer ended and update again appropriately. Existing knowledge must be preserved as it is part of the underlying knowledge base of the hotels. In this way the world remains the same until new knowledge causes an update.

The thesis shows how these kind of real-life problems are translated through the AF and how dynamically changing environment are handled. For this matter, a hotel booking problem was mapped onto the AF, and is presented in Section 3.3.2.

Example 6 (Hotel Booking cont.) Hotels are the available options, that are **parameterized**: Options = $\{o_1, \ldots, o_n\}$. Each hotel has different facilities, unique location, and special offers that are mapped to this hotel through its parameter space. Therefore, options can be seen as follows:

Options = $\{o_1(p_{1_1}, p_{2_1}, \ldots, p_{k_1}), \ldots, o_n(p_{1_n}, p_{2_n}, \ldots, p_{k_n})\}$, where $p_1, p_2, \ldots, p_k$ are parameters. Parameters can be time dependent (e.g., (a) every Tuesday it is Italian night, (b) the swimming pool is under construction for the next two days), where we can clearly see that they are time dependent.

Defining different spaces: (a) option space and (b) user preference space that constantly changes over time (**dynamic environments**). Under the abstract theoretical framework, argumentation techniques can be used to reach a final conclusion.

3.3.1 Representing the Problem in our AF

Here, how the problem can be represented our AF of Section 3.2 is presented. Specifically, the AF and language of $Gorgias$ [1, 64] is used to demonstrate how these representations can be implemented. In $Gorgias$ [64], arguments are built from rules. Rules have the form $rule(Label, Head, Body)$, where $Label$ is the name of the rule. $Head$ and $Body$ are the head and body of the rule respectively, and can be a positive or a negative program fluent. With operators $neg/1$ and $complement/2$, a contradictory conclusion can be defined in the program. The predicate $prefer/2$ was used, which defines priorities among the $Labels$ (i.e., $prefer(Label1, Label2)$ shows that if $Label1$ and $Label2$ are in conflict, $Label1$ is preferred.

Given a program, queries can be written through the predicate $prove(Query, Delta)$. $Gorgias$ will then try to compute an answer which will be Yes or No. When the answer is Yes, all rules and priorities that were used to reach to this answer will be shown to the user under the variable $Delta$.

An example is presented where a user tries to choose a hotel to stay at (see Listing 3.1). There are three facts that are labelled $f1\_o(X)$, $f2\_o(X)$ and $f3\_o(X)$, two rules that are labelled $r1\_o(X)$ and $r2\_o(X)$ and one priority rule that is labelled $pr1\_(X)$. A rule labelled $f1\_o(X)$

has as a head the fluent $hotel(X)$ and states that there are twenty hotels. There are also rules with the heads fluent $special\_rates(X)$ and $crowded(X)$. Hotels $X \in [1,3] \cup [10,13]$ have special rates, while hotels $X^{'} \in [2,7] \cup [12,17]$ are crowded. Rule labelled $r1\_o(X)$ has as a head the fluent $stay\_in\_hotel(X)$. This rule becomes active when its body, the fluents $hotel(X)$ and $special\_rates(X)$ become true. The rule labelled $r2\_o(X)$ contradicts the rule labelled $r1\_o(X)$ as their heads contradict each other. Among contradictory arguments, those from rule $r2\_o(X)$ are preferred, as described by the priority rule labeled $pr1\_(X)$.

Listing 3.1: Hotel Booking (Simple)

```
% Hotel example
rule(f1_o(X), hotel(X), []) :- between(1,20,X).

rule(f2_o(X), special_rates(X), [hotel(X)]) :-
                        between(1,3,X);between(10,13,X).
rule(f3_o(X), crowded(X),       [hotel(X)]) :-
                        between(2,7,X);between(12,17,X).

rule(r1_o(X), stay_in_hotel(X), [hotel(X),special_rates(X)]).
rule(r2_o(X), neg(stay_in_hotel(X)), [hotel(X),crowded(X)]).

rule(pr1_(X), prefer(r2_o(X),r1_o(X)), []).
```

Listing 3.2: Answers for Listing 3.1

```
?-prove([stay_in_hotel(X)],Delta).
```

```
X = 1,
Delta = [f1_o(1), f2_o(1), f1_o(1), r1_o(1)] ;
X = 10,
Delta = [f1_o(10), f2_o(10), f1_o(10), r1_o(10)] ;
X = 11,
Delta = [f1_o(11), f2_o(11), f1_o(11), r1_o(11)] ;
false.
```

To identify what hotels one can stay in, $Gorgias$ is asked to build admissible arguments for the option '$stay\_in\_hotel(X)$'. When the query $prove([stay\_in\_hotel(X)], Delta)$ is asked (Listing 3.2) $Gorgias$ presents an answer followed by a bag of rules and priorities that are used to reach this answer. These rules are given as an explanation in $Delta$. For example, hotels $X \in \{1, 10, 11\}$ qualify and are presented by $Gorgias$ when all solutions are required. The set of qualified options can be narrowed down by adding more rules, as shown in the extended example (see Listing 3.3). Three more facts $f4\_o(X)$, $f5\_o(X)$, and $f6\_o(X)$ are added that separate hotels in two areas. Hotels $X \in [1, 10]$ belong to area 1, while hotels $X \in [11, 20]$ belong to area 2. Furthermore, the body of the rule labelled $r1\_o(X)$ is extended and contains the fluent $near(X, 2)$. Thus, all suggested hotels will be constructed from the first example (Listing 3.1) that is also in area 2 (see Listing 3.4).

Listing 3.3: Hotel Booking (Extended)

```
% Hotel example extended
rule(f1_o(X), hotel(X), []) :- between(1,20,X).

rule(f2_o(X), special_rates(X), [hotel(X)]) :-
```

```
                between(1,3,X);between(10,13,X).
rule(f3_o(X), crowded(X),        [hotel(X)]) :-
                between(2,7,X);between(12,17,X).

rule(f4_a(A), area(A), []) :- between(1,2,A).

rule(f5_o(X,A), near(X,A), [hotel(X),area(A),A=1]) :-
                between(1,10,X).
rule(f6_o(X,A), near(X,A), [hotel(X),area(A),A=2]) :-
                between(11,20,X).

rule(r1_o(X), stay_in_hotel(X), [hotel(X),
          special_rates(X),near(X,2)]).
rule(r2_o(X), neg(stay_in_hotel(X)), [hotel(X),crowded(X)]).

rule(pr1_(X), prefer(r2_o(X),r1_o(X)), []).
```

Listing 3.4: Answers for Listing 3.3

```
?-prove([stay_in_hotel(X)],Delta).
X = 11,
Delta = [f4_a(2), f1_o(11), f6_o(11, 2), f1_o(11),
          f2_o(11), f1_o(11), r1_o(11)] ;
false.
```

Gorgias can be used in web applications with several benefits. By adding priorities through preferred rules, options can be qualified that are based on the user's profile, needs, and habits. As a result, the 'stronger' options are identified.

In summary, with this type of application, argumentation is used to (1) construct a system that can preserve knowledge and accept new observation [48, 38, 40], (2) extend the knowledge base dynamically, and make decisions that best suit the user.

3.3.2 Formalization of the Application 'Hotel for ME'

In this subsection, the application problem of 'Hotel for ME' is formalized through argumentation based on the general theory that is presented in Section 3.2.

Definition 12 (Decision problem) Recommend amongst a set of available hotels, those that best suit the user's needs and preferences.

Definition 13 (Options) A set of available hotels with any information that characterizes this hotel, e.g. $o_i = \{hotel_i, cyprus, area, facilities, children\,care, near\}$.

In this decision problem, the **options are the hotels**. Each hotel has some parameters that characterize it including: 1) area it is in - whether it is near Nicosia, Larnaca, Limasol, Paphos or Troodos and 2) its facilities - whether the hotel has a pool, a playground, or a conference room and whether it is crowded. To solve this problem, existing parametric spaces that can be used for this problem are identified. If such parametric spaces did not exist, then they were built. Parametric spaces should be complete and able to be updated if there is an area that does not cover. For this example, the parametric space that was built was used as it is shown in Figure 30 in Appendix B that covers the most important factors for our simple implementation application. This space was built studying online hotel booking web-cites such as trip advisor[2] .

[2] https://www.tripadvisor.com/

Users also need to be parameterized by their needs and preferences to differentiate them from each other. This includes information such as the area that they are interested in, Nicosia, Larnaca, Limasol, Paphos, or Troodos, or if they are traveling for business or pleasure. It is also includes who they travel with, whether they are alone, with friends or family, and if they have children. Hotels are parameterized based on ontology parameters and pragmatic parameters.

Definition 14 (Ontological class) Let $\mathcal{P}_j, \forall j$ be ontological hierarchies for a hotel. This class categorizes the features that a hotel offers.

Ontological Classes: Examples are $\mathcal{P}_1$ = Sports Facilities, $\mathcal{P}_2$ = Children Care $\mathcal{P}_3$ = Nightlife, and $\mathcal{P}_4$ = Food.

Definition 15 (Pragmatic class) Let $\mathcal{P}_p$ be the pragmatic hierarchies for a hotel. This class categorizes what holds for one hotel at a specific time point.

Pragmatic Classes: Examples are availability, pool is not under construction, and hotel has special rates.

Hotels are then placed in these parametric spaces, and together they form the general parameter space of a hotel. Certain pragmatic classes should be assigned with a higher priority. For example, if a hotel does not have rooms available then no matter how good of a fit it is to a specific user, it should not be recommended.

Users express their needs and preferences to allow the labelling of each parameter of a hotel. If a parameter is labelled 1 then the user shows interest, if it is labelled -1 then the user shows no interest. This can be done through a short dialogue with the user; this is discussed below in the next subsection.

Definition 16 (Labelling function) Given a user, U, and $\mathcal{P}$, its general parameter space, sets $p_i^U(1)$, $p_i^U(0)$ and $p_i^U(-1)$ that partly characterize the user, where: $p_i^U(1)$ is a set of parameters that the user is interested in, $p_i^U(-1)$ is a set of parameters that the user is not interested in, and $p_i^U(0)$ are all parameters that are not included in the sets $p_i^U(1)$ and $p_i^U(-1)$.

These preferences are then prioritized, and if a parameter is preferred by the user then all hotels that have this parameter in their parameter space will be favorable. Arguments are then built in a way that can prioritize hotels that suit the user's preferences and needs. In this way, recommended hotels will make sense to each user. When an argument supports a hotel, different arguments may exist that attack it by an argument of higher priority for another hotel.

Definition 17 (Arguments) $arg(o; Sup(o); Par(o))$ is an argument that us built to support each hotel o. $Sup(o)$ is a set of timed fluents of properties of the hotel and $Par(o)$ is a set of conditions that hold or do not hold under a given time period.

For example, the following argument shows a hotel with the name hotel$_1$ that has a pool, a conference room, and at time 1, the pool of the hotel is under construction. $arg(hotel_1; []; [pool\ under\ construction\ at\ t = 1])$. Among the existing arguments an attack relation that depends on the time point one argument will qualify over the other.

Priorities and attacks amongst arguments can then be given by applying Definitions 9 and 10 in Section 3.2. To illustrate this, let $arg_1(hotel_1; []; [pool\ under\ construction\ at\ t = 1])$ and $arg_2(hotel_2; []; [pool\ under\ construction\ at\ t = 2, conference\ room\ under\ construction])$ be two arguments. For this example, $hotel_1$ attacks $hotel_2$ at time 2, regarding the parameter pool and $hotel_2$ attacks $hotel_1$ at time 1 for the same parameter. Similarly, $hotel_1$ attacks $hotel_2$ regarding the parameter conference room at all time points as $hotel_1$ has a conference room and $hotel_2$'s conference room is under construction.

The hotel that this system will recommend is the one that tries to satisfy all of the criteria. Since the admissible qualified hotels that are recommended depend on the user's needs and preferences, each user will receive different recommendations.

3.4 Implementation: 'Hotel for ME'

The parametric spaces for the hotels were studied. The following tree (see Figure 30 in Appendix B) is the result of this process. The application 'Hotel for ME' was implemented as shown in Appendix D. Here, how such a problem that is given in an abstract AF is realized and how it can be implemented in a theoretical level is shown. In this application, users try to find hotels in Cyprus by providing information that will define their needs and preferences. Based on this information, the user's parametric space is created, and with the existing parametric spaces of the hotels they are matched in the best way possible (this is dependent on the attack relation of the program) to the hotels. Hotel(s) fulfilling these predefined constrains will be suggested to the user as a recommendation.

First we create the profile of the user based on facts. Rules are then created in a form that $Gorgias$ can handle. Based on argumentation and a priority relation embedded in $Gorgias$, we prioritize potential recommendation. Based on priorities and the user's needs $Gorgias$ will then suggest the 'winning' option. The benefit of using $Gorgias$ is that all rules used to reach a conclusion are presented to the user which can then be reevaluated if the active user desires.

To empirical test our approach we have presented a 'benchmark' application. The narrative for this application is as follows: There are 500 hotels throughout Cyprus (see fact $f1\_o(H)$[3]). Hotels $\{1, \ldots, 100\}$ are in Nicosia, hotels $\{101, \ldots, 200\}$ are in Larnaca, hotels $\{201, \ldots, 300\}$

[3]Referring rules and facts can be found in Appendix D

are in Limasol, hotels $\{301, \ldots, 400\}$ are in Paphos, and hotels $\{401, \ldots, 500\}$ are in Troodos (see facts $f5\_a(A)$, $f6\_n(H, A), \ldots, f10\_n(H, A)$), and these are the available **options**.

For each time point or time period, the state of the world must be known. For this system, twelve time points (see fact $f2\_t(T)$) have been introduced, where each one represents a month of the year. For example, time 1 is January and time 8 is August. At different time points or time periods, hotels parametric spaces may change. For example, one hotel may be expensive throughout the entire year, but if it offers special rates then it becomes less expensive (see rule $r1\_s(H, T)$). Each area has a different cool season; for example, during the winter, the destination Troodos is preferred whereas during spring it is not. Of course this is also relevant to the state of the hotel (see rule $r2\_cs(H, T)$) as each hotel is crowded at different time points (see facts $r3\_c(H, T)$ and $r4\_c(H, T)$). In addition, some hotels have pools (see fact $f11\_p(H, T)$) but during the winters months, the pools at Troodos, where the water in the pools is frozen are not usable. Thus during these months, these hotels are considered not to have pools (see fact $f12\_p(H, T)$), and if the pool is under construction then the hotel is considered to be without a pool until the pool is fixed. The conference rooms and the playground are similar. Some hotels may have conference rooms or a playground (see rules $f11\_cr(H, T)$ and $r13\_p(H, T)$), but if the conference room or the playground is under maintenance then the hotel is not consider to have these facilities. In addition, as mentioned above, in Troodos during the winter months no one can use the playgrounds since they are covered with snow (see rules $f12\_cr(H, T)$, $r14\_p(H, T)$ and $r15\_p(H, T)$). Finally, some hotels offer entertainment during the night (see rule $r16\_n(H, T, A)$), and among these hotels some have bad reputations (see rule $r17\_n(H, T, A)$).

By including time as part of this system, it can handle the dynamic changes and updates of each hotel separately and manage to maintain a good representation of the world. Each option can also be time dependent. **Ontological class** of parameters contains information of the following type:

whether the hotel offers special rates and at what time point, whether it is a cool season, the area of the hotel, whether the hotel has a pool and a playground, whether the hotel offers entertainment at night, and whether it has a conference room. It also contains the following information about the user: the reason for visiting; and whether they are traveling with family, alone, or with friends. **Pragmatic class** of parameters relates to the following information: the availability of the hotel, whether the hotel is crowded, whether the pool, the conference room, or the playground is under construction and whether the hotel has a bad reputation for its entertainment at night.

To test the system we have a preparation phase of a 'dialogue' with the user. The user is asked to provide relative and personal information, such as the time period, the area of interest, the reason for visiting, and who will travel with the user (if any exists). Through this question, the profile of the user is constructed. Using argumentation through the $Gorgias$ system we compute, recommend by deciding amongst a set of available hotels which ones best suit the user needs but also the working environment. We use the following questions to $Gorgias$:

Question 1: $prove([hotel\_for\_me(U, H, T, A, R, W)], Delta)$.

Question 2: $prove([hotel\_for\_me1(U, H, T, A, R, W, L1)], Delta)$.

Question 3: $prove([hotel\_for\_me2(U, H, T, A, R, W, DL1)], Delta)$.

Question 4: $prove([hotel\_for\_me3(U, H, T, A, R, W, DL1, L2)], Delta)$.

Question 5: $prove([hotel\_for\_me4(U, H, T, A, R, W, L1, L2)], Delta)$.

Question 6: $prove([hotel\_for\_me5(U, H, T, A, R, W, L1, L2, L3)], Delta)$.

where U = User and can take one of the following values [klelia, alexandros, john, christos, giorgos], H = Hotel and can be a hotel between values [1,500], T = Time point that takes the values [1,12] and we think of it as the months of a year where 1 represents January and 12 represents December, A = Area and can take one of the following values [nicosia, larnaca, limasol, paphos, troodos], R = Reason for visiting that is either business or pleasure, W = traveling with and can

take on of the following values [alone, family(children(N)), friends], where N takes the values [0,5]. Additionally, users can specify what they like, L1 and L2, and what they do not like DL1 and these can be one of these parameters: special_rates(H,T), cool_season(H,T), crowded(H,T), pool(H,T), play_ground(H,T), good_nightlife(H,T,A). With that explained, question 2 is used when user specifically know one parameter the recommended hotels must have while question 3 is used when user can specify three parameters that the recommended hotels must have. Similarly for all other questions.

For the purpose of the experiment an agent was constructed that, under the conditions of the world, is able to recommend the 'best' option based on the user's needs. There are five users: $U = \{$Klelia, Alexandros, John, Christos, Giorgos$\}$. To understand what the users' needs and preferences are we ask the following questions:

1) Who is the user?

2) Reason of your visit? (Business or Pleasure)

3) Who do you travel with? (alone, family(children(N)),friends)

4) Are there any specific parameters that a hotel must have (in order to be considered as an option)?

5) Is there more than one parameter that hotel must have?

6) Are there any specific parameters that a hotel must not have?

Needs: n_1 = area, n_2 = special rates, n_3 = cool season, n_4 = crowded, n_5 = pool, n_6 = conference room, n_7 = playground.

To find the parameters that the user is interested in, questions 1,2,5, and 6 are used, and to find which parameters the user is not interested in, question 3 is used. Question 4 allows the users to specify one parameter that they like and one that they do not like.

Based on the user's question, arguments are built for or against an option. Options to be qualified must first satisfy the Sup class; they must hold in the snapshot of the world for this option

to be enabled. For example, for question 1, all hotels that satisfy at least one of the parameters that are specified by the user will be qualified, and question 2 allows higher priority in the options that the parameter L1 specifies. Question 5 allows higher priority in two parameters, L1 and L2, that users show interest in, while question 6 allows higher priority in three parameters that are specified by the user (L1, L2, and L3). Question 3 allows higher priority in a parameter that the user specifies that he or she does not like (DL1), whereas question 4 allows higher priority in one parameter that the user does not like (DL1), and in one that the user likes (L2).

The user's preferences are turned into priorities over arguments for the different hotels depending on their parameters as formulated in general is Section 3.3.

This implementation has two main variables: the users personal interest and time based on a single user. This is so that we can examine how this system can adapt to personal interest and dynamically adjust to the working environments. Examples are shown below, where under the same time point or time period, this system may suggest different hotels for different users since they have different needs that are provided as an input to the system (see Listing 3.5); for a single user, the system may recommend different hotels at different times. This happens because at different time points, different constraints hold that the world is changing (see Listing 3.6).

Listing 3.5: Test 1: Different Users

```
% User klelia
?- findall(H, prove([hotel_for_me(klelia,H,5,nicosia,business,
family(children(3)))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [1, 2, 3, 4, 5, 8, 9, 10, 11|...],
Number_of_solutions = 33.
```

```
% User giorgos
?- findall(H, prove([hotel_for_me(giorgos,H,1,paphos,pleasure,
family(children(1)))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [301, 302, 303, 304, 305, 306, 307, 308, 309|...],
Number_of_solutions = 45.
```

Listing 3.6: Test 2: Single User Different Time Points

```
% At time 1
?- findall(H, prove([hotel_for_me(klelia,H,1,larnaca,pleasure,
family(children(0)))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [101, 102, 147, 148, 149, 156, 157, 158, 159|...],
Number_of_solutions = 12.

% At time 8
?- findall(H, prove([hotel_for_me(klelia,H,8,larnaca,pleasure,
family(children(0)))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [101, 102, 103, 104, 147, 148, 149, 156, 157|...],
Number_of_solutions = 14.
```

To test the approach we have also experiment with different attack relations by changing the priority among arguments, which will result in new recommendations. As we have seen from Listing 3.5 and Listing 3.6, *Gorgias* provides a sufficient recommendation, and more solutions can be requested. Different priority over the parametric arguments resulting in the fact that the attack relations of *Gorgias* changes are shown below in Listing 3.7:

Listing 3.7: Test 3: Different Priority Over the Parametric Arguments

```
% Question 2

?- findall(H, prove([hotel_for_me1(alexandros,H,T,nicosia,pleasure,
alone,pool(H,T))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [7, 7, 7, 7, 7, 7, 7, 7, 7|...],
Number_of_solutions = 203.


% Question 3

?- findall(H, prove([hotel_for_me2(alexandros,H,T,nicosia,pleasure,
alone, play_ground(H,T))],Delta),Bag),
length(Bag,Number_of_solutions).
Bag = [24, 24, 25, 25, 26, 26, 28, 28, 29|...],
Number_of_solutions = 16.


% Question 4

?- findall(H, prove([hotel_for_me3(alexandros,H,T,nicosia,pleasure,
alone,play_ground(H,T),pool(H,T))],Delta),Bag),
length(Bag,Number_of_solutions).
Bag = [24, 24, 25, 25, 26, 26, 28, 28, 29|...],
Number_of_solutions = 16.


% Question 5

?- findall(H, prove([hotel_for_me4(alexandros,H,T,nicosia,pleasure,
alone,pool(H,T),good_nightlife(H,T,nicosia))],Delta),Bag),
```

```
length(Bag, Number_of_solutions).
Bag = [7, 7, 7, 7, 7, 7, 7, 7, 7|...],
Number_of_solutions = 203.

% Question 6
?-findall(H, prove([hotel_for_me5(alexandros,H,T, nicosia, pleasure,
alone, pool(H,T), good_nightlife(H,T, nicosia), special_rates(H,T))],
Delta),Bag), length(Bag, Number_of_solutions).
Bag = [7, 7, 8, 8, 9, 9, 10, 10, 31|...],
Number_of_solutions = 48.
```

As shown in Listing 3.7, question 2, gives priority to the need for a pool, provides 203 answers, and question 5, which allows user to pick two needs (for this example pool and nightlife) also provides 203 answers. Therefore, all 203 answers from question 2 have a good nightlife. Question 6 is more focused with 48 answers, as users pick three of their most favorite needs (for this example special rates are also added). Questions 3 and 4 allow user to say which need they do not want (for this example playground) and the latter question allows user to specify one need that they like the most (for this example a pool). Both question 3 and 4 produce 16 solutions. Adding negative information to the system significantly reduces the number of solutions.

3.4.1 Evaluation

To evaluate this approach, an experimentation process is used to study i) how well it adapts when the world changes and ii) how well it focuses with more information from the user. Here the

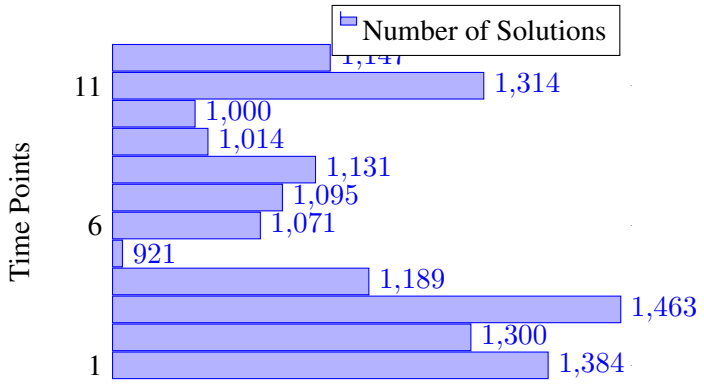


Figure 6: Fixed User Dynamic World

information is categorized in different ways: a) how much information, b) whether the information is prioritized or not and c) whether the information is positive as well as negative.

Evaluation 1 - Fixed user dynamic world: The program is first run to show that at different time points, which indicates that the world is different, the number of solutions that are produced by the system vary. To accomplish this result, the user, the type of question, the type of attack, and finally the information provided are fixed. For this example, question $prove([hotel\_for\_me(alexandros, H, T, A, R, W)], Delta)$ is used and all solutions are counted. The only parameters that vary are time and number of solutions produced, which are depicted on Figure 6. More tests have been produced, where the priority may vary and these results can be found in Appendix E.

Question 1: Matching parameters at a general level when time changes (see Listing E.1).

Question 2: Allowing higher priority for one parameter that the user likes (see Listing E.2).

Question 3: Allowing higher priority for one parameter that the user does not like (see Listing E.3).

Question 4: Allowing higher priority for one parameter that the user does not like and in one parameter that the user likes (see Listing E.4).

Question 5: Allowing higher priority for two parameters that the user likes (see Listing E.5).

Question 6: Allowing higher priority for three parameters that the user likes (see Listing E.6).

Questions 1,2,5, and 6 are based on one of the priorities and have attack relations that were created, as they do not accept negative information, whereas questions 3 and 4 concern a different attack relation that allows negative information. Specifically, question 3 only allows negative information and question 4 is a combination of these two attack relation that allows both positive and negative information from the user. Question 1, which embodies the first attacking relation, assigns the same priority to all of the parameters. Our second attack relation is performed through questions 2, 5, and 6, where the user assigns higher priority to certain predefined parameters. The parameters that are assigned to have a higher priority amongst the remaining parameters are then assigned to have the same priority amongst them. Finally, the questions 3 and 4 are the third attacking relations, where a user can assign parameters that they do not like.

Evaluation 2 - Many users, fixed world: The question was fixed to a specific time point where the world was frozen to be the same for any user. For example consider time 5 and users to seeking a hotel in Cyprus.

The **first user**, $user_1$ = Klelia, is traveling in Nicosia for pleasure with her family, including three children, and wants a hotel with a playground. The question that is used here is question 1: $prove([hotel\_for\_me1(klelia, H, 5, nicosia, pleasure, family(children(3)),$ $play\_ground(H, 5))], Delta)$, and all of the recommendations are counted and listed. The 17 recommendations for Klelia are: $\{1, 2, 3, 4, 5, 8, 9, 10, 32, 33, 34, 35, 36, 37, 38, 39, 40\}$.

The **second user** $user_2$ = Christos is traveling alone in Larnaca for pleasure and wants a hotel with a good nightlife. The question used here is question 1:

$prove([hotel\_for\_me1(christos, H, 5, larnaca, pleasure, alone,$

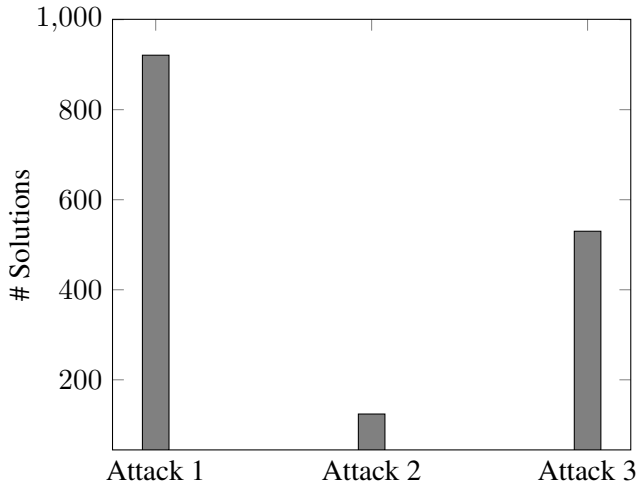


Figure 7: Fixed User Fixed World

$good\_nightlife(H, 5, larnaca))], Delta)$ and all of the recommendations were counter and listed. The five recommendations for Christos are: $\{101, 102, 103, 104, 105\}$.

When questions are different the number of solutions will differ as well as the recommended hotels. For this evaluation method a bar chart was not necessary.

Evaluation 3 - Fixed user, fixed world, different attack relations: For this, we fixed the user and run question 1, which embodies the first attacking relation where it assigns the same priority to all of the parameters. For this test, the way that the systems behave at all time points was tracked and for this reason, many graphs (see Appendix E) were created based on the type of question that was asked, which can then be compared. In Figure 7, a summary of these results is presented. The fixed time = 5, the 'imaginary' x-axis represents the attacking relation, and the 'imaginary' y-axis represents the number of solutions. These results were taken from Figure 37, Figure 39, and Figure 41 respectively. Attack 2 that allows negative information gives more focused results.

Evaluation 4 - Fixed user, fixed world, more information from user: In this test, the user was fixed and all questions were run in order to compare questions 1,2,5, and 6 between them and

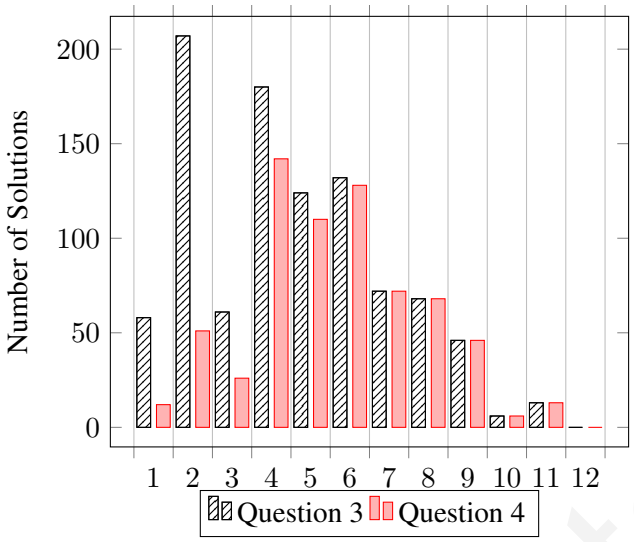


Figure 8: Fixed User Fixed World More Information 1

to compare question 3 with question 4. Again, all time points were compared and this is the reason that all results are shown in Appendix E and only the comparison is discussed here.

The demonstrated tests shows us that by allowing higher priority to more than one parameter, the solutions are more focused and in many cases no solution is reached. As the user provides more information to the system, available solutions are narrowed down. Furthermore, allowing a specific profile to the user is incorrect. For the application 'Hotel for ME', question 4 or 5 seems to be more appropriate. The bar chars that are presented in this section are a summary of the bar charts that are presented in Appendix E. The number of solutions may seem high, but the questions that were asked here are general. The area, the reason for visiting, and who the user travels with are left as variables and can be instantiated with any parameter that can be instantiated with them.

We now present an example where the user Alexandros specifies how parameters H = hotel, A = area, R = reason for visiting, and W = traveling with will be instantiated when the question is asked. The program was run at different time points to show that the application 'Hotel for ME' is time dependent and to understand how fast solutions are narrowed down in a realistic setting of

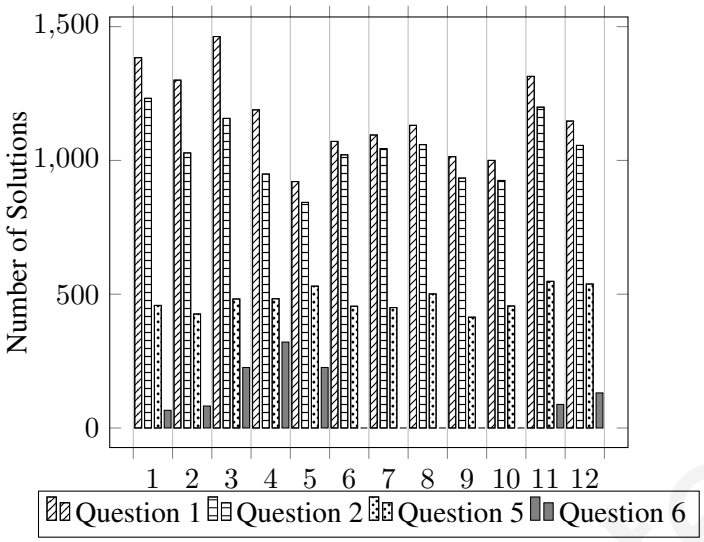


Figure 9: Fixed User Fixed World More Information 2

a user providing incrementally more information to the system. In this way, a good understanding is obtained of how much information is necessary and whether negative information can focus the solutions.

Question 1: In this general question, our user Alexandros wants to find a hotel in Nicosia at time 5. He travels alone for pleasure. This query is shown in Listing 3.8.

Listing 3.8: Time 5: Realistic Example for Question 1

```
?- findall(H, prove([hotel_for_me(alexandros,H,5,nicosia,
pleasure,alone)],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [7, 7, 8, 8, 9, 9, 10, 10, 24|...],
Number_of_solutions = 36.
```

If the user now provides more information and specifically asks for hotels with similar characteristics as before, but also with pool at time 5, the question in Listing 3.9 is provided. The number of solutions did not change from question 1 to question 2. Therefore, all of the available hotels at time 5 that where recommended in question 1 have a pool.

Listing 3.9: Time 5: Realistic Example for Question 2

```
?- findall(H, prove([hotel_for_me1(alexandros,H,5,nicosia,pleasure,
alone,pool(H,5))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [7, 7, 8, 8, 9, 9, 10, 10, 24|...],
Number_of_solutions = 36.
```

This knowledge is extended, as the user also requests hotels with similar facilities to question 2, but with special rates at time 5. Listing 3.10 depicts these results. Available solutions are narrowed down as expected; from 36 hotel recommendations, 28 were left. If the user had asked for a good nightlife instead of a special rates then the solutions would have not have been narrowed down, as indicated in Listing 3.11. This is surprising as one would expect the number of solutions to be reduced as a user provides more information. Instead, the user receives the same solutions from question 1 and question 5. Asking for a hotel with a pool is common in our database of hotels and this is the reason that the number of recommendations remain the same. As expected, running the question with the user seeking a hotel with a pool, special rates, and a good nightlife at time 5 will generate 28 solutions. These solutions are the same as those shown in Listing 3.11.

Listing 3.10: Time 5: Realistic Example 1 for Question 5

```
?- findall(H, prove([hotel_for_me4(alexandros,H,5,nicosia,pleasure,
alone,pool(H,5),special_rates(H,5))],Delta),Bag),
length(Bag,Number_of_solutions).
Bag = [7, 7, 8, 8, 9, 9, 10, 10, 31|...],
Number_of_solutions = 28.
```

Listing 3.11: Time 5: Realistic Example 2 for Question 5

```
?- findall(H, prove([hotel_for_me4(alexandros,H,5,nicosia,pleasure,
alone,pool(H,5),good_nightlife(H,5,nicosia))],Delta),Bag),
length(Bag, Number_of_solutions).
Bag = [7, 7, 8, 8, 9, 9, 10, 10, 24|...],
Number_of_solutions = 36.
```

Finally, the attack relation is run, which allows negative information, through questions 3 and 4. If a user asks for a hotel without playground, eight results are obtained (see Listing 3.12), a question is asked with negative information as before and with one parameter that the user wants, such as good nightlife, pool, or special rate, eight eight, and two recommendations are obtained, respectively. Therefore, as positive parameters are more likely to be offered by many hotels, it is beneficial to obtain negative information.

Another way for narrowing down the recommendations from the system is to guide the information that the user provides. As demonstrated, many hotels have similar parameters, and these parameters may differ wrt specific details. A query can be created that will be dynamic, in the sense that the program can be run as the information is provided by the user. Given the last example where the user does not want hotels with playgrounds if the user provides one parameter that they prefer, as demonstrated previously, the recommendations will be the same whether the user asks for a good nightlife or a pool. Since this information in known beforehand, it can be uses as an advantage. Since the solutions will not be more focused if such information is provided, the user can be asked if they want special rates instead. If the answer is yes then two solutions remain, as shown above. However, if user says no, this information can be used to narrow down the solutions because instead of having one negative piece of information, there are two negative pieces of information.

Listing 3.12: Time 5: Realistic Example for Question 3

```
?- findall(H, prove([hotel_for_me2(alexandros,H,5,nicosia,pleasure,
alone,play_ground(H,5))],Delta),Bag),
length(Bag, Number_of_solutions).
Bag = [24, 25, 26, 28, 29, 30, 31, 31],
Number_of_solutions = 8.
```

In a case where this system does not provide any recommendations a meta-level, a query can be applied that can inform the user that no hotels are found and ask the user to alter the information that has been provided. Other hotels can be recommended, even if they do not meet all of the criteria that is provided by the user. In this case, what criteria each recommendation has can be clearly indicated, leaving the user to make decisions from there.

Until this point, random data for the hotel database has been used. Our Tables 3 and 4 may show that the application recommends many hotels, but after investigating these results the recommended solutions do not differ, as the recommended hotels have similar parameter spaces. We have also prepared another hotel database for the application 'Hotel for ME' (see Appendix D) to accept time 0; at this time point, arithmetic progression was added to indicate which hotels offer special rates $(1, 3, 5, \ldots)$, a cool season $(2, 5, 8, \ldots)$, pool $(5, 9, 13, \ldots)$, have a conference room $(2, 7, 12, \ldots)$, and have a playground $(4, 7, 10, \ldots)$. The program ran again at time 0 and the results in Listing 3.13 and the Bar Chart in Figure 10 were obtained. Here, as more information is provided, the number of solutions becomes more focused, and at the end, only one recommendation remains.

Listing 3.13: Time 0: Discrete Hotels

```
% At Time 0
```

Time	Questions				% of solution loss		
	Q1	Q2	Q5	Q6	Q1 to Q2	Q2 to Q5	Q5 to Q6
1	1384	1232	458	66	11%	63%	86%
2	1300	1028	426	82	21%	59%	81%
3	1463	1157	482	226	21%	58%	53%
4	1189	949	483	321	20%	49%	34%
5	921	843	530	226	8%	37%	57%
6	1071	1021	455	0	5%	55%	100%
7	1095	1044	450	0	5%	57%	100%
8	1131	1059	501	0	6%	53%	100%
9	1014	934	414	0	8%	56%	100%
10	1000	925	456	0	8%	51%	100%
11	1314	1199	548	88	9%	54%	84%
12	1147	1056	538	131	8%	49%	76%
	Average				11%	53%	81%

Table 3: Number of Solutions at Different Time Points 1

```
?- findall(H, prove([hotel_for_me(alexandros,H,0,nicosia,business,
family(children(4)))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [1, 3, 5, 7, 9, 11, 13, 15, 17|...],
Number_of_solutions = 50.
```

```
?- findall(H, prove([hotel_for_me1(alexandros,H,0,nicosia,business,
```

Time	Questions		% of solution loss
	Q3	Q4	Q3 to Q4
1	58	12	79%
2	207	51	75%
3	61	26	57%
4	180	142	21%
5	124	110	11%
6	132	128	3%
7	72	72	0%
8	68	68	0%
9	46	46	0%
10	6	6	0%
11	13	13	0%
12	0	0	0%
	Average		21%

Table 4: Number of Solutions at Different Time Points 2

```
family(children(4)),pool(H,0))],Delta),Bag),
length(Bag,Number_of_solutions).
Bag = [5, 9, 13, 17, 21, 25, 29, 37, 41|...],
Number_of_solutions = 21.

?-findall(H,prove([hotel_for_me4(alexandros,H,0,nicosia,business,
```

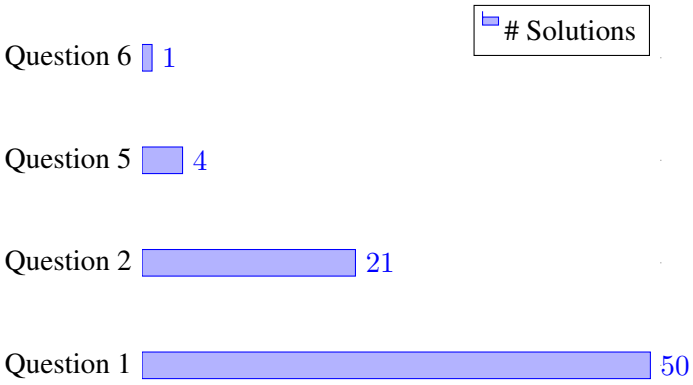


Figure 10: Discrete Hotels at Time 0

```
family(children(4)),pool(H,0),play_ground(H,0))],Delta),Bag),
length(Bag,Number_of_solutions).
Bag = [13, 37, 61, 85],
Number_of_solutions = 4.

?-findall(H,prove([hotel_for_me5(alexandros,H,0,nicosia,business,
family(children(4)),pool(H,0),play_ground(H,0),
conference_room(H,0))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [37],
Number_of_solutions = 1.
```

3.5 Summary

In this Chapter we show how real-life problems such as personalized and adaptive hotel booking can be formalized. We first formalize the general problem in time based preference argumentation. Then we study specifically an application, 'Hotel for ME', and show how it operates in a

time-based AF for decision making. Under different time points, the current state of the world, and the user's needs our system is able to adapt and make decisions that are relevant to the user. Through evaluations, we show that our system qualifies options based on the user, the pragmatic considerations, and the user's preferences. Argumentation techniques and predefined priority relations can be applied to qualify one option over another.

Chapter 4

Adaptation Over Time

"Do not fear mistakes. There are none."

Miles Davis

4.1 Introduction

The need to have a dynamic system is more obvious by what has already been discussed in previous Chapters. As the world changes and the user's needs as time passes by may change, systems that are able to keep track of these changes and are able to adapt dynamically are needed. Additionally, arguments may not cease to exist but their strength may be altered at different time points, thus, new systems should be able to prioritized arguments and attacks on arguments. This is what we will discuss in this Chapter.

We have created in an abstract level a system based on argumentation, that keeps track of the world and based on different time points, different values are then given to each option. In this way changes over time can now be tracked and under the user needs, and the options parameter space one option will be stronger over another option under a given time period. Adaptation over

time is handled through argumentation. Therefore, we will use argumentation on decision making but also for tracking decisions over time. Based on the 'Hotel for ME' example, available options can then be prioritized and narrowed down.

Specifically, we have created an argumentation based formulation of the action language $\mathcal{E}$ for RAC, extending an existing based argumentation based formulation of $\mathcal{E}$ to accommodate arguments based on property observations, and arguments for backwards persistence. Given a time line, we introduce priorities on arguments and observations to manage and to always have a good representation of the world.

RAC has been an important problem in the development of AI. Starting with the early and foundational work of the Situation and Event Calculi [58, 51] there has been a wide interest in understanding and solving the central problems that underly RAC, namely the frame, ramification and qualification problems leading to several 'action languages' such as the $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{E}$ languages [33, 57, 48] and the fluent calculus [79].

The relatively recent realization that non-monotonic reasoning can be addressed using argumentation (e.g. [45, 24, 13, 10]) has led to the study of RAC through argumentation in works such as [49, 82]. Hence given some narrative information we can use argumentation to capture temporal projection from this and general knowledge about the causal laws of our problem domain. As shown in [49], where the language $\mathcal{E}$ ([48]) for RAC was formalized in terms of argumentation, default persistence over time can be captured by assigning higher priority to arguments that are based on later events over arguments based on earlier events.

We extend this argumentation based formulation of language $\mathcal{E}$ by introducing arguments based on property observations that one typically finds in any given narrative. We will also introduce in the framework new arguments for backward persistence. This will allow us to recover and also extend language $\mathcal{E}$, giving a semantic meaning to domains that cannot be interpreted

under $\mathcal{E}$. With this form of backward persistence the extended interpretation of the language $\mathcal{E}$ comes closer to the spirit of the original Event Calculus (EC) [51] which also includes notions for backward temporal conclusions. Our work can thus also be seen as a way of increasing the expressiveness of the original EC by allowing for example positive but also negative observations.

We will be mainly concerned with domains that language $\mathcal{E}$ cannot give models and exogenous qualification is required. From a purely technical point of view we extend the argumentation formulation of language $\mathcal{E}$ giving a semantics to domains where $\mathcal{E}$ fails to do so. From a conceptual point of view we aim to address the qualification problem in RAC (see e.g. [78]) in a natural and modular way. This will rely on capturing a relative strength between arguments where, for example, arguments based on observations are stronger than other types of arguments.

RAC is an area that is concerned with the study of how (some of) the properties of our problem domain, normally called *Fluents*, change when new information is acquired such as the occurrence of *Actions* and how this view of the problem world is affected by the explicit observation of some properties holding or not at a particular stage (or time) in the flow of change. The change is governed by domain specific *causal laws* that describe how the particular properties (or fluents) of interest are generated or terminated by actions and the domain independent principle of *frame inertia* that properties do not change but rather persist, unless some action occurrence causes them to change via some causal law. The main aim of the reasoning is then to maintain, normally under the case of incomplete information, an accurate view of the problem world as things occur and/or are observed with the passage of time given in a specific *narrative* of interest.

To illustrate the types of problems that will concern us and the general approach that we will follow let us consider a standard example in RAC, that has the name the 'Car Park Domain' [50]. This domain when expressed in the language $\mathcal{E}$ contains one action constant $ParkingCar$ and one property fluent $CarInParkingSpace$ together with the causal law that 'parking the car causes

the car to be in the parking space'. This causal law together with the specific narrative information that we park the car at time 4 and that later at time 8 we observe that the car is not where it was parked, is represented by the following *domain description*:

$ParkingCar$ initiates $CarInParkingSpace$ (Δ_1)

$ParkingCar$ happens-at 4 (Δ_2)

$\neg CarInParkingSpace$ holds-at 8 (Δ_3)

For domains like this, where a fluent (e.g. $CarInParkingSpace$) changes its truth value without any known causal explanation, language $\mathcal{E}$ does not have a model. Our extended AF of the language $\mathcal{E}$ that includes arguments for observations and for backwards persistence will allow arguments for both truth values of the fluent within the time interval $(4, 8)$. Forwards persistence from the action $ParkingCar$ (Δ_2) that initiates $CarInParkingSpace$ for every time point $t > 4$ (Δ_1) comes into conflict with backwards persistence from the observation argument $\neg CarInParkingSpace$ (Δ_3). Allowing the same priority between conflicting forward persistence and backwards persistence arguments will give the natural interpretation of unknown value for the fluent $CarInParkingSpace$ for every $t \in (4, 8)$.

Pictorially, the way our AF behaves as we update the domain description for the parking domain example, is illustrated in Figure 11 where KB denotes the domain description, PC the action $ParkingCar$ and CPS the fluent $CarInParkingSpace$. The arrows in the pictures show the different arguments (for the fluent CSP and its negation) that exist in each case.

Suppose now that we extend the parking domain example by adding in the narrative another observation at time 12 of the form:

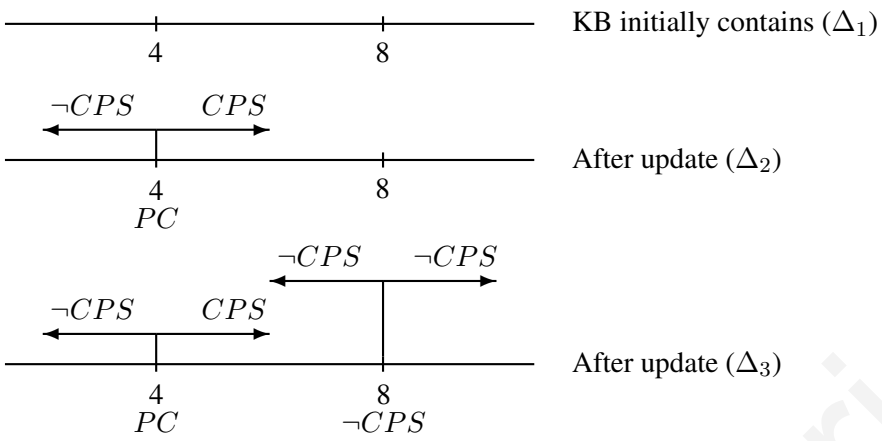


Figure 11: Parking Domain

$CarInParkingSpace$ holds at time 12 (Δ_4)

This will also result in an unknown value for the fluent $CarInParkingSpace$ in the interval $(8, 12)$. This arises analogously from conflicting arguments based on the two observation arguments (Δ_3) and (Δ_4). Forward persistence from $\neg CarInParkingSpace$ (Δ_3) is equally strong to backwards persistence from $CarInParkingSpace$ (Δ_4) and hence we can build equally good (or strong) arguments both for the truth and falsity of the fluent at any time point in this interval. Finally, we note that by giving higher priority to conflicting forward persistence over backwards persistence we will see that our framework can fully recover language $\mathcal{E}$ and hence in this respect it forms a conservative extension of $\mathcal{E}$.

The argumentation based formalization that we will develop in this Section is essentially a preference based (see e.g. [63]) realization of an abstract AF [24] under the admissibility semantics. Following the approach in [49] we will build the preference based AF in terms of logic programming rules with a priority relation over these rules. Our work then can be seen as an example where the general theory of argumentation, that has been extensively and widely developed

over the past two decades in AI (see e.g. [10, 67]), finds concrete application in addressing the foundational problems of temporal persistence and knowledge qualification.

This synthesis of ideas opens up possibilities of extending the application of argumentation from 'static problems' to variations of these where the problem environment is dynamically changing as new information becomes available. Using argumentation for reasoning about changes in the problem's world domain offers a principled way to manage the changes in the AF under which the application problem is expressed, thus extending the use of argumentation from static to 'dynamic problems'.

4.2 A Brief Review of Language $\mathcal{E}$

Language $\mathcal{E}$ [48] is inspired by the EC [51]. As most other action languages, language $\mathcal{E}$ has a set of *action constants,* $\mathcal{AC}$*,* that name actions that can occur and a set of *fluent constants,* $\mathcal{F}$*,* that name the different properties that can change in the problem domain. *Fluent literals* are either positive or negative fluent constants. It also has a set of *time points,* $\mathcal{T}$*,* s.t. for time points T_1 and T_2, the notation $T_1 < T_2$ indicates that time point T_1 comes before time point T_2 and $T_1 \neq T_2$.

The language uses three kinds of sentences or propositions defined as follows, where A is an action constant, F is a fluent constant, T is a time point, L is a fluent literal and C is a set of fluent literals:

1. **c-propositions** (c stands for causes), of the form 'A initiates F when C' or 'A terminates F when C',

2. **h-propositions** (h stands for happens) of the form 'A happens-at T',

3. **t-propositions** (t stands for time-point) of the form 'L holds-at T'.

The first type of sentence is used to express the causal laws of our domain (i.e., how action occurrences generate a property or stop a property from holding). The other two types of sentences are used to describe the particular narratives of interest. The first is used to state what actions have occurred while the latter type of sentences, namely the t-propositions, which will also be called **observations**, are used to represent properties of the world known to hold at particular time points.

A **domain description** D is a set of t-propositions, h-propositions and c-propositions. The c-propositions form the background **world knowledge** of the domain and the t-propositions and h-propositions the **narrative** of the domain.

The semantics of language $\mathcal{E}$ is given in terms of its **models**. These assign a truth value, $true\, or\, false$ to every fluent at every time point in the domain in a way that satisfies a set of constraints which reflect the axiom of **persistence** of the truth value of fluents. The following definitions (taken from [48]) formalize the notion of a model.

Definition 18 (Interpretation) Given a domain description D, an interpretation of D is a mapping

$$H : \mathcal{F} \times \mathcal{T} \mapsto \{true, false\}$$

where $\mathcal{F}$ is the set of fluent constants in D and $\mathcal{T}$ is the set of time points in $\mathcal{E}$. A fluent literal F (resp. $\neg F$) is satisfied at T by H iff $H(F, T) = true$ (resp. $H(F, T) = false$).

An interpretation may or may not support the initiation or termination of a fluent property depending on whether the preconditions in the causal laws of the domain description are satisfied or not under the interpretation.

Definition 19 (Initiation-termination point) Given an interpretation H of a domain description D, a time point T is an initiation (resp. termination) point of a fluent F in H in D when, D contains a combination of a c-proposition 'A initiates (resp. terminates) F when C' and an h-proposition 'A happens-at T', s.t. C at T is satisfied by the interpretation H.

An interpretation is then a model when it satisfies the property of *persistence*, namely that, within any time interval the truth value assigned by a model to any fluent remains the same or *persists*, changing from false to true (resp. from true to false) *only* at an initiation (resp. termination) point for that fluent that is supported by some action occurrence and associate causal law under which the action changes the fluent.

Definition 20 (Model) Let D be a domain description. An interpretation H is a **model** of D iff for every fluent constant F and time points T_1, T_2, T_3 the following properties hold:

1. If there is no initiation-point or termination-point T_2 for F in H in D s.t. $T_1 \leq T_2 < T_3$, then $H(F, T_1) = H(F, T_3)$.

2. If T_1 is an initiation-point for F in H in D, and there is no termination-point T_2 for F in H in D s.t. $T_1 \leq T_2 < T_3$, then $H(F, T_3) = true$.

3. If T_1 is a termination-point for F in H in D, and there is no initiation-point T_2 for F in H in D s.t. $T_1 \leq T_2 < T_3$, then $H(F, T_3) = false$.

4. For all t-propositions in D of the form 'F holds-at T_1', $H(F, T_1) = true$, and for all t-propositions in D of the form '$\neg F$ holds-at T_2', $H(F, T_2) = false$.

The property of persistence is expressed by the first three conditions with the second and third condition also capturing the semantic meaning of the causal laws expressed by the c-propositions. The fourth condition imposes the requirement that models must respect all the observations (t-propositions) of fluents literals contained in the narrative of the given domain description.

The above definition of a model is a slightly extended definition of the original definition in [48] that allows **non-deterministic models** when we have domains where there exist time points that can be simultaneously initiation and termination points for the same fluent. For this

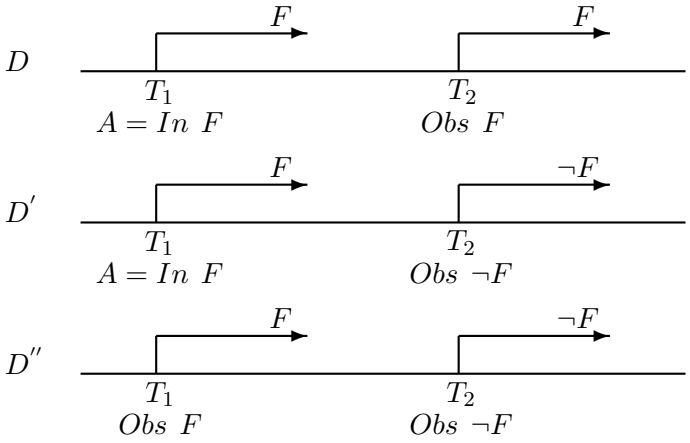


Figure 12: Example Domains

we have replaced, in the second and third condition of the original definition, the restriction on the intermediate time point, T_2, to be strictly greater than T_1 (i.e., the restriction $T_1 < T_2$, by $T_1 \leq T_2$).

Given the notion of a model, entailment and consistency of formulae of the form 'L holds-at T', where L is a fluent literal are then defined in the usual way. Hence 'L holds-at T' is **consistent in D** iff there exists a model, M, of D s.t. $M(F,T) = true$ when $L = F$ and $M(F,T) = false$ when $L = \neg F$. Similarly, 'L holds-at T' is **entailed by D** when the above holds for every model, M, of D.

The following simple examples illustrate the (above variant of) language $\mathcal{E}$. Let us consider the domain descriptions $D, D^{'}$ and $D^{''}$ shown in Figure 12, where A and B are an action constants and $T_1 < T_2$ are two time points.

Example 7 (Domain D)

A initiates F

A happens-at T_1

F holds-at T_2

In domain D we have an initiation point at time T_1 and at time T_2 we observe F. Models of language $\mathcal{E}$, for domain D, require F to be true for all $T > T_1$ whereas, for $T^{'} \leq T_1$ a model can assign F to be either true or false at all such time points. Suppose we replace the observation that F holds at T_2 with the observation that it does not hold, i.e. we have the domain $D^{'}$ given by:

Example 8 (Domain $D^{'}$)

A initiates F

A happens-at T_1

$\neg F$ holds-at T_2.

Then this domain, which is similar to the car parking domain discussed in the introduction of this Chapter, does not have any models. The generation of F holding onwards from T_1 cannot be reconciled with the observation of $\neg F$ at T_2. Similarly, for the domain $D^{''}$, given by:

Example 9 (Domain $D^{''}$)

F holds-at T_1

$\neg F$ holds-at T_2

language $\mathcal{E}$ is inconsistent and has no models. The persistence of F holding onwards from T_1 cannot be reconciled with the observation of $\neg F$ at T_2. Finally, consider the domain $D^{'''}$ given by:

Example 10 (Domain $D^{'''}$)

A initiates F

B terminates F

A happens-at T_1

B happens-at T_1

We have two possible models for times after time T_1. In one F holds true for all such time points and in the other F is false for all time points after time T_1. Hence if we extend this domain with either of the observations that F holds-at T_2 or $\neg F$ holds-at T_2 the extended domain remains consistent.

4.3 Argumentation Formulation

The language $\mathcal{E}$ has been reformulated in terms of argumentation [49] within the preference-based AF of logic programs with priorities, $LPwNF$, under its admissibility semantics [46, 22]. In this the information from t-propositions (observations) is not used in the argumentation process but rather these observations are imposed as a-posteriori constraints on the argumentation formulation.

We will extend this reformulation so that t-propositions are taken into account directly within the argumentation. To do so we will generalize the original formulation by introducing new arguments that are rooted or based on observations. In addition, we will use backward temporal persistence arguments as well as forward ones to allow us to address the qualification problem.

To translate a given domain description D into an argumentation program in $LPwNF$, all individual h- and c-propositions translations are included in the background theory as follows. We will consider time to be discrete scalar.

Definition 21 (Background theory for D) The background theory for D is the theory $B(D)$ given by:

- for all time points T and T' and action constants A:

  $B(D) \vdash T \leq T'$ iff $T \leq T'$,

  $B(D) \vdash T < T'$ iff $T < T'$,

$HappensAt(A,T) \in B(D)$ iff 'A happens-at T' is in D, and

$Observation(L,T) \in B(D)$ iff 'L holds-at T' is in D.

- for each c-proposition 'A initiates F when $\{L_1, \ldots L_n\}$' in D, $B(D)$ contains the rule

  $Initiation(F,t) \leftarrow HappensAt(A,t), \Lambda(L_1), \ldots, \Lambda(L_n)$, and

- for each c-proposition 'A terminates F when $\{L_1, \ldots L_n\}$' in D, $B(D)$ contains the rule

  $Termination(F,t) \leftarrow HappensAt(A,t), \Lambda(L_1), \ldots, \Lambda(L_n)$

  where $\Lambda(L_i) = HoldsAt(F_i,t)$ if $L_i = F_i$, and $\Lambda(L_i) = \neg HoldsAt(F_i,t)$ if $L_i = \neg F_i$, for any fluent constant F_i,

- $B(D)$ contains no other rules.

The above definition is essentially the same as in [49] with the exception that the given observations are also added in $B(D)$ and hence the whole narrative forms part of the background knowledge. Any such background knowledge is then extended by the following logic program rules and priorities between these to give the full **argumentation logic program with priorities** corresponding to a given domain description.

Definition 22 (Argumentation program of D) The argumentation program corresponding to a domain D is $AD \equiv (B(D), \mathcal{A}, <)$ where:

- $B(D)$, is the corresponding background knowledge of D.

- The set, $\mathcal{A}$, consists of the following **argument rules**. For all time points t_1, t_2 and t s.t. $t_1 < t < t_2$,

Persistence: [1]

$HoldsAt(f, t_2) \leftarrow HoldsAt(f, t)$ $PFP[f, t_2; t]$

$HoldsAt(f, t_1) \leftarrow HoldsAt(f, t)$ $PBP[f, t_1; t]$

$\neg HoldsAt(f, t_2) \leftarrow \neg HoldsAt(f, t)$ $NFP[f, t_2; t]$

$\neg HoldsAt(f, t_1) \leftarrow \neg HoldsAt(f, t)$ $NBP[f, t_1; t]$

Local Generation Arguments:

$HoldsAt(f, t+1) \leftarrow Initiation(f, t)$ $PG_F[f, t]$

$\neg HoldsAt(f, t) \leftarrow Initiation(f, t)$ $NG_B[f, t]$

$\neg HoldsAt(f, t+1) \leftarrow Termination(f, t)$ $NG_F[f, t]$

$HoldsAt(f, t) \leftarrow Termination(f, t)$ $PG_B[f, t]$

Local Observation Arguments:

$HoldsAt(f, t) \leftarrow Observation(f, t)$ $PO[f, t]$

$\neg HoldsAt(f, t) \leftarrow Observation(\neg f, t)$ $NO[f, t]$

Initial Assumptions:

$HoldsAt(f, 0)$ $PA[f, 0]$

$\neg HoldsAt(f, 0)$ $NA[f, 0]$

- The priority (or strength of argument) relation, $<$, between these arguments is given as follows. Let t, t^*, t_1 and t_2 be time points:

  Set 1: If $t_1 < t_2$

  $PFP[f, t^*; t_1] < NFP[f, t^*; t_2]$ and $NFP[f, t^*; t_1] < PFP[f, t^*; t_2]$,

[1] PFP (resp. PBP) stands for Positive Forward (resp. Backward) Persistence, NFP (resp. NBP) stands for Negative Forward (resp. Backward) Persistence, PG_F (resp. PG_B) stands for Positive Forward (resp. Backward) Generation, NG_F (resp. NG_B) stands for Negative Forward (resp. Backward) Generation, PO (resp. NO) stands for Positive (resp. Negative) Observation and finally PA (resp. NA) stands for Positive (resp. Negative) Assumption.

$PBP[f, t^*; t_2] < NBP[f, t^*; t_1]$ and $NBP[f, t^*; t_2] < PBP[f, t^*; t_1]$.

Set 2: If $t_1 \leq t_2$

$NFP[f, t_2 + 1; t_1] < PG_F[f, t_2]$ and $PFP[f, t_2 + 1; t_1] < NG_F[f, t_2]$.

Set 3: At 0,

$PA[f, 0] < NO[f, 0]$ and $NA[f, 0] < PO[f, 0]$.

Set 4: For any t,

$NG_B[f, t] < PO[f, t]$ and $PG_B[f, t] < NO[f, t]$.

$PG_F[f, t] < NO[f, t + 1]$ and $NG_F[f, t] < PO[f, t + 1]$.

Set 5: If $t_1 < t_2$

$NFP[f, t_2; t_1] < PO[f, t_2]$ and $PFP[f, t_2; t_1] < NO[f, t_2]$,

$NBP[f, t_1; t_2] < PO[f, t_1]$ and $PBP[f, t_1; t_2] < NO[f, t_1]$.

This definition introduces four different types of arguments and the relative strength between them. These types of arguments express in a natural way the defeasible conclusions that we can draw or support from the non-defeasible information in $B(D)$.

Let us explain the various parts of this definition. Consider first the various arguments rules.

- **Persistence Rules:** These rules express the possibility of fluent properties persisting forward and or backward in time. The first rule captures the forward persistence from a property holding at t to holding at a latter point t_2, while the second rule captures the backward

persistence from a property holding at t to holding at an earlier point t_1. The other two rules are similar but for the persistence of properties not holding.

- **Local Generation Arguments:** These are argument rules for the causation of a property by an event. The first rule says that the effect of an initiation point at time t starts at the very next time point, while the second rules says that from this initiation point we have an argument that its effect does not hold at the time of the event that generates the property. Analogously for the following two rules based on termination points.

- **Local Observation Arguments:** Observations give directly arguments for fluents holding or not at the time of observation.

- **Initial Assumptions:** Assumptions can only be introduced at time 0 and can be for a property holding or not holding.

Let us now comment on the **priority relation**. We first note that the priority relation, $<$, in an argumentation program is *independent* of the domain description, D. It is a universal relation that will us capture the general principles of persistence and qualification.

- **Set 1:** The first set of priorities in the above definition make forward persistence arguments that are based on a later time-point stronger than conflicting forward persistence arguments that are based on an earlier time-point. Analogously, backward persistence arguments that are based on earlier time points are stronger than conflicting backward persistence arguments that are based on later time-points.

- **Set 2:** This set of priorities expresses the fact that the generation of effects is stronger than opposing persistence.

- **Set 3 - Set 5:** The final three sets of priorities give, as expected, higher strength to observation arguments over any other type of conflicting arguments.

We note that included in the types of arguments are the two local generation arguments, $NG_B[f,t]$ and $PG_B[f,t]$, which are not usually found in formulations of many RAC frameworks, with a notable exception of the original EC. It is important though to stress that these arguments only support the possibility for the occurrence of events and their initiation or termination of effects to affect the past. The argumentation formulation allows us to capture this possibility in a weak form, by saying that there exists an argument for something holding in the past due to the occurrence of events. Then depending on the priority we give to this argument over other arguments, its effect on the semantics would vary. For example, as we will see below if we assign forward persistence arguments to be stronger than these local generation arguments (or more generally stronger than backwards persistence arguments) then these arguments do not have any effect on the semantics (i.e., no maximal admissible extension can contain these local generation arguments).

In the priority relation there is no priority between conflicting forward and backward arguments. Such priorities can be additionally set when we wish to impose further properties on the temporal reasoning. Finally, we note that the assumption arguments are not chosen to be weaker than the conflicting backwards persistence arguments to allow the freedom of incomplete initial states.

The semantics of these $LPwNF$ argumentation programs is given through the standard argumentation notion (see [24, 46]) of *maximally admissible* subsets of the given argumentation program, called **admissible extensions**, as follows.

Definition 23 (Background logic) The background monotonic logic is the tuple $(\mathcal{L}, \vdash)$, where the language $\mathcal{L}$ consists of all ground rules of the form $l_0 \leftarrow l_1, l_2, \ldots, l_n$ $(n \geq 0)$ that belong to $B(D)$ or $\mathcal{A}$ with each l_i a classical literal. l_0 is called the head of the rule and $l_1, l_2, \ldots, l_n$ is called the body of the rule. $\vdash$ is obtained by the repeated application of the modus ponens rule.

The above definition expresses formally the logical reasoning that allows the link of the background knowledge with the argument rules so that arguments supporting a fluent literal conclusion can be constructed. These arguments are grounded on the narrative given in the domain description D and translated in the corresponding background knowledge B(D).

Definition 24 (Argument) Let an argumentation program $(B(D), \mathcal{A}, <)$ be given. An argument Δ is a subset of argument rules from $\mathcal{A}$. An argument supports any conclusion derived under the backround logic from $B(D) \cup \Delta$.

The attacking relation is then defined amongst sets of argument rules that have conflicting conclusions by taking into account the priorities on the argument rules as given in Definition 22.

Definition 25 (Attack) Let G, W be two non-empty sets of argument rules. G attacks W iff there exists a literal l of the form $HoldsAt(f, t)$ or $\neg HoldsAt(f, t)$ and sets $G_1 \subseteq G$ and $W_1 \subseteq W$ s.t.:

(i) $B(D) \cup G_1 \vdash_{min} l$ [2] and $B(D) \cup W_1 \vdash_{min} \neg l$

(ii) if there exist $r' \in G_1, r \in W_1$ s.t. $r' < r$ then there exist $r' \in G_1, r \in W_1$ s.t. $r < r'$.

Hence the attacking relation is defined by lifting the priorities from the level of single argument rules to sets of these. For a set G of argument rules to attack another set W it must contain at least one rule of higher priority than a rule in the attacked set W or otherwise the set W must (also) not

[2] $B(D) \cup X \vdash_{min} l$ iff $B(D) \cup X \vdash l$ and there does not exist $X' \subset X$ s.t. $B(D) \cup X' \vdash l$

contain any rule of higher priority than some rule in the attacking set G. In this way we ensure that the attacking set is not 'weaker' than the attacked set as either it contains a stronger rule or the two sets are non-comparable. Note that if a set is inconsistent (i.e., it derives both a literal and its negation) then it attacks itself as the second condition in the definition of the attacking relation is trivially satisfied.

The following definition introduces some technical conditions on sets of argument rules that are more specific to the use of argumentation to the particular application of RAC.

Definition 26 (Closed, complete and compact) A set Δ of argument rules is closed if all the bodies of its rules are derived via $\vdash$ from the background theory $B(D)$ extended by Δ. Δ is complete if for any fluent f and time point t, $\Delta \vdash HoldsAt(f,t)$ or $\Delta \vdash \neg HoldsAt(f,t)$. Finally, Δ is compact if Δ is closed and does not contain a pair of argument rules, $PFP[f,t;\_]$ and $PBP[f,t;\_]$ or $NFP[f,t;\_]$ and $NBP[f,t;\_]$, for any fluent f and any time point t.

The condition of closeness is a technical condition to avoid considering unnecessarily argument sets that contain rules which cannot be used to support conclusions. Similarly, the compactness property imposes a uniqueness of support for the conclusions drawn through persistence arguments requiring that any such conclusion (timed fluent literal) is supported in a uni-directional way, i.e. either by a forwards persistence or by backwards persistence but not (redundantly) by both such arguments. These technical conditions are conditions that avoid redundancy and help to simplify the framework and the proofs of its formal properties, particularly the link with the model theory of language $\mathcal{E}$ when the two approaches coincide.

Finally, the property of completeness is a desired property for the semantics of the approach so that the admissible extensions, that will define the semantics, are able to decide for any fluent at any time point its status, i.e they can populate the whole time line with a decision on whether

any fluent holds or not. This condition is also motivated by the technical consideration of relating the semantics to the model theory semantics of the language $\mathcal{E}$.

Definition 27 (Admissibility) Let D be a domain description and $AD \equiv (B(D), \mathcal{A}, <)$ its corresponding argumentation program. Let S be a closed subset of $\mathcal{A}$. Then S is admissible iff:
(i) $B(D) \cup S$ does not derive a literal l and its complement $\neg l$ and
(ii) for any $S^{'} \subseteq \mathcal{A}$ if $S^{'}$ attacks S then S attacks $S^{'}$.

This is the standard definition of admissibility. A subset of arguments is admissible if it does not derive both $HoldsAt(f,t)$ and $\neg HoldsAt(f,t)$ for any fluent and time point and therefore it does not attack itself and it can counter-attack any subset of arguments that attacks it.

The semantics of domain descriptions is then given by putting these notions together to define admissible extensions of the corresponding argumentation program.

Definition 28 (Admissible extension) Let D be a domain description and $AD \equiv (B(D), \mathcal{A}, <)$ its corresponding argumentation program. Let S be a closed subset of $\mathcal{A}$. Then $\Delta = B(D) \cup S$ is an admissible extension of AD iff S is a complete and compact admissible set.

We note that alternatively we could have omitted the explicit requirement of completeness and replaced this with the maximality condition as we normally have with the preferred semantics of argumentation. Nevertheless, remaining within the spirit of RAC and as complete extensions are maximal it is natural to define the semantics only in terms of the complete admissible extensions provided that complete extensions exist, which as we will see in the next Section, is indeed the case. Complete admissible extensions then correspond more closely to the stable semantics of argumentation as they cannot be extended consistently (i.e., without self-conflict) by any other

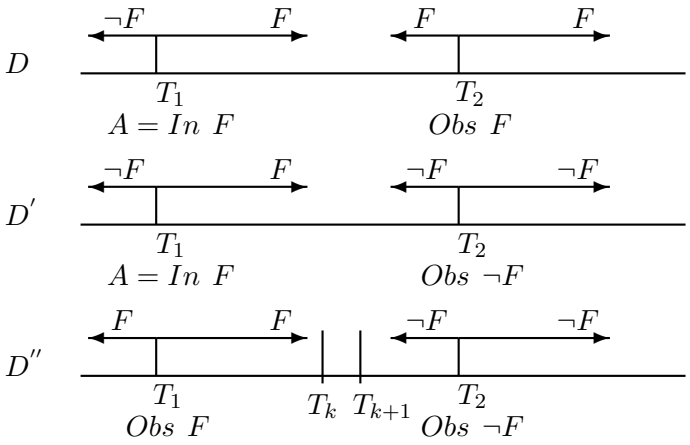


Figure 13: Example Domains and Arguments

subset of argument rules that support a conclusion that is not already supported by the complete extension.

Conclusions from an admissible extension are then drawn using the derivability relation, $\vdash$, given in Definition 23, with *credulous conclusions* those that can be derived from at least one admissible extension and *sceptical conclusions* as those that are derived in every admissible extension.

To illustrate the above definition of the argumentation semantics let us consider our earlier example domains D (Example 7), $D^{'}$ (Example 8) and $D^{''}$ (Example 9) within the AF defined above (Figure 13). In domain D, for all $T > T_1$ the strongest (and hence admissible) argument is for F to hold through a local generation argument at $T_1 + 1$ and then by forwards persistence arguments at any other time after T_1. Otherwise, from T_2 onwards we can use the observation argument at T_2 and forwards persistence arguments from this to support strongly F for times greater than T_2. For $T^{'} \leq T_1$ we can have admissible arguments for F or its negation $\neg F$ depending on the assumption we make at the initial time point. Note also that the backwards persistence arguments for F from T_2 to some time point equal to or before T_1 would not be stronger than the forwards persistence of

$\neg F$ starting from time 0 through an assumption argument for $\neg F$. Nor would it be stronger than the backwards negative generation argument for $\neg F$ based on the initiation point T_1 for F.

For the domain $D^{''}$ the strongest argument for all time points $T \geq T_2$ is the observation argument for $\neg F$ at T_2 combined with a forward persistence argument from T_2 to any such time point T. For times between T_1 and T_2 we have admissible arguments for either F or $\neg F$: at some time point T_k, $T_1 \leq T_k < T_2$, the fluent F changes from true to false at T_{k+1}. This indicates that the given narrative has some missing information within this time interval that would explain the change in F. Similar results hold for $D^{'}$ where also in this case there exists an admissible extension where $\neg F$ holds for all times T, s.t. $T_1 \leq T \leq T_2$. This captures the possibility that the generation of F at T_1 has failed through some exogenous qualification.

The examples that follow illustrate further the argumentation formulation. Let f be any fluent and t, T time points:

Example 11 Let D_1 be a domain with no h-propositions and t-propositions. The only possible admissible extensions is for f (resp. $\neg f$) to hold for all times t as there does not exist a stronger argument that attack the set $S = \{PA[f, 0], PFP[f, t; 0]\}$, for all $t > 0$. Possible attacks on S contain the argument $NA[f, 0]$ and S counterattacks it through $PA[f, 0]$. Admissible extensions in this example domain are the same as the (maximally) admissible extensions of language $\mathcal{E}$.

Example 12 Let D_2 be a domain with no t-propositions containing h-propositions so that it necessarily has two initiation points of f at t_1, t_2 s.t. $t_2 > t_1$. In this case, language $\mathcal{E}$ models require f to hold for all $T > t_1$. In our extended AF, f will hold for all $T > t_2$ in all admissible extensions. For time points $T^{'} \in (t_1, t_2]$ we can build an argument for f, $(\{PG_F[f, t_1], PFP[f, T^{'}; t_1 + 1]\})$ and an argument for its negation $\neg f$, $(\{NG_B[f, t_2], NBP[f, T^{'}; t_2]\})$ which are both admissible. Hence between the two initiation points we cannot conclude that f is entailed, like language $\mathcal{E}$,

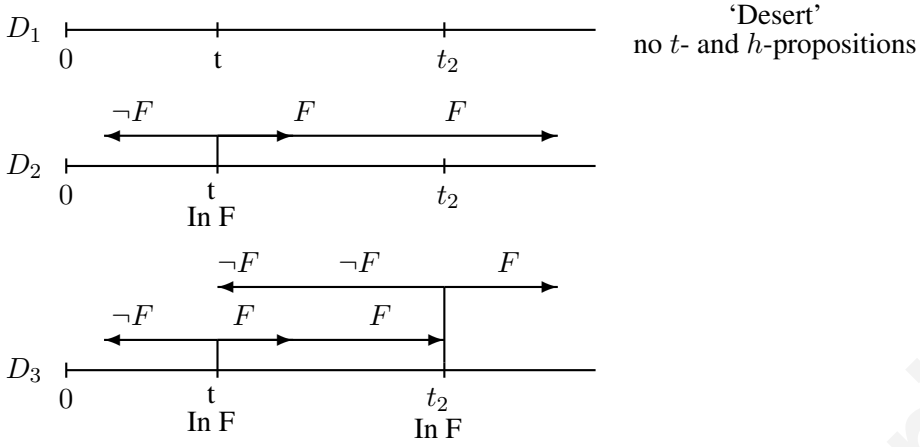


Figure 14: Examples

as f does not hold in all admissible extensions. We will see that our framework is closer to the original EC in this respect. Also we will see that if we wish to fully recover language $\mathcal{E}$ we could do this by assigning higher priority to forward persistence arguments over conflicting backwards persistence arguments.

Note that if we additionally have in the domain D_2 a termination point for f at time $T^{''} \in (t_1, t_2)$ then the second type of argument sets that derives $\neg f$ at points in between the two initiation points before $T^{''}$ would not be admissible anymore as it would be attacked by the stronger backward persistence argument for f from $T^{''}$, namely from $(\{PG_B[f, T^{''}], PBP[f, T^{'}; T^{''}]\})$. This can be understood as an indication that the given narrative in D_2 has some missing information between the time points t_1 and t_2 of the two initiation points.

4.4 Formal Results

In this Section we present a set of formal results that show the various properties that the argumentation formulation given above has wrt persistence and qualification, how this formulation

gives meaning to any domain description theory and how it relates to and extends the original model theoretic formulation of language $\mathcal{E}$.

Firstly, as expected, we show that admissible extensions are consistent with the observations in the given narrative. From now on we will consider only **point-wise consistent** domain descriptions (i.e., domain descriptions that do not contain any pair of t-propositions of the form 'f holds-at t' and'$\neg f$ holds-at t').

Proposition 1 Let D be a point-wise consistent domain description and E an admissible extension of the corresponding argumentation program $AD \equiv (B(D), \mathcal{A}, <)$. Then E is consistent with D, i.e. there does not exist a t-proposition, 'f holds-at t' (resp. '$\neg f$ holds-at t') in D s.t. $E \vdash \neg\text{HoldsAt}(f, t)$ (resp. $E \vdash \text{HoldsAt}(f, t)$).

Proof 1 see Appendix A.1.

This is a simple basic result that is needed for the main result of showing the existence of admissible extensions for any description domain and for linking admissible extensions with the models of language $\mathcal{E}$, whenever such models exist. Indeed, it corresponds to the fourth condition in the definition of a model of language $\mathcal{E}$ (see Definition 20).

The next result shows that the admissible extensions satisfy the important property of *persistence of derivation* of fluent properties along the time line. Informally, it says that if between two time points there is no information, either causal or observational in nature, that a fluent has changed value then an admissible extension will derive a constant value for the fluent in this time interval.

Lemma 2 Let D be a point-wise consistent domain description and E an admissible extension of D. Let f be a fluent and $t_n < t_m$ two time points s.t. there does not exist an initiation point

nor a termination point for the fluent f in E [3] at any time $t_1 \in [t_n, t_m)$ nor there exists an observation point for the fluent f in E at any time $t_2 \in (t_n, t_m)$. Then if $E \vdash$HoldsAt(f, t_n) and $E \vdash$HoldsAt(f, t_m) hold or (respectively, $E \vdash \neg$HoldsAt(f, t_n) and $E \vdash \neg$HoldsAt(f, t_m) hold) then $E \vdash$HoldsAt(f, t) (respectively, $E \vdash \neg$HoldsAt(f, t)) holds, for every $t \in [t_n, t_m]$.

Proof 2 see Appendix A.1.

Therefore the notion of admissibility in argumentation helps us to capture the requirement of persistence. Admissible extensions must have this property of persistence as otherwise they will have attacks that they cannot counter-attack. This plays an important role in showing the next theorem which gives the central result of the argumentation formulation, namely that any domain description D (with a countable narrative) is always consistent (i.e., there always exists an admissible extension of the corresponding argumentation program of D).

Theorem 1 Let D be a point-wise consistent domain description with time structure that of the natural numbers and a countable number of h-propositions and t-propositions. Then there exists an admissible extension E of the corresponding argumentation program $AD \equiv (B(D), \mathcal{A}, <)$.

Proof 3 see Appendix A.1.

Hence any domain description D is given a meaning under the above argumentation semantics. For example, as we have seen above, the example domains D' (Example 8) and D'' (Example 9) are consistent in our extended framework. These domains do not have any model in the language $\mathcal{E}$ or an admissible extension in the earlier argumentation based reformulation of $\mathcal{E}$ in [49].

The formal link of our argumentation formulation of the language $\mathcal{E}$ with its original model theoretic semantics is given by the following theorem.

[3] An initiation or termination point for a fluent f in an admissible extension E is defined as in Definition 19 where now the preconditions C of the c-proposition are satisfied at T in E when the corresponding $HoldsAT$ conclusions are derived by E.

Theorem 2 Let D be a point-wise consistent domain description with time structure that of the natural numbers and a countable number of h-propositions. Then for every language $\mathcal{E}$ model, M, of D there exists an admissible extension, E, of the corresponding argumentation program $AD \equiv (B(D), \mathcal{A}, <)$ s.t. E corresponds to M (i.e., $E \vdash HoldsAt(f,T)$ iff $M(f,T) = true$ and $E \vdash \neg$HoldsAt(f,T) iff $M(f,T) = false$).

Proof 4 see Appendix A.1.

This theorem shows that when language $\mathcal{E}$ models exist then our AF always has a corresponding admissible extension. Our formulation therefore forms a conservative extension of $\mathcal{E}$.

The next theorem provides an interpretation of how the extended argumentation formulation handles the qualification problem (see Appendix C.1) when it provides a semantics to any domain description. It shows that an admissible extension E of any domain D can be interpreted as a language $\mathcal{E}$ model of D or of D extended by some h-propositions of unknown events.

Theorem 3 Let D be a point-wise consistent domain description with a finite number of h-propositions and t-propositions. For every admissible extension E of D there exists a domain $D^{'}$ obtained from D by adding a (possibly empty) set of new h-propositions s.t. there exist a language $\mathcal{E}$ model, M, of $D^{'}$ that corresponds to E (i.e., $E \vdash$HoldsAt(f,t) (resp. $E \vdash \neg$HoldsAt(f,t)) iff $M(f,t) = true$ (resp. $M(f,t) = false$)).

Proof 5 see Appendix A.1.

For example consider the domain $D^{''}$ (Example 9) where we observe F at time T_1 and $\neg F$ at time T_2. Then given any time point $T_k \in (T_1, T_2)$ this domain has admissible extensions E where F is derived for any time point between T_1 up to T_k and $\neg F$ is derived for all time points after T_k up to T_2. The domain $D^{''}$ does not have any language $\mathcal{E}$ models. But by interpreting time T_k

as a termination point for F through an unknown (in D) new action that can terminate F we can 'explain' within $\mathcal{E}$ the meaning given to D'' by the above admissible extensions in the extended argumentation formulation of the language.

4.5 Qualification Extensions

In this Section we study in more detail how the argumentation semantics that we have provided for action theories addresses the qualification problem. In particular, we will examine how it helps us understand qualifications of action occurrences and how we can find, within the given domain description, explanations for exogenous qualifications.

The argumentation formulation allows us to identify exogenous qualifications imposed by the lack of information in the given narrative of the domain description.

Definition 29 Let D be a domain description and E an admissible extension of D s.t. there exist time points t_1, t_2 with $t_1 < t_2$ and a fluent F so that the following conditions hold:

- The t-proposition '$\neg F$ holds-at t_2' (respectively 'F holds-at t_2') belongs to D and there is no t-proposition 'F holds-at t' (respectively '$\neg F$ holds-at t') in D for any t s.t. $t_1 < t < t_2$.

- There exists an h-proposition 'A happens-at t_1' and a c-proposition 'A initiates F when C' (respectively 'A terminates F when C') in D s.t. t_1 is an initiation (respectively termination) point for F in E, and there is no other initiation (respectively termination) point t for F in E s.t. $t_1 < t < t_2$ [4] .

- There exists no termination point for F at t in E s.t. $t_1 \leq t < t_2$.

[4] For simplicity of presentation we are assuming that there can only be one generating (initiating or terminating) action of a given fluent at any time point.

Then E is called an **exogenous qualification extension on** F **at** $(t_1, t_2]$ of D. The exogenous qualification is said to be **based on the t-proposition** '$\neg F$ holds-at t_2' (respectively 'F holds-at t_2').

Furthermore, if there does not exist a time point $t \in (t_1, t_2)$ s.t. $E \vdash HoldsAt(F, t)$ (respectively $E \vdash \neg HoldsAt(F, t)$) then E is called an **exogenous qualification of the action occurrence of** A **at** t_1. Otherwise, E is an **exogenous qualification of the persistence of** F **in** $(t_1, t_2]$.

Qualification extensions are therefore separated into two types. In the first case the action occurrence of A at t_1 is exogenously qualified failing to generate its effect and consequently in the whole of the interval, $(t_1, t_2]$, $\neg HoldsAt(F, t)$ (respectively $HoldsAt(F, t)$) holds in E. In the second case the action occurrence of A has succeeded to initiate its effect but its persistence was terminated before t_2 by some unknown exogenous event.

The above Definition of exogenous qualification captures instances of **direct qualification** in the sense that the qualification is linked directly to the fluent through which the disparity between what is observed and what is described by the domain and its semantic extensions, manifests itself. The domain description implies that F (respectively $\neg F$) should hold at t_2 but the opposite i.e. $\neg F$ (respectively F), is observed at t_2. In the first case of an exogenous qualification of an action, the failure of the action to produce its effect could be due to some exogenous reason for (at least) one of the preconditions of the c-proposition through which F would be initiated (respectively terminated), not to hold at the time t_1 of the action occurrence. For example, consider the following domain description, also depicted in Figure 15.

Example 13 (Domain QD_1)

A initiates F when L

A happens-at T_1

L holds-at T_0

$\neg F$ holds-at T_2

Figure 15: Domain QD_1

All the admissible extensions of this domain are exogenous qualification extensions on F. In particular, one of these extensions is an exogenous qualification of the action occurrence of A at T_1. But the failure of the action A to produce its effect F might be due to some other exogenous reason for the precondition L, of the c-proposition that initiates F, not to hold at the time of occurrence of A.

We can thus translate or **explain the exogenous qualification** of the action A at T_1 **endogenously** by the hypothesis that $\neg L$ holds-at T_1 and transfer the exogenous qualification of the domain to the earlier persistence of L from T_0 to T_1. Indeed, if we add in QD_1 this t-proposition then the extended domain description does not anymore have an exogenous qualification extension for F between T_1 and T_2. Such extensions of a given domain description with additional

t-propositions so that the exogenous qualification of actions is pushed back into the past in terms of exogenous qualifications of preconditions of c-propositions will be called qualification explanation domains and are defined as follows.

Definition 30 Given a domain description D and E an exogenous qualification extension of an action occurrence A on F at t_1 a **qualification explanation domain for** A **at** t_1 is a domain $D^{'}$ obtained from D by adding a set H of t-propositions on the narrative time points of D s.t.:

- H is point-wise consistent with the t-propositions already in D and H does not contain a t-proposition of the form 'F holds-at t' for $t > t_1$
- E is no longer an exogenous qualification extension of the action occurrence A at t_1 in $D^{'}$ in the sense that starting from the same initial state as in E there is no admissible exogenous qualification extension of the action occurrence A at t_1 in $D^{'}$.

H is called a **qualification explanation (wrt** E**)** (or the initial state of E) of the exogenous failure of action A at t_1.

In addition, as is usual with abductive explanations, we can require that qualification explanations are minimal wrt set inclusion.

Hence in the previous Example 13, $H = \{\neg L$ holds-at $T_1\}$, is a qualification explanation of the exogenous failure of action A at T_1. Suppose now that we extend the previous Example 13 as follows:

Example 14 (Domain QD_2)

A initiates F when L

A happens-at T_1

B initiates L when K

$B^{'}$ terminates L when $K^{'}$

B happens-at $T^{'}$

$\neg F$ holds-at T_2

where $T_0 < T^{'} < T_1 < T_2$. Depending on the initial value of the fluents K and L in an admissible extension, i.e. whether a positive or a negative assumption argument at time T_0 for these fluent is included in an extension, this will determine whether the extension is an exogenous qualification of the action occurrence of A at T_1. This can happen in two cases (a) 'L holds-at T_0' or (b) 'K holds-at T_0' under which the action occurrence of A at T_1 gives a generation (initiation) argument for F. Figure 16 depicts this example and three possible qualification explanations, $H, H^{'}, H^{''}$, for the exogenous qualification of the action occurrence of A at T_1, as will be discussed below.

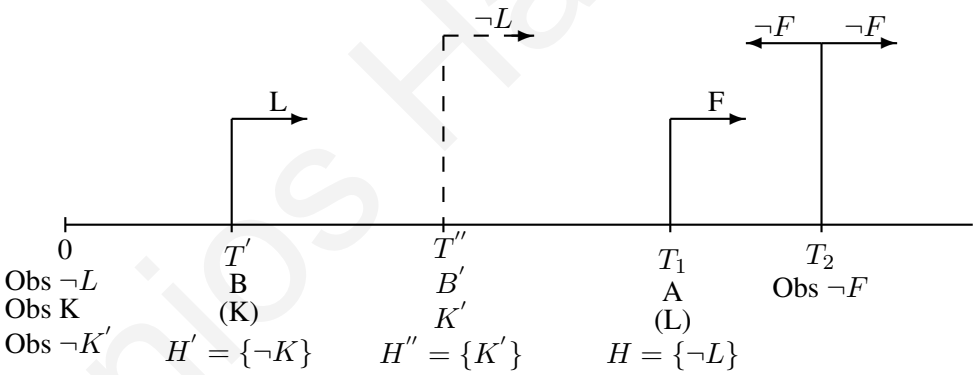


Figure 16: Qualification Explanations for Domain QD_2

Hence under either of these cases, as in the previous example, a qualification explanation for the exogenous qualification of the action occurrence of A at T_1 is given by '$\neg L$ holds-at T_1'. The extended domain obtained from QD_2 by adding this t-proposition does not anymore have an exogenous qualification extension of A at T_1 because the potential initiation argument of F at T_1

cannot be applied. But this extended domain, in the case where we have 'K holds-at T_0', will now have an exogenous qualification extension for L at $T^{'}$ based on the new t-proposition that we have now added to the domain [5] (see Figure 17).

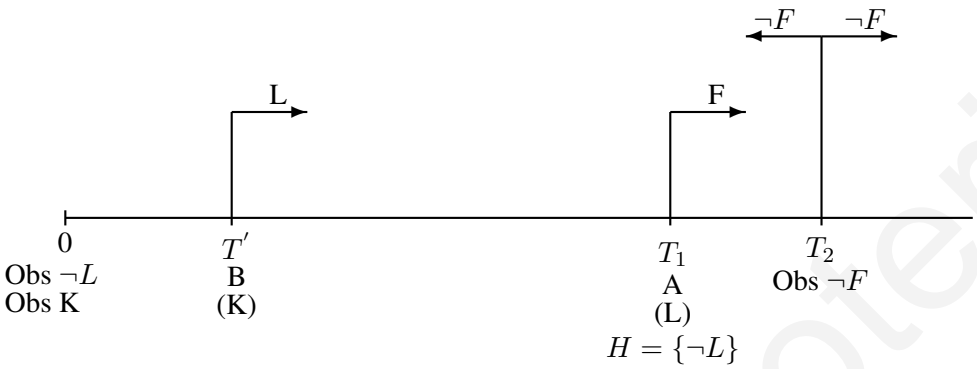


Figure 17: Qualification Explanation H for Domain QD_2

We can then explain this new exogenous qualification by adding the extra t-proposition, '$\neg K$ holds-at $T^{'}$', to the extended domain. This pushes the exogenous qualification back to the persistence of K from T_0 to $T^{'}$.

Then if also the initial state is s.t. $\neg L$ is derived at T_0, e.g. when '$\neg L$ holds-at T_0' is in the domain, there will be no exogenous qualification of A at T_1 in the domain extended by both '$\neg L$ holds-at T_1' and '$\neg K$ holds-at $T^{'}$' (see Figure 18). Note that Figure 18 shows '$\neg L$ holds-at T_1' as an additional observation.

Furthermore, the new t-proposition, '$\neg K$ holds-at $T^{'}$', is sufficient alone, without the need for the earlier explanation '$\neg L$ holds-at T_1', to explain the exogenous qualification of A at T_1. In other words, starting from the explanation $H = \{$ $\neg L$ holds-at $T_1\}$ for the exogenous qualification of A at T_1 we can replace (one of) its element(s), i.e. '$\neg L$ holds-at T_1', by an explanation, $H^{'} = \{\neg K$

[5] When we have the other case of 'L holds-at T_0' the extended domain will have admissible extensions with an exogenous qualification of persistence of L in $(T_0, T_1]$.

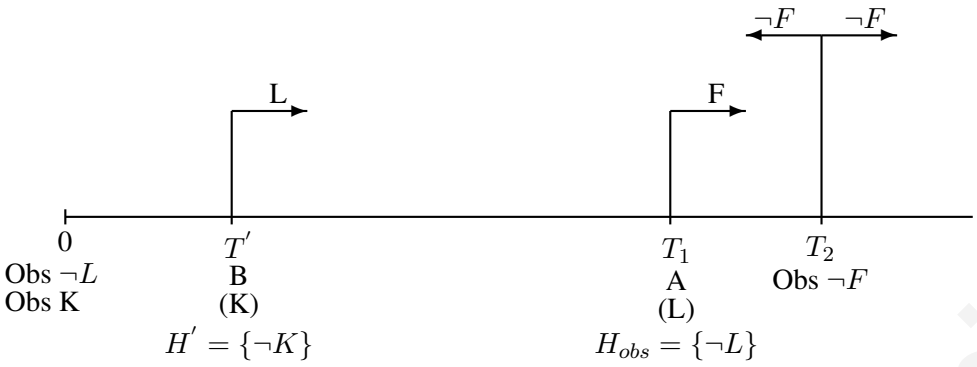


Figure 18: Qualification Explanation $H^{'}$ for Domain QD_2

holds-at $T^{'}$}, of the exogenous qualification of B at $T^{'}$ based on this element, resulting in a new (minimal) explanation of the exogenous qualification of A at T_1.

In general, there are two types of qualification explanation domains: ones that block a generation point as we have seen in the examples above and explanations that activate an intermediate generation point to block persistence. Let us consider an example of the latter type. In the example domain QD_2 another explanation of the exogenous qualification of A at T_1 would be possible if we had in the domain a known occurrence of the action $B^{'}$ at some time point $T^{''} \in (T^{'}, T_1)$ (shown in Figure 16 with a dotted line). So let us assume that the domain QD_2 also contains the h-proposition $B^{'}$ happens-at $T^{''}$. Then $H^{''} = \{K^{'}$ holds-at $T^{''}\}$ is a qualification explanation of the exogenous qualification of B at $T^{'}$ as it enables a termination point for L at $T^{''}$. In turn this means that $H^{''}$ is also a qualification explanation of the exogenous qualification of A at T_1. In other words, again starting from the explanation $H = \{\neg L$ holds-at $T_1\}$ for the exogenous qualification of A at T_1 we can generate a new explanation $H^{''}$ for this by finding an explanation for an earlier action that is exogenous qualified by the addition of H in the domain.

Hence as we have seen in the example above, qualification explanations can be built iteratively by pushing the exogenous qualification they explain to an earlier action and replacing (elements

of) the initial explanation by an explanation of the exogenous qualification of the earlier action. The following property linking the qualification explanations of actions holds in general.

Property Let D be a domain, E an exogenous qualification extension of an action occurrence A at t_1 and H a qualification explanation of A wrt E. Let also 'L holds-at T' belong to H and $H^{'}$ be a qualification explanation wrt E of an action occurrence $A^{'}$ at T_1 ($T_1 < T$) based on L holds-at T. Then given $H^{''} = (H - \{\ L$ holds-at $T\ \}) \cup H^{'}$ one of the following holds:

- $H^{''}$ is also a qualification explanation of A at t_1 wrt E
- or $D \cup H^{''}$ has an admissible extension $E^{''}$ with the same initial state as that of E s.t. $E^{''}$ is an exogenous qualification extension of an action occurrence $A^{''}$ at T_2 with $T_2 < T_1$.
- or $E \vdash \neg HoldsAt(L, 0)$ and the persistence of the fluent literal $\neg L$ from 0 to T is exogenously qualified in $D \cup H$ wrt E.

Hence as we have seen above in the example domain QD_2 $H = \{\neg L$ holds-at $T_1\}$ can be replaced by $H'' = \{\neg K$ holds-at $T^{'}\}$ to either give a new qualification explanation of A at T_1 in the case where $E \vdash \neg HoldsAt(L, 0)$ or when $E \vdash HoldsAt(L, 0)$ the persistence of L from 0 to T_1 is exogenously qualified in $D \cup H$.

Based on this property we can then develop algorithms to find qualification explanations of action failures by systematically following backwards the time line from the time of an unexpected observation, such as '$\neg F$ holds-at T_2' in the above examples, and examining the preconditions of the possible initiation and termination points for F. These explanations can be further compared according to some criterium, e.g. that of minimality of the number of actions that are exogenously qualified. This issue of the minimization of exogenous qualification (and its argumentation interpretation) is beyond the scope of this thesis. It is important though to note that the computation of

qualification explanations is made possible by the generality of the argumentation semantics and its ability to give meaning to any domain description.

4.6 Related Work and Summary

Our work follows the EC approach for RAC and the realization of this through argumentation. It extends the earlier argumentation based approach for the EC language $\mathcal{E}$ [48] by allowing the observations in a domain description to be used directly in the argumentation process rather than to be imposed as a-posteriori constraints. As a result any domain description in $\mathcal{E}$ can be given a meaning under the extended AF.

Interestingly, our formulation comes closer to the original EC [51]. Like the original EC it has a symmetrical treatment of forward and backward reasoning from events (see Appendix C), unlike the later formulations of the Simplified EC [74, 60]) where the emphasis is mainly on reasoning forward from events. The original EC is based upon the notion of events and time periods generated by the events. There are two types of time periods, $after(e\ u)$ and $before(e\ u)$ generated by an event e and during which a fluent u holds. The first of these names a time period after the occurrence of the event e while the second names a time period before the occurrence of the event e. The ends of these periods is initially undefined until extra information that can help us derive this is provided. These time periods are derived through the rules in the EC of the following form:

$Holds(after(e\ u))$ if $Initiates(e\ u)$

$Holds(before(e\ u))$ if $Terminates(e\ u)$

Such rules are analogous to the arguments $\{PG_F[u, e]$, $PFP[u, t; e]\}$, where t is any time point after the time of the event e and $NG_B[u, e]$ and $NBP[u, t; e]$, where t is any time point before or equal to the time of the event e.

This symmetrical nature of reasoning with time in the EC is mirrored with the symmetry of forward and backwards persistence arguments in our formulation. Hence both formalisms have similar effects especially when reasoning in the past. To illustrate this consider the simple example:

Example 15 (Domain $D^{''''}$)

A initiates F

B initiates F

A happens-at T_1

B happens-at T_2

In this domain the original EC concludes that there exists an end point i of the time period $after(A\ F)$ s.t. $A < i \leq B$. After i the value of F is unknown. The effect of this is that the value of F cannot be concluded at any time point between the time of A, i.e. T_1, and the time of B, i.e. T_2 (as the exact time of i is unknown). Similarly, in our formulation there exists admissible extensions of this domain s.t. for any time point $T_i \in (T_1, T_2)$ the fluent F changes it truth value from true to false and hence the value of F within this time period cannot be sceptically determined.

Hence both formulations essentially conclude the existence of at least one unknown event at some intermediate time point T_i between T_1 and T_2 that has terminated F and hence we cannot decide sceptically on F between T_1 and T_2. Through different mechanisms, our argumentation and the original EC give the same end result in terms of properties holding. We also note that both formalisms allow that in these situations of 'uncertainty' between two time points, the possibility of several changes in the value of the fluent.

The above example may be a little specialized. The more natural case of this effect of incorporating backwards persistence is seen in the Examples 8 and 9 where observations are involved. The observations indicate a change in the value of a fluent at some unknown time within a certain time period. This is captured by our formulation thus showing how the original EC could be extended to allow observations in its domain descriptions.

Another approach to RAC using argumentation is that of [82]. This work addresses the main problems of RAC, in particular the frame, and qualification problems, using a formulation that is based on argumentation and in particular the ABA AF of [13]. As in [13] it uses argumentation assumptions that complement the logical knowledge to draw inferences from a domain description. There are two types of assumptions associated to each fluent literal: *frame* and *qualification* assumptions. The frame assumptions allow the persistence of a fluent literal as these assumptions can be assumed at any time point. Unlike our approach which relies directly on argumentation to capture persistence through the use of persistence arguments, the effect of persistence in [82] is obtained indirectly by the freedom of their formalism to make these frame assumptions. This though imposes the need to control this freedom by additional requirements imposed on top of a first structure they obtain via the argumentation requirement of preferred frame assumptions, called pre-models, in order to narrow down the possibilities of frame assumptions. This extra requirement is outside the standard argumentation notions imposing a 'minimization of change' on the pre-models.

Subsequently, in order to handle also the qualification problem another type of assumptions is introduced, the qualification assumptions. These are 'externally' linked with each of the causal rules that generate fluent literals and are thought off as encompassing all the exogenous (unknown) qualification conditions that can block the generation of the fluent literal through the causal law.

Then again a further technical machinery is developed on top of this with notions such as, plausible, unsound and rejected sets of assumptions, which need to be imposed on the initial pre-models, i.e outside the argumentation formulation. In this filtering it is also necessary to impose carefully a preference of qualification assumptions over frame assumptions in order to solve both the frame and qualification problems together.

Hence although the two approaches have similarities as they are both based on argumentation they have crucial differences which can have an effect both at the representation and computational level. For example, the assumption arguments in [82] are of various types and are operational in nature rather than expressing some declarative knowledge in the domain. In contrast, our approach works directly on the given declarative knowledge of the problem: assumptions can only made at the initial time point and importantly they are declarative statements.

Furthermore, our approach addresses the problem of RAC entirely within standard argumentation notions with no reliance on any other formal structure. One of the reasons why this is not the case for the case of [82] is the overly liberal freedom to make assumptions in the first place. In our case the arguments considered need to be grounded (or supported) by the given domain description and particularly the narrative it contains. We can thus apply the argumentation process directly on the given knowledge capturing the required reasoning in a natural and simple way. This difference in the two approaches is highlighted by the way each approach uses priorities between arguments. In [82] the priorities are used on top to filter pre-structures whose definition depends on notions of argumentation only at a first level whereas in our case the priority is integrated within the argumentation process allowing the notions of argumentation to be sufficient to directly capture the semantics and the reasoning required.

As [82] argue, one of the main advantages of an argumentation based approach to the central challenges of RAC and default reasoning more generally, is that the reasoning can be made more

explicit compared to other non-monotonic approaches. This is a general feature shared by most argumentation based application frameworks including our own in this Chapter, whose relative simplicity and direct use of argumentation makes it easy to exploit this advantage.

The main contribution of this Chapter is that we have extended an existing based argumentation based formulation of language $\mathcal{E}$ by introducing (a) arguments based on property observations that one typically finds in any given narrative and (b) arguments for backward persistence. This allow us to recover and also extend language $\mathcal{E}$, giving a semantic meaning to domains that cannot be interpreted under $\mathcal{E}$. Our work then can be seen as an example where the general theory of argumentation, finds concrete application in addressing the foundational problems of temporal persistence and knowledge qualification.

Chapter 5

Computing Dynamic Argumentation

"We do not see things as they are. We see things as we are."

Rabbi Shemuel ben Nachmani, as quoted in the Talmudic tractate Berakhot (55b.)

In Chapters 3 and 4, argumentation is used for reaching conclusions by tracking and evaluating changes in the problem environment. This Chapter, discusses a new approach of creating argumentation through matrices. Since any abstract AF with many finite arguments can be presented in a matrix form, matrix operations, theorems from mathematics and graph theory were used to go deeper and investigate AF under the scope of the matrices. In this way, the strengths and weaknesses of matrix operations are migrated from a mathematical representation to a computer science interpretation.

As time changes the problem, narrative also changes abstractly and the world representation may be altered. For example, existing attack(s) or argument(s) may cease to exist, or new attack(s) or argument(s) may arise. Furthermore, at different time points under a snapshot of the world the strength of many attacks may differ. At this level of our investigation, how changes occur is not explained. Rather, they are allowed, observed and accepted and their affect is evaluated on argumentation.

Abstract AF is interpreted in terms of matrix multiplication and present matrix operation algorithms that can answer whether a given set of arguments is part of an argumentation extension under the various semantics of AF. We describe a new way of answering the question of whether or not a set of arguments, $\mathcal{S}$, is a conflict-free/admissible/complete/stable/grounded extension of a given AF. Based on this concept, the algorithms ASSA and ASSA$_G$[1] were created. In recent months, our algorithms were optimized and effective and efficient ways to handle dynamically changing environments were identified through matrix methods.

5.1 Introduction

Agents can construct arguments for and against a specific goal in order to reach a conclusion. The construction of these arguments often follows semantics that are given in an abstract AF, either by extensions [24] or by labellings [17, 84]. The former was used to base the ASSA algorithm and the latter for the the ASSA$_G$ algorithm.

The aim of this chapter is to introduce matrices and their operations that were studied, and to explore AFs in order to propose new approaches to identifying extensions. The theoretical part must be studied in order to understand the theory of abstract AF in terms of matrices. Within the practical part, efficient algorithms and systems that compute argumentation are developed. A matrix-based approach to compute extensions for abstract AFs is presented according to various semantics. When AFs are represented by their adjacency matrices and set as characteristic vectors, then matrix operations and additional tests can be used to verify whether a given set of arguments satisfies certain semantical criteria. This is a new approach to addressing the question of whether or not a set of arguments, $\mathcal{S}$, is a conflict-free/admissible/complete/stable/grounded extension of a given AF.

[1] Inspired from left and right matrix multiplication: AS and SA, and the author's children's names Artemis-StylianaStylianaArtemis and Giorgos

We have developed an algorithm that answers whether a given set of arguments is an extension, implemented in a system, and presented at the ICCMA'15 [2] competition [80]. Our solver, called ASSA [37], finds the stable extension(s) of an AF.

5.1.1 Related Work

There is a lot of interest in computing extensions, with a competition starting in 2015 (see ICCMA'15). Other approaches that tackle similar results exist, such as the ASPARTIX (Answer Set Programming Argumentation Reasoning Tool) and DIAMOND [27, 28, 19, 83], but none of them use matrices to compute extensions. They are based on SAT-solving techniques, where the argumentation problems are transformed to SAT problems.

A matrix-based mathematical approach to answering question in abstract argumentation of the form: 'Is set A an extension?' is introduced. Similar to this approach, is the work presented in [85, 18]. Their approach is structured as follows: They consider the adjacency matrix of an AF and then define several parts of the adjacency matrix, which they call sub-blocks. Finding all sub-blocks that have zeroes everywhere is similar to finding the conflict-free sets of the AF. This result can be seen in Definition 31. Each one of these sets was mapped into its matrix representation normal form, with respect to a specific semantics (stable, admissible, complete). Based on matrix criteria they find among the conflict-free sets that qualify as stable, admissible, or complete extensions.

As we will see, our approach differs in that this study not use sub-blocks, and creates sets of arguments in a matrix representation to define tests for the different argumentation semantics. Our work, when considering all possible subsets of arguments, comes close to the work of [18], as it answers similar questions but with a different algorithm. Both techniques are time consuming with

[2] http://argumentationcompetition.org/index.html

high computational complexity in many problems. In the work [18], they first labell all arguments in the AF with a distinct name and use the notion of the adjacency matrix to represent the AF as a square matrix. Instead of having only one representation of the AF as we have, they can rename arguments whenever needed to map the AF to a different matrix. The renaming process can also be done through a series of dual interchanges inside of their initially created adjacency matrix, which is an interchange of j row with k row and j column with k column. This process, when done constructively, manages to move all of the arguments that are being considered to the top, left corner of the matrix, where they represent each AF into a matrix that has one of the two standard forms that have defined. Taking into consideration the full potentials of the adjacency matrix represented in a standard form, and the basic definitions of extensions, they manage to find a mapping relationship between extensions under various semantics and the internal structure of the matrix by observing predefined sub-blocks for various extensions, such as admissible, stable, and complete.

Various optimization techniques have been applied in [18], such as finding various sub-blocks of the matrix and comparing their values. The standard form is partitioned into three parts (a) set S, which includes the arguments under investigation; (b) the set of arguments that are under attack by S; and (c) the other arguments in the AF or even considering the set as a single argument when it reduces the dimensions of the matrix significantly.

To find the conflict-free sets the work of [18] uses the adjacency matrix properties. To consider all possible conflict-free sets, they must create all possible sets (which is time consuming and not practical) and determine whether these sets satisfy the definition. If they already know which arguments they want to check, then they create the standard form matrix based on this set and compare the elements of a particular sub-block. If all of the elements are zero, they conclude that the set it is conflict free. Furthermore, by taking different sub-blocks of the standard form matrix,

they follow the definition of stable set, admissible set, or complete set to check by comparing the elements of the sub-blocks if, the conflict-free set is also a stable, admissible, or a complete extension.

As we will see, in both techniques, the definitions of semantics are strictly followed to find conflict-free, admissible, stable, and complete extensions. The approaches are different, as our study uses matrix multiplication and a comparison with the resulting vector, while in [18] the authors create the standard form of the adjacency matrix based on the given set and then compared. The philosophy might be the same but the implementation varies. Nonetheless, our approach has been implemented in the ASSA and AASA$_G$ algorithms, while their approach has not been implemented yet.

5.2 Matrix Approach to Argumentation

"If you're not having fun, you're doing something wrong."

Groucho Marx

The basic information presented here is used throughout this Chapter on matrices and argumentation extensions that one can find in books (see e.g. [75, 24, 5, 25]). Based on matrix operations, an algebraic method was constructed that computes extensions for an abstract AF. A link between the fields of graph theory and logic programs has been presented in [23, 81], and it is shown that stable extensions correspond to the kernels of the adjacency matrix. This approach was not followed, despite the similarities found through general comparison.

A **matrix** is a structure that is designed in rows and columns, where each one of its elements contains information. When the number of rows and columns are equal, the matrix is called a

square matrix. Square matrices have diagonals; a row vector is a $1 \times n$ matrix $\begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix}$ and a column vector is a $n \times 1$ matrix $\begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix}^T$.

Computers can perform matrix operations relatively quickly. The computational complexity for multiplying two matrices with n digit numbers is $\mathcal{O}(n^3)$ [76], and there are methods that can optimize this result [76, 65]. This algorithm for computing argumentation extensions and verifying extensions is based on matrix multiplication. Thus, it is of logarithmic space complexity [28, 19]. This sections is not concerned with complexity issues.

This approach is based on matrices because: (a) matrices can be easily represented and handled by a computer; (b) illustrated information is compact; (c) through different operation tools, matrices can be easily manipulated; and (d) matrices can capture all of the information in an abstract AF. From a theoretical point of view, matrices are interesting to investigate in depth, as characterizations of semantical definitions are simple to recognize.

Square matrices can capture the arguments and the attacking relations of an AF in a relatively easy way. First, each argument must be labelled with a distinct natural number, using the rows and columns of the matrix to represent the arguments and the attacks, respectively. For example, the third row $a_{3,*}$ of a square matrix consists of the elements $\{a_{3,1}, a_{3,2}, \dots, a_{3,n}\}$. If argument a_3 attack argument a_4 then the element $a_{3,4}$ of the square matrix will be one (1), otherwise it will be zero (0). The value of the element $a_{3,3}$ will show if argument a_3 is self-attacking. The formal definition is as follows:

The next definitions and results are adapted straightforwardly from graph theory as can be found in [52, 14, 75].

Definition 31 (Mapping an AF to a matrix. The reader is also referred to [52]) Let $A = (a_{i,j})$ be the **adjacency matrix** of an AF $= \langle \mathcal{A}, \mathcal{R} \rangle$ s.t.: $a_{i,j} = \begin{cases} 1 & \text{if } (a_i, a_j) \in \mathcal{R} \\ 0 & \text{if } (a_i, a_j) \notin \mathcal{R} \end{cases}$

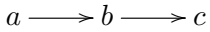


Figure 19: A Simple AF

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Figure 20: Adjacency Matrix of Figure 19

Figure 21: Figures 19 and 20 Show Respectively the Directed Graph and Matrix Representation of Example 16

Note that if arguments are selected in a different order the adjacency matrix will differ. Thus, different ordering will generate different adjacency matrices. It is important to know who attacks who. The attacker is represented by the row of the adjacency matrix and each column represents the attacked argument. Therefore, $a_{3,4}$ represents the attack from argument a_3 to argument a_4 while $a_{4,3}$ represents the attack from a_4 to a_3.

$$\begin{array}{c} \textbf{to} \\ \textbf{from} \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} \end{array}$$

Example 16 Let $\{a, b, c\} = \mathcal{A}$ be three arguments s.t. $\{(a, b), (b, c)\} = \mathcal{R}$. Figure 21 depicts this example.

Example 17 Let $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$ and $\mathcal{R} = \{(a_1, a_1), (a_1, a_2), (a_1, a_4), (a_2, a_3), (a_2, a_4), (a_4, a_2), (a_4, a_3)\}$. Its directed graph and matrix representation are shown in Figure 24.

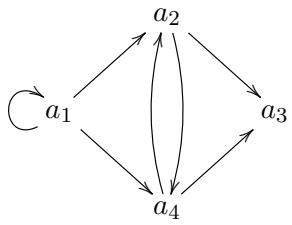


Figure 22: A More Complicated AF

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Figure 23: Adjacency Matrix of Figure 22

Figure 24: Figure 22 Shows Example 17's Directed Graph and 23 Captures its Adjacency Matrix

Matrix operations can be constructed under standard mathematical rules. When these operations are performed on the adjacency matrix of an AF, an interpretation exists, connecting the result of the operation with the AF.

Definition 32 (Representing a set of arguments as a vector) Let AF $= \langle \mathcal{A}, \mathcal{R} \rangle$ with A its adjacency matrix and $\mathcal{S} \subseteq \mathcal{A}$. Set $\mathcal{S}$ is represented by a column vector $S_{n\times 1} = (s_{i,1})$, where

$$s_{i,1} = \begin{cases} 1 & \text{if } a_i \in \mathcal{S} \\ 0 & \text{if } a_i \notin \mathcal{S} \end{cases}$$

Proposition 2 Let A be the adjacency matrix of AF $= \langle \mathcal{A}, \mathcal{R} \rangle$ and let $\mathcal{S} \subseteq \mathcal{A}$ be a set of arguments with S as its column vector representation. The product AS is a column vector where the entry $(AS)_{i,1}$ shows how many times argument $a_i \in \mathcal{A}$ attacks $\mathcal{S}$.

Proof 6 See Appendix, Proof 22

Proposition 3 Let A be the adjacency matrix of AF $= \langle \mathcal{A}, \mathcal{R} \rangle$ and $\mathcal{S} \subseteq \mathcal{A}$ be a set with S^T as its row vector representation. The product $S^T A$ is a row vector where entry $(S^T A)_{1,i}$ shows how many times argument $a_i \in \mathcal{A}$ is attacked by $\mathcal{S}$.

Proof 7 The proof is similar to the proof of Proposition 2.

From a mathematical perspective [75] the notions of a walk and length of a walk of a directed graph, have a different meaning than from a computer science perspective. These mathematical notions are used to provide an interpretation of an AF.

Definition 33 (The reader is also referred to [52]) A **walk** in an AF is a sequence of arguments and attacks where each attack is directed from the attacking argument to the argument under attack.

Length of a walk in an AF is the smallest number of attacks occurring in the AF that connect argument a_i to argument a_j.

Theorem 4 [The reader is also referred to Theorem 5.4.4 at [52]] Let A be the adjacency matrix of AF $= \langle \mathcal{A}, \mathcal{R} \rangle$. The $(A^m)_{i,j}$ entry is the number of walks of length m from a_i to a_j in AF.

Proof 8 See Appendix, Proof 23

At this point, several questions have been generated. To interpret this result under the scope of an abstract AF, $A^2, A^3, \ldots, A^n, \ldots$ is computed. But when to stop is unclear. To determine whether there is a limit, or an upper bound where no more information can be extracted if it is passed, or if it carries on indefinitely, the following restriction is added: A path that has been used can not be retaken. With this limitation an upper bound can be identified that is relative to the number of arguments in the AF. When an AF contains n arguments, A^n can be computed in order to find all of the loops containing up to n arguments. This result is interesting when investigating elements $(A^n)_{i,i}$ on the main diagonal.

Example 18 Consider the AF depicted in Figure 22. To find the number of walks with length 2 and 3 starting from argument a_1 to a_4, matrices A^2 and A^3 are computed respectively. Note that in $A^2, a_{1,4} = 2$ which indicates that two walks of length 2 exist from argument a_1 to argument a_4.

These two walks are $\{a_1, a_1, a_4\}$ and $\{a_1, a_2, a_4\}$. In addition, $A^4 = A^2 \times A^2$; thus, any element in A^2 that is not zero will not be a zero in A^4.

$$A^2 = \begin{pmatrix} \boxed{1} & 2 & 2 & 2 \\ 0 & \boxed{1} & 1 & 0 \\ 0 & 0 & \boxed{0} & 0 \\ 0 & 0 & 1 & \boxed{1} \end{pmatrix} \quad A^3 = \begin{pmatrix} \boxed{1} & 3 & 4 & 3 \\ 0 & \boxed{0} & 1 & 1 \\ 0 & 0 & \boxed{0} & 0 \\ 0 & 1 & 1 & \boxed{0} \end{pmatrix} \quad A^4 = \begin{pmatrix} \boxed{1} & 4 & 6 & 4 \\ 0 & \boxed{1} & 1 & 0 \\ 0 & 0 & \boxed{0} & 0 \\ 0 & 0 & 1 & \boxed{1} \end{pmatrix}$$

Proposition 4 Let A be the adjacency matrix of an AF $= \langle \mathcal{A}, \mathcal{R} \rangle$. The entry $(A^m)_{i,i}$, $m \leq n$ of A^m is the number of loops of length m from a_i to a_i in AF. When m is odd (resp. even) the number of odd (resp. even) loops is shown.

Proof 9 The proof follows directly from Theorem 4.

Based on Theorem 4, $(A^m)_{i,j}$ entry shows the number of walks of length m. The main diagonal of a matrix illustrates loops, as loops are walks at which the starting and the ending point is the same. The number of loops of length m (i.e., containing m arguments) is shown in the A^m main diagonal when m is not a multiplier of an already existing loop. When $m =$ odd (resp. even) number, the number of odd (resp. even) loops is shown in an AF of any length. The matrix $A = A^1$ diagonal shows the 'self-attacking' arguments as it is a walk from an argument to itself of length 1 (i.e., in Figure 23 $a_{1,1} = 1$). When a loop with length n exists (i.e., n arguments) it will be shown in the matrix A^n main diagonal. Furthermore, any other matrix $A^{n \times k}$, $k \in \mathbf{N}$ will also contain this loop (as we cannot allow restrictions to fall directly into matrix operations). Additionally, other loops of length $n \times k$ will also be shown if they exist on the main diagonal of $A^{n \times k}$.

With the help of matrix multiplication, questions such as which arguments in A attack (resp. are attacked by) a specific set of arguments can be answered. Even though this question can be

answered by observing either the AF or the adjacency matrix, when the frameworks and hence their matrices are large, it may be easier to use matrix operations. In this way, it is similar to using a series of matrix multiplications to navigate through the AF. To illustrate this statement, let us expand Example 17.

Example 19 (Based on Example 17) To answer the question what arguments argument a_2 attacks or is attacked, the vector $S = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = (\,0\;1\;0\;0\,)^T$ is constructed. S contains as many entries as the arguments in the AF (four in this case). Since this example is only interested in a_2, all of the entries in S will be 0 except for the second entry, which will be 1. To find what arguments are attacked by argument a_2 the matrix $S^T A$ is evaluated. To find what arguments in AF can attack argument a_2 the matrix AS is evaluated. Here, $S^T A = (\,0\;0\;1\;1\,)$, which means that $\mathcal{S}$ attacks arguments a_3 and a_4, as the third and fourth elements are not zero and $AS = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ which means that $\mathcal{S}$ is attacked by arguments a_1 and a_4.

Let us consider the arguments a_2, a_3, and a_4. In this particular instance, $S = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$. Evaluating the matrix $S^T A$, $S^T A = (\,0\;1\;2\;1\,)$ is obtained. This shows which arguments in A are attacked by $\mathcal{S} = \{a_2, a_3, a_4\}$. Specifically, a_2 and a_4 are attacked by one of the arguments in $\mathcal{S}$ and a_3 is attacked by two of the arguments in $\mathcal{S}$, but it is unclear by which arguments. This information is lost through the matrix multiplication and one can retrieve it when multiplication is done indestructibly, which can then be used to easily evaluate the results in dynamic environments.

By following definitions for labelling arguments [17, 84], which arguments are labelled in, out, or $undec$ can be identified. Furthermore, the interpretation of other multiplications can be provided as well. For example, $S^T A^2 = (S^T A)A$ (resp. $S^T A^k$ for $k \in \{1, 2, \ldots\}$) shows the arguments in A that can be reached/attacked by the $\mathcal{S}$ arguments in two steps (resp. k steps). In addition, $A^2 S = A(AS)$ (resp. $A^k S$ for $k \in \{1, 2, \ldots\}$) shows the arguments in A that

can reach/attack the $\mathcal{S}$ arguments in two steps (resp. k steps). Using the notion 'reach' conveys a positive meaning where the notion of 'attack' conveys a negative meaning (i.e., depending whether the length of the walk is an odd or an even number), we present it here as an observation and the mapping to an abstract AF should be done with gentle manipulations. The formal results are shown below.

Proposition 5 Let A be the adjacency matrix of an AF $= \langle \mathcal{A}, \mathcal{R} \rangle$. Let $\mathcal{S}$ be a set of arguments with S as its column vector representation. The entry $(A^k \mathcal{S})_{i,1}$ of the product $A^k \mathcal{S}$ shows how many times arguments $a_i \in \mathcal{A}$ can reach the set S in a k length walk.

Proof 10 Let AF $= \langle \mathcal{A}, \mathcal{R} \rangle$ with $A = (a_{i,j})$ as its adjacency matrix, and let $A^k = B$. By Theorem 4 and Proposition 2 is proven.

Proposition 6 Let A be the adjacency matrix of AF $= \langle \mathcal{A}, \mathcal{R} \rangle$. Let $\mathcal{S}$ be a set of arguments with S as its column vector representation. The entry $(S^T A^k)_{1,i}$ of the product $S^T A^k$ shows how many times arguments $a_i \in \mathcal{A}$ are reached by the set $\mathcal{S}$ in a k length walk.

Proof 11 The proof is similar to the proof of Proposition 5. It follows directly from Theorem 4 and Proposition 3.

5.2.1 Theory and Algorithms

In this section, algorithms are provided to determine whether a set of arguments, is conflict-free, admissible, stable, or complete.

Conflict-free test: Given a set of arguments we can check if this set is conflict-free by running a conflict-free test as follows:

Proposition 7 (Conflict-free test) Let AF $= \langle \mathcal{A}, \mathcal{R} \rangle$ with A as its adjacency matrix. Let $\mathcal{S} \subseteq \mathcal{A}$ be a given set of arguments with S as its column vector representation. Let $\Gamma = S^T A$. S passes the conflict-free test iff whenever $\gamma_i \neq 0 \in \Gamma$ then $f_i = 0 \in \mathcal{S}$.

Proof 12 See Appendix, Proof 24

Computing a matrix multiplication can determine whether a given set of arguments, $\mathcal{S}$, is conflict-free. When a row matrix passes (resp. fails) the test, $\mathcal{S}$ is (resp. is not) conflict-free. The empty set always passes the conflict-free test, as the generated matrix Γ has zeroes everywhere.

Example 20 (a) Consider Example 16 illustrated in Figure 21. Let $\mathcal{S}_1 = \{a_1\}$ with $S = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ and $S^T A = \Gamma = (0\ 1\ 0)$. $\gamma_i \neq 0$ when $i = 2$ and $s_2 = 0$. For this reason it passes the conflict-free test. Therefore, the set $\mathcal{S}_1 = \{a_1\}$ is conflict-free (i.e., not self-attacking). Let us now consider the set $\mathcal{S}_2 = \{a_1, a_2\}$ with $S = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$ and $S^T A = \Gamma = (0\ 1\ 1)$. $\gamma_2 = s_2 = 1$. Thus, the set $\mathcal{S}_2$ fails the conflict-free test.

(b) Similarly, for Example 17 depicted in Figure 24, the set $\{a_1\}$ is not conflict-free as it attacks itself: $(\boxed{1}\,0\,0\,0) \times A = (\boxed{1}\,1\,0\,1)$.

Admissible extension test: To check the admissibility of a given set of arguments $\mathcal{S}$, two tests are performed: (a) the conflict-free test and (b) the defending test. For $\mathcal{S}$ to be admissible it must pass both tests.

Proposition 8 (Defending test) Let AF $= \langle \mathcal{A}, \mathcal{R} \rangle$ with adjacency matrix A and let $\mathcal{S} \subseteq \mathcal{A}$ be a set of arguments. Let S be the column vector representation of $\mathcal{S}$ and $\Gamma = (\gamma_i) = AS$. For every i, s.t. $\gamma_i \neq 0$, a column vector $\Delta^{(i)} = (\delta_j^{(i)})$ is created, s.t.: $\delta_j^{(i)} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$. Options holds: (1) Γ is a zero matrix, or (2) $E^{(i)} = A\Delta^{(i)}$ and $\forall i, \exists e_k^{(i)} \in E^{(i)}$ s.t. $e_k^{(i)} \neq 0$ and $0 \neq s_k \in S$.

Proof 13 See Appendix, Proof 25

The defending test is based on matrix multiplication. $\Gamma = AS$ shows if $\mathcal{S}$ is under attack, i.e. if any arguments in AF can attack $\mathcal{S}$. When $\Gamma = \mathbf{0}$, set $\mathcal{S}$ passes the defending test as no attackers exist and there are no arguments to defend against. When $\Gamma \neq \mathbf{0}$, set $\mathcal{S}$ is under attack and we have to check if $\mathcal{S}$ counter attacks them. Since we do not know who attacks who, instead we get an indication of the number of attacks an argument or a set of arguments has, we need to separate the arguments under attack and create the vectors $\Delta^{(i)}$. Based on another matrix multiplication, $E^{(i)} = A\Delta^{(i)}$, we find $E^{(i)}$ that shows any arguments in A that attacks $\Delta^{(i)}$. To confirm that counter attacks existing come from arguments in $\mathcal{S}$, we compare S with $E^{(i)}$s. If the comparison shows that any attack is counter attacked by $\mathcal{S}$, we conclude that $\mathcal{S}$ has passed the defending test based on S.

Example 21 (Defending test examples) In Example 16, given the sets $\mathcal{S}^{'} = \{\}$ and $\mathcal{S} = \{a, c\}$, for $\mathcal{S}^{'}$: all entries for its column vector representation are zeroes, and since $\mathcal{S}^{'}$ is represented by a zero matrix, the empty set passes the defending test.

For $\mathcal{S}$: $S = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ and $AS = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \Gamma$. Using $S = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$, the thesis asks which arguments can attack arguments a and c. The answer is $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \Gamma$, which means that argument b attacks the argument set $\mathcal{S} = \{a, c\}$. From Γ and its non zero entries we find $\Delta^{(1)} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$. To check if all attackers (i.e. b) are attacked back by $\mathcal{S}$ we evaluate $E^{(1)} = A\Delta^{(1)} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$. The result is $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ and since $e_1^{(1)} \neq 0 \in E^{(1)}$ and $s_1 \neq 0 \in S$, $\mathcal{S}$ has passed the defending test.

Proposition 9 (Admissibility test) Let $\mathcal{S}$ be a given set of arguments. For $\mathcal{S}$ to be admissible its matrix representation S has to pass both the conflict-free and defending tests.

Proof 14 The proof follows directly from Definition 1, Proposition 7 and Proposition 8.

Stable extensions test: Stable extensions are a subclass of admissible extensions. For any given set, for $\mathcal{S}$ to be stable it first has to pass the conflict-free test. Then, whether $\mathcal{S}$ can attack all other arguments which do not belong to it is determined. This is done with the stable extension test.

Proposition 10 (Stable extensions test) Let AF $= \langle \mathcal{A}, \mathcal{R} \rangle$ with adjacency matrix A. Let $\mathcal{S} \subseteq \mathcal{A}$ be a given set of arguments and S the column vector of $\mathcal{S}$ and $\Gamma = S^T A$. The set $\mathcal{S}$ passes the stable extensions test iff:

1. S passes the conflict-free test, and
2. $\forall i$ s.t. $s_i = 0, \gamma_i \neq 0$ (where $s_i \in S, \gamma_i \in \Gamma$).

Proof 15 See Appendix, Proof 26

For the stable extension test, the admissibility test is not used. Intuitively, a set attacking anything that is 'outside' of it means that it attacks all of its potential attackers. This is true since passing the conflict-free test shows that attacks coming from 'inside' of it are nonexistent. Thus, any existing attacks should be from 'outside' and it would counterattack regardless.

Example 22 (Stable extension test) For Example 16, illustrated in Figure 21, wether $\mathcal{S} = \{a\}$ is stable id determined. To do this, $\mathcal{S}$ must pass the stable extension test. $\{a\}$ passes the conflict-free test as shown below. $S = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ and $S^T A = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} = \Gamma$, where $\gamma_2 \neq 0$ and $b_2 = 0$. The matrix $S^T = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$ is used. This is similar to asking which arguments in $\mathcal{A}$ are attacked by a. The answer is matrix $\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$, which means that a attacks b. Therefore it is not stable as it does not attack all of the arguments found elsewhere (i.e., arguments $\{b, c\}$). To conclude this result based on Proposition 10 matrices $\Gamma = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$ and $(S)^T = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$ are compared, but $s_3 = 0$ and $\gamma_3 = 0$. For this reason, it fails the stable extension test.

The set $\mathcal{S} = \{a, c\}$, $S = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ passes the stable extension test. $S^T A = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} = \Gamma$. First, it passes the conflict-free test as $\gamma_2 \neq 0$ and $s_2 = 0$. Using $S^T = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$ the thesis asks which arguments in A are attacked by the set $\mathcal{S} = \{a, c\}$. The answer is $\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$, which means that it only attacks argument b. Based on the stable extension test, we compare matrices $\Gamma = S^T A = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$ and $(S)^T = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$. $s_2 = 0$ and $\gamma_2 = 1$, and therefore it is stable.

Complete extensions test: Let $\mathcal{S}$ be a given set of arguments. For $\mathcal{S}$ to be complete it must be admissible and for this reason its matrix representation has to pass the admissibility test first. Wether it contains all of the arguments that it defends must then be determined. Note that if an argument has no attackers, it is trivially defended by any set. This check can be done through two matrix multiplications. The first multiplication $B = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}_{1 \times n} A_{n \times n}$ finds all of the arguments that are not under attack and these arguments should be in any complete extension. The second multiplication determines whether the set $\mathcal{S}$ contains all of the arguments that it defends. $\Gamma = (S^T A)A = S^T A^2$, where $Z = S^T A$ will show (when mapped to the AF) all arguments that are under attack by the set $\mathcal{S}$, and $\Gamma = ZA$ will show all arguments that set $\mathcal{S}$ can defend. To identify all complete extensions, the admissibility test must be run with an extended set $\mathcal{S}$ that contains all possible combinations of the arguments in $\mathcal{A}$. This technique is time consuming and its computational complexity increasingly grows as the number of arguments that are not in $\mathcal{S}$ become bigger. To consider all possible combinations with n (many arguments), a matrix with 2^n number of columns has to be created.

Proposition 11 (Complete extensions test) Let AF $= \langle \mathcal{A}, \mathcal{R} \rangle$ with adjacency matrix $A_{n \times n}$. Let $\mathcal{S} \subseteq \mathcal{A}$ with S its column vector. $\mathcal{S}$ passes the complete extension test iff:

1. S passes the admissibility test

2. Compute $B = (1\ 1\ \ldots\ 1)_{1\times n}\, A$ and $\Gamma = SA^2$. For each entry $b_i = 0$ then $s_i \subseteq S$, and for each entry $\gamma_j \neq 0$ then $s_j \subseteq S$.

Proof 16 The proof follows directly from Proposition 9, Definition 1, and Definition 2.

Example 23 Consider Example 16, its set $\mathcal{S} = \{a, c\}$ and its matrix representation $S^T = (1\ 0\ 1)$. Condition 1 is satisfied as S passes the admissibility test (see Example 21) and therefore $\mathcal{S}$ is admissible. To determine whether $\mathcal{S}$ is complete, condition 2 should also be satisfied. Evaluate $B = (1\ 1\ 1)\, A = (1\ 1\ 1) \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} = (0\ 1\ 1)$, and $\Gamma = S^T A^2 = (1\ 0\ 1) \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = (0\ 0\ 1)$. Note that $b_1 = 0$ and $s_1 \subseteq S$. In addition, $\gamma_3 \neq 0$ and $s_3 \subseteq S$. Thus, condition 2 is satisfied and $\mathcal{S} = \{a, c\}$ is complete.

Building the grounded extension: To build the grounded extensions, a constructive algorithm for generating the grounded labelling, taken from [61], is followed. First, all arguments that are not under attacked are labelled in. If such arguments do not exist then the grounded extension set is empty. Then, the arguments that are under attack by an argument that is labelled in are identified, and the labelling out is assigned to these arguments. The in labell is then assigned to all arguments whose attackers are out. Continue until no new arguments can be labelled either in or out. If any arguments have not been assigned a labell then assign those with the labell $undec$.

Algorithm for grounded semantics using the labelling approach

1. $\mathcal{L}_0 = (\emptyset, \emptyset, \emptyset)$ % $(in(\mathcal{L}), out(\mathcal{L}), undecided(\mathcal{L}))$
2. **repeat**
3. $in(\mathcal{L}_{i+1}) = in(\mathcal{L}_i) \cup \{x |$ x is not labelled $in(\mathcal{L}_i)$, and $\forall y$: if $(y, x) \in \mathcal{R}$ then $y \in out(\mathcal{L}_i)\}$
4. $out(\mathcal{L}_{i+1}) = out(\mathcal{L}_i) \cup \{x |$ x is not labelled $in\mathcal{L}_i$, and $\exists y$: if $(y, x) \in \mathcal{R}$ then $y \in in(\mathcal{L}_i)\}$
5. **until** $\mathcal{L}_{i+1} = \mathcal{L}_i$

6. $\mathcal{L}_{grounded}$ = $(in(\mathcal{L}_i), out(\mathcal{L}_i), \mathcal{A} \setminus (in(\mathcal{L}_i) \cup out(\mathcal{L}_i)))$

To map this algorithm to the matrix approach, the following steps are taken. Let $\mathcal{G}$ be the set that contains the arguments that are qualified as grounded. Start with $\mathcal{G}_0 = \{\}$.

Step1:

Let AF be the existing argumentation framework with $A_{n\times n}$ as its adjacency matrix. Let $B_{1\times n}$ be a row matrix that has ones in every position. Evaluate $BA = C$. Let $\mathcal{G}_1$ be the set containing all of the arguments that are represented by the zero elements from matrix C (these are the arguments that are labelled in). Let $\mathcal{G}_2 = \mathcal{G}_1 \cup \mathcal{G}_0$.

Step 2:

Find which arguments are labelled out. To find them matrix D must be created first, which is a matrix that has zeroes in positions that matrix C has elements that are not zeroes and ones at the positions where the matrix C has zeroes. Then, the matrix multiplication $DA = E$ must be evaluated. The positions that do not have zeros in matrix E are the arguments that are labelled out. Let these arguments be represented by the set $\mathcal{F}$.

Step 3:

Take the arguments that are labelled in and the arguments that are labelled out and remove them from the existing AF. By removing these arguments, all attacks that are related to these arguments are removed as well. That is, the existing AF is taken and the set $\mathcal{G}_1 \cup \mathcal{F}$.

Step 4:

If $\mathcal{G}_1 = \{\}$ or $\mathcal{A} = \{\}$ go to ***Step 5*** **else** go to ***Step 1*** with $\mathcal{G}_0 = \mathcal{G}_2$.

Step 5:

Grounded = $\mathcal{G}_2$

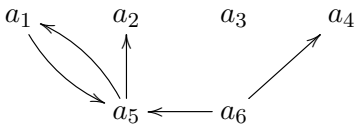


Figure 25: Example for Grounded Extension

5.2.2 Implementation Systems

A new way of answering the question of whether or not a set of arguments, $\mathcal{S}$, is a stable extension or a grounded extension of a given AF is implemented through the algorithms ASSA and ASSA$_G$ respectively.

5.2.2.1 ASSA

"Many hands make light work."

John Heywood

A system called ASSA [37, 36] was implemented that is able to answer various questions for stable semantics for any given abstract AF. This program creates all possible instances of selected set $\mathcal{S}$ and builds a matrix $S^{'}$. That is, each set of arguments is represented as one column of the matrix $S^{'}$. By handling matrix $S^{'}$ in a similar way to vector S and performing necessary tests, all stable extensions were found that exist in an abstract AF.

The solver ASSA is written in Java; it uses the trivial graph format and the package of matrix multiplication. It performs relatively well, despite the fact that it is in its primary stage. The authors participated in the ICCMA'15 competition for the following computational tasks with respect to the stable semantics:

Given an abstract AF the following must be determined:

1. Some extension $(SE - ST)$

2. All extensions $(EE - ST)$

Given an abstract AF and some argument, decide whether the given argument is:

1. Credulously inferred $(DC - ST)$

2. Skeptically inferred $(DS - ST)$

Example 17 can be presented in tgf format as follows:

a_1

a_2

a_3

a_4

#

a_1,a_1

a_1,a_2

a_1,a_4

a_2,a_3

a_2,a_4

a_4,a_2

a_4,a_3

The benchmark graphs that were use for these results (e.g., $(SE - ST)$, $(EE - ST)$, $(DC - ST)$, and $(DS - ST)$) were based on three different graph models: (a) A group consisting of graphs with a large grounded extension and many nodes, (b) a group consisting of graphs with

a rich structure of strongly connected components, and, (c) a group consisting of graphs which feature many complete/preferred/stable extensions.

To find several extensions or all extensions, a matrix $S^{'}$ must be created where each one of its columns represent one subset of arguments in the AF. Then, by running tests under $S^{'}$, similar questions can be addressed by comparing each column of the resulting matrices with the appropriate vector of the test.

Example 24 (cont.) Creating the matrix $S^{'}$ of Example 17. The number of arguments are four, thus $S^{'}$ contains $2^4 = 16$ columns. Each column of $S^{'} = \begin{pmatrix} 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \end{pmatrix}$ is mapped to selected sets of arguments in $\mathcal{A}$ (e.g., the fifth column refers to argument a_2). Therefore, information related to arguments a_2 will be found when comparing the fifth columns of matrices $AS^{'}$ and ${S^{'}}^{T}A$. Here, $AS^{'} = \begin{pmatrix} 0&1&0&1&1&2&1&2&1&2&1&2&2&3&2&3 \\ 0&1&1&2&0&1&1&2&0&1&1&2&0&1&1&2 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&1&2&2&0&0&1&1&1&1&2&2 \end{pmatrix}$ and ${S^{'}}^{T}A = \begin{pmatrix} 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 0&1&0&1&0&1&0&1&1&2&1&2&1&2&1&2 \\ 0&1&0&1&1&2&1&2&0&1&0&1&1&2&1&2 \\ 0&0&0&0&1&1&1&1&1&1&1&1&2&2&2&2 \end{pmatrix}$.

ASSA is ineffective at this time as it needs to create a matrix with 2^n number of columns. Specifically, under the first group of benchmark graphs where many nodes exist, ASSA suffers the most. The authors plan to study ways to make this more effective, including (a) optimizations that come from the properties of argumentation semantics and (b) optimizations that exploit general optimizations on matrix operations. On the other hand, ASSA is not affected by the density of the problem (i.e. problems with a rich structure of strongly connected components and problems that feature many stable extensions) and because it is based the adjacency matrix one can understand the algorithm and the definitions fully with basic mathematic knowledge. Based on these advantages ASSA scored relatively well in the ICCMA'15 competition as shown in Figure 26.

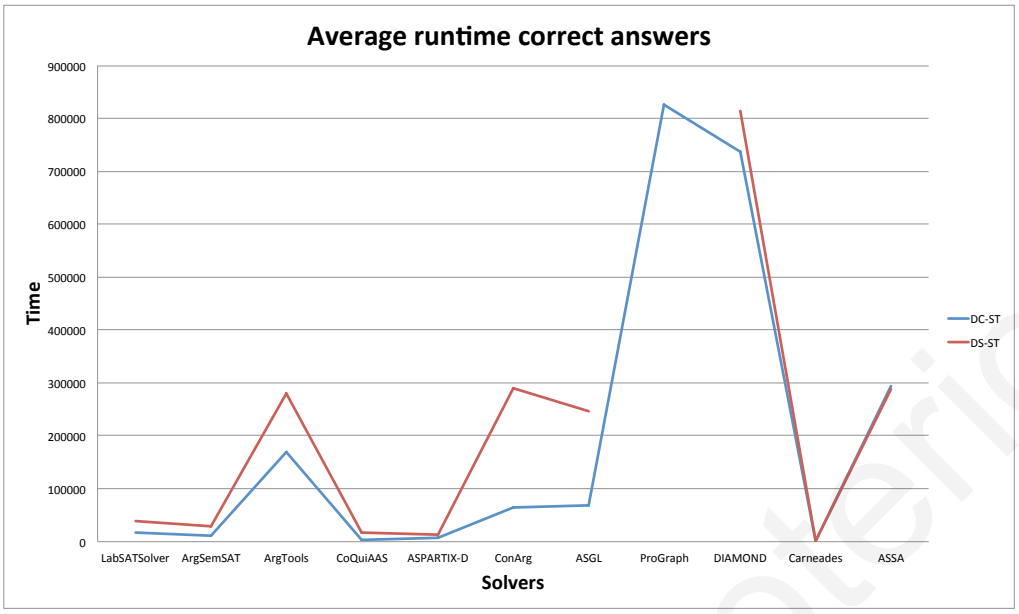


Figure 26: Average Runtime Correct Answers Based on ICCMA'15 Competition

5.2.2.2 ASSA$_G$

"If I knew how to take a good photograph, I'd do it every time."

Robert Doisneau

The following example discusses how the ASSA$_G$ algorithm is implemented and tested.

Example 25 Let AF $= \langle \mathcal{A}, \mathcal{R} \rangle$, where $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5, a_6\}$

and $\mathcal{R} = \{(a_1, a_5), (a_5, a_1), (a_5, a_2), (a_6, a_5)\}$, depicted in Figure 25.

Step 1: (Start with $\mathcal{G}_0 = \{\}$)

For the AF in Example 25, $n = 6$. Let $A_{6\times6} = \begin{pmatrix} 0&0&0&0&1&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 1&1&0&0&0&0 \\ 0&0&0&1&1&0 \end{pmatrix}$ be the adjacency matrix of AF and

$B_{1\times6} = (1\ 1\ 1\ 1\ 1\ 1)$.

To find matrix C, evaluate:

$B_{1\times6}A_{6\times6} = (1\ 1\ 1\ 1\ 1\ 1) \begin{pmatrix} 0&0&0&0&1&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 1&1&0&0&0&0 \\ 0&0&0&1&1&0 \end{pmatrix} = (1\ 1\ 0\ 1\ 2\ 0) = C$. From C, create set $\mathcal{G}_1 = \{a_3, a_6\}$.

$\mathcal{G}_2 = \mathcal{G}_1 \cup \mathcal{G}_0$ = $\{a_3, a_6\} \cup \{\}$ = $\{a_3, a_6\}$.

Step2:

From matrix C, create matrix $D = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$:

$DA = E = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$ = $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$ = E. From matrix $E = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$ find set $\mathcal{F} = \{a_4, a_5\}$.

Step3:

From the existing AF that contains arguments $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, create the new AF by removing the arguments $\{a_3, a_4, a_5, a_6\}$ and the attacks to and from these arguments. Let $\text{AF}_1 = \langle \mathcal{A}_1, \mathcal{R}_1 \rangle$ be the new argumentation framework that is depicted in Figure 27, where $\mathcal{A}_1 = \{a_1, a_2\}$ and $\mathcal{R}_1 = \{\}$. $\mathcal{G}_1 \neq \{\}$; thus, one can continue by going to ***Step 1*** with $\mathcal{G}_0 = \mathcal{G}_2 = \{a_3, a_6\}$.

For AF_1, $n = 2$. Let $A_{2\times2} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$, $B_{1\times2} = \begin{pmatrix} 1 & 1 \end{pmatrix}$

To find matrix C, evaluate:

$B_{1\times2}A_{2\times2} = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ = $\begin{pmatrix} 0 & 0 \end{pmatrix}$ = C. From C we create set $\mathcal{G}_1 = \{a_1, a_2\}$. $\mathcal{G}_2 = \mathcal{G}_1 \cup \mathcal{G}_0$ = $\{a_1, a_2\} \cup \{a_3, a_6\}$ = $\{a_1, a_2, a_3, a_6\}$.

Step2:

From matrix C, matrix $D = \begin{pmatrix} 1 & 1 \end{pmatrix}$ is created.

$DA = E = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ = $\begin{pmatrix} 0 & 0 \end{pmatrix}$ = E. From matrix $E = \begin{pmatrix} 0 & 0 \end{pmatrix}$, set $\mathcal{F} = \{\}$ is found.

Step3:

From the existing AF that contains arguments $\mathcal{A} = \{a_1, a_2\}$, the new AF is created by removing the arguments $\{a_1, a_2\}$ and the attacks to and from these arguments. Let $\text{AF}_2 = \langle \mathcal{A}_2, \mathcal{R}_2 \rangle$ be the new argumentation framework, where $\mathcal{A}_2 = \{\}$ and $\mathcal{R}_2 = \{\}$. $\mathcal{A} = \{\}$; thus, one must continue by going to ***Step 5*** with Grounded = $\mathcal{G}_2 = \{a_1, a_2, a_3, a_6\}$.

a_1 a_2

Figure 27: Example for Grounded Extension After the First Pass

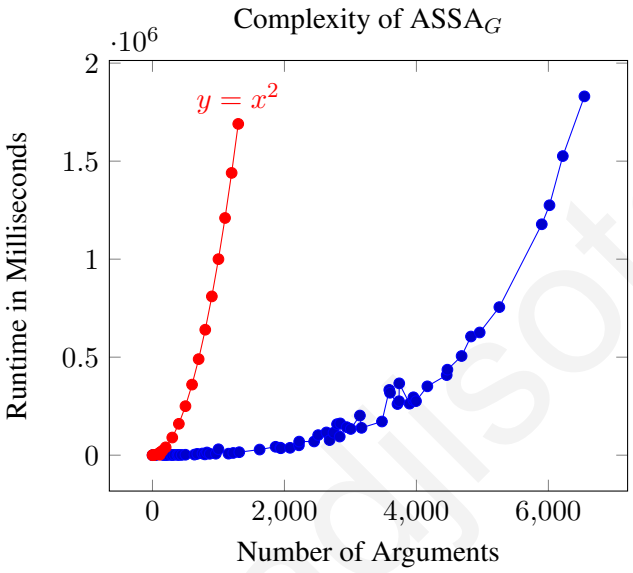


Figure 28: ASSA$_G$ is of Polynomial Complexity

The algorithm that was used to compute the grounded extensions is of polynomial complexity. The ASSA$_G$ algorithm was run to various examples, asking to find under a given AF the grounded extension. This demonstrates that implementing a polynomial algorithm with the matrix approach remains polynomial. The benchmark results were taken from the ICCMA'15 competition. In Figure 28, the x-axis represents the number of arguments, and the y-axis represents the runtime of ASSA$_G$ in milliseconds. Furthermore, the graph $y = x^2$ is also added for comparison.

The next experiment is more targeted since how the algorithm works is known it has to do an indebt check to find grounded extension. For this reason, Example 29 was created, which needs to go step by step in a predefined way to reach the end. Argument a_1 is labelled in, a_2 is labelled

$$a_1 \longrightarrow a_2 \longrightarrow \ldots \longrightarrow a_{1999} \longrightarrow a_{2000}$$

Figure 29: Experiment for Grounded Extension

out, a_3 is labelled in, and so forth. The runtime of ASSA$_G$ is 365185 milliseconds. The attack (a_1,a_{1000}) that labells argument a_{1000} out is then added, which was already labelled out. Adding this attack provided a similar outcome to splitting the AF into two smaller AFs that run parallel in the ASSA$_G$ algorithm. This example needed 164087 milliseconds to complete. Extending this thought process, the attacks $\{(a_1, a_{500}), (a_{501}, a_{1000}), (a_{1001}, a_{1500})\}$ are added to the initial AF; this example ran in 118998 milliseconds. The experiment continued by **removing arguments and their attacks**. By removing argument a_{1000} and the attacks that are related to this argument, which are $\{(a_{999}, a_{1000}), (a_{1000}, a_{1001})\}$, ASSA$_G$ finished in 187031 milliseconds. Finally, arguments a_{500}, a_{1000} and a_{1500} were removed and the attacks that are related to these arguments, which are $\{(a_{499}, a_{500}), (a_{500}, a_{501}), (a_{999}, a_{1000}), (a_{1000}, a_{1001}), (a_{1499}, a_{1500}), (a_{1500}, a_{1501})\}$. For this final experiment, ASSA$_G$ needed 84724 milliseconds to complete.

5.3 Dynamic Argumentation

Previously, a matrix was shown as a rectangular array, arranged in n of rows and m of columns ($n \times m$ matrix). An element of the matrix can be anything that provides information. For example, the following matrix provides the name, age, preferred color, and preferred fruit of my three children.

$$A = \begin{pmatrix} Styliana & Artemis & Giorgos \\ 5 & 3 & 1 \\ pink & purple & undefined \\ allfruits & banana & undefined \end{pmatrix}$$. It can be represented as a row matrix or as a column matrix. That is $A = \left(\begin{array}{c} Row_1 \\ \hline Row_2 \\ \hline Row_3 \\ \hline Row_4 \end{array}\right) = \left(\begin{array}{ccc} Styliana & Artemis & Giorgos \\ \hline 5 & 3 & 1 \\ \hline pink & purple & undefined \\ \hline allfruits & banana & undefined \end{array}\right)$

$$= \left(\begin{array}{c|c|c} Column_1 & Column_2 & Column_3 \end{array}\right) = \left(\begin{array}{c|c|c} Styliana & Artemis & Giorgos \\ 5 & 3 & 1 \\ pink & purple & undefined \\ allfruits & banana & undefined \end{array}\right).$$

Matrices can be large, and it might be difficult or confusing to follow and understand their structure. For this reason, matrices are written into **blocks**. Nice patterns such as the identity matrix or the zero matrix might be good partition points, or even repetitions of ones, zeroes, or other elements. For square matrices, it is often useful to choose the same partition points for columns and rows. In addition, there may be as many levels of partition points as desired. Blocks are matrices and therefore, they can be written as blocks using sub-blocks.

Systems that are continuously changing, such as decision making in argumentation can benefit from dynamic systems. Two types of changes exist in an AF: 1) adding or removing arguments and 2) adding or removing attacks. A naive and time consuming approach to find the new extensions would be to construct the extensions from scratch, ignoring all of the work that was done before the AF was updated. All of the knowledge that is gained and the mathematical operations that are

done are used when evaluating all of the extensions for the initial AF before it was updated, to quickly evaluate the new extensions for the updated AF. This is done using block matrices.

Block Matrices: Matrix multiplication can be a difficult and computational, complex procedure. When the AF is represented by an adjacency matrix to capture dynamic changes of the framework, matrix multiplication can be a repetitive procedure. For this reason, it might be useful to split matrices into smaller matrices that are often called sub matrices, and handle changes of the main matrix by working solely on the sub matrices. The disadvantage is a matrix must be split into sub matrices wisely, as dimensions need to match up in order to produce valid operations.

A block matrix is any matrix that is partitioned into smaller matrices called a block of matrices. Any matrix may be interpreted as a block matrix in many ways. The usual rules of matrix multiplication hold as long as the block sizes correspond. For example:

$$A = \begin{pmatrix} B & C \\ D & E \end{pmatrix} = \left(\begin{array}{cc|ccc} b_{1,1} & b_{1,2} & c_{1,1} & c_{1,2} & c_{1,3} \\ b_{2,1} & b_{2,2} & c_{2,1} & c_{2,2} & c_{2,3} \\ \hline d_{1,1} & d_{1,2} & e_{1,1} & e_{1,2} & e_{1,3} \\ d_{2,1} & d_{2,2} & e_{2,1} & e_{2,2} & e_{2,3} \\ d_{3,1} & d_{3,2} & e_{3,1} & e_{3,2} & e_{3,3} \end{array}\right), \text{ where } B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix},$$

$C = \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \end{pmatrix}$, $D = \begin{pmatrix} d_{1,1} & d_{1,2} \\ d_{2,1} & d_{2,2} \\ d_{3,1} & d_{3,2} \end{pmatrix}$ and $E = \begin{pmatrix} e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,1} & e_{3,2} & e_{3,3} \end{pmatrix}$. Let also consider block matrix $A_1 = \begin{pmatrix} B_1 & C_1 \\ D_1 & E_1 \end{pmatrix}$ and perform a block matrix multiplication $\Gamma = AA_1 = \begin{pmatrix} B & C \\ D & E \end{pmatrix}\begin{pmatrix} B_1 & C_1 \\ D_1 & E_1 \end{pmatrix} = \begin{pmatrix} BB_1 + CD_1 & BC_1 + CE_1 \\ DB_1 + ED_1 & DC_1 + EE_1 \end{pmatrix}$. As shown by the matrix multiplication $\Gamma = AA_1$, smaller matrix multiplications are evaluated to obtain the

end result. Specifically, the results, BB_1, CD_1, BC_1 CE_1, DB_1, ED_1, DC_1 and EE_1 are computed. When an attack is added or subtracted from the AF, this will reflect specific elements of the adjacency matrix, and will therefore only change some of the sub matrices that were created. In this way, the end result can be computed quickly by only computing the affected blocks.

Matrices can be partitioned in many ways, up to the point where each element of the matrix is solely considered. They can be partitioned by splitting them into columns or rows, as we shown below for the matrix multiplication $C_{n\times m} = A_{n\times k}B_{k\times m}$.

(**Partitions by columns:**) Partition matrices B and C (which is the resulting matrix), and get the result: $C = \left(\begin{array}{c|c|c|c} c_1 & c_2 & \dots & c_m \end{array}\right) = AB = A\left(\begin{array}{c|c|c|c} b_1 & b_2 & \dots & b_m \end{array}\right) = \left(\begin{array}{c|c|c|c} Ab_1 & Ab_2 & \dots & Ab_m \end{array}\right)$

(**Partitions by rows:**) Partition C and A by rows and to obtain the following result:

$$C = \left(\begin{array}{c} c_1 \\ \hline c_2 \\ \hline \dots \\ \hline c_n \end{array}\right) = AB = \left(\begin{array}{c} a_1 \\ \hline a_2 \\ \hline \dots \\ \hline a_n \end{array}\right) B = \left(\begin{array}{c} a_1B \\ \hline a_2B \\ \hline \dots \\ \hline a_nB \end{array}\right)$$

(**Partitions by rows and columns:**) Partition matrices A and B to obtain the result: $C = AB = \left(\begin{array}{c|c|c|c} a_1 & a_2 & \dots & a_k \end{array}\right) \left(\begin{array}{c} b_1 \\ \hline b_2 \\ \hline \dots \\ \hline b_k \end{array}\right) = a_1b_1 + a_2b_2 + \ldots + a_kb_k.$

With the result of each column or row in memory, which lines changed is shown and a new matrix multiplication can be computed solely for that line or row. In this way, matrix multiplication can be quickly identified adjusted when the elements of matrix S partially changed by evaluating only the affected block of matrices.

In the case where new arguments are added to the framework, the adjacency matrix $A_{n\times n}$ will become a new matrix B with dimensions $n+1 \times n+1$. These cases we can be partitioned as follows: $B = \left(\begin{array}{c|c} & c_{1,1} \\ A & \vdots \\ & c_{1,n} \\ \hline c_{n+1,1} \ldots c_{n+1,n} & c_{n+1,n+1} \end{array}\right)$

If an existing argument (i.e., argument a_k) was deleted, then the k^{th} row and column of the adjacency matrix would be deleted and therefore partition there. If more arguments where removed, then more partitions could be applied increased convenience.

We have extended ASSA and ASSA$_G$ to work into dynamic environments. We first run the program with the available information and provide the finished result. With this result, a dialogue is also presented to the user. In this dialogue, available arguments and attacks are written on the screen for the user to see and decide which of these arguments s/he wants to remove. The user can then end the program, deciding which of the available arguments or attacks they want to have removed or if new arguments with new attacks will be added to the system. The systems are then run, with consideration for the new information that has been provided. This approach, as already stated, is naïve, as it does not take results into consideration before the AF has been updated. As future work, the authors want to extend this dynamic system to conform with the block theory on matrices in order to run faster.

5.3.1 Summary

The main contribution of this chapter is the new approach to formalizing and computing argumentation through matrices in order to identify extensions. We show a new way of answering the question of whether or not a set of arguments, $\mathcal{S}$, is a conflict-free/admissible/complete/stable

and grounded extension of a given AF. Based on this concept, the algorithms ASSA and ASSA$_G$ were created. One of the features of ASSA is that it is not affected by the density of the problem (i.e. problems with a rich structure of strongly connected components and problems that feature many stable extensions) and because it is based the adjacency matrix one can understand the algorithm and the definitions fully with basic mathematic knowledge. Similarly, ASSA$_G$ can be fully understood with basic mathematic knowledge. We have shown that implementing a polynomial algorithm with the matrix approach remains polynomial.

Chapter 6

Conclusion and Future Work

"Twelve significant photographs in any one year is a good crop."

Ansel Adams

Argumentation is ubiquitous. Humans try to find the reason behind behavior, and use argumentation to make decisions and to explain behavior. Reasoning depends on the user, the current state of the world and the final goal. The need for an adaptive and dynamic argumentation that can help people reason in order to solve decision making problems, with incomplete information in a continually changing world is becoming a necessity.

A theoretical framework was created [35] that can adjust dynamically to changes and maintain a good representation of the operating world. Our argumentation, based formulation of the action language $\mathcal{E}$ for RAC, extends the argumentation based formulation of $\mathcal{E}$ [48] to accommodate arguments based on property observations, and arguments for backwards persistence [48, 38, 40, 39] in a domain description that will be used directly in the argumentation process rather than imposed as a-posteriori constraints.

Thus, this work can be seen as an example where the general theory of argumentation, which has been extensively and widely developed over the past two decades in AI (see e.g. [10, 67]),

finds a concrete application in addressing the foundational problems of temporal persistence and knowledge qualification. This synthesis of ideas opens up possibilities of extending the application of argumentation from 'static problems' to variations of these, where the problem environment is dynamically changing as new information becomes available. Using argumentation for reasoning about changes in the problem world domain offers a principled way to manage the changes in the AF, under which the application problem is expressed, thus extending the use of argumentation from static to 'dynamic problems'.

A real life type of application called 'Hotel for ME' was studied that uses argumentation as a decision-making process in order to show how such problems can be formalized and implemented in argumentation. The framework is influenced by the working environment and the user's preferences. The applications become personalized since the user profile is analyzed, including their needs and habits, by extracting them through a short dialogue with the user. Implementing the application in $Gorgias$ [1, 64], all of the rules and priorities that were used to reach the final conclusion were tracked which can the be used to convince the user to follow a recommended option. Through examples we have shown that for a single user at different time point, different results are produced illustrating the dynamic and personalized nature of the approach. We can extend this application to have a meta-level stage that are approximate closely enough or pass a predefined threshold.

For future work the authors will examine how to integrate the AF for reasoning about properties over time in Chapter 4 with argumentation-based approaches for decision making in Chapter 3 such that we have a fully argumentation based framework. Similarly, the authors want to apply this framework to planning problems and in particular, to the revision of plans as new unexpected information is acquired.

In particular, as the dialectic nature of argumentation is close to human reasoning, with recent studies from Cognitive Psychology [59] reinforcing this view, our argumentation-based approach for RAC can help its integration with wider forms of human reasoning, such as that of discourse comprehension, dialogue, and debate. Recent work on story comprehension [21] has shown how argumentation can play a significant role in formulating and automating the human process of comprehension. Similarly, in multi-agent system interaction and communication, several works (e.g., [56, 47]), have shown the suitability of using argumentation to model agent dialogues by exploiting the dialectic and game theoretic form of argumentation.

The dynamic nature of dialogues lends itself to a uniform argumentation based formalization for integrating reasoning about changes in the environment of communication with the various dialogue protocols. In particular, the incremental process of a dialogue can be modeled in terms of dynamic argumentation by generalizing the proof and game theories of computation [26, 62] for static argumentation. Allowing arguments to be time dependent permits adaptation over time by tracking the changes of arguments and the attacking relation between them. Real-life problems can then be mapped to AFs and constructed incrementally from the dynamic knowledge of the problem (e.g., the information exchanged at different time points of a dialogue).

For future work we will examine how we can integrate our framework for reasoning about properties over time with argumentation based approaches for decision making so that these decisions can be context sensitive over time. This will allow us to develop applications of advisory or recommendation systems, as in [20], that can adapt themselves as their external environment evolves. Similarly, we want to apply our framework to planning problems and in particular to the revision of plans as new unexpected information is acquired.

Another way to address these problems of dynamic development is to examine the link between our work and that of dynamic argumentation (see e.g. [7, 54]), where RAC with new

information in the time line of an application problem realizes a dynamic AF. New arguments and attacks between arguments are enabled as the information unfolds, particularly by observations that cause exogenous qualification. Researches can also explore how various problems that have been studied in the general setting of dynamic abstract argumentation can help address problems in RAC, such as overcoming an exogenous qualification by identifying what actions must occur such that certain conclusion would necessarily follow.

A matrix-based approach has been created that uses argumentation in order to reason. Through this approach extensions were computed for abstract AFs according to various semantics. When AFs are represented by their adjacency matrices and set as characteristic vectors, then matrix operations and additional tests can be used to verify whether a given set of arguments satisfies certain semantical criteria. This is a new way of answering the question of whether or not a set of arguments, $\mathcal{S}$, is a conflict-free/admissible/complete/stable/grounded extension of a given AF. This approach has been implemented in the ASSA and ASSA$_G$ systems [37, 36]. ASSA is ineffective as it needs to create a matrix with 2^n number of columns when trying to find all stable extensions which is an NP-complete problem. For this reason, all possible combinations of sets of arguments must be computed and each one must be checked if it passes the criteria. For this reason, ASSA cannot scale up to problems with more than twenty arguments, but it is not affected by the density of the attacks provided. Of course, ASSA can easily check if a given set is stable, as the complexity for this manner does not matter. On the other hand, ASSA$_G$ implements a polynomial complexity problem, and we have shown that our technique does not affect its complexity.

As future work, we plan to apply different optimization methods to improve computability, and to expand ASSA to all of the other semantics. We plan to study ways to make this more effective: (a) optimizations that come from the properties of argumentation semantics, (b) optimizations that exploit general optimizations on matrix operations, or finally (c) as stated in the work of [85, 18],

when given a set S, instead of considering it as a set, it should be considered as a single argument to continue from there. In this way, we can be mapped the problem into a simpler one and will likely produce better results.

The ASSA and ASSA$_G$ have been extended to work in dynamic environments. At this point, the user can add or remove arguments or attacks after the initial program provides an answer. All of the available nodes are then presented to the user with the possible attacks in order to avoid confusion when the user decides to remove arguments or attacks. If the user decides to add arguments or attacks then this information will also be needed, as the user may want to add attacks that does not exist. At this point, the new information that is provided by the user is considered while discarding all operations that were done up to this point and the program is run again. As future work we want to extend this dynamic system to conform with the block theory on matrices so the the dynamic re-computation can be improved.

We have shown that adding or removing targeted attacks and arguments in the AF can reduce the runtime of ASSA$_G$. We are interested in studying the addition of targeted dummy attacks to specific arguments of the AF, in order to split the AF into smaller virtual AF that will run parallel and provide labelling to arguments at an earlier time point of the procedure. If such labelling is done wisely the runtime of ASSA$_G$ will be reduced significantly. Other optimization techniques to examine are: Allowing our systems to use matrix calculation packages will improve its performance, and taking into the odd numbers of loops into consideration will reduce the number of checks that must be performed. Furthermore, distributed systems are used to manage all necessary matrix operations in order to have a system that can answer questions for a larger number of arguments.

Bibliography

[1] *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*. ACM, 2003.

[2] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. Symb. Log.*, 50(2):510–530, 1985.

[3] Leila Amgoud and Henri Prade. Using arguments for making and explaining decisions. *Artif. Intell.*, 173(3-4):413–436, 2009.

[4] Marko Balabanovic and Yoav Shoham. Content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, 1997.

[5] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011.

[6] Pietro Baroni and Massimiliano Giacomin. *Argumentation in Artificial Intelligence*, chapter Semantics of Abstract Argument Systems, pages 25–44. Springer US, Boston, MA, 2009.

[7] Ringo Baumann and Gerhard Brewka. Expanding argumentation frameworks: Enforcing and monotonicity results. In *Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010.*, pages 75–86, 2010.

[8] T Bench-Capon and D Dunne. Argumentation and dialogue in artificial intelligence. *IJCAI 2005 tutorial notes*, 2005.

[9] Trevor J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.*, 13(3):429–448, 2003.

[10] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artif. Intell.*, 171(10-15):619–641, 2007.

[11] Philippe Besnard and Anthony Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128(12):203 – 235, 2001.

[12] Guido Boella, Dov M. Gabbay, Alan Perotti, Leendert van der Torre, and Serena Villata. *Theorie and Applications of Formal Argumentation: First International Workshop, TAFA 2011. Barcelona, Spain, July 16-17, 2011, Revised Selected Papers*, chapter Conditional Labelling for Abstract Argumentation, pages 232–248. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[13] Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.*, 93:63–101, 1997.

[14] John-Adrian Bondy and U. S. R. Murty. *Graph theory*. Graduate texts in mathematics. Springer, New York, London, 2007. OHX.

[15] Gerhard Brewka. On the relationship between defeasible logic and well-founded semantics. In *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, pages 121–132, 2001.

[16] Martin W. A. Caminada, Walter Alexandre Carnielli, and Paul E. Dunne. Semi-stable semantics. *J. Log. Comput.*, 22(5):1207–1254, 2012.

[17] Martin W. A. Caminada and Dov M. Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2-3):109–145, 2009.

[18] Claudette Cayrol and Yuming Xu. The matrix approach for abstract argumentation frameworks. Technical Report RR–2015-01–FR, IRIT, Universit Paul Sabatier, Toulouse, 2015.

[19] Günther Charwat, Wolfgang Dvorák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.*, 220:28–63, 2015.

[20] Carlos Iván Chesñevar, Ana Gabriela Maguitman, and Guillermo Ricardo Simari. A first approach to argument-based recommender systems based on defeasible logic programming. In *NMR*, pages 109–117, 2004.

[21] Irene-Anna Diakidoy, Antonis C. Kakas, Loizos Michael, and Rob Miller. Story Comprehension through Argumentation. In *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA 2014)*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 31–42, Scottish Highlands, U.K., September 2014. IOS Press.

[22] Yannis Dimopoulos and Antonis C. Kakas. Logic programming without negation as failure. In *Logic Programming, Proceedings of the 1995 International Symposium, Portland, Oregon, USA, December 4-7, 1995*, pages 369–383, 1995.

[23] Yannis Dimopoulos and Alberto Torres. Graph theoretical structures in logic programs and default theories. *Theor. Comput. Sci.*, 170(1-2):209–244, 1996.

[24] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

[25] Phan Minh Dung, Paolo Mancarella, and Francesca Toni. Computing ideal sceptical argumentation. *Artif. Intell.*, 171(10-15):642–674, 2007.

[26] P.M. Dung, R.A. Kowalski, and F. Toni. Dialectic proof procedures for assumption-based, admissible argumentation. *Artificial Intelligence*, 170(2):114 – 159, 2006.

[27] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. ASPARTIX: implementing argumentation frameworks using answer-set programming. In *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, pages 734–738, 2008.

[28] Stefan Ellmauthaler and Hannes Strass. The DIAMOND system for computing with abstract dialectical frameworks. In *Computational Models of Argument - Proceedings of COMMA 2014, Atholl Palace Hotel, Scottish Highlands, UK, September 9-12, 2014*, pages 233–240, 2014.

[29] Morten Elvang-Gøransson, Paul J. Krause, and John Fox. Dialectic reasoning with inconsistent information. *CoRR*, abs/1303.1467, 2013.

[30] Xiuyi Fan, Robert Craven, Ramsay Singer, Francesca Toni, and Matthew Williams. Assumption-based argumentation for decision-making with preferences: A medical case study. In *CLIMA*, pages 374–390, 2013.

[31] Xiuyi Fan and Francesca Toni. Decision making with assumption-based argumentation. In *TAFA*, pages 127–142, 2013.

[32] Alejandro Javier García and Guillermo Ricardo Simari. Defeasible logic programming: An argumentative approach. *TPLP*, 4(1-2):95–138, 2004.

[33] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Logic Programming*, 17:301–322, 1993.

[34] Marie Pierre Gleizes, Andrea Omicini, and Franco Zambonelli, editors. *Engineering Societies in the Agents World V, 5th InternationalWorkshop, ESAW 2004, Toulouse, France, October 20-22, 2004, Revised Selected and Invited Papers*, volume 3451 of *Lecture Notes in Computer Science*. Springer, 2005.

[35] E. Hadjisoteriou and A. Kakas. Reasoning about actions and change in argumentation. *Argument & Computation*, 0(0):1–27, 2016.

[36] Evgenios Hadjisoteriou. Computing Argumentation with Matrices. In Claudia Schulz and Daniel Liew, editors, *2015 Imperial College Computing Student Workshop (ICCSW 2015)*, volume 49 of *OpenAccess Series in Informatics (OASIcs)*, pages 29–36, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[37] Evgenios Hadjisoteriou and Michael Georgiou. Assa: Computing stable extensions with matrices. *System Descriptions of the First International Competition on Computational Models of Argumentation (ICCMA15)*, pages 62–65, 2015.

[38] Evgenios Hadjisoteriou and Antonis Kakas. *Logic Programs, Norms and Action: Essays in Honor of Marek J. Sergot on the Occasion of His 60th Birthday*, chapter Argumentation and the Event Calculus, pages 103–122. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[39] Evgenios Hadjisoteriou and Antonis C. Kakas. Argumentation and temporal persistence. In *Proceedings of the 7th Panhellenic Logic Symposium*, pages 89–94, 15-19 July, Patras, 2009.

[40] Evgenios Hadjisoteriou and Antonis C. Kakas. Argumentation and temporal persistence. In *ICCSW*, pages 31–38, 2011.

[41] Steve Hanks and Drew V. McDermott. Nonmonotonic logic and temporal projection. *Artif. Intell.*, 33(3):379–412, 1987.

[42] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems - An Introduction*. Cambridge University Press, 2010.

[43] Antonis Kakas and Pavlos Moraitis. Argumentation based decision making for autonomous agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 883–890, New York, NY, USA, 2003. ACM.

[44] Antonis C. Kakas, Leila Amgoud, Gabriele Kern-Isberner, Nicolas Maudet, and Pavlos Moraitis. Aba: Argumentation based agents. In *ArgMAS*, pages 9–27, 2011.

[45] Antonis C. Kakas, Robert A. Kowalski, and Francesca Toni. Abductive logic programming. *J. Log. Comput.*, 2(6):719–770, 1992.

[46] Antonis C. Kakas, Paolo Mancarella, and Phan Minh Dung. The acceptability semantics for logic programs. In *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994*, pages 504–519, 1994.

[47] Antonis C. Kakas, Nicolas Maudet, and Pavlos Moraitis. Flexible agent dialogue strategies and societal communication protocols. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, pages 1434–1435, 2004.

[48] Antonis C. Kakas and Rob Miller. A simple declarative language for describing narratives with actions. *J. Log. Program.*, 31(1-3):157–200, 1997.

[49] Antonis C. Kakas, Rob Miller, and Francesca Toni. An argumentation framework of reasoning about actions and change. In *LPNMR*, pages 78–91, 1999.

[50] Henry A. Kautz. The logic of persistence. In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science.*, pages 401–405, 1986.

[51] R Kowalski and M Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.

[52] Eric Lehman, Tom Leighton, and Albert R Meyer. Mathematics for computer science. Technical report, 2010.

[53] Beishui Liao. *Efficient Computation of Argumentation Semantics*. Intelligent systems series. Academic Press, 2014.

[54] Beishui Liao, Li Jin, and Robert C Koons. Dynamics of argumentation systems: A division-based method. *Artificial Intelligence*, 175(11):1790–1814, 2011.

[55] Tariq Mahmood and Francesco Ricci. Improving recommender systems with adaptive conversational strategies. In *Hypertext*, pages 73–82, 2009.

[56] Peter McBurney and Simon Parsons. Dialogue games for agent argumentation. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 261–280. Springer US, 2009.

[57] Norman McCain and Hudson Turner. Causal theories of action and change. In *In Proc. AAAI-97*, pages 460–465, 1997.

[58] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[59] H. Mercier and D. Sperber. Why Do Humans Reason? Arguments for an Argumentative Theory. *Behavioral and Brain Sciences*, 34(2):57–74, 2011.

[60] Rob Miller and Murray Shanahan. The event calculus in classical logic - alternative axiomatisations. *Electron. Trans. Artif. Intell.*, 3(A):77–105, 1999.

[61] S. Modgil and Martin W.A. Caminada. Proof theories and algorithms for abstract argumentation frameworks. In I. Rahwan and G. Simari, editors, *Argumentation in Artif. Intell.*, pages 105–129. Springer Publishing Company, Incorporated, 2009.

[62] Sanjay Modgil and Martin Caminada. Proof theories and algorithms for abstract argumentation frameworks. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 105–129. Springer US, 2009.

[63] Sanjay Modgil and Henry Prakken. A general account of argumentation with preferences. *Artif. Intell.*, 195:361–397, 2013.

[64] Victor Noël and Antonis C. Kakas. Gorgias-c: Extending argumentation with constraint solving. In *LPNMR*, pages 535–541, 2009.

[65] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[66] Michael J. Pazzani and Daniel Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.

[67] I. Rahwan and G.R. Simari. *Argumentation in Artificial Intelligence*. Springer Publishing Company, Incorporated, 2009.

[68] Iyad Rahwan and Guillermo R. Simari. *Argumentation in Artificial Intelligence*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[69] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[70] Paul Resnick and Hal R. Varian. Recommender systems - introduction to the special section. *Commun. ACM*, 40(3):56–58, 1997.

[71] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[72] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[73] Scott G Schreiber. *Aristotle on false reasoning: language and the World in the Sophistical Refutations*. SUNY Press, 2003.

[74] Murray Shanahan. *Solving the frame problem - a mathematical investigation of the common sense law of inertia*. MIT Press, 1997.

[75] Gilbert W Stewart. Introduction to matrix computations. 1973.

[76] Andrew James Stothers. On the complexity of matrix multiplication. 2010.

[77] Loren G. Terveen and David W. McDonald. Social matching: A framework and research agenda. *ACM Trans. Comput.-Hum. Interact.*, 12(3):401–434, 2005.

[78] M. Thielscher. The Qualification Problem: A Solution to the Problem of Anomalous Models. *AIJ*, 131(1–2):1–37, 2001.

[79] Michael Thielscher. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111:277–299, 1999.

[80] M. Thimm and S. Villata. System Descriptions of the First International Competition on Computational Models of Argumentation (ICCMA'15). *ArXiv e-prints*, October 2015.

[81] A. Torres. Negation as failure to support. In L. M. Pereira and A. Nerode, editors, *Logic Programming and Non-Monotonic Reasoning: Proc. of the Second International Workshop. Cambridge*, pages 223–243. MIT Press, MA, 1993.

[82] Quoc Bao Vo and Norman Y. Foo. Reasoning about action: An argumentation-theoretic approach. *Journal of Artificial Intelligence Research*, 24:465–518, 2005.

[83] Toshiko Wakaki and Katsumi Nitta. Computing argumentation semantics in answer set programming. In *New Frontiers in Artificial Intelligence, JSAI 2008 Conference and Workshops, Asahikawa, Japan, June 11-13, 2008, Revised Selected Papers*, pages 254–269, 2008.

[84] Y. Wu and M.W.A. Caminada. A labelling-based justification status of arguments. *Studies in Logic*, 3(4):12–29, 2010.

[85] Yuming Xu. A matrix approach for computing extensions of argumentation frameworks. *CoRR*, abs/1209.1899, 2012.

Appendix A

Proofs

A.1 Chapter 4 Proof

Proposition 1

Proof 17 By contradiction. Let D be a domain and E be an admissible extension that is not consistent with the t-propositions in D. Therefore, there exists a fluent f and a time point t s.t. 'f holds-at t' belongs to D and $E \vdash \neg\text{HoldsAt}(f,t)$ (resp. '$\neg f$ holds-at t' belongs to D and $E \vdash\text{HoldsAt}(f,t)$). Consider the case $E \vdash \neg\text{HoldsAt}(f,t)$ (the respective case is completely analogous). Let B be the set of argument rules $B = \{PO[f,t]\}$. We will show that B is an attack against E that is not attacked back by E, thus contradicting the admissibility of E. B attacks E because $B(D) \cup B \vdash\text{HoldsAt}(f,t)$ minimally and $E \vdash \neg\text{HoldsAt}(f,t)$ and there is no argument that has higher priority than $PO[f,t]$ at t. This attack can only be attacked back by deriving $\neg\text{HoldsAt}(f,t)$. The (minimal) derivation of $\neg\text{HoldsAt}(f,t)$ by E will require either a persistence argument or a local generation argument whose conclusion will be $\neg\text{HoldsAt}(f,t)$. The priority relation in Definition 22 assigns to these lower priority than $PO[f,t]$, and hence since B does not contain any other argument rule such derivations cannot form an attack against B. The only other possibility for E to derive $\neg\text{HoldsAt}(f,t)$ is through $PO[f,t]$ but this is not possible due to the point wise consistency of D.

Lemma 2

Proof 18 Let f be a fluent, $t_n < t_m$ two time points s.t. $E \vdash\text{HoldsAt}(f,t_n)$ and $E \vdash\text{HoldsAt}(f,t_m)$ (the case where $E \vdash \neg\text{HoldsAt}(f,t_n)$ and $E \vdash \neg\text{HoldsAt}(f,t_m)$ is analogous). Assume that $E \vdash \neg\text{HoldsAt}(f,t)$ for some time $t \in (t_n,t_m)$. We will show that this leads to a contradiction of the admissibility of E. As time is discrete and so there are finitely many time points in the interval (t_n,t_m) we can consider the first time point T in $[t_n,t_m)$ at which we have a change of the value of f by E, i.e. $E \vdash\text{HoldsAt}(f,T)$, $E \vdash \neg\text{HoldsAt}(f,T+1)$. Consider now the set $S = E_1 \cup \{PFP(f,T+1;T)\}$ where E_1 is a minimal subset of E that derives f at T. Then $S \vdash\text{HoldsAt}(f,T+1)$ and so it is a potential attack on E. In fact, since there are no termination points for f in $[t_n,t_m)$ and observations of $\neg f$ in $[t_n,t_m)$ the extension E cannot derive $\neg f$ at $T+1$ with an argument stronger than $PFP(f,T+1;T)$ and so S attacks E. The only possible way for E to attack back S is via a conflict at $T+1$ by deriving $\neg f$ at $T+1$, as it cannot attack S on its E_1 subset since this would mean that E is inconsistent.
Again, since there are no termination points for f in $[t_n,t_m)$ and observations of $\neg f$ in $[t_n,t_m)$

the extension E can derive $\neg f$ at $T+1$ either by a forward persistence of $\neg f$ starting from a time point before T or by backward persistence of $\neg f$ from a time point after $T+1$, but not both due to the compactness property of an admissible extension (see Definition 28) that explicitly excludes this. Note that if there is an initiation point for f at t_m and hence we have the possibility to have in the AF a $NG_B[f, t_m]$ argument for $\neg f$ at t_m this cannot belong to E as then E would be inconsistent (at t_m since E derives f at t_m). In the first case, this cannot constitute an attack as $PFP(f, T+1; T)$ is stronger than the forward persistence of $\neg f$, starting from a time point before T.
In the second case this backward persistence of $\neg f$ must have its origin from a point after t_m as there are no termination points or observations before t_m. But, then the set $S^{'} = E^{'} \cup \{PBP(f, T+1; t_m)\}$ where $E^{'}$ is a minimal subset of E that derives f at t_m attacks E. This can only be defended by E with a backwards persistence argument starting before t_m which is not possible as there are no termination points for f or observation points for $\neg f$. Note that if we have a chain of backward persistence starting from a time after t_m to an intermediate time point, t_i, before t_m but after $T+1$ and then another backwards persistence from this to $T+1$, the extension E still needs to contain a persistence argument for $\neg f$ starting after t_m and reaching this intermediate time point before t_m. Then the attack, $S^{''} = E^{'} \cup \{PBP(f, t_i; t_m)\}$, analogous to $S^{'}$, cannot be attacked back by E. Similarly, if the way that E derives $\neg f$ at $T+1$ is through a forward persistence from a time point before t_n to a time point, $t^{'}$, after $T+1$ and then a backwards persistence from $t^{'}$ to $T+1$, then E would not be able to attack back the attack, $S^{'''} = E_1 \cup \{PFP(f, t^{'}; T)\}$, where E_1 is a minimal subset of E that derives f at T. Hence one of the attacks, S, $S^{'}$, $S^{''}$ or $S^{'''}$, cannot be attacked back by E and so E would not be admissible. Contradiction.

Theorem 1

Proof 19 Let D be given with a countable number of h-propositions and t-propositions. We will show the existence of an admissible extension E by induction on the number of h-propositions in D.

Base Case: Number of h-propositions is zero.
To prove this case, a second induction on the number of t-propositions was used, which we have also assumed to be countable.
Base case: Number of t-propositions is zero.
Let $E = \bigcup_f \{PA[f, 0], PFP[f, t; 0]$ for every fluent f and every time point $t > 0\}$. By construction E is complete, consistent, and clearly satisfies the compactness property of an admissible extension. It remains to show that E attacks all of its attacks. To have an attack A on E, A needs to derive $\neg$HoldsAt(f, t) for some fluent f and time point t, so as to have a contrary conclusion with E. This can only happen if A contains the argument $NA(f, 0)$. But then E attacks back A through its argument $PA(f, 0)$.
Induction Step: Suppose that there exist a (complete) admissible extension, $E^{'}$, of the corresponding AF $(B(D^{'}), \mathcal{A}, <)$ for any domain $D^{'}$ that contains up to k t-propositions and no h-propositions. We will prove that the statement holds when a domain D contains $k+1$ t-propositions (and no h-propositions).
Consider the last t-proposition and let its time point be t_{k+1}. If it is removed from D, the resulting domain, $D^{'}$ will only contain k many t-propositions. By the induction hypothesis, let $E^{'}$ be a (complete) admissible extension of $D^{'}$. We will use $E^{'}$ to construct an extension, E, for D as follows. Let f be the fluent to which the t-proposition taken out of D refers to and consider the last

t-proposition in $D^{'}$ on f. Let its time be t_l, $t_l \leq t_{k+1}$. We will construct an admissible extension E for D as the union, $E = E_f^- \cup E_f^+$, of two sets of arguments obtained and extended from $E^{'}$. The following cases are obtained:

Case(a) there does not exist such a t-proposition, i.e. the observation at t_{k+1} is the only one in D that refers to f. Assume that this observation at t_{k+1}, is that f holds (the case where $\neg f$ holds at t_{k+1} is analogous). Then let
$E_f^- = E^{'} \setminus \{$All arguments in $E^{'}$ that refer to f for any time point$\}$ and
$E_f^+ = \{PO[f, t_{k+1}], PFP[f, t_1; t_{k+1}], PBP[f, t_2; t_{k+1}] \mid t_1 > t_{k+1}, t_2 < t_{k+1}\}$.

Case(b) the (last) observation at time t_l also states that f holds (i.e. as does the observation for the fluent f at time t_{k+1}). Let $E_f^- = E^{'}$ and $E_f^+ = \emptyset$.

Case(c) the (last) observation at time t_l is opposite to the observation for the fluent f at time t_{k+1}, i.e. at time t_l f is observed not to hold whereas at t_{k+1} it is observed to hold. Then let
$E_f^- = E^{'} \setminus \{$All arguments in $E^{'}$ that refer to f for any time point, $t \geq t_l\}$. Choose a time point $T_i \in [t_l, t_{k+1})$ and let
$E_f^+ = \{NO[f, t_l], NFP[f, t; t_l] \mid t \in (t_l, T_i]\} \cup \{PO[f, t_{k+1}], PFP[f, t_1; t_{k+1}], PBP[f, t_2; t_{k+1}]$
$\mid t_1 > t_{k+1}, t_2 \in (T_i, t_{k+1})\}$.

By construction, $E = E_f^- \cup E_f^+$ in every case is consistent, complete and compact. To show that E is admissible let us consider the possible new attacks on E that result from the changes to $E^{'}$ for the three cases above. Case(a): new attacks need to prove $\neg HoldsAt(f, t)$ for some time point t. This can only happen by a potential attack A that contains the argument $NA[f, 0]$ as in D there does not exist any h-propositions and the observation at t_{k+1} is the only one referring to f. But any attack of this form on $E_f^+ = \{PO[f, t_{k+1}], PFP[f, t_1; t_{k+1}], PBP[f, t_2; t_{k+1}] \mid$ $t_1 > t_{k+1}, t_2 < t_{k+1}\}$ can be counterattacked by E_f^+ on the argument $NA[f, 0]$ of A. Case(b): E does not differ from $E^{'}$ and hence it remains admissible. Case(c): let us consider new attacks on E that are not attacks on $E^{'}$ after and before t_{k+1}.
(i) after t_{k+1}: any potential new attack on E on the fluent f can only come through forward persistence of $\neg f$ from some time point $T \leq t_l < t_{k+1}$ as there aren't any h-propositions in D and the last observation for f before t_{k+1} is at time t_l. But this is not an attack on $E_f^+ = \{NO[f, t_l], NFP[f, t; t_l] \mid t \in (t_l, T_i]\} \cup \{PO[f, t_{k+1}], PFP[f, t_1; t_{k+1}],\ PBP[f, t_2; t_{k+1}] \mid$ $t_1 > t_{k+1}, t_2 \in (T_i, t_{k+1})\}$ as forward persistence arguments from latter time points are stronger.
(ii) before t_{k+1} and t_l new potential attacks on E can only occur by the observation at t_{k+1}. Let us then consider a potential new attack, $A^{'}$ on E given by $\{PO[f, t_{k+1}], PBP[f, T; t_{k+1}]$ for some $T < t_l\}$. We then consider how $E^{'}$ and E derive $\neg f$ at T. There are two possibilities, either by $\{NO[f, t_l], NBP[f, T; t_l]$ in which case $A^{'}$ does not attack $E^{'}$, or through some forward persistence argument from some other observation for $\neg f$ before t_l or from a negative assumption for $\neg f$ at 0. In both of these latter cases, $E^{'}$ would counter attack the attack $A^{'}$ as forward and backwards persistence arguments are non-comparable.
(iii) before t_{k+1} but after t_l attacks on E are based on a conflict between forward and backwards persistence in the interval (t_l, t_{k+1}). These are counter-attacked by $E_f^+ = \{NO[f, t_l], NFP[f, t; t_l]$ $\mid t \in (t_l, T_i]\} \cup \{PO[f, t_{k+1}], PFP[f, t_1; t_{k+1}],\ PBP[f, t_2; t_{k+1}] \mid t_1 > t_{k+1}, t_2 \in (T_i, t_{k+1})\}$ in E since conflicting forward and backwards persistence arguments have the same priority.

Induction step: Suppose that there exist an (complete) admissible extension, $E^{'}$, of the corresponding AF $(B(D^{'}), \mathcal{A}, <)$, for any domain $D^{'}$ that contains k or less h-propositions. We will prove that the statement holds when a domain D contains $k+1$ h-propositions.
Consider the last h-proposition that occurred at time point t_{k+1}. If it is removed from D, the resulting domain, $D^{'}$, will only contain k many h-propositions. By the induction hypothesis, let

$E^{'}$ be a (complete) admissible extension of $D^{'}$. We will use $E^{'}$ to construct an extension E for D as follows. Let E be given by:

$$E = \bigcup_{f} E_f^- \cup \bigcup_{f} E_f^+$$

where E_f^- is defined from $E^{'}$ by removing some arguments for the fluent f and E_f^+ are some new arguments for the fluent f. For any fluent f we have the following cases:

(Case1) t_{k+1} is an initiation point for f in $E^{'}$.

(Case2) t_{k+1} is a termination point for f in $E^{'}$. This case is similar to case 1. Note that t_{k+1} can be both an initiation and a termination point for the same fluent f. In such a case we can pick either case in the construction.

(Case3) t_{k+1} is neither an initiation or a termination point for f in $E^{'}$. In this case $E_f^- = E_f^{'}$ and $E_f^+ = \{\}$, i.e. nothing (related to the fluent f) is removed or added.

Hence let us assume that t_{k+1} is an initiation point for f in $E^{'}$ and consider the following changes in $E^{'}$ in order to build an admissible extension E.

These changes to $E^{'}$ will all be after time point t_{k+1}. Note that there is no need to make any changes before t_{k+1}, as the existence of the initiation point for f at t_{k+1} can enable new possible attacks on $E^{'}$ (and the resulting E that we are constructing) only through the argument $NG_B[f, t_{k+1}]$ and either backwards persistence of $\neg f$ from this or forwards persistence. In the case of backwards persistence, this attack will always be counter attacked by $E^{'}$ as backwards persistence arguments are non-comparable to other arguments, or they will be weaker than earlier backwards persistence in $E^{'}$. For the new forwards persistence of $\neg f$ from $NG_B[f, t_{k+1}]$ possible attacks we are shown below that these persistence arguments will be weaker than the new arguments E_f^+ that we add in $E^{'}$.

Consider therefore, the first t-proposition referring to f after t_{k+1} at time, $t_n > t_{k+1}$. The following cases are obtained:

(a_1) If there does not exist such a t-proposition let
$E_f^- = E^{'} \backslash \{$All arguments in $E^{'}$ that refer to f, for any $t > t_{k+1}\}$ and
$E_f^+ = \{PG_F[f, t_{k+1}], PFP[f, t; t_{k+1} + 1] \mid t > t_{k+1} + 1\}$.
(a_2) The first t-proposition after t_{k+1}, at time t_n, confirms (i.e. we observe that f holds at t_n) the initiation point for the fluent f at time t_{k+1}. Let
$E_f^- = E^{'} \backslash \{$All arguments in $E^{'}$ that refer to f, for any $t \in (t_{k+1}, t_n)\}$ and
$E_f^+ = \{PG_F[f, t_{k+1}], PFP[f, t; t_{k+1} + 1] \mid t \in (t_{k+1} + 1, t_n)\}$.
(a_3) The first t-proposition after t_{k+1} at time t_n is opposite (i.e. we observe that f does not holds at t_n) to the initiation point for the fluent f at t_{k+1}. Choose $T_i \in [t_{k+1}, t_n)$. Let
$E_f^- = E^{'} \backslash \{$All arguments in $E^{'}$ that refer to f, for any $t \in (t_{k+1}, t_n]\}$ and
$E_f^+ = \{PG_F[f, t_{k+1}], PFP[f, t; t_{k+1} + 1] \mid t \in (t_{k+1} + 1, T_i]\} \cup \{NO(f, t_n),$
$NBP[f, t^{'}; t_n] \mid t^{'} \in (T_i, t_n)\}$.

(b) Changes to $E^{'}$ before or at time point t_{k+1}:
Consider the last t-proposition referring to f or h-proposition that can generate f or $\neg f$ in $E^{'}$ at time $t_m \leq t_{k+1}$.
(b_1) If there does not exist such a t-proposition or an h-proposition, i.e. the h-proposition at t_{k+1} is the only one in D that refers to f and there does not exist a t-proposition in D at $t_m \leq t_{k+1}$ that refers to f. Let
$E_f^- = E^{'} \backslash \{$All arguments in $E^{'}$ that refer to f, for any $t \leq t_{k+1}\}$ and
$E_f^+ = \{NA[f, 0], NFP[f, t; 0] \mid t < t_{k+1}\}$.

(b_2) If there exists a t-proposition or an h-proposition at $t_m \leq t_{k+1}$ opposite to the initiation point for the fluent f at time t_{k+1} (i.e. the observation at t_m is that f does not hold at t_m or at t_m we have a termination point for f), then no change before t_{k+1} of $E^{'}$ is needed.
(b_3) Similarly, if there exists a t-proposition at $t_m \leq t_{k+1}$ where f is observed to hold at t_m or there exits an h-proposition at $t_m \leq t_{k+1}$ s.t. t_m is also an initiation point for the fluent f, then no change before t_{k+1} of $E^{'}$ is needed.

By construction E is complete, consistent, and compact. It must further demonstrated that E attacks back at any of its attacks. Consider all possible new attacks on E through the different cases that are given above. Case a_1. Any new potential attack on E after t_{k+1} needs to prove $\neg HoldsAt(f,t)$ for some $t > t_{k+1}$. Such a minimal proof can only be built from either an observation for the fluent $\neg f$ before or equal to t_{k+1}, a termination for the fluent f before or equal to t_{k+1} or an assumption for $\neg f$ at 0. But all three cases do not constitute an attack on the $E_f^+ = \{PG_F[f,t_{k+1}], PFP[f,t;t_{k+1}+1] \mid t > t_{k+1}+1\}$ as between conflicting forward persistence arguments higher priority have the arguments occurring at a latter time point. Hence there are no new attacks on this new part of E.
Case a_2. A potential new attack on E needs to prove $\neg HoldsAt(f,t)$ for $t \in (t_{k+1}, t_n)$. Such a proof can only be built from an observation or a generation point before or equal to t_{k+1} or by an assumption at 0. Similar to case a_1 these are not attacks. Another possibility are proofs starting after t_n by observations of $\neg f$ by backward persistence. This attacks E but it is counter attacked by E because no priority is given between conflicting forward persistence arguments over backward persistence arguments.
Case a_3. Any new potential attack on E at $(t_{k+1}, T_i]$ needs to prove $\neg HoldsAt(f,t)$ for any $t \in (t_{k+1}, T_i]$. Such a minimal proof can only be built from either an assumption for the fluent $\neg f$ at 0, an observation for the fluent $\neg f$ before or equal to t_{k+1} or by a termination for the fluent f before or equal to t_{k+1}. But similarly to case a_1 all three are not attacks. Other new attacks require a proof of $\neg f$ starting after t_n. But these attacks are counter attacked (similarly to case a_2). Also new attacks on E at $(T_i, t_n]$ need to prove $HoldsAt(f,t)$ for any $t \in (T_i, t_n]$. Such a minimal proof can only be built from either an assumption for the fluent f at 0, an observation for the fluent f before T_i or by an initiation for the fluent f before T_i. All three are attacks but can be counter attacked by the constructed E.

Theorem 2

Proof 20 Let D be given and let M be a language $\mathcal{E}$ model of D. We will show the existence of a corresponding admissible extension E by induction on the number of h-propositions in D.

Base case: Number of h-propositions is zero.
Case(1): Number of t-propositions (observations) in D is zero.
By definition of the Models of $\mathcal{E}$, for every fluent f either $M(f,t) = true$ for every t or $M(f,t) = false$ for every t. For any fluent f s.t. $M(f,t) = true$ for every t, let
$E_f = \{PA(f,0)\} \cup \{PFP(f,t;0) | for\ every\ t\}$ and for any fluent $f^{'}$ s.t. $M(f^{'},t) = false$ for every t, let $E_{f'} = \{NA(f^{'},0)\} \cup \{NFP(f^{'},t;0) | for\ every\ t\}$. Let us then define E as follows:

$$E = \bigcup_{f|M(f,t)=true} E_f \cup \bigcup_{f'|M(f',t)=false} E_{f'}$$

By construction E corresponds to the model M. We will show that E is an admissible extension. By construction E is consistent, complete and compact. In order for an argument set A, to attack E, for some fluent f and time point t a contrary conclusion should exist, i.e. $A \vdash \neg\text{HoldsAt}(f,t)$ when $M(f,t) = true$ (resp. $A \vdash \text{HoldsAt}(f^{'},t)$ when $M(f^{'},t) = false$) This is only possible if A contains $NA(f,0)$ (resp. contains $PA(f^{'},0)$. E therefore attacks back A through its argument $PA(f,0)$ (resp. $NA(f^{'},0)$. Furthermore, by construction E corresponds to the given model M.
Case(2): The domain D contains t-propositions (observations).
Given the existence of the model, M, of D and the fact that there are no h-propositions in D all observations in D for a given fluent f are either all for f to hold or all for $\neg f$ to hold at the different time points where we have observations. Therefore as in Case(1), all fluents have a constant truth value in M at all time points. Thus we can define E as before:

$$E = \bigcup_{f|M(f,t)=true} E_f \cup \bigcup_{f'|M(f',t)=false} E_{f'}$$

New attacks on E based on observations are not possible since by construction of E such an attack would require an observation for a fluent f which is contrary to the constant truth value for f in M.

Induction Step: Suppose that the statement holds for any domain s.t. the number of h- propositions is less or equal to k.
Let D be a domain with $k+1$ h-propositions and let M be a model of D. Consider the last h-proposition and assume that this occurred at the time point t_{k+1}. We then consider the new domain $D^{'}$ resulting by taking out of D this last h-proposition as well as all the t-propositions after this time point. The domain $D^{'}$ contains k many h-propositions. By the induction hypothesis there exist an admissible extension $E^{'}$ for $(B(D^{'}), \mathcal{A}^{'}, <)$ corresponding to the model, $M^{'}$, of $D^{'}$ obtained from the model M by replacing the truth value of every fluent, for every time after t_{k+1}, to be the same as its truth value in M at t_{k+1}. We then define E, from $E^{'}$ by considering for each fluent the initiation or termination status of the time point t_{k+1} and the possible t-propositions in D after this time point.

Let f be any fluent:

(i) Let t_{k+1} be an initiation point for the fluent f in M and that M assigns true to f after t_{k+1}[1] . Let $E_f^- = E_f^{'} \setminus \{$All arguments in $E^{'}$ with conclusion $\neg HoldsAt(f,T)$ or $HoldsAt(f,T)$, for any $T > t_{k+1}$ $\}$ and $E_f^+ = \{PG(f,t_{k+1})\} \cup \{PFP(f,t_1;t_{k+1}+1) \mid t_1 > t_{k+1}+1\}$. Here and below $E_f^{'}$ denotes the subset of $E^{'}$ of all the arguments that refer to the fluent f.

(ii) Let t_{k+1} be a termination point for the fluent f in M. Let $E_f^- = E_f^{'} \setminus \{$All arguments in $E^{'}$ with conclusion $HoldsAt(f,T)$ or $\neg HoldsAt(f,T)$, for any $T > t_{k+1}$ $\}$ and $E_f^+ = \{NG(f,t_{k+1})\} \cup \{NFP(f,t_1;t_{k+1}+1) \mid t_1 > t_{k+1}+1\}$.

(iii) Let t_{k+1} be neither an initiation nor a termination point for the fluent f in M. Let $E_f^- = E_f^{'}$ and $E_f^+ = \emptyset$.
(iv) Let t_{k+1} is both an initiation and a termination point for f in M. The model M of $\mathcal{E}$ will non deterministically have chosen the value $true$ or $false$ for f after t_{k+1}. If this value is $true$ then E_f^- and E_f^+ are defined as in case (i). Otherwise, if it is $false$ then E_f^- and E_f^+ are defined as in case (ii).

[1]Note that t_{k+1} can be both an initiation and a termination point for f in M but any given model the fluent f will have either the value true or false (see case (iv)).

We then define E as follows:

$$E = \bigcup_f E_f^- \cup \bigcup_f E_f^+$$

By construction and the inductive hypothesis on $E^{'}$, E is consistent, complete and compact. Also by construction E corresponds to M. We also note that if there exists t-propositions on a fluent f after time t_{k+1} then they must all give the same truth value for f since otherwise the domain D would not have a model. This truth value of f in M will coincide with the derivation of f or $\neg f$ after t_{k+1} by the constructed E.

It remains to show that E attacks all its attacks. New attacks on E that were not attacks on $E^{'}$ can occur by the possible generation point at t_{k+1}. Since t_{k+1} is the last generation point new attacks on E can be built by backwards persistence from this, conflicting with forward persistence arguments from $t < t_{k+1}$. But then, E can counter attack any of these attacks since forward persistence has same priority than conflicting backwards persistence. Note also that observations of f after t_{k+1} cannot generate any new attacks on E since if t_{k+1} is an initiation point in M for a fluent f then these observations must all be for f to be true (otherwise M would not be a model) and hence by the construction of E the observations cannot generate an argument conflicting with the arguments in E. Similarly, when t_{k+1} is a termination point in M.

Theorem 3

Proof 21 Let E be a given admissible extension of D and consider the interpretation H^E that correspond to E. H^E is well-defined as admissible extensions are complete. Suppose that H^E is not a model of D in language $\mathcal{E}$. Then one of the four conditions of the definition 20 of a model must be violated. Given the result of Proposition 1, H^E cannot violate property (4) of Definition 20, since admissible extensions are consistent wrt the t-propositions in the domain. Thus H^E violates one of properties (1-3).

Violation of property 1: There exist a fluent f and time points t_1, t with $t_1 < t$ s.t. $E \vdash$HoldsAt(f, t_1) and $E \vdash \neg$HoldsAt(f, t) (or vice-versa) and there exists no termination point in $[t_1, t)$ relative to E (or H^E). Since time is discrete we can choose t_1, t to be consecutive times. We then add in D a new h-proposition 'A happens-at t_1' s.t. 'A terminates F'. In the new domain D_1 obtained from this addition this violation of the model property is removed.

Violation of property 2: There exist a fluent f and time points $t_1 < t$ s.t. t_1 is an initiation point of f relative to H^E, $E \vdash \neg$HoldsAt(f, t) and there exists no termination point in (t_1, t) for f in H^E. Due to the discrete nature of time we can choose t to be the closest time point to t_1 where there is such a violation, i.e. t is the closest time point after t_1 where $E \vdash \neg$HoldsAt(f, t).
We then add in D a new h-proposition 'A happens-at $T^{'}$' at $T^{'} = t - 1$ s.t. 'A terminates F'. In the new domain D_2 obtained from this addition the violation of the model property is removed. Note that $T^{'}$ can be equal to t_1 in which case language $\mathcal{E}$ has models for either f or $\neg f$ to hold after t_1.

Violation of property 3: The treatment of this violation is analogous to the violation of property (2) and results in a new domain D_3 that removes one such violation.

We then consider the domain $D_{123} = D_1 \cup D_2 \cup D_3$. If H^E is not a model of D_{123} then we repeat the above construction. Due to the finiteness of the h-propositions and t-propositions in D and the discrete nature of the time line this process terminates and results in the required new domain $D^{'}$.

A.2 Chapter 5 Proof

Proposition 2

Proof 22 Let $AF = \langle \mathcal{A}, \mathcal{R} \rangle$ with n arguments and $A = (a_{i,j})$ its adjacency matrix. Let $\mathcal{S} \subseteq \mathcal{A}$ be a set of arguments with S its column vector representation. The product (of the two matrices A and S) AS is defined as follows: $(AS)_{i,1} = \sum_{t=1}^{n}((i,t)_{th}$ element of $A \times (t,1)_{th}$ element of $S) = \sum_{t=1}^{n} a_{i,t}s_{t,1}$. Based on Definition 32 and Definition 31, $(AS)_{i,1}$ is an addition of zeroes if at least one of the entries $a_{i,t}$ or $s_{t,1}$ is zero as $0 \times 1 = 1 \times 0 = 0 \times 0 = 0$ or an addition of ones if both entries $a_{i,t} = s_{t,1} = 1$ since $1 \times 1 = 1$. Intuitively, it is an addition of ones if and only if there exists an attack from a_i to a_t in the AF and $a_t \in \mathcal{S}$.

Theorem 4

Proof 23 Let $AF = \langle \mathcal{A}, \mathcal{R} \rangle$ and $A = (a_{i,j})$ its adjacency matrix. We will show that A^m is the number of walks of length m from a_i to a_j in AF. By induction on m.

Base case: $m = 1$. $A^1 = A$ and by Definition 31, it is true.

Induction Step: Suppose the theorem holds for any $A^{m-1}, m-1 > 0$ and we will prove that it is true for A^m. Let $A^{m-1} = (b_{i,j})$ and $A^m = (c_{i,j})$. Since $A^m = A^{m-1} \times A$, from the matrix multiplication we get:
$(c_{i,j}) = \sum_{t=1}^{n}((i,t)_{th}$ element of $A^{m-1} \times (t,j)_{th}$ element of $A) = \sum_{t=1}^{n} b_{i,t}a_{t,j}$.
Every walk from argument i to j of length m is constructed by a walk of length $m-1$ from i to t and by a walk of length 1 from t to j. Since there are $b_{i,t}$ walks of length $m-1$ from a_i to a_t (Induction Step) and a_t attacks a_j (walks of length 1), the total number of walks with length m is $\sum_{t=1}^{n} b_{i,t}a_{t,j} = (c_{i,j})$.

Proposition 7

Proof 24 Based on Proposition 3, $\gamma_i \in \Gamma$ shows how many times argument $a_i \in \mathcal{A}$ is attacked by S. Therefore, when $\gamma_i \neq 0 \in \Gamma$ means that a_i is connected to $\mathcal{S}$ in γ_i ways, (i.e. argument a_i is attacked in γ_i ways by the arguments in $\mathcal{S}$). This does not conform with the definition of a conflict free set. To pass the test arguments that are attacked should not be part of S, thus $\int_i = 0 \in \mathcal{S}$.

Proposition 8

Proof 25 Based on Definition 1. Matrix $\Gamma = (\gamma_i) = AS$ shows if $\mathcal{S}$ is under attack (see Proposition 2). When (1) $\Gamma = \mathbf{0}$, $\mathcal{S}$ passes the defending test as no attackers exist and there are no arguments to defend against. When (2) $\Gamma \neq \mathbf{0}$, i.e. attackers exist, we check if $\mathcal{S}$ counter attacks them. $\gamma_i \neq 0$ Proposition 2, shows the number of attacks. To retrieve this critical information we create $\Delta^{(i)}, \forall i$ that $\gamma_i \neq 0$ and evaluate $E^{(i)} = A\Delta^{(i)}$. By Proposition 2, $E^{(i)}$ shows if any arguments in A attack $\Delta^{(i)}$. To ensure that when counter attacks exist, they come from arguments in $\mathcal{S}$, we allow the restriction $\forall i, \exists e_k^{(i)} \in E^{(i)}$ when $e_k^{(i)} \neq 0$, $s_k \neq 0$, i.e. for every attack there exists at least one argument counter attacking it and this argument belongs to $\mathcal{S}$.

Proposition 10

Proof 26 Let $AF = \langle \mathcal{A}, \mathcal{R} \rangle$,$\mathcal{S} \subseteq \mathcal{A}$ be a set of arguments with S its column vector representation. Based on Definition 2 $\mathcal{S}$ should be conflict-free, i.e. it should pass the conflict-free test and every argument not in $\mathcal{S}$ should be attacked by an argument in $\mathcal{S}$. $\Gamma = S^T A$ is a row vector where its entry $(S^T A)_{i,1}$ shows (see Proposition 3) how many times argument $a_i \in \mathcal{A}$ is attacked by $\mathcal{S}$. Fulfilling the constrain $\forall i$ s.t. $s_i = 0, \gamma_i \neq 0$ we make sure that every argument not in $\mathcal{S}$ should be attacked by an argument in $\mathcal{S}$.

Appendix B

Parametric Space for Hotels

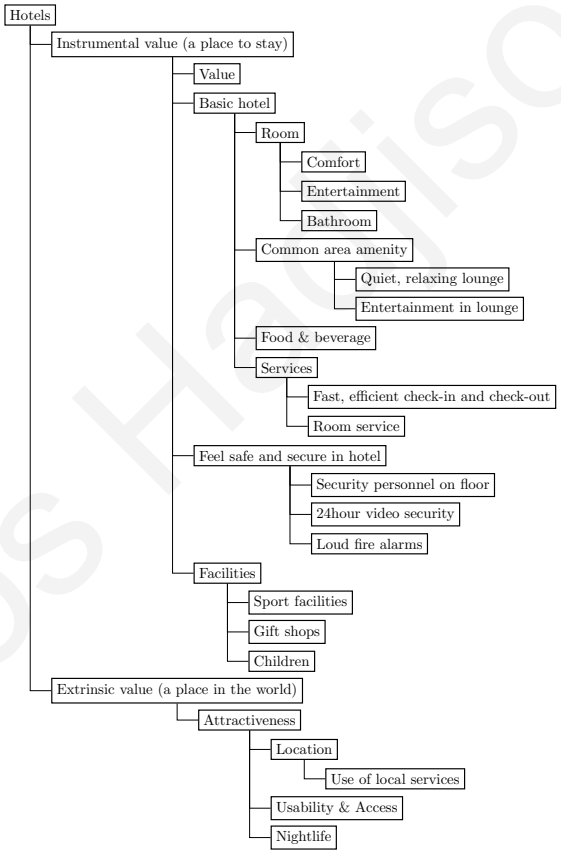


Figure 30: Hotels

Appendix C

Original Event Calculus

The original EC is formalized as a logic program using negation as failure to capture persistence. It was designed to deal with incomplete narrative information where updates to the theory (knowledge base) are done additively and incrementally. New information in the form of new events can cause the start or the end of zero or more time periods. In particular, they can affect the end points of existing $after(e\ u)$ and $before(e\ u)$ time periods and this in turn affects the possible conclusion of whether the property u holds or not at some time point. The following example is taken from [51].

Example 26 Let John be a person working for company C at position P_1. The event promoting John from position P_1 to position P_2 will end the period of John working at position P_1 and start a new period of John working at position P_2. The previous knowledge is not deleted as promoting John does not delete the period where he was working at position P_1.

The original EC uses the two time period terms, $after(e\ u)$ and $before(e\ u)$ where:

$after(e\ u)$ names a time period after the event e as a function of the event e. The parameter u names the relationship that is associated with the period. The end of $after(e\ u)$ is undefined unless extra information that can help us derive the end of the period $after(e\ u)$ is added to the theory.

$before(e\ u)$ names a time period before the event e as a function of the event e. The parameter u names the relationship associated with the period. The start of $before(e\ u)$ is undefined unless extra information that can help us derive the start of the period $before(e\ u)$ is added to the theory.

Predicates $after(e\ u)$ and $before(e\ u)$ are treated symmetrically and the latter allows backwards reasoning. Stating that the end of $after(e\ u)$ (resp. the start of $before(e\ u)$) is undefined means that $after(e\ u)$ (resp. $before(e\ u)$) might last forever or until a new event with extra information will determines its end. Some events can inform us about the future (e.g., the event hire denotes the start of a period), the past (e.g., the event fire or leave denotes the end of a period). Furthermore, there are events (e.g., promote) that inform us about the end of a period and the beginning of a new one.

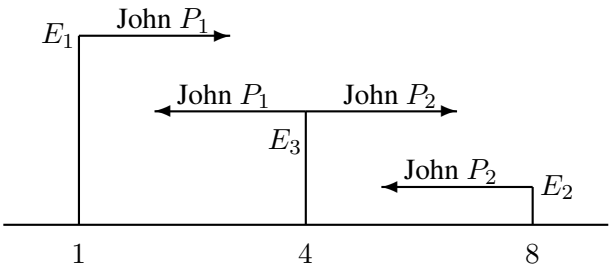


Figure 31: After Update (3)

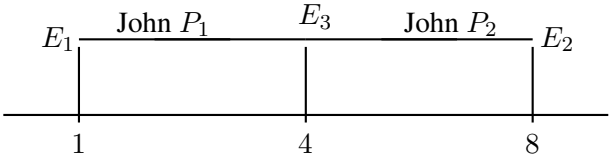


Figure 32: Conclusion

Example 27 Let the following narrative:
(1) John was hired for the position P_1 at time 1.
(2) John left from position P_2 at time 8.
(3) John was promoted from position P_1 to position P_2 at time 4.

Updating the knowledge base (initially it is empty) with the narratives (1)-(3) can be represented pictorially as illustrated in Figure 31. Each sentence (event) in Example 27 can be seen as an update to the knowledge base.

After update (1), the term $after(E_1\ P_1)$ names the time period after the event E_1. The end of $after(E_1\ P_1)$ is undefined; therefore at this point it is assumed that it lasts forever. Similarly, after update (2); the term $before(E_2\ P_2)$ names the time period before the event E_2. The starting end of $before(E_2\ P_2)$ is also undefined. After the final update (3), we can conclude that the ends of $after(E_1\ P_1)$ and $before(E_2\ P_2)$ are revealed, and $after(E_1\ P_1) = before(E_3\ P_1)$ and $after(E_3\ P_2) = before(E_2\ P_2)$. Pictorially this is shown in Figure 32.

Let e and $e^{'}$ be two events. In order to conclude that two periods are the same (i.e., $after(e\ u) = before(e^{'}\ u)$) original EC uses rules like the following one:

$$
\begin{aligned}
after(e\ u) = before(e^{'}\ u) \text{ if } & Holds(after(e\ u)) \\
\text{and } & Holds(before(e^{'}\ u)) \\
\text{and } & e < e^{'} \\
\text{and not } & Broken(e\ u\ e^{'})
\end{aligned}
$$

Here, $Holds(p)$ expresses the fact that there is a time period p that an associated property holds. $Broken(e\ u\ e^{'})$ indicates that, given the information that is available, the relation u was interrupted between the events e and $e^{'}$.

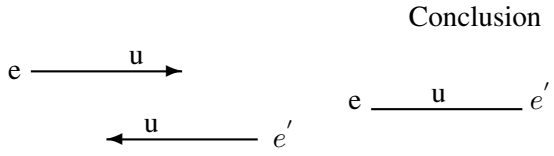


Figure 33: Identical

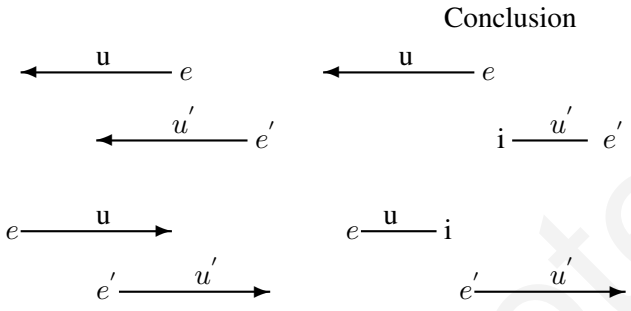


Figure 34: Exclusive

C.0.1 Deriving the End Points

The original EC uses several rules to determine the beginning of a period or the end of a period. When two time periods interact, there are three cases that cover all possible scenarios: (1) identical, (2) exclusive, and (3) incompatible. Assume that $e < e^{'}$ are two events.

In the first case, as shown above, the two periods that the events generate become identical, as illustrated in Figure 33 (i.e., $after(e\ u) = before(e^{'}\ u)$). In this case, it is natural to conclude that the start and the end of the time periods were found.

As shown in Figure 34 when $before(e\ u)$ comes into conflict with $before(e^{'}\ u^{'})$ (e.g., when $u^{'} = \neg u$) then there must be a start point i of $before(e^{'}\ u^{'})$ at or after e. When $after(e\ u)$ comes into conflict with $after(e^{'}\ u^{'})$ then there must be an end point i of $after(e\ u)$ at or before $e^{'}$.

Finally, as shown in Figure 35, when $after(e\ u)$ comes into conflict with $before(e^{'}\ u^{'})$ then there is an end of $after(e\ u)$ at or before the start of $before(e^{'}\ u^{'})$.

In both of the last two cases, the narrative has some missing information.

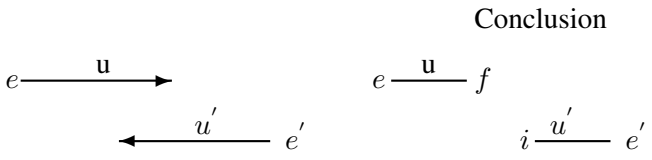


Figure 35: Incompatible

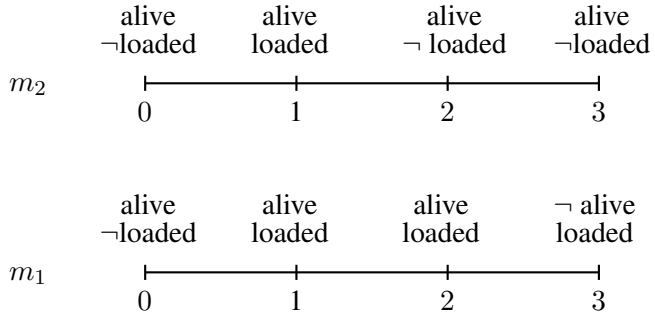


Figure 36: Yale Shooting Problem

C.1 Frame and Qualification Problem

The **qualification problem** exists when at a time point t_1 there is an initiation point for fluent F, but at a latter time point where $t_2 > t_1$, $\neg F$ is observed, or when at t_1 F is observed and at $t_2 > t_1$ $\neg F$ is observed. When this result cannot be explained, it is called a qualification problem. In these two cases, there is an exogenous qualification problem since exogenous reason exists to 'explain' these behaviors that our description does not conform.

Consider the Yale Shooting Problem introduced by [41], which is directly related to the frame problem (fluent values do not change unless an action occurs that has the ability to change them). The Yale shooting Problem scenario contains a gun and a victim. There are two fluent names alive and loaded and three actions are produced in the following way: 1) load (the gun), 2) wait and 3) shoot (the victim). Initially, the victim is alive and the gun is not loaded. One possible model m_1 would be for the victim to not be alive after performing the action shoot. Another possible model m_2, again logically correct, would be for the gun to mysteriously become unloaded before shooting and therefore the victim survives; these two models are depicted in Figure 36.

Appendix D

Codes

To download and run the programs please visit our website[1] .

D.1 Code: 'Hotel for ME'

Listing D.1: Hotel for Me

```
% Some facts
rule(f1_o(H), hotel(H), []) :- between(1,500,H).
rule(f2_t(T), timepoint(T), []) :- between(1,12,T).

% Some priority rules
rule(pr1_(H,T), prefer(f12_p(H,T), f11_p(H,T)), []).
rule(pr1cr_(H,T), prefer(f12_cr(H,T), f11_cr(H,T)), []).
rule(pr2_(H,T), prefer(r14_p(H,T), r13_p(H,T)), []).
rule(pr3_(H,T), prefer(r15_p(H,T), r13_p(H,T)), []).
rule(pr4_(H,T), prefer(r17_n(H,T,A), r16_n(H,T,A)), []).

% Users: For this narrative we have 5 users
% klelia, alexandros john, christos and giorgos
rule(r16_u(U),user(U),[]):-member(U,[klelia,
        alexandros,john,christos,giorgos]).

% User may travel for business or pleasure
% R = Reason for visiting
rule(r16_r(R), reason(R), []) :- member(R,[business,pleasure]).

% User may travel alone, with family (+children 0-5) or
% with friends, W = Travelling with
rule(r19_r(W), travel_with(W), []) :- member(W,[alone,
        family(children(N)),friends]),between(0,5,N).
```

[1] http://www.mertjiandata.com/assa.html

```
% If user travels for business alone or with family and no
% children user only cares about availability near the desired
% area and for the hotel to have a conference_room:
% U = User, H = Hotel, T = Timepoint, A = Area, R =
% Reason for visiting, W = Travelling with
rule(r17_all(U,H,T,A,R,W),hotel_for_all(U,H,T,A,R,W),[user(U),
                    hotel(H),timepoint(T),area(A),
                    conference_room(H,T),near(H,A),reason(R)]):-
        member(A,[nicosia,larnaca,limasol,paphos,troodos]),
    R = business, member(W,[alone,family(children(0))]).

% If user travels for business with 1-3 children, user
% only cares about play_ground and pool
% U = User, H = Hotel, T = Timepoint, A = Area, R = Reason
% for visiting,W = Travelling with
rule(r17_all(U,H,T,A,R,W),hotel_for_all(U,H,T,A,R,W),[user(U),
            hotel(H),timepoint(T),area(A),near(H,A),reason(R),
            play_ground(H,T),pool(H,T)]):-
        member(A,[nicosia,larnaca,limasol,paphos,troodos]),
        R = business,
        member(W,[family(children(N))]),
        between(1,3,N).

% If user travels for business with friends user
% only cares about pool and good night life
% U = User, H = Hotel, T = Timepoint, A = Area, R = Reason
% for visiting, W = Travelling with
rule(r17_all(U,H,T,A,R,W),hotel_for_all(U,H,T,A,R,W),[user(U),
                hotel(H),timepoint(T),
                area(A),near(H,A),reason(R),pool(H,T),
                good_nightlife(H,T,A)]):-
        member(A,[nicosia,larnaca,limasol,paphos,troodos]),
        R = business,
        member(W,[friends]).

% If user travels for business with 4-5 children, user
% only cares about special_rates
% U = User, H = Hotel, T = Timepoint, A = Area, R = Reason
% for visiting,W = Travelling with
rule(r17_all(U,H,T,A,R,W),hotel_for_all(U,H,T,A,R,W),[user(U),
            hotel(H),timepoint(T),area(A),near(H,A),
            reason(R),special_rates(H,T)]):-
        member(A,[nicosia,larnaca,limasol,paphos,troodos]),
        R = business,
        member(W,[family(children(N))]),
        between(4,5,N).
```

```
% If user travels for pleasure with friends user only cares
% about cool_season, crowded and good night life
rule(r18_all(U,H,T,A,R,W),hotel_for_all(U,H,T,A,R,W),[user(U),
        hotel(H),timepoint(T),reason(R),area(A),near(H,A),
        cool_season(H,T),crowded(H,T),
        good_nightlife(H,T,A)]):-
    member(A,[nicosia,larnaca,limasol,paphos,troodos]),
    R = pleasure,
    member(W,[friends]).

% If user travels for pleasure with family and children
% user only cares about pool, play_ground and special_rates
rule(r20_all(U,H,T,A,R,W),hotel_for_all(U,H,T,A,R,W),[user(U),
        hotel(H),timepoint(T),reason(R),area(A),
        near(H,A),play_ground(H,T),
        pool(H,T),special_rates(H,T)]):-
    member(A,[nicosia,larnaca,limasol,paphos,troodos]),
    R = pleasure,
    member(W,[family(children(N))]),
    between(1,5,N).

% If user travels for pleasure with family and no children,
% user only cares about good_nightlife and pool
rule(r21_all(U,H,T,A,R,W),hotel_for_all(U,H,T,A,R,W),[user(U),
        hotel(H),timepoint(T),reason(R),
        area(A),near(H,A),pool(H,T),
        good_nightlife(H,T,A)]):-
    member(A,[nicosia,larnaca,limasol,paphos,troodos]),
    R = pleasure,
    member(W,[family(children(0))]).

% If user travels for pleasure and travels alone only cares
% about pool good_nightlife and to be crowded
rule(r21_all(U,H,T,A,R,W),hotel_for_all(U,H,T,A,R,W),[user(U),
    hotel(H),timepoint(T),reason(R),area(A),near(H,A),
    crowded(H,T),pool(H,T),good_nightlife(H,T,A)]):-
    member(A,[nicosia,larnaca,limasol,paphos,troodos]),
    R = pleasure,
    member(W,[alone]).

% Initial program: Match all parameters
rule(r21_me(U,H,T,A,R,W),hotel_for_me(U,H,T,A,R,W),[
    hotel_for_all(U,H,T,A,R,W)]).
% New attack relation
% Allow user to specifically say which parameter is the
% strongest among: special_rates(H,T),cool_season(H,T),
% crowded(H,T),pool(H,T), play_ground(H,T),good_nightlife(H,T,A),
```

```
% conference_room(H,T). If user prefers special_rates
rule(r22_me(U,H,T,A,R,W),hotel_for_me1(U,H,T,A,R,W,L1),[
                                hotel_for_all(U,H,T,A,R,W),L1]).

% New attack relation
% Allow user to specifically say which parameter does not
% want among: special_rates(H,T),cool_season(H,T),crowded(H,T),
% pool(H,T),play_ground(H,T), good_nightlife(H,T,A),
% conference_room(H,T)
%
rule(r32_me(U,H,T,A,R,W),hotel_for_me2(U,H,T,A,R,W,DL1),[
                                hotel_for_all(U,H,T,A,R,W),neg(DL1)]).

% New attack relation
% Allow user to specifically say which parameter does not want
% (DL1) among:
% special_rates(H,T),cool_season(H,T),crowded(H,T),pool(H,T),
% play_ground(H,T),
% good_nightlife(H,T,A), conference_room(H,T) and again
% among thouse one that user likes (L2)
rule(r32a_me(U,H,T,A,R,W),hotel_for_me3(U,H,T,A,R,W,DL1,L2),
            [hotel_for_me2(U,H,T,A,R,W,DL1),L2]).
% or the following rule which is the same
% rule(r32a_me(U,H,T,A,R,W),hotel_for_me3(U,H,T,A,R,W,DL1,L2),
% [hotel_for_me1(U,H,T,A,R,W,L2),neg(DL1)]).
% This use gives duplicates
% rule(r32a_me(U,H,T,A,R,W),hotel_for_me3a(U,H,T,A,R,W,DL1,L2),
%[hotel_for_me2(U,H,T,A,R,W,DL1),hotel_for_me1(U,H,T,A,R,W,L2)]).

% New attack relation
% Allow user to specifically say two parameters (L1 and L2) that
% user likes (are the strongest) among:
% special_rates(H,T),cool_season(H,T),crowded(H,T),pool(H,T),
% play_ground(H,T),good_nightlife(H,T,A), conference_room(H,T)
%
rule(r42_me(U,H,T,A,R,W),hotel_for_me4(U,H,T,A,R,W,L1,L2),
[hotel_for_me1(U,H,T,A,R,W,L1),L2]).

% Final attack relation
% Allow user to specifically say three parameters (L1 and L2 L3)
% that user likes (are the strongest) among:
% special_rates(H,T),cool_season(H,T),crowded(H,T),pool(H,T),
% play_ground(H,T),good_nightlife(H,T,A), conference_room(H,T)
rule(r43_me(U,H,T,A,R,W),hotel_for_me5(U,H,T,A,R,W,L1,L2,L3),
[hotel_for_me4(U,H,T,A,R,W,L1,L2),L3]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Questions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Question 1 —— General:
% prove([hotel_for_me(U,H,T,A,R,W)], Delta).
% prove([hotel_for_me1(U,H,T,A,R,W,L1)], Delta).
% prove([hotel_for_me2(U,H,T,A,R,W,DL1)], Delta).
% prove([hotel_for_me3(U,H,T,A,R,W,DL1,L2)], Delta).
% prove([hotel_for_me4(U,H,T,A,R,W,L1,L2)], Delta).
% prove([hotel_for_me5(U,H,T,A,R,W,L1,L2,L3)], Delta).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Question 2 —— Specific:
% prove([hotel_for_me(klelia,H,5,nicosia,business,family(
% children(3)))], Delta).
% prove([hotel_for_me(klelia,H,1,larnaca,pleasure,family(children
% (0)))], Delta).
% prove([hotel_for_me(klelia,H,1,larnaca,pleasure,friends)], Delta).
% prove([hotel_for_me(alexandros,H,4,troodos,business,alone)], Delta).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Question 3
% prove([hotel_for_me(alexandros,H,T,nicosia,pleasure,alone)],
% Delta).
% prove([hotel_for_me1(alexandros,H,T,nicosia,pleasure,alone,
% pool(H,T))], Delta).
% prove([hotel_for_me2(alexandros,H,T,nicosia,pleasure,alone,
% play_ground(H,T))], Delta).
% prove([hotel_for_me3(alexandros,H,T,nicosia,pleasure,alone,
% play_ground(H,T),pool(H,T))], Delta).
% prove([hotel_for_me4(alexandros,H,T,nicosia,pleasure,alone,
% pool(H,T),good_nightlife(H,T,A))], Delta).
% prove([hotel_for_me5(alexandros,H,T,nicosia,pleasure,alone,
% pool(H,T),good_nightlife(H,T,A),special_rates(H,T))], Delta).
```

D.2 Code: Main sections of '*ASSA*'

Listing D.2: Main Code for ASSA

```
public int isConflictFreeNode(Group p){
int exit_code=1;
if (CONFLICTFREELIST.size() > 0){
 for (int i=0; i<CONFLICTFREELIST.size(); i++){
  Group e= (Group)CONFLICTFREELIST.elementAt(i);
  if (e.isEqual(p)==0){
   exit_code=0;}}}
return exit_code;}
//*************************************************
public void calculateStable(Matrix m1, Matrix m2){
double [ ][ ] B = m2.getArray();
double [ ][ ] C = m1.getArray();
int rows = B.length;
int cols = B[0].length;
int nodesSize = rows;
double[ ][ ] CV = new double [rows][cols];
CV=B;
CV = transposeMatrix (CV);    // ALL
C  = transposeMatrix (C);
rows = C.length;
cols = C[0].length;
nodesSize = rows;
int j=0;
int q=0;
int flag=0;
for (int i=0; i<cols; i++){
 Group group = new Group();
 while(j<rows){
  if ((CV[j][i]==0)&&(C[j][i]>=1)){
    flag ++;}
  else if ((CV[j][i]==1)&&(C[j][i]>=0)){
   flag ++;
   Node node = new Node(getNodeByPosition(j));
   group.addIntoGroup(node);}
  else{
   flag --;}
  j++;}
  if (flag==nodesSize){
   flag=0;
   if (isConflictFreeNode(group)==0){
        STABLELIST.add(group);}}
  else{
                group = new Group();
```

```
                    flag=0;}
  j=0;}}
//*************************************************
public void calculateConflictFree(Matrix m1, Matrix m2){
double[ ][ ] A = m1.getArray();
double[ ][ ] B = m2.getArray();
int rows = A.length;
int cols = A[0].length;
int j=0;
int n=0;
int t=0;
boolean found = false;
for (int i=0; i<cols; i++){
Group group = new Group();
while(j<rows){
 if (A[j][i] > 0){
  found =true;
  n++;
  Node node = new Node(getNodeByPosition(j));
  group.addIntoGroup(node);
  if (B[j][i]==0){
   t++;}}
  j++;}
j=0;
if (found){
 if (t==n){
  CONFLICTFREELIST.add(group);}
 else{}
 t=n=0;}}}
//*************************************************
public String [][] getAllCombinationsMatrix(int numberOfNodes){
String[ ][ ] matrix = null;
String[ ][ ] tmp = null;
int columns=numberOfNodes;
int rows=powerOf(2,numberOfNodes);
matrix = new String[rows][columns];
tmp= new String[1][numberOfNodes];
for (int i=rows-1; i>=0; i--){
 tmp=getbinaryMatrix (numberOfNodes,i);
 for (int w=0; w<1; w++){
        for (int j=0; j<numberOfNodes; j++){
                        matrix[i][j]=tmp[w][j];
        }}
}return matrix;}
```

D.3 Code: Main sections of 'ASSA$_G$'

Listing D.3: Main Code for ASSA$_G$

```
public void calculateGrounded(){
double[ ][ ] A = null;
double[ ][ ] B = null;
int inNodes=0;
int outNodes=0;
A=covertMatrixtoDouble(getMatrix());
B=create_1XN_Matrix_Double();
Matrix Am = new Matrix(A);
Matrix Bm = new Matrix(B);
Matrix Cm = Bm.times(Am);
inNodes=getGroundedInNodes(Cm);
Matrix Dm = convert_1XN_Matrix_Double(Cm);
Matrix Em = Dm.times(Am);
outNodes =  getGroundedOUTNodes(Em);
int nodesSize = this.NODES.size();
while ((inNodes > 0)&&(nodesSize>0)){
A=covertMatrixtoDouble(getMatrix());
B=create_1XN_Matrix_Double();
Am = new Matrix(A);
Bm = new Matrix(B);
Cm = Bm.times(Am);
inNodes=getGroundedInNodes(Cm);
Dm = convert_1XN_Matrix_Double(Cm);
Em = Dm.times(Am);
outNodes =  getGroundedOUTNodes(Em);
nodesSize=this.NODES.size();}
printGrounded();}
//*************************************************
public int getGroundedInNodes(Matrix m){
int x=0;
double[ ][ ] E = m.getArray();
int rows = 1;
int columns = E[0].length;
for (int i=0; i<rows; i++){
for (int j=0; j<columns; j++){
 if (E[i][j]>0){}
 else{
  String nodeName = getNodeByPosition(j);
  Node n = new Node(nodeName);
  this.IN.add(n);
  x++;}}}
return x;}
//*************************************************
```

```
public int getGroundedOUTNodes(Matrix m){
int x=0;
double[ ][ ] E = m.getArray();
int rows = 1;
int columns = E[0].length;
for (int i=0; i<rows; i++){
for (int j=0; j<columns; j++){
 if (E[i][j]>0){
  String nodeName = getNodeByPosition(j);
  Node n = new Node(nodeName);
  this.OUT.add(n);
  x++;}
   else{}}}
for (int i=0; i<IN.size(); i++){
Node n= (Node)IN.elementAt(i);
removeNode(n.getName());
breakEdge(n.getName());}
for (int i=0; i<OUT.size(); i++){
Node n= (Node)OUT.elementAt(i);
removeNode(n.getName());
breakEdge(n.getName());}
return x;        }
//*************************************************
public double[][] create_1XN_Matrix_Double(){
if (this.NODES.size() > 0){
int rows = 1 ;
int columns = this.NODES.size();
double[ ][ ] matrix = new double [rows][columns];
for (int i=0; i<rows; i++){
 for (int j=0; j<columns; j++){
  matrix[i][j]=1.0;}}
return matrix;}
else{
return null;}}
//*************************************************
public int removeNode(String nodeName){
Vector V = new Vector();
int x =0;
if (NODES.size() > 0){
for (int i=0; i<NODES.size(); i++){
 Node n = (Node)NODES.elementAt(i);
 if (n.getName().equalsIgnoreCase(nodeName)){
  x++;}
 else{
  V.add(n);}}}
if (x>0){
 this.NODES=V;}
```

```
return x;}
//*****************************************************
public void breakEdge(String nodeName){
Vector E = new Vector();
for (int i=0; i<EDGES.size(); i++){
Edge e= (Edge)EDGES.elementAt(i);
Node startNode = e.getStart();
Node endNode = e.getEnd();
if (endNode.getName().equalsIgnoreCase(nodeName)){
 endNode.setName("X");
 Edge e1 = new Edge(startNode,endNode);
 EDGES.add(i, e1);}}
for (int i=0; i<EDGES.size(); i++){
Edge e= (Edge)EDGES.elementAt(i);
Node startNode = e.getStart();
Node endNode = e.getEnd();
if (startNode.getName().equalsIgnoreCase(nodeName)){}
else{
 E.add(e);}}
this.EDGES=E;}
//*****************************************************
private Edge parseTrivialEdges(String line){
String tmp="";
Node startNode = new Node();
Node endNode = new Node();
for (int i = 0; i < line.length(); i++){
 tmp += line.charAt(i);
 if (Character.isWhitespace(line.charAt(i))){
  startNode.setName(tmp.trim());
  tmp="";}}
endNode.setName(tmp.trim());
Edge e = new Edge(startNode,endNode);
return e;}
//*****************************************************
public void parseTrivialFiles(String f ){
BufferedReader br = null;
boolean found= false;
Node n = null;
try{
String sCurrentLine;
br = new BufferedReader(new FileReader(f));
while ((sCurrentLine = br.readLine()) != null){
 if (!sCurrentLine.equalsIgnoreCase("#") && (!found)){
  n = new Node(sCurrentLine);
  this.NODES.add(n);}
 else{
  found=true;}
```

```
 if (found){
  if (!sCurrentLine.equalsIgnoreCase("#")){
   Edge e = parseTrivialEdges(sCurrentLine);
   this.EDGES.add(e);}   }}}
catch (IOException e){
e.printStackTrace();}
finally{
try{
 if (br != null)br.close();}
catch (IOException ex){
        ex.printStackTrace();}}}
```

Appendix E

'Hotel for ME' Queries

For this set of queries Alexandros is used as the user. Parameters, such as H = hotel, A = area, R = reason for visiting, and W = traveling with, are left as variable to be instantiated with all available values. The program is run at different time points to show that the application 'Hotel for ME' is time dependet.

E.1 Single User Dynamic World: Question 1

This question is general. To run a query $hotel\_for\_me/6$, the questions illustrated in Listing E.1 were used and the bar chart illustrated in Figure 37 was generated for different time points.

Listing E.1: Question 1

```
% Any time point
?- findall(H, prove([hotel_for_me(alexandros,H,T,A,R,W)], Delta), Bag),
length(Bag, Number_of_solutions).
Bag = [7, 7, 7, 7, 7, 7, 7, 7, 7|...],
Number_of_solutions = 14029.

% At Time 1
?- findall(H, prove([hotel_for_me(alexandros,H,1,A,R,W)], Delta), Bag),
length(Bag, Number_of_solutions).
Bag = [7, 8, 9, 10, 11, 12, 13, 14, 15|...],
Number_of_solutions = 1384.
```

Figure 37 shows the number of solutions that our system produces when question 1 is asked. At time 1 there are 1384 answers, at time 2 there are 1300 answers, etc. 1463 is the largest number of solution which is produced at time time 4 and the least number of solutions, which is 921 is generated at time 5. As observed in this bar chart, where the 'imaginary' x-axis represents the number of solutions and the 'imaginary' y-axis represents the time-points, available solutions vary. This is because the working environment changes and at different time point different parameters hold. This is similar for all other bar charts.

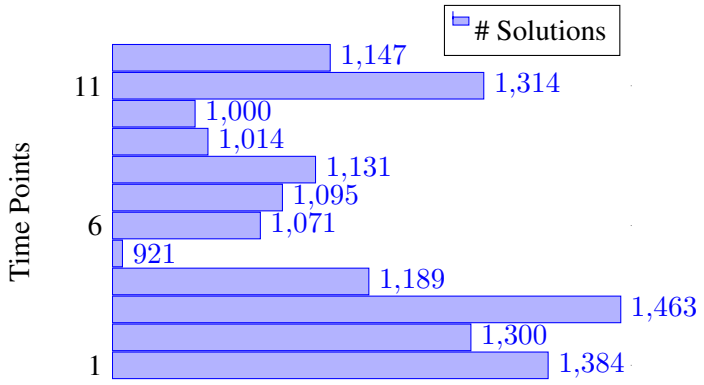


Figure 37: Single User Dynamic World: Question 1

E.2 Single User Dynamic World: Question 2

Question 1 is extended to question 2, where the user can specify one parameter that he or she is interested in. For this example, the user Alexandros, allows higher priority to the need pool. This is similar for all other needs. To run query $hotel\_for\_me1/7$, the questions illustrated in Listing E.2 were used and the bar chart illustrated in Figure 38 was generated for different time points.

Listing E.2: Question 2

```
% Any time point
findall(H,prove([hotel_for_me1(alexandros,H,T,A,R,W,pool(H,T))],
Delta),Bag),length(Bag,Number_of_solutions).
Bag = [7, 7, 7, 7, 7, 7, 7, 7, 7|...],
Number_of_solutions = 12447.

% At Time 1
?-findall(H,prove([hotel_for_me1(alexandros,H,1,A,R,W,pool(H,1))],
Delta),Bag),length(Bag,Number_of_solutions).
Bag = [7, 8, 9, 10, 11, 12, 13, 14, 16|...],
Number_of_solutions = 1232.
```

E.3 Single User Dynamic World: Question 3

Question 3 is an attack relation where the user can specify a need that he or she does not prefer. For this example, the user Alexandros, specifies that he does not like the hotel to have a play ground. This is similar for all other needs. The query $hotel\_for\_me2/7$ was run as indicated in Listing E.3, and all of the results are shown in Figure 39 for different time points.

Listing E.3: Question 3

```
% Any time point
?-findall(H,prove([hotel_for_me2(alexandros,H,T,A,R,W,
```

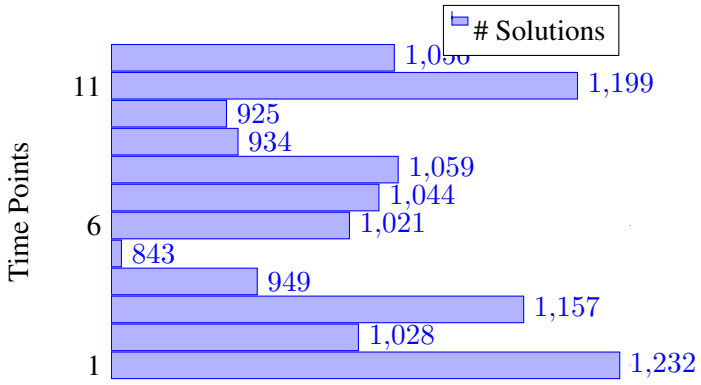


Figure 38: Single User Dynamic World: Question 2

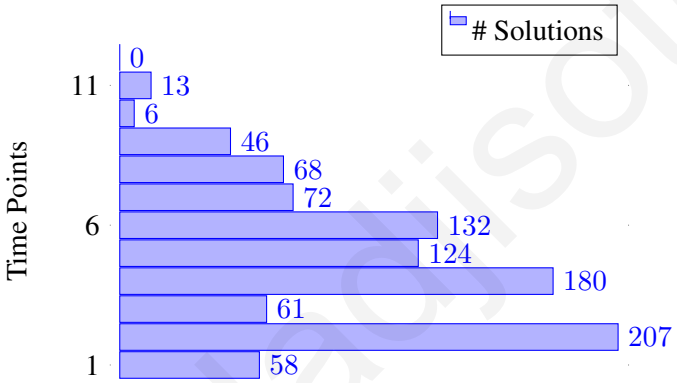


Figure 39: Single User Dynamic World: Question 3

```
play_ground(H,T))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [23, 23, 23, 23, 23, 24, 24, 24, 24|...],
Number_of_solutions = 967.

% At Time 1
?-findall(H,prove([hotel_for_me2(alexandros,H,1,A,R,W,
play_ground(H,1))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [401, 402, 403, 404, 405, 406, 407, 408, 435|...],
Number_of_solutions = 58.
```

E.4 Single User Dynamic World: Question 4

Question 3 has been extended to question 4, where the user can specify one parameter that he or she is not interested in, as in question 3, and one parameter that he or she is interested in. For this example, the user Alexandros, does not like playground and likes pools. This is similar for all other needs. To run query $hotel\_for\_me3/8$, the questions illustrated in Listing E.4 were used and the bar chart illustrated in Figure 40 was generated for different time points.

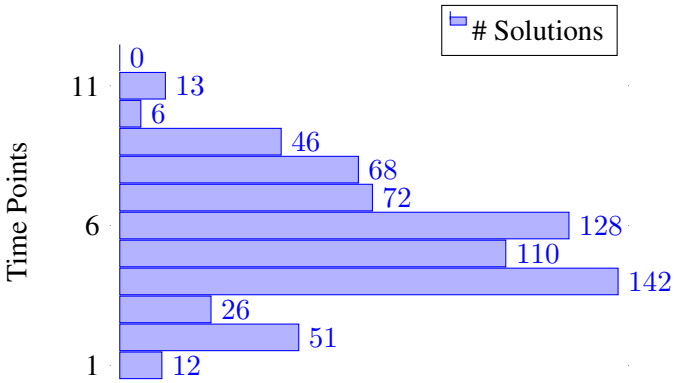


Figure 40: Single User Dynamic World: Question 4

Listing E.4: Question 4

```
% Any time point
?-findall(H,prove([hotel_for_me3(alexandros,H,T,A,R,W,play_ground(
H,T),pool(H,T))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [23, 23, 23, 23, 23, 24, 24, 24, 24|...],
Number_of_solutions = 674.

% At Time 1
?-findall(H,prove([hotel_for_me3(alexandros,H,1,A,R,W,play_ground(
H,1),pool(H,1))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [101, 102, 295, 296, 295, 296, 101, 102, 295|...],
Number_of_solutions = 12.
```

E.5 Single User Dynamic World: Question 5

If the user provides more information, question 5 is an attack relation where the user can specify two needs that s/he likes. For this example, the user Alexandros, specifies that he likes pools and good nightlife, which is an extension of question 1. Similarly for all other needs. The query $hotel\_for\_me5/8$, was run as indicated in Listing E.5 and all of the results are shown in Figure 41 for different time points.

Listing E.5: Question 5

```
% Any time point
?-findall(H,prove([hotel_for_me4(alexandros,H,T,A,R,W,pool(H,T),
good_nightlife(H,T,A))],Delta),Bag),length(Bag,Number_of_solutions).
Bag = [7, 7, 7, 7, 7, 7, 7, 7, 8|...],
Number_of_solutions = 5741.

% At Time 1
?-findall(H,prove([hotel_for_me4(alexandros,H,1,A,R,W,pool(H,1),
good_nightlife(H,1,A))],Delta),Bag),length(Bag,Number_of_solutions).
```

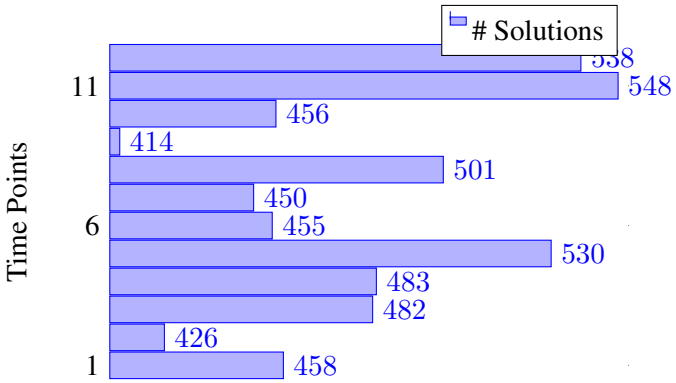


Figure 41: Single User Dynamic World: Question 5

```
Bag = [7, 8, 9, 10, 24, 25, 26, 28, 29|...],
Number_of_solutions = 458.
```

E.6 Single User Dynamic World: Question 6

Focussing the recommendations. Question 6 allow the user to specify three needs that they like. For this example, the user Alexandros, likes pools, good nightlife, and special rates, which is an extension of question 5. This is similar for the other needs. The query $hotel\_for\_me6/9$ was run as indicated in Listing E.6 and Figure 42 was generated for different time points.

Listing E.6: Question 6

```
% Any time point
?-findall(H, prove([hotel_for_me5(alexandros,H,T,A,R,W, pool(H,T),
good_nightlife(H,T,A), special_rates(H,T))], Delta), Bag),
length(Bag, Number_of_solutions).
Bag = [7, 8, 9, 10, 31, 31, 32, 32, 33|...],
Number_of_solutions = 1140.

% At Time 1
?-findall(H, prove([hotel_for_me5(alexandros,H,1,A,R,W, pool(H,1),
good_nightlife(H,1,A), special_rates(H,1))], Delta), Bag),
length(Bag, Number_of_solutions).
Bag = [149, 149, 147, 148, 149, 147, 148, 149, 147|...],
Number_of_solutions = 66.
```

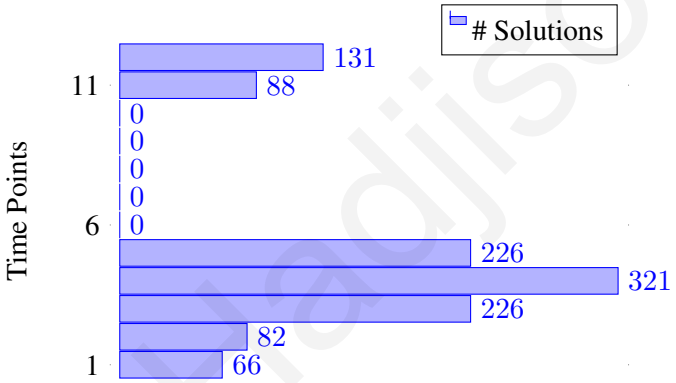


Figure 42: Single User Dynamic World: Question 6

Appendix F

Table of Basic Notations

Notation	Meaning
$+$	plus
$-$	minus
$\therefore$	therefore
$\in$	is an element of
$<$	is less than
$>$	is greater than
$\geq_u$	priority relation among arguments
$\sqsupseteq_u$	priority relation among parametric spaces
$\notin$	is not an element of
$=$	is equal to
$\neq$	is not not equal to
$\exists$	there exists
$\nexists$	there doe not exists
$\forall$	for all
$\emptyset$	empty set
$\subseteq$	subset
$\supseteq$	superset
$\cup$	union
$\langle \mathcal{A}, \mathcal{R} \rangle$	an abstract AF
$\mathcal{A}$	a set of arguments
$\mathcal{S}$	a set of arguments
$\mathcal{R}$	an attack relation over arguments
$a \rightarrow b$	argument a attacks argument b
$attacks(a, b)$	argument a attacks argument b
$(a, b) \in \mathcal{R}$	argument a attacks argument b
$attacks(\mathcal{S}, b)$	set $\mathcal{S}$ attacks argument b
$(\mathcal{S}, b) \in \mathcal{R}$	set $\mathcal{S}$ attacks argument b
$L \leftarrow l_1, l_2, \ldots, l_n$	rule in $LPwNF$
KB	domain description
PC	action $ParkingCar$
CPS	fluent $CarInParkingSpace$

K	belief in revision theory
a	formula in revision theory
$K + a$	Expansion in revision theory
$K - a$	Contraction in revision theory
$K * a$	Revision in revision theory
$p_i^U(1)$	user U, is interested in parameter p_i
$p_i^U(0)$	user U, does not specify an interested in parameter p_i
$p_i^U(-1)$	user U, is not interested in parameter p_i
$arg_1 \geq_u arg_2$	priority relation amongst arguments
$Par_1 \sqsupseteq_u Par_2$	priority relation amongst parametric spaces
$\mathcal{AC}$	action constants in language $\mathcal{E}$
$\mathcal{F}$	fluent constants in language $\mathcal{E}$
$\mathcal{T}$	time points in language $\mathcal{E}$

Table 5: Table of Basic Notations