Modeling arguments and uncertain information — A non-monotonic reasoning approach

Juan Carlos Nieves Sánchez
Advisors:
Ulises Cortés Ph. D.
Mauricio Osorio Ph. D.

Software Department (LSI)
Artificial Intelligence Ph D Program

Technical University of Catalonia

A thesis submitted for the degree of

Ph. D. in Artificial Intelligence

November 17th, 2008

I would like to dedicate this thesis to my mother Cristina and to the memory of my father Tomás.

Acknowledgements

Quiero agradecer muy especialmente a los Doctores Ulises Cortés y Mauricio Osorio por todo el apoyo brindado en el desarrollo de este trabajo. Así también agradecer les, me hayan compartido de su experiencia y perspicacia en mi formación académica. Gracias Dr. Mauricio por motivar me a emprender esta gran aventura en mi vida académica.

De igual forma, quiero agradecer el apoyo de mi madre, mi hermana, mi hermano, mis sobrinos y por su puesto a Paty por dejarme sentir su cariño aún estando lejos de casa.

Por otra parte, me gustaría agradecer a cada uno de mis incondicionales amigos que me han brindado su apoyo a lo largo de estos últimos años. Como a mis compañeros de despacho quienes me han apoyado en la edición de algunos de mis trabajos. También me gustaría agradecer a todos mis amigos de la UPC con quienes he compartido grandes momentos en nuestras múltiples reuniones. Ustedes bien saben que formamos una gran familia.

Aprovecho la ocasión para agradecer a mis grandes amigos externos a la UPC, como mis amigos de salsa, quienes me han hecho sentir como en casa aún estando muy lejos de ella.

Finalmente quiero expresar mi gratitud al Concejo Nacional de Ciencia y Tecnología de México (CONACyT) por la beca brindada al apoyo de mi formación académica. Así como también a los revisores anónimos de este documento y de los artículos publicados que contribuyen a este documento, por sus invaluables comentarios que en suma contribuyeron a esta tesis.

Modeling arguments and uncertain information — A non-monotonic reasoning approach

Juan Carlos Nieves Sánchez
Advisors:
Ulises Cortés Ph. D.
Mauricio Osorio Ph. D.

Software Department (LSI)
Artificial Intelligence Ph D Program
Technical University of Catalonia

A thesis submitted for the degree of Ph. D. in Artificial Intelligence

November 17th, 2008

In this thesis, we define a possibilistic disjunctive logic programming approach for modeling uncertain, incomplete and inconsistent information. This approach introduces the use of possibilistic disjunctive clauses which are able to capture incomplete information and incomplete states of a knowledge base at the same time. This approach is computable and moreover allows encoding uncertain information by using either numerical values or relative likelihoods. In order to define the semantics of the possibilistic disjunctive programs, three approaches are defined:

1. The first is strictly close to the proof theory of possibilistic logic and answer set models;
2. The second is based on partial evaluation, a fix-point operator and answer set models; and
3. The last is also based on the proof theory of possibilistic logic and pstable semantics.

In order to manage inconsistent possibilistic logic programs, a preference criterion between inconsistent possibilistic models is defined; in addition, the approach of cuts for restoring consistency of an inconsistent possibilistic knowledge base is adopted.

Argumentation theory is also explored in this work. In particular, we explore how to model abstract argumentation semantics from a point of view of non-monotonic logic programming semantics. Based on a suitable mapping of an argumentation framework into a normal logic program, we define a direct relationship between argumentation semantics (*e.g.*, the preferred semantics) and logic

programming with answer sets models (which is one of the most successful approaches of non-monotonic reasoning of the last two decades). As a consequence of this result, we are able to suggest an easy-to-use method for implementing argumentation systems under the platform of answer set solvers. In fact, we point out that we can use answer sets solvers as DLV-system for implementing argumentations systems under the preferred semantics.

Another interesting point of exploring argumentation semantics, from the point of view of logic programming semantics, is that one can deal with part of the problems of the argumentation semantics *e.g.*, emptiness, non-existence. Hence, by considering the idea that argumentation semantics can be viewed as a special form of logic programming semantics with negation as failure, we show that any logic programming semantics as the answer set semantics, the minimal models, the pstable semantics *etc.*, can define candidate argumentation semantics. These candidate argumentation semantics will overcome some of the problems of the Dung's argumentation semantics that have been discussed in the literature. The new argumentation semantics are based on a new recursive framework for logic programming semantics. This framework generalizes any logic programming semantics in order to build logic programming semantics which are always defined, satisfy the property of relevance and agree with the answer set semantics for the class of stratified programs.

As an extension of our possibilistic logic programming approach, we also present a possibilistic argumentation approach which is based on our possibilistic logic programming approach. This approach offers some natural mechanisms for dealing with reasoning under inconsistent information. In fact, this approach does not requite to apply cuts to an inconsistent possibilistic knowledge base, as it is done in possibilistic logic programming, in order to manage the non-existence of possibilistic models.

Contents

List of Figures

Chapter 1

Introduction

Since the very moment Artificial Intelligence (AI) emerged as an alternative branch of the area of computer science, scientists have dreamed of creating intelligent machines capable of solving our problems. In fact, the first systems developed by AI scientists were provided by knowledge bases derived from experts in order to build intelligent systems which were able to perform problem analysis for their users. These systems were called expert systems and now they are known as knowledge-based systems. The success of these systems for supporting human decision-making have extensively documented in literature (16; 47; 102; 105). However, we cannot say that there is a magic system able to support all our decisions in any situation.

In the design of knowledge-based systems for supporting decision-making, we can identify two basic problematic issues:

- how to model knowledge, and
- how to model rational criteria for choosing one decision over other.

In the following subsection, we will identify some challenges *w.r.t.* these issues.

1.1 Modeling uncertain information

In computer science literature, we can find formal languages in order to give answer to the issue of how to model knowledge. The most common forms for modeling knowledge are based on symbolic logic. Even thought, the diversity of formal languages is wide and the question of how to model uncertain information has caused much heated debate. Maybe, the most common form of representing uncertain information is based on probability theory (56). In fact, we can find successful approaches based on probability theory as Bayesian Networks.

However, there are several authors which disagree with probability theory for modeling uncertain information.

- McCarthy and Hayes in (73) pointed out that attaching probabilities to a statement has objections. For example, they say that

  > The information necessary to assign numerical probabilities is not ordinary available. Therefore, a formalism that required numerical probabilities would be epistemologically inadequate.

- Halpern has remarked in (56) that probability has its problems. For one thing, the numbers are not always available. For another, the commitment to numbers means that any two events must be comparable in terms of their probabilities: either one event is more probable than the other, or they have equal probability.

- Dubois and Prade in (43) have pointed that there are at least three worth noticing difficulties when casting the probability calculus into a logic framework for handling uncertain information:

  1. Probabilities do not fit very well with logical entailment, in the sense that a set of propositions true with a probability greater or equal to some threshold, say α, is not deductively closed. Indeed, from the constraints $Prob(\neg p \vee q) \geq \alpha$ and $Prob(p) \geq \alpha$, one can only deduce in general that $Prob(q) \geq max(0, 2\alpha - 1)$, where $max(0, 2\alpha - 1) \leq \alpha$ (except if $\alpha = 1$ or 0). This means that probability reasoning does not maintain probability bounds across inference steps.
  2. The probabilistic counterpart of the resolution rule is not sufficient as a local computation tool for computing the best lower bound of the probability of a formula from a set of probabilistic constraints.
  3. There is a strong discrepancy between the probability of a material conditional $Prob(\neg p \vee q)$ and a conditional probability $Prob(q|p)$, which raises the question of the proper modeling of an uncertain rule: if p then q.

Since probability has its problems for modeling uncertain information, it is not surprising that many other approaches of uncertainty have been considered in computer science literature.

In the mid-1980s, it was introduced a logic framework called *Possibilistic Logic* (42). Possibilistic logic is a logic of uncertainty tailored for reasoning under incomplete evidence and partially inconsistent knowledge. In this approach all the formulæ are attached by degrees of uncertain. These degrees are quantifications of necessity or possibility of the corresponding possibilistic formulæ. At the mathematical level, degrees of possibility and necessity are closely related to fuzzy set and, possibilistic logic is especially well adapted to automated reasoning when the available information is pervaded with vagueness. In general terms, we can say that possibilistic logic is a tool for reasoning under uncertainty based on the idea of ordering rather than counting, on the contrary to probabilistic logic (42).

An important feature of possibilistic logic is that the degrees of uncertainty of a possibilistic formula do not belong necessarily to a totally ordered set. This feature allows to explore a diversity of uncertain degrees *e.g.*, *non-numerical uncertain degrees*. In psychology literature, we can find significant observations which are worth mentioning when we are designing an approach for modeling uncertain information. Tversky and Kahneman have observed in (107) [1], that many decisions that we make in our common life are based on beliefs concerning the likelihood of uncertain events. In fact, we commonly use statements such as "*I think that* ...", "*chances are* ...", "*it is probable that* ...", "*it is plausible that* ...", *etc.*, for supporting our decisions. In this kind of statements usually we appeal to our experience or our commonsense. It is not surprising to think that a reasoning based on these kind of statements could reach to *biased conclusions*. However, these conclusions could reflect an expert's experience or commonsense. Pelletier and Elio pointed out in (96) that people simply have tendencies to ignore certain information because of the (evolutionary) necessity to make decisions quickly. This gives rise to *biases* in judgments concerning what they *really* want to do.

Based on the flexibility of possibilistic logic for defining degrees of uncertainty, we believe that when numerical information is not available for capturing degrees of necessity or possibility of a statement, one can use adjectives/labels like *probable*, *plausible*, *etc.*, for capturing degrees of uncertain information. In fact the handling of this kind of degrees of uncertainty will define a heuristic for performing uncertain reasoning.

1.2 Modeling rational criteria

Nowadays, there have been proposed several decision-making approaches for supporting decisions which are based on arguments. Argumentation theory has become an increasingly important and exciting research topic in Artificial Intelligence (AI), with research activities ranging from developing theoretical models,

[1]It is worth mentioning that Kahneman (an author of (59)) is the winner of the 2002 Nobel Prize in Economics for having integrated insights from psychological research into economic science, especially concerning human judgment and decision-making under uncertainty

prototype implementations, and application studies (16). The main purpose of argumentation theory is to study the fundamental mechanism, humans use in argumentation, and to explore ways to implement this mechanism on computers.

Since humans currently use arguments for explaining choices which are already made, or for evaluating potential choices where each potential choice has usually *pros* and *cons* of various stretch, argumentation theory is also a suitable approach for practical and uncertain reasoning, (6). The reasoning in argumentation theory is not explained in terms of the interpretation of a defeasible condition. It is explained by the interactions between conflicting arguments. Surveys of this research field are (30; 99).

Although, several approaches have been proposed for modeling argumentation, Dung's approach, presented in (44), is a unifying framework which has played an influential role on argumentation research and AI. In fact, Dung's approach has been influencing subsequent proposals for argumentation systems, *e.g.*, (15; 18; 58; 111). Besides, Dung's approach is mainly relevant in fields where conflict management plays a central role. For instance, Dung showed that his theory naturally captures the solutions of the theory of n-person games and the well-known stable marriage problem (44).

The interaction of conflicting arguments in Dung's approach is supported by four abstract argumentation semantics: *stable semantics*, *preferred semantics*, *grounded semantics*, and *complete semantics*. The central notion of these semantics is the *acceptability of the arguments*. An argument is called *acceptable* if and only if it belongs to a set of arguments which is called *extension*.

Since Dung introduced his abstract argumentation approach, he proved that his approach can be regarded as a special form of logic programming with *negation as failure*. This result points out to the existence of a direct relationship between abstract argumentation semantics and logic programming semantics. As consequences of this relationship, we have that:

1. We can deploy argumentation systems based on logic programming systems.

2. We can study abstract augmentation semantics based on logic programming semantics. This means that a proper logic programming semantics can describe the interaction of conflicting arguments.

It is worth mentioning that Dung only characterizes the grounded and stable semantics by two logic programming semantics; however, Dung did not give any characterization of the preferred semantics in terms of logic programming semantics.

According to Bench-Capon and Dunne, the three principal abstract argumentation semantics introduced in (44) are the grounded, preferred and stable semantics. However, these semantics exhibit a variety of problems which have illustrated in the literature (13; 16; 26; 27; 99). In (16), Bench-Capon and Dunne summarize three main problems of these argumentation semantics:

(P1) *Emptiness*: this problem happens when even though an extension satisfying the prescribed conditions always exists, there are argumentation frameworks for which the only such extension is the empty set. This problem can arise with both the grounded and preferred semantics.

(P2) *Non-existence*: there are argumentation semantics as the stable argumentation semantics that when it exists is never empty, but there are argumentation frameworks for which no extension meeting the required criteria exist.

(P3) *Multiplicity*: in an argumentation framework AF there may be several incompatible extensions, *i.e.* S_1 and S_2 which are well-defined extensions of AF but with $S_1 \cup S_2$ failing to be so. This problem does not happen with Dung's grounded semantics; however, argumentation frameworks are easily constructed in which both the preferred and stable semantics exhibit this phenomenon.

Based on the fact that non-monotonic reasoning is one of the most explored areas of IA where actually there are many results *w.r.t.* logic programming semantics properties and logic programming semantics implementations, we believe that these problems can be confronted by identifying proper logic programming semantics which overcome these problems *w.r.t.* non-monotonic reasoning side. It is worth mentioning that the logic programming semantics implementations could improve the implementation cycle of argumentation systems.

1.3 Contribution of this thesis

In the previous section, we have outlined the two main issues that are concerned in this thesis: *modeling uncertain information* and *modeling argumentation theory under a non-monotonic approach*. We will group the contributions of this work in two main sections. More details about the impact of our results will be discussed at the end of each chapter and also they are discussed as part of the general conclusions of the thesis in the Chapter 7.

The first group, on our contributions *w.r.t.* the modeling of uncertain information under a non-monotonic approach, includes:

- a definition of a possibilistic disjunctive logic programming approach which is able to deal with reasoning under uncertainty, incomplete and inconsistent information (Chapter 3). Some properties of our approach are:

1. It permits to encode uncertain information by using either numerical values or relative likelihoods (§3.2).
2. It is computable.

- the definition of three approaches for capturing the semantics of possibilistic disjunctive logic programs :
  - the first is strictly close to the proof theory of possibilistic logic and *answer set models* (§3.3.1);
  - the second is based on *partial evaluation*, a fix-point operator and *answer set models* (§3.3.2); and
  - the last is also based on the proof theory of possibilistic logic and *pstable semantics* (§3.3.3).
- the definition of a criterion of preference between possibilistic models in order to manage the inconsistency of possibilistic models (§3.4).
- the definition of a possibilistic-based argumentation approach based on our possibilistic disjunctive logic programming approach (Chapter 6).

In the second group, our contributions *w.r.t.* the modeling of argumentation theory under a non-monotonic approach, we include:

- the definition of some basic conditions for studying abstract argumentation semantics in terms of logic programming semantics (§4.2).
- the definition of a suitable codification which is able to characterize the grounded, stable and preferred semantics (§4.3).
- a study of the preferred semantics in terms of minimal models and answer set models — this study will suggest some practical methods for implementing the preferred semantics (§4.5).
- a study of the grounded semantics in order to define some intermediate argumentation semantics between the grounded and the preferred semantics (§4.6).
- a definition of an approach in order to build new abstract argumentation semantics based on logic programming semantics. Some properties of the argumentation semantics constructed under our approach are (Chapter 5):

  1. They can be always defined.

2. They can be constructed under any logic programming semantics.

- the outline of an approach for describing the interaction of arguments based on rewiring systems and logic programs (§4.6.2).

Since we are interested in capturing real domains as the medical domain, we consider the following contributions which are not only related to AI community.

- We outline how to model medical scenarios in our possibilistic disjunctive logic programming approach. In particular, we consider the domain of human organ transplanting (§3.1).
- We outline a possibilistic argumentation engine architecture in order to show that by considering the results presented in thesis, we can implement real argumentation systems.

1.4 Thesis overview

Chapter 2 contains, all the necessary terminology and relevant definitions in order to have a self-contained document. In particular, we present a general description of the syntax and semantics for logic programs. Also, general descriptions of Dung's argumentation approach, rewriting systems and possibilistic logic are presented.

Chapter 3 The major part of our contribution *w.r.t.* modeling uncertain and incomplete information is included in this chapter. It includes the definition of a possibilistic disjunctive logic programming approach. Part of the results presented in this chapter were originally published in (81; 82; 94). The proofs of this chapter are included in Appendix A.

Chapter 4 In this chapter, we present our contribution *w.r.t.* the modeling of argumentation theory under a non-monotonic approach. For instance, we present our study of how to characterize and/or construct argumentation semantics in terms of non-monotonic logic programming semantics. The early ideas of this chapter were published in (86; 87). After those publications we explore new ways that lead to the results that have been presented in this chapter. In fact, part of these results were published in (29; 78; 79; 84). The proofs of this chapter are included in Appendix B.

Chapter 5 In this chapter, by considering the idea that argumentation semantics can be viewed as a special form of logic programming semantics with negation as failure, we show that any logic programming semantics as the answer set semantics, the minimal models, *etc.*, can define candidate argumentation semantics. These new argumentation semantics will overcome some of the problems of the Dung's argumentation semantics that have been discussed in the literature.

Chapter 6 In this chapter, we define a possibilistic-based argumentation approach. This approach has as specification language the logic programming approach defined in Chapter 3. The interaction of possibilistic arguments is managed by Dung's argumentation semantics style. Hence, all the results presented in Chapter 4 and Chapter 5 can be applied to this possibilistic argumentation approach. Some of the first ideas of this chapter were published in (77; 80; 83).

Chapter 7 We finally, in this last chapter, present a general discussion and overview of the material presented in this thesis.

Chapter 2

Background

In this chapter we introduce all the necessary terminology and relevant definitions in order to have a self-contained document. We want to point out that this chapter is not intended to be of a tutorial nature; please consult the references for a more detailed presentation. We assume that the reader has familiarity with basic concepts of *classic logic*, *logic programming* and *lattices*. If this is not the case, the reader can refer himself to (11; 32; 68; 74).

2.1 Logic programs: Syntaxis

The language of a propositional logic has an alphabet consisting of

(i) proposition symbols: $p_0, p_1, ...$

(ii) connectives : $\vee, \wedge, \leftarrow, \neg,$ *not*, $\bot, \top$

(iii) auxiliary symbols : (,)

where $\vee, \wedge, \leftarrow$ are binary-place connectives, $\neg$, *not* are unary-place connective and $\bot$ is zero-ary connective. The proposition symbols, $\bot$ and $\top$ stand for the indecomposable propositions, which we call *atoms*, or *atomic propositions*. Atoms negated by $\neg$ will be called *extended atoms*.

Remark 2.1 *We will use the concept of atom without paying attention if it is an extended atom or not.*

The negation sign $\neg$ is regarded as the so called *strong negation* by the ASP's literature and the negation *not* as the *negation as failure*. A literal is an atom, a, or the negation of an atom *not* a. Given a set of atoms $\{a_1, ..., a_n\}$, we write *not* $\{a_1, ..., a_n\}$ to denote the set of literals $\{not\ a_1, ..., not\ a_n\}$. An extended disjunctive clause, C, is denoted:

$$a_1 \vee \ldots \vee a_m \leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n$$

where $m \geq 0$, $n \geq 0$, $m + n > 0$, each a_i is an atom[1]. When $n = 0$ and $m > 0$ the clause is an abbreviation of $a_1 \vee \ldots \vee a_m \leftarrow \top$ such that $\top$ is the proposition symbol that always evaluates to true; clauses of these form some times are written just as $a_1 \vee \ldots \vee a_m$. When $m = 0$ the clause is an abbreviation of $\bot \leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n$ such that $\bot$ is the proposition symbol that always evaluates to false. Clauses of this form are called constraints (the rest, non-constraint clauses). An extended disjunctive program P is a finite set of extended disjunctive clauses. By $\mathcal{L}_P$, we denote the set of atoms in the language of P.

Sometimes we denote an extended disjunctive clause C by $\mathcal{A} \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$, where $\mathcal{A}$ contains all the head literals, $\mathcal{B}^+$ contains all the positive body literals and $\mathcal{B}^-$ contains all the negative body literals. When $\mathcal{B}^- = \emptyset$, the clause is called positive disjunctive clause. A set of positive disjunctive clauses is called a positive disjunctive logic program. When $\mathcal{A}$ is a singleton set, the clause can be regarded as a normal clause. A normal logic program is a finite set of normal clauses. Finally, when $\mathcal{A}$ is a singleton set and $\mathcal{B}^- = \emptyset$, the clause can be also regarded as a definite clause. A finite set of definite clauses is called a definite logic program.

We will manage the strong negation ($\neg$), in our logic programs, as it is done in ASP (11). Basically, it is replaced each extended atom $\neg a$ by a new atom symbol a' which does not appear in the language of the program. For instance, let P be the normal program:

$a \leftarrow q.$
$\neg q \leftarrow r.$
$q \leftarrow \top.$
$r \leftarrow \top.$

Then replacing each extended atom by a new atom symbol, we will have:

$a \leftarrow q.$
$q' \leftarrow r.$
$q \leftarrow \top.$
$r \leftarrow \top.$

In order to disallow models with complementary atoms *i.e.* q and $\neg q$, usually it is added a constraint of the form $\bot \leftarrow q, q'$ to the logic program. We will omit

[1]Notice that these atoms can be *extended atoms*.

this constraint in order to allow models with complementary atoms. However, the user could add this constraint without losing generality.

When we treat a logic program as a theory, each negative literal *not a* is replaced by $\sim a$ such that $\sim$ is regarded as the classical negation in classic logic. Formulæ are constructed as usual in classic logic by the connectives: $\vee, \wedge, \leftarrow, \sim, \bot, \top$. A theory T is a finite set of formulæ. By $\mathcal{L}_T$, we denote the signature of T, namely the set of atoms that occur in T.

Given a set of proposition symbols S and a theory Γ in a logic X. If $\Gamma \vdash_X S$ if and only if $\forall s \in S\ \Gamma \vdash_X s$.

2.2 Interpretations and models

In this section we define some relevant concepts *w.r.t.* semantics. The first basic concept that we introduce will be *interpretation*.

Definition 2.1 *Let T be a theory, an interpretation I is a mapping from $\mathcal{L}_T$ to $\{0,1\}$ meeting the conditions:*

1. $I(a \wedge b) = min\{I(a), I(b)\}$,
2. $I(a \vee b) = max\{I(a), I(b)\}$,
3. $I(a \leftarrow b) = 0$ *if and only if* $I(b) = 1$ *and* $I(a) = 0$,
4. $I(\sim a) = 1 - I(a)$,
5. $I(\bot) = 0$.
6. $I(\top) = 1$.

It is standard to provide interpretations only in terms of a mapping from $\mathcal{L}_T$ to $\{0,1\}$. Moreover, it is easy to prove that this mapping is unique by virtue of the definition by recursion (108). Also, it is standard to use sets of atoms to represent interpretations. The set corresponds exactly to those atoms that evaluate to 1.

An interpretation I is called a (2-valued) model of the logic program P if and only if for each clause $c \in P$, $I(c) = 1$. A theory is consistent if it admits a model, otherwise it is called inconsistent. Given a theory T and a formula α, we say that α is a logical consequence of T, denoted by $T \models \alpha$, if every model I of T holds that $I(\alpha) = 1$. It is a well known result that $T \models \alpha$ if and only if $T \cup \{\sim \alpha\}$ is inconsistent.

We say that a model I of a theory T is a minimal model if there does not exist a model I' of T different from I such that $I' \subset I$. Maximal models are defined in the analogous form.

Since we will use 3-valued semantics, we will define some definitions *w.r.t.* 3-valued logic semantics. A partial interpretation based on a signature $\mathcal{L}$ is a disjoint pair of sets $\langle I_1, I_2 \rangle$ such that $I_1 \cup I_2 \subseteq \mathcal{L}$. A partial interpretation is total if $I_1 \cup I_2 = \mathcal{L}$. Given two interpretations $I = \langle I_1, I_2 \rangle$, $J = \langle J_1, J_2 \rangle$, we set $I \leq_k J$ *if, by definition,* $I_i \subseteq J_i$, $i = 1, 2$. Clearly $\leq_k$ is a partial order. We may also see an interpretation $\langle I_1, I_2 \rangle$ as the set of literals $I_1 \cup \textit{not}\ I_2$. When we look at interpretations as sets of literals then $\leq_k$ corresponds to $\subseteq$.

2.3 Rewriting systems

In this section, we define some basic concepts *w.r.t.* abstract rewriting systems and some basic transformation rules for normal logic programs.

It is well accepted that rewriting is a very powerful method for dealing computation with equations (33). In fact, rewriting has the computational power of Markov algorithms — and of recursive functions and Turing machines. In the context of logic programming, it has been defined program transformations which allow to define a *calculus of logic program transformations* (22; 38). These logic program transformations can be seen as declarative because they express precise logic programming semantics properties; moreover, they represent a procedural method for computing logic programming semantics.

As we will see in Chapter 4, the combination of logic programming semantics and rewriting systems will help to define a *calculus of argumentation frameworks* in order to define abstract argumentation semantics and to define a procedural method for computing abstract argumentation semantics.

An *abstract rewriting system* is a pair $\langle S, \rightarrow \rangle$ where $\rightarrow$ is a binary relation on a given set S. Let $\rightarrow^*$ be the reflexive, and transitive closure of $\rightarrow$. When $x \rightarrow^* y$ we say that x *reduces to* y. An *irreducible* element is said to be in *normal form*. We say that a rewriting system is:

noetherian: if there is no infinite chain $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_i \rightarrow x_{i+1} \rightarrow \ldots$, where for all i the elements x_i and x_{i+1} are different,

confluent: if whenever $u \rightarrow^* x$ and $u \rightarrow^* y$ then there is a z such that $x \rightarrow^* z$ and $y \rightarrow^* z$,

locally confluent: if whenever $u \rightarrow x$ and $u \rightarrow y$ then there is a z such that $x \rightarrow^* z$ and $y \rightarrow^* z$.

In a noetherian and confluent rewriting system, every element x reduces to a unique normal form that we denote by $norm(x)$.

In literature, it has defined several transformation rules for logic programs (22; 38; 95). In particular, we only define seven basic transformation rules.

Let $Prog_{\mathcal{L}}$ be the set of all normal logic programs with atoms from the signature $\mathcal{L}$. Given a normal program P, we define $\text{HEAD}(P) = \{a | \ a \leftarrow \mathcal{B}^+, \ not \ \mathcal{B}^- \ \in P\}$ — the set of all head-atoms of P.

Definition 2.2 (Transformation Rules) *(39) A transformation rule is a binary relation on* $Prog_{\mathcal{L}}$. *The following transformation rules are called* basic. *Let a program* $P \in Prog_{\mathcal{L}}$ *be given.*

RED$^+$: *This transformation can be applied to P, if there is an atom a which does not occur in $HEAD(P)$.* ***RED***$^+$ *transforms P to the program where all occurrences of not a are removed.*

RED$^-$: *This transformation can be applied to P, if there is a rule $a \leftarrow \top \in P$.* ***RED***$^-$ *transforms P to the program where all clauses that contain not a in their bodies are deleted.*

Success: *Suppose that P includes a fact $a \leftarrow \top$ and a clause $q \leftarrow body$ such that $a \in body$. Then we replace the clause $q \leftarrow body$ by $q \leftarrow body \setminus \{a\}$.*

Failure: *Suppose that P contains a clause $q \leftarrow body$ such that $a \in body$ and $a \notin HEAD(P)$. Then we erase the given clause.*

Loop: *We say that P_2 results from P_1 by* Loop$_A$ *if, by definition, there is a set A of atoms such that:*

1. *for each rule $a \leftarrow body \in P_1$, if $a \in A$, then $body \cap A \neq \emptyset$,*
2. *$P_2 := \{a \leftarrow body \in P_1 | body \cap A = \emptyset\}$,*
3. *$P_1 \neq P_2$.*

LLC$'$ *Let a be an atom that occurs negatively in P and also appears in the head of some rule of P. Let P_1 be the program that results from P by removing not a from every clause of P. Let* ***Success**** *denote the reflexive and transitive closure of the relation* ***Success***. *Suppose that P_1 relates to P_2 by* ***Success**** *and $a \in P_2$. In this case, we add $a \leftarrow \top$ to P.*

Weak-Cases *Let us suppose the following condition holds: $C_1 \in P$, $C_2 \in P$, C_1 is of the form $a \leftarrow l$ and C_2 is of the form $a \leftarrow \ not \ l$. Then the Weak-Cases transformation replaces the clauses C_1 and C_2 in P by the single clause $a \leftarrow \top$.*

Based on these transformation rules, we define the following rewriting systems:

$\mathcal{CS}_0 := \{RED^+, RED^-, Success, Failure, Loop\}$

$\mathcal{CS}_1 := \mathcal{CS}_0 \cup \{LLC'\}$

$\mathcal{CS}_2 := \mathcal{CS}_0 \cup \{\,Weak\text{-}Cases\,\}$

$\mathcal{CS}_3 := \mathcal{CS}_0 \cup \{LLC', Weak\text{-}Cases\,\}$

We denote the uniquely determined normal form of a program P with respect to the system $\mathcal{CS}$ by $norm_{\mathcal{CS}}(P)$. In order to illustrate these transformation rules, let us consider the following example.

Example 2.1 *Let P be the following normal program:*

$$d(b) \leftarrow not\ d(a). \qquad d(b) \leftarrow \top. \qquad d(c) \leftarrow not\ d(b). \qquad d(c) \leftarrow d(a).$$

Now, let us apply $\mathcal{CS}_0$ to P. Since $d(a) \notin HEAD(P)$, then, we can apply ***RED***$^+$ *to P. Thus we get:*

$$d(b) \leftarrow \top. \qquad d(c) \leftarrow not\ d(b). \qquad d(c) \leftarrow d(a).$$

Notice that we can apply ***RED***$^-$ *to the new program, thus we get:*

$$d(b) \leftarrow \top. \qquad\qquad d(c) \leftarrow d(a).$$

Finally, we can apply ***Failure*** *to the new program, thus we get: $d(b) \leftarrow \top$. This last program is the* normal form *of P* w.r.t. $\mathcal{CS}_0$*, because none of the transformation rules from $\mathcal{CS}_0$ can be applied.*

We use $P_1 \rightarrow^T P_2$ for denoting that we get P_2 from P_1 by applying the transformation rule T to P_1. It is worth mentioning that the rewriting systems $\mathcal{CS}_0, \mathcal{CS}_1, \mathcal{CS}_2$ and $\mathcal{CS}_3$ are *confluent* and *noetherian.*

2.4 Logic programming semantics

In this section, we will define three logic programming semantics: *answer set semantics*, *pstable semantics* and *the well-founded semantics.* The first two semantics represent a two-valued semantics approach and the last one represents a three-valued semantics approach. These semantics will be of vital importance for the study of this thesis. For instance, in Chapter 3 we introduce a couple of possibilistic logic programming semantics based on the philosophy of the answer set semantics and the pstable model semantics. Also, we will see, in Chapter 4, that all semantics presented in this section are of vital importance for studying abstract argumentation semantics.

2.4.1 Answer set semantics

By using ASP, it is possible to describe a computational problem as a logic program whose answer sets correspond to the solutions of the given problem. It represents one of the most successful approaches of non-monotonic reasoning of the last two decades (11). The number of applications of this approach has been increased thanks to the efficient implementations of the answer set solvers that exist.

The answer set semantics was first defined in terms of the so called *Gelfond-Lifschitz reduction* (52) and it is usually studied in the context of syntax dependent transformations on programs. The following definition of an answer set for extended disjunctive logic programs generalizes the definition presented in (52) and it was presented in (53): Let P be any extended disjunctive logic program. For any set $S \subseteq \mathcal{L}_P$, let P^S be the positive program obtained from P by deleting

(i) each rule that has a formula *not* a in its body with $a \in S$, and then

(ii) all formulæ of the form *not* a in the bodies of the remaining rules.

Clearly P^S does not contain *not* (this means that P^S is either a positive disjunctive logic program or a definite logic program), hence S is called an answer set of P if and only if S is a minimal model of P^S. In order to illustrate this definition let us consider the following example:

Example 2.2 *Let us consider the set of atoms $S := \{b\}$ and the following normal logic program P:*

$b \leftarrow not\ a.$ $\qquad$ $b \leftarrow \top.$

$c \leftarrow not\ b.$ $\qquad$ $c \leftarrow a.$

We can see that P^S is:

$b \leftarrow \top.$ $\qquad$ $c \leftarrow a.$

Notice that this program has three models: $\{b\}$, $\{b, c\}$ and $\{a, b, c\}$. Since the minimal model amongst these models is $\{b\}$, we can say that S is an answer set of P.

In the answer set definition, we will normally omit the restriction that if S has a pair of complementary literals then $S := \mathcal{L}_P$. This means that we allow that an answer set could have a pair of complementary atoms. For instance, let us consider the program P:

$a.$ $\qquad$ $\neg a.$ $\qquad$ $b.$

then, the only answer set of this program is : $\{a, \neg a, b\}$. In Section 3.4, we will discuss how we will manage the inconsistency in our logic programs.

It is worth mentioning that in literature there are several forms for handling an inconsistency program. For instance, by applying the original definition (53) the only answer set of P is: $\{a, \neg a, b, \neg b\}$. On the other hand, the DLV system (40) returns no models if the program is inconsistent.

2.4.2 Pstable semantics

Pstable semantics is a recently introduced logic programming semantics which is inspired in paraconsistent logics. This semantics is defined by a fixed point operator in terms of classical logic. The expressiveness of pstable semantics is at least as the answer set semantics for disjunctive logic program (88). In Chapter 3, we will define an extension of this semantics for possibilistic programs and in Chapter 4 we will see that this semantics is able to characterize one of the most acceptable abstract argumentation semantics *i.e. preferred semantics.*

First, to define pstable semantics, we will introduce some basic concepts. By $\vdash_C$, we denote logic consequence in classic logic. Given a normal program P, if $M \subseteq \mathcal{L}_P$, we write $P \Vdash M$ when: $P \vdash_C M$ and M is a classical 2-valued model of P.

Pstable semantics is defined in terms of a single reduction which is defined as follows:

Definition 2.3 *(92) Let P be a normal program and M a set of atoms. We define*

$$RED(P, M) := \{a \leftarrow \mathcal{B}^+,\ not\ (\mathcal{B}^- \cap M) | a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- \in P\}$$

Let us consider the set of atoms $M_1 := \{a, b\}$ and the following normal program P_1:

$a \leftarrow\ not\ b,\ not\ c.$
$a \leftarrow b.$
$b \leftarrow a.$

We can see that $RED(P_1, M)$ is:
$a \leftarrow\ not\ b.$
$a \leftarrow b.$
$b \leftarrow a.$

By considering the reduction RED, it is defined the pstable semantics for normal programs as follows:

Definition 2.4 *(92) Let P be a normal program and M a set of atoms. We say that M is a pstable model of P if $RED(P, M) \Vdash M$. We use Pstable to denote the semantics operator of pstable models.*

Let us consider again M_1 and P_1 in order to illustrate the definition. We want to verify whether M_1 is a pstable model of P_1. First, we can see that M_1 is a model of P_1, *i.e.* $\forall\, c \in P_1$, M_1 evaluates c to true. Now, we have to prove each atom of M_1 from $RED(P_1, M_1)$ by using classical inference, *i.e.* $RED(P_1, M_1) \vdash_C M_1$. Let us consider the proof of the atom a, which belongs to M_1, from $RED(P_1, M_1)$.

1. $(a \vee b) \rightarrow ((b \rightarrow a) \rightarrow a)$	Tautology
2. $\sim b \rightarrow a$	Premise from $RED(P_1, M_1)$
3. $a \vee b$	From 2 by logical equivalency
4. $(b \rightarrow a) \rightarrow a$	From 1 and 3 by Modus Ponens
5. $b \rightarrow a$	Premise from $RED(P_1, M_1)$
6. a	From 4 and 5 by Modus Ponens

Remember that the formula $\sim b \rightarrow a$ corresponds to the normal clause $a \leftarrow\ not\ b$ which belongs to the program $RED(P_1, M_1)$. The proof for the atom b, which also belongs to M_1, is similar to the proof of the atom a. Then we can conclude that $RED(P_1, M_1) \Vdash M_1$. Hence, M_1 is a *pstable model* of P_1.

2.4.3 Well-Founded Semantics

The Well-Founded Semantics (WFS) was introduced by A. V. Gelder *et al.* in (50). This semantics is one of the most accepted and studied semantics in logic programming. In fact this semantics is called a *well-behaved semantics* (36). Initially, this semantics was introduced in terms of a fix-point operator; however, in (22) it was introduced a characterization of WFS in terms of rewriting systems.

Since the characterization of WFS in terms of rewriting systems is most appropriated for the study of this thesis, we will define this characterization. Also we will define three extensions of this semantics which are based on two transformations rules: *LLC′* and *Weak-Cases*. In Chapter 4, we will see how the combination of WFS and rewriting systems will help to describe a *calculus of argumentation framework transformations* in order to define new abstract argumentation semantics and to define a procedural method for computing abstract argumentation semantics.

In order to define WFS, let us introduce some concepts. A general semantics SEM is a function on $\text{Prog}_{\mathcal{L}}$ which associates with every program a partial interpretation. Given a signature $\mathcal{L}$ and two semantics SEM_1 and SEM_2, we define $\text{SEM}_1 \leq_k \text{SEM}_2$ if for every program $P \in \text{Prog}_{\mathcal{L}}$, $\text{SEM}_1(P) \leq_k \text{SEM}_2(P)$. What are the minimal requirements we want to impose on a semantics? Certainly, we want to have facts that requirements, *i.e.* rules with empty bodies, treated as being true. Dually, if an atom does not occur in any head, then its negation should be true. This gives rise to the following definition, which will play an important role later.

Definition 2.5 (SEM) *For any logic program P, we define $HEAD(P) = \{a | \ a \leftarrow \mathcal{B}^+, \ \neg\mathcal{B}^- \ \in P\}$ — the set of all head-atoms of P. We also define $SEM(P) = \langle P^{true}, P^{false} \rangle$, where $P^{true} := \{p| \ p \leftarrow \top \in P\}$ and $P^{false} := \{p| \ p \in \mathcal{L}_P \backslash HEAD(P)\}$.*

Every rewriting system $\mathcal{CS}$ induces a semantics $\text{SEM}_{\mathcal{CS}}$ as follows:

$$\text{SEM}_{\mathcal{CS}}(P) := \text{SEM}(norm_{\mathcal{CS}}(P))$$

By considering the rewriting systems $\mathcal{CS}_0$ and the function SEM, WFS is characterized as follows:

Lemma 2.1 *(22) CS_0 is a confluent rewriting system. It induces a 3-valued semantics that it is the Well-founded Semantics.*

In order to illustrate the well-founded semantics. Let us consider the program P in Example 2.1. We have already seen that the *normal form* of P *w.r.t.* $\mathcal{CS}_0$ is:

$$d(b) \leftarrow \top.$$

This means that $WFS(P) := \langle \{d(b)\}, \{d(a), d(c)\} \rangle$.

This characterization of WFS based on rewriting systems has been useful for defining some extensions of WFS. We will define three extensions of WFS which were introduced in (39).

Lemma 2.2

a) *$\mathcal{CS}_1$ is a confluent rewriting system. It induces a 3-valued semantics that we call $WFS^{LLC'}$.*

b) *$\mathcal{CS}_2$ is a confluent rewriting system. It induces a 3-valued semantics that we call WFS^{WK}.*

c) *$\mathcal{CS}_3$ is a confluent rewriting system. It induces a 3-valued semantics that we call $WFS^{WK+LLC'}$.*

Observe that the differences between WFS and the semantics introduced by this lemma are made by the transformation rules: *LLC′* and *Weak-Cases*. In Chapter 4, we will see that these new semantics will be helpful for defining extensions of the so call grounded semantics in argumentation theory (See Section 2.6).

2.5 Possibilistic Logic

In this section, we will define some basic concepts of possibilistic logic for the case of necessity-valued formulæ.

Possibilistic logic is a weighted logic introduced and developed in the mid-1980s, in the setting of artificial intelligence, with the view to develop a simple an rigorous approach to automated reasoning from uncertain or prioritized incomplete information. Possibilistic logic is especially adapted to automated reasoning when the available information is pervaded with vagueness. In fact, possibilistic logic is a natural extension of classical logic where the notion of total order/partial order is embedded in the logic.

For the case of necessity-valued formulæ, a necessity-valued formula is a pair $(\varphi\ \alpha)$ where φ is a classical logic formula and $\alpha \in (0, 1]$ is a positive number. The pair $(\varphi\ \alpha)$ expresses that the formula φ is certain at least to the level α, *i.e.* $N(\varphi) \geq \alpha$, where N is a necessity measure modeling our possibly incomplete state knowledge (42). α is not a probability (like it is in probability theory) but it induces a certainty (or confidence) scale. This value is determined by the expert providing the knowledge base. A necessity-valued knowledge base is then defined as a finite set (*i.e.* a conjunction) of necessity-valued formulæ.

The following properties hold *w.r.t.* necessity-valued formulæ:

$$N(\varphi \wedge \psi) = min(\{N(\varphi), N(\psi)\}) \tag{2.1}$$

$$N(\varphi \vee \psi) \geq max(\{N(\varphi), N(\psi)\}) \tag{2.2}$$

$$\text{if } \varphi \vdash \psi \text{ then } N(\psi) \geq N(\varphi) \tag{2.3}$$

Dubois *et al*, in (42) introduced a formal system for necessity-valued logic which is based in the following axioms schemata (propositional case):

(A1) $(\varphi \rightarrow (\psi \rightarrow \varphi)\ 1)$

(A2) $((\varphi \rightarrow (\psi \rightarrow \xi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \xi))\ 1)$

(A3) $((\neg\varphi \rightarrow \neg\psi) \rightarrow ((\neg\varphi \rightarrow \psi) \rightarrow \varphi)\ 1)$

Inference rules:

(GMP) $(\varphi\ \alpha), (\varphi \rightarrow \psi\ \beta) \vdash (\psi\ min\{\alpha, \beta\})$

(S) $(\varphi\ \alpha) \vdash (\varphi\ \beta)$ if $\beta \leq \alpha$

According to Dubois *et al*, in (42), basically we need a complete lattice in order to express the levels of uncertainty in Possibilistic Logic. Dubois *et al*, extended the axioms schemata and the inference rules for considering partially ordered sets. We shall denote by $\vdash_{PL}$ the inference under Possibilistic Logic without paying attention if the necessity-valued formulæ are using either a totally ordered set or a partially ordered set for expressing the levels of uncertainty.

The problem of inferring automatically the necessity-value of a classical formula from a possibilistic base was solved by an extended version of *resolution* for possibilistic logic (see (42) for details).

One of the main basic principle of possibilistic logic is that:

Remark 2.2 *The strength of a conclusion is the strength of the weakest argument used in its proof.*

According to Dubois and Prade (43), the contribution of possibilistic logic setting is to relate this principle (measuring the validity of an inference chain by its weakest link) to fuzzy set-based necessity measures in the framework of Zadeh's possibilistic theory, since the following pattern then holds:

$$N(\sim p \vee q) \geq \alpha \text{ and } N(p) \geq \beta \text{ imply } N(q) \geq min(\alpha, \beta)$$

This interpretive setting provide a semantics justification to the claim that the weight attached to a conclusion should be the weakest among the weights attached to the formulæ involved in the derivation.

By considering the basic principles of possibilistic logic *e.g.*, Remark 2.2, we will define some possibilistic logic programming semantics (see Chapter 3) which in turn will define a semantics for possibilistic disjunctive logic programs. Moreover, we will define a possibilistic-based argumentation framework (see Chapter 6).

2.6 Abstract Argumentation Theory

Dung's approach, presented in (44), is a unifying framework which has played an influential role on argumentation research and AI. This approach is mainly orientated to manage the interaction of arguments. This interaction is managed by four abstract argumentation semantics: *stable semantics*, *preferred semantics*, *grounded semantics*, and *complete semantics*. The central notion of these semantics is the *acceptability of the arguments*. In this section, we will define some basic concepts of Dung's argumentation approach.

An argumentation framework captures the relationships between the arguments (all the definitions of this subsection were taken from the seminal paper (44)).

Definition 2.6 *An argumentation framework is a pair $AF := \langle AR, attacks \rangle$, where* AR *is a finite set of arguments, and* attacks *is a binary relation on* AR*,* i.e. attacks $\subseteq AR \times AR$*.*

Notice that the relation *attacks* does not yet tell us with which arguments a dispute can be won; it only tells us the relation of two conflicting arguments. In order to illustrate this definition let us consider the following example which was taken from (99).

Example 2.3 *Consider three arguments a, b and c such that a* attacks *b and b* attacks *c. A concrete version of this example is:*

c	=	*Tweety flies because it is a bird.*
b	=	*Tweety does not fly because it is a penguin.*
a	=	*The observation that Tweety is a penguin is unreliable.*

Notice that the argumentation framework which capture this example is $AF := \langle \{a,b,c\}, \{(a,b),(b,c)\} \rangle$. It is worth mentioning that any argumentation framework could be regarded as a directed graph. For instance, the graph representation of AF is presented in Figure 2.1.

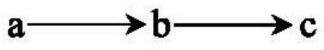

Figure 2.1: Graph representation of $AF := \langle \{a,b,c\}, \{(a,b),(b,c)\} \rangle$.

Definition 2.7 *A set S of arguments is said to be conflict-free if there are no arguments a, b in S such that a attacks b.*

A central notion of Dung's framework is *acceptability*. It captures how an argument that cannot defend itself, can be protected by a set of arguments.

Definition 2.8 *(1) .An argument $a \in AR$ is* acceptable w.r.t. *a set S of arguments iff for each argument $b \in AR$: If b attacks a then b is attacked by an argument in S. (2) A conflict-free set of arguments S is* admissible *iff each argument in S is acceptable* w.r.t. *S*

To illustrate this definition, let us consider Example 2.3. We can see that c is acceptable *w.r.t.* $\{a\}$, $\{a,b\}$, $\{a,c\}$ and $\{a,b,c\}$, but not *w.r.t.* $\{\}$ and $\{b\}$. Notice that $\{a,b,c\}$ and $\{a,b\}$ could not be admissible sets because they are not conflict-free sets. We can say that an admissible set represents a defendable point of view. For instance, in Example 2.3 there are three admissible sets: $\{\}$, $\{a\}$ and $\{a, c\}$. Intuitively, an admissible set is a coherent point of view. Since an argumentation framework could have several coherent point of views, one can take the maximum admissible sets in order to get maximum coherent point of views of an argumentation framework. This idea is captured by Dung's framework with the concept of *preferred extension*.

Definition 2.9 *A preferred extension of an argumentation framework AF is a maximal (*w.r.t. *inclusion) admissible set of AF.*

Since an argumentation framework could have more than one preferred extension, the preferred semantics is called credulous. The argumentation framework of Figure 2.1 has just one preferred extension which is $\{a, c\}$.

The grounded semantics is defined in terms of a *characteristic function.*

Definition 2.10 *The characteristic function, denoted by F_{AF}, of an argumentation framework $AF = \langle AR, attacks\rangle$ is defined as follows:*
$F_{AF} : 2^{AR} \rightarrow 2^{AR}$
$F_{AF}(S) = \{a|\ a \textit{ is acceptable} \text{ w.r.t. } S\ \}$

Definition 2.11 *The grounded extension of an argumentation framework AF, denoted by GE_{AF}, is the least fixed point of F_{AF}*

In order to illustrate the definition, let us consider the argumentation framework of Figure 2.1. Then

$$\begin{array}{ll} F^0_{AF}(\emptyset) := & \{a\}, \\ F^1_{AF}(F^0_{AF}(\emptyset)) := & \{a, c\}, \\ F^2_{AF}(F^1_{AF}(F^0_{AF}(\emptyset))) := & \{a, c\}, \end{array}$$

since $F^1_{AF}(F^0_{AF}(\emptyset)) = F^2_{AF}(F^1_{AF}(F^0_{AF}(\emptyset)))$, then $GE_{AF} = \{a, c\}$. Therefore the grounded extension of AF is $\{a, c\}$.

Another interesting semantics which was introduced in (44) is *stable semantics.*

Definition 2.12 *A conflict-free set of arguments S is called a stable extension if and only if S attacks each argument which does not belong to S.*

Dung showed that this semantics coincides with the notion of stable solutions of n-person games (44). There is an interesting relationship between the stable semantics and the preferred semantics which is that every stable extension is a preferred extension, but not vice versa.

Remark 2.3 *Hence we will say that any argument is defeated if and only if it is attacked by an acceptable argument.*

(44) defined some important concepts *w.r.t.* the relationship between arguments when they are taking part of a sequence of attacks.

- An argument b *indirectly attacks* a if there exists a finite sequence $a_0, \ldots, a_{2n+1}$ such that 1) $a = a_0$ and $b = a_{2n+1}$, and 2) for each i, $0 \leq i \leq 2n$, a_{i+1} attacks a_i.
- An argument b *indirectly defends* a if there exists a finite sequence $a_0, \ldots, a_{2n}$ such that 1) $a = a_0$ and $b = a_{2n}$ and 2) for each i, $0 \leq i \leq 2n$, a_{i+1} attacks a_i.
- An argument b is said to be *controversial w.r.t.* a if b indirectly attacks a and indirectly defeats a.
- An argument is *controversial* if it is controversial *w.r.t.* some argument a.

In (44), Dung suggested a general method for generating metainterpreters in terms of logic programming for argumentation systems. This is the first approach which regards an argumentation framework as a logic program. This metainterpreter is divided in two units: Argument Generation Unit (AGU), and Argument Processing Unit (APU). The AGU is basically the representation of the attacks in an argumentation framework and the APU consists of two clauses:

(C1) $acc(x) \leftarrow not\ d(x)$

(C2) $d(x) \leftarrow attack(y, x), acc(y)$

C1 suggests that the argument x is acceptable if it is not defeated and C2 suggests that an argument is defeated if it is attacked by an acceptable argument. Formally, the Dung's metainterpreter is defined as follows:

Definition 2.13 *Given an argumentation framework* $AF = \langle AR, attacks \rangle$, P_{AF} *denotes the logic program defined by* $P_{AF} = APU + AGU$ *where* $APU = \{C1, C2\}$ *and*

$$AGU = \{attacks(a, b) \leftarrow \top | (a, b) \in attacks\}$$

For each extension E of AF, m(E) is defined as follows:
$m(E) = AGU \cup \{acc(a) | a \in E\}$
$\cup \{defeat(b) | b\ is\ attacked\ by\ some\ a \in E\}$

Based on P_{AF}, Dung was able to characterize the stable semantics and the grounded semantics.

Theorem 2.1 *Let AF be an argumentation framework and E be an extension of AF. Then*

1. *E is a stable extension of AF if and only if $m(E)$ is an answer set of P_{AF}*
2. *E is a grounded extension of AF if and only if $m(E) \cup \{\neg defeat(a) | a \in E\}$ is the well-founded model of P_{AF}*

2.7 Lattices and order

In mathematics, especially order theory formalizes the intuitive concept of an ordering or an arrangement of the elements of a set. Indeed, the study of order has let to a great unification of results in algebra and logic. More recently, it has infused into theoretical computer science, particularly into programming language semantics (32). As we will see in Chapter 3, an order set will be a suitable structure for capturing uncertain information, particularly into answer set programs.

In this section, we will present some fundamental definitions of lattice theory in order to make this document self contained (see (32) for more details).

Definition 2.14 *Let $\mathcal{Q}$ be a set. An order (or partial order) on $\mathcal{Q}$ is a binary relation $\leq$ on $\mathcal{Q}$ such that, for all $x, y, z \in \mathcal{Q}$,*

(i) $x \leq x$

(ii) $x \leq y$ *and* $y \leq x$ *imply* $x = y$

(iii) $x \leq y$ *and* $y \leq z$ *imply* $x \leq z$

These conditions are referred to, respectively, as reflexivity, antisymmetry and transitivity.

A set $\mathcal{Q}$ equipped with an order relation $\leq$ is said to be an ordered set (or partial ordered set). It will be denoted by $(\mathcal{Q},\leq)$.

Definition 2.15 *Let $(\mathcal{Q},\leq)$ be an ordered set and let $S \subseteq \mathcal{Q}$. An element $x \in \mathcal{Q}$ is an upper bound of S if $s \leq x$ for all $s \in S$. A lower bound is defined dually. The set of all upper bounds of S is denoted by S^u (read as 'S upper') and the set of all lower bounds by S^l (read as 'S lower').*

If S^u has a least element x, then x is called the least upper bound ($\mathcal{LUB}$) of S. Equivalently, x is the least upper bound of S if

(i) x is an upper bound of S, and

(ii) $x \leq y$ for all upper bound y of S.

The least upper bound of S exists if and only if there exists $x \in \mathcal{Q}$ such that

$$(\forall y \in \mathcal{Q})[((\forall s \in S)s \leq y) \Longleftrightarrow x \leq y],$$

and this characterizes the $\mathcal{LUB}$ of S. Dually, if S^l has a greatest element, x, then x is called the greatest lower bound ($\mathcal{GLB}$) of S. Since least element and greatest element are unique, $\mathcal{LUB}$ and $\mathcal{GLB}$ are unique when they exist.

The least upper bound of S is called the supremum of S and it is denoted by *sup* S; the greatest lower bound of S is also called the infimum of S and it is denoted by $inf\ S$.

Definition 2.16 *Let* ($\mathcal{Q}$,$\leq$) *be a non-empty ordered set.*

(i) *If sup*$\{x, y\}$ *and* $inf\{x, y\}$ *exist for all* $x, y \in \mathcal{Q}$*, then* $\mathcal{Q}$ *is called lattice.*

(ii) *If sup* S *and* $inf\ S$ *exist for all* $S \subseteq \mathcal{Q}$*, then* $\mathcal{Q}$ *is called a complete lattice.*

Example 2.4 *Let us consider the set of labels* $\mathcal{Q} := \{$*Certain, Confirmed, Probable, Plausible, Supported, Open*$\}$[1] *and let* $\preceq$ *be a partial order such that the following set of relations holds:* $\{$*Open* $\preceq$ *Supported, Supported* $\preceq$ *Plausible, Supported* $\preceq$ *Probable, Probable* $\preceq$ *Confirmed, Plausible* $\preceq$ *Confirmed, Confirmed* $\preceq$ *Certain*$\}$. *A graphic representation of* S *according to* $\preceq$ *is showed in Figure 2.2. It is not difficult to see that* $(\mathcal{Q}, \preceq)$ *is a lattice and even more it is a complete lattice.*

[1]This set of labels was taken from (49). In this paper, authors argue that we can construct a set of labels (they call those: *modalities*) in a way that this set provides a simple scale for ordering the claims of our beliefs. In the following chapter, we will use this kind of labels for quantifying the uncertainty degree of a statement.

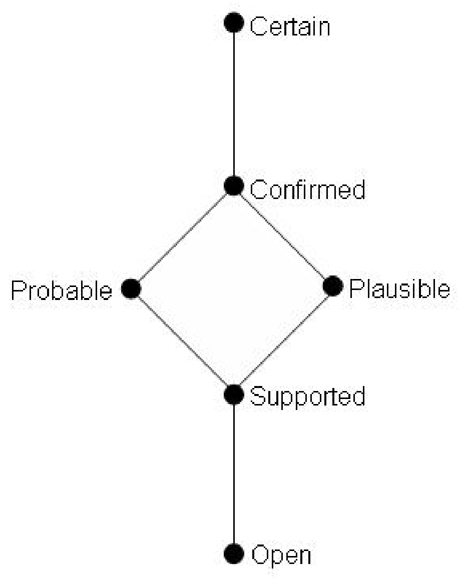

Figure 2.2: A graphic representation of a lattice where the following relations holds: $\{Open \preceq Supported$, $Supported \preceq Plausible$, $Supported \preceq Probable$, $Probable \preceq Confirmed$, $Plausible \preceq Confirmed$, $Confirmed \preceq Certain\}$.

Chapter 3

Possibilistic Disjunctive Logic programs

In this chapter, we will explore an approach for modeling uncertain, incomplete and inconsistent information. This approach will be based on Answer Set Programming and ideas of Possibilistic Logic.

3.1 Introduction

Uncertain and incomplete information is an unavoidable feature of daily decision-making. In order to deal with uncertain and incomplete information intelligently, we need to be able to represent it and reasoning about it.

Answer Set Programming (ASP) is one of the most successful logic programming approaches in Non-monotonic Reasoning and Artificial Intelligence applications (11; 51). In (76), a possibilistic framework for reasoning under uncertainty was proposed. This framework is a combination between ASP and possibilistic logic (42).

Possibilistic Logic is based on possibilistic theory where at the mathematical level, degrees of possibility and necessity are closely related to fuzzy sets (42). Thanks to the natural properties of possibilistic logic and ASP, Nicolas *et al*'s approach allows to deal with reasoning that is at the same time *non-monotonic* and *uncertain*. Nicolas *et al*'s approach is based on the concept of *possibilistic stable model* which defines a semantics for *possibilistic normal logic programs*.

To say that possibilistic disjunctive logic programs are required for representing incomplete information could be questioned; however, Gelfond and Lifschiz observed in (53) that:

> *An important limitation of traditional logic programming as a knowledge representation tool, in comparison with classical logic, is that*

> *logic programming does not allow us to deal directly with incomplete information.*

In order to overcome this limitation they suggested to include strong negation and disjunctive clauses in ASP for dealing incomplete information. In possibilistic answer set programming, one can also expect to use strong negation and possibilistic disjunctive clauses for dealing with incomplete information. An important feature of the possibilistic disjunctive clauses is that they are able to capture *incomplete information* and *incomplete states of a knowledge base* at the same time.

Let us consider a medical domain, in order to illustrate a scenario where uncertain and incomplete information is always presented. The particular medical domain that we consider is *human organ transplanting*. In human organ transplanting, one of the most sophisticated and complex processes is the organ donor selection. There are several factors that make this process sophisticated and complex. For instance:

- the transplant acceptance criteria vary ostensibly between transplant teams from the same geographical area and substantially between more distant transplantation teams (71). This means that the acceptance criteria applied in one hospital could be invalid or at least arguable in another hospital.
- there are lots of factors that make unpredictable if an organ donor's disease will be diagnosed in the organ recipient. For instance, if an organ donor D has hepatitis, then an organ recipient R could be infected by an organ of D. According to (70), there are cases where the infection can occur; however the recipient can *spontaneously* clear the infection *e.g.*, hepatitis. This means that an organ donor's infection can be *present* or *not-present* in the organ recipient. It is worth to comment that there are infections which can be prevented by applying a post-transplant treatment to the organ recipient.
- the clinical state of an organ recipient can be affected by several factors *e.g.*, malfunctions of the graft. This means that the clinical state of an organ recipient can be *stable* or *unstable* after the graft because the graft can have *good graft functions*, *delayed graft functions* and *terminal insufficient functions*[1].

It is worth mentioning that the transplant acceptance criteria may be dependant to the kind of organ (kidney, heart, liver, *etc.*) that will be considered for transplanting and the clinical situation of the potential organ recipients.

[1]Usually, when a doctor says that an organ has *terminal insufficient functions*, it means that there exists no clinical treatment for improving the organ's functions.

Let us consider the particular case of kidney transplantation with organ donors that have a kind of infection *e.g.*, endocarditis, hepatitis. As already stated, the clinical situation of the potential organ recipients is relevant in the organ transplanting process. We will denote the clinical situation of an organ recipient by the predicate $cs(t, T)$, such that t can be *stable*, *unstable*, 0-*urgency* and T denotes a moment in the time. Another important factor that we will consider is the state of the organ's functions. We will denote by the predicate $o(t, T)$ the state of the organ's functions, such that t can be *terminal-insufficient functions*, *good-graft functions*, *delayed-graft functions*, *normal-graft functions* and T denotes a moment in the time. Also, we will consider in our scenario the state of an infection in both the organ recipient and the organ donor, this condition will be denoted by the predicates $r\_inf(present, T)$ and $d\_inf(present, T)$ respectively such that T denotes a moment in the time. The last predicate that we introduce, will be $action(t, T)$ such that t can be *transplant*, *wait*, *post-transplant treatment* and T denotes a moment in the time. This predicate denotes the possible actions of a doctor. In Figure 3.1[1], a finite state automata is presented — for a formal presentation of automata theory see, (57). In this automata each node represents a possible situation where an organ recipient can be found and the arrows represent the possible doctor's actions. Observe that we are assuming that in the initial state the organ recipient is clinically stable and he does not have an infection; however, he has a kidney whose functions are terminal insufficient. From the initial state, the doctor's actions can be either to make a kidney transplantation or just wait[2].

According to Figure 3.1, an organ recipient can be found in different situations after a graft. In fact, the organ recipient may require another graft and the state of the infection can be unpredictable. Let us introduce extended disjunctive clauses which describe some situations presented in Figure 3.1

$$\begin{aligned} r\_inf(present, T2) \vee \neg r\_inf(present, T2) \leftarrow\ & action(transplant, T), \\ & d\_inf(present, T), T2 = T + 1. \end{aligned}$$

$$\begin{aligned} & o(good\_graft\_funct, T2) \vee o(delayed\_graft\_funct, T2) \vee \\ & o(terminal\_insufficient\_funct, T2) \leftarrow action(transplant, T), T2 = T + 1. \end{aligned}$$

The intended meaning of the first clause is that if the organ donor has an infection, then the infection can be *present* or *not-present* in the organ recipient, after the graft, and the intended meaning of the second one is that the graft's functions can

[1]This finite state automata was built under the supervision of Francisco Caballero M. D. Ph. D. from the Hospital de la Santa Creu I Sant Pau, Barcelona, Spain.

[2]In the automata of Figure 3.1, we are not considering that there is a waiting list where the organ recipient waits for an organ. This waiting list has different politics for assigning an organ to an organ recipient.

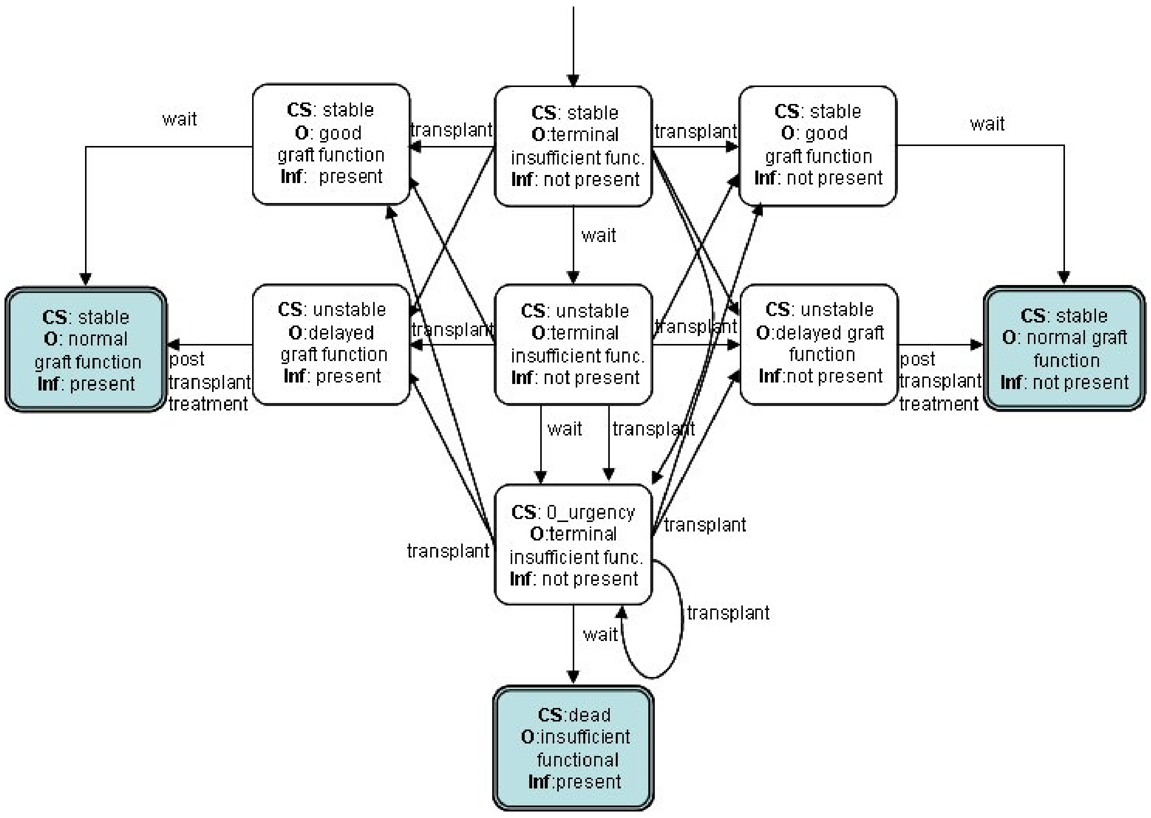
CS: stable O: good graft function Inf: present
CS: stable O:terminal insufficient func. Inf: not present
CS: stable O: good graft function Inf: not present
CS: stable O: normal graft function Inf: present
CS: unstable O:delayed graft function Inf: present
CS: unstable O:terminal insufficient func. Inf: not present
CS: unstable O:delayed graft function Inf:not present
CS: stable O: normal graft function Inf: not present
CS: 0_urgency O:terminal insufficient func. Inf: not present
CS:dead O:insufficient functional Inf:present
wait
transplant
post transplant treatment

Figure 3.1: An automata of finite states and actions for considering infections in kidney organ transplanting.

be: *good*, *delayed* and *terminal*, after the graft. Observe that these clauses are not capturing the uncertainty that is involved in each statement. For instance, *w.r.t.* the first clause, one can wish to represent an uncertainty degree in order to capture the uncertainty that is involved in this statement — remember that the organ recipient can be infected by the infection of the donor's organ; however, the infection can be *spontaneously* cleared by the organ recipient as it is the case of hepatitis (70).

In Section 1.1, we remarked that Tversky and Kahneman have observed in (107) that we commonly use statements such as "*I think that* ...", "*chances are* ...", "*it is probable that* ...", "*it is plausible that* ...", *etc.*, for supporting our decisions. Observe that these statements have as common denominator adjectives which quantify the information. These adjectives are of the form: *probable*, *plausible*, *etc.* Based on this observation, we propose to use adjectives/labels of the same kind in order to quantify the uncertain information of a knowledge base. The only formal requirement is that this set of adjectives/labels must be a *complete lattice*. For instance, for the case of our medical scenario a transplant coordinator[1] can suggest a set of labels in order to quantify a medical knowledge base and of course to define an order between those labels. Based on those labels we can have possibilistic clauses as:

probable: $r\_inf(present, T2) \vee \neg r\_inf(present, T2) \leftarrow action(transplant, T), d\_inf(present, T), T2 = T + 1.$

Informally speaking, the reading of this clause is: *if the organ donor has an infection, then it is* probable *that the organ recipient can be infected or not after a graft.*

In this chapter, we will introduce the use of *possibilistic disjunctive clauses* which are able to capture *incomplete information* and *incomplete states of a knowledge base* at the same time. In order to capture the semantics of possibilistic disjunctive programs, we will present three possible approaches. Two of these approaches will be based on answer set models and the other one will be based on pstable models. As part of our study, we will present some approaches for managing the inconsistency of a possibilistic knowledge. Also, we will take care of the relationship that there is between the approach presented by Nicolas *et al*, in (76) and our approach.

The rest of the chapter is divided as follows: In §3.2, the syntax of our possibilistic framework is presented. In §3.3, three approaches for defining the semantics of possibilistic disjunctive programs are presented. In §3.4, some criteria for managing inconsistent possibilistic logic programs are defined. Finally, in the last section, we present our conclusions.

3.2 Syntax

In this section, we will define the general syntax for possibilistic disjunctive logic programs. This syntax will be based on the standard syntax of extended disjunctive logic programs (see Section 2.1). First of all, we start defining some relevant concepts[2].

In all the document, we will only consider finite lattices — remember that any finite lattice is complete. This convention was taken based on the assumption that in real applications we will rarely have an infinite set of labels for expressing the incomplete state of a knowledge base.

[1]A transplant coordinator is an expert in all the process of transplanting (71).

[2]Some concepts presented in this section extend some terms presented in (76).

A *possibilistic atom* is a pair $p = (a, q) \in \mathcal{A} \times \mathcal{Q}$, where $\mathcal{A}$ is a finite set of atoms and $(\mathcal{Q}, \leq)$ is a lattice. We apply the projection $*$ to any possibilistic atom p as follows: $p^* = a$. Given a set of possibilistic atoms S, we define the generalization of $*$ over S as follows: $S^* = \{p^* | p \in S\}$.

Now, we define the syntax of a valid possibilistic logic program. Let $(\mathcal{Q}, \leq)$ be a lattice. A possibilistic disjunctive clause R is of the form:

$$\alpha : \mathcal{A} \leftarrow \mathcal{B}^+, \ not\ \mathcal{B}^-$$

where $\alpha \in \mathcal{Q}$ and $\mathcal{A} \leftarrow \mathcal{B}^+, \ not\ \mathcal{B}^-$ is an extended disjunctive clause as defined in Section 2.1. The projection $*$ for a possibilistic clause is $R^* = \mathcal{A} \leftarrow \mathcal{B}^+, \ not\ \mathcal{B}^-$. $n(R) = \alpha$ is a necessity degree representing the certainty level of the information described by R. A possibilistic constraint C is of the form:

$$\mathcal{TOP}_{\mathcal{Q}} : \bot \leftarrow \mathcal{B}^+, \ not\ \mathcal{B}^-$$

where $\mathcal{TOP}_{\mathcal{Q}}$ is the top of the lattice $(\mathcal{Q}, \leq)$ and $\bot \leftarrow \mathcal{B}^+, \ not\ \mathcal{B}^-$ is a constrain as defined in Section 2.1. As in possibilistic clauses, the projection $*$ for a possibilistic constraint is : $C^* = \bot \leftarrow \mathcal{B}^+, \ not\ \mathcal{B}^-$. Observe that any possibilistic constraint must have the top of the lattice $(\mathcal{Q}, \leq)$, this restriction is motivated on the fact that like a constraint in standard ASP, the purpose of a possibilistic constraint is to eliminate possibilistic models. Hence, it is assumed that there does not exists doubt about the veracity of the information captured by a possibilistic constraint.

A possibilistic disjunctive logic program P is a tuple of the form $\langle(\mathcal{Q}, \leq), N\rangle$, where N is a finite set of possibilistic disjunctive clauses and possibilistic constraints. The generalization of $*$ over P is as follows: $P^* = \{r^* | r \in N\}$. Notice that P^* is an extended disjunctive program. When P^* is a normal program, P is called a possibilistic normal program. Also, when P^* is a positive disjunctive program, P is called a possibilistic positive logic program and so on. A given set of possibilistic disjunctive clauses $\{\gamma, \dots, \gamma\}$ is also represented as $\{\gamma; \dots; \gamma\}$ to avoid ambiguities with the use of the comma in the body of the clauses.

Given a possibilistic disjunctive logic program $P = \langle(\mathcal{Q}, \leq), N\rangle$, we define the *$\alpha$-cut* and the *strict α-cut* of P, denoted respectively by P_α and $P_{\overline{\alpha}}$, by

$$P_\alpha = \langle(\mathcal{Q}, \leq), N_\alpha\rangle \text{ such that } N_\alpha = \{c | c \in N \text{ and } n(c) \geq \alpha\}$$
$$P_{\overline{\alpha}} = \langle(\mathcal{Q}, \leq), N_{\overline{\alpha}}\rangle \text{ such that } N_{\overline{\alpha}} = \{c | c \in N \text{ and } n(c) > \alpha\}$$

Example 3.1 *In order to illustrate a possibilistic program, let us go back to our scenario described in Section 3.1. We will consider the lattice of Example 2.4; hence, let $(\mathcal{Q}, \preceq)$ be the lattice of Figure 2.2 such that the relation $A \preceq B$ means that A is less possible that B. The possibilistic program $P := \langle(\mathcal{Q}, \preceq), N\rangle$ will be the following set of possibilistic clauses:*

In the introduction, we presented the following possibilistic clause:

probable: $r\_inf(present, T2) \vee \neg r\_inf(present, T2) \leftarrow action(transplant, T),$
$d\_inf(present, T), T2 = T + 1.$

Remember that the intended meaning of the first clause is that if the organ donor has an infection, then it is probable that the organ recipient can be infected or not after a graft.
The intended meaning of the following clause is that it is confirmed that the organ's functions can be: good, delayed and terminal after a graft.

confirmed: $o(good\_graft\_funct, T2) \vee o(delayed\_graft\_funct, T2) \vee$
$o(terminal\_insufficient\_funct, T2) \leftarrow action(transplant, T), T2 = T + 1.$

The intended meaning of the following clause is that it is confirmed that if the organ's functions are terminal insufficient then it is necessary a transplanting.

confirmed: $action(transplant, T) \leftarrow o(terminal\_insufficient\_funct, T).$

The intended meaning of the following clause is that it is plausible that the clinical situation of the organ recipient can be stable if the graft's functions are good.

plausible: $cs(stable, T) \leftarrow o(good\_graft\_funct, T).$

The intended meaning of the following clause is that it is plausible that the clinical situation of the organ recipient can be unstable if the graft's functions are delayed.

plausible: $cs(unstable, T) \leftarrow o(delayed\_graft\_funct, T).$

The intended meaning of the following clause is that it is plausible that the clinical situation of the organ recipient can be of 0-urgency if the graft's functions are terminal insufficient after the graft.

plausible: $cs(\textit{0-urgency}, T2) \leftarrow o(terminal\_insufficient\_funct, T2),$
$action(transplant, T), T2 = T + 1.$

The intended meaning of the following possibilistic constraint is that it is certain that the doctor cannot do two actions at the same time.

certain: $\perp \leftarrow action(transplant, T), action(wait, T).$

The intended meaning of the following possibilistic constraint is that it is certain that a transplanting cannot be done if the organ recipient is dead.

certain: $\bot \leftarrow action(transplant, T), cs(dead, T)$.

The initial state of the automata of Figure 3.1 is captured by the following possibilistic clauses:

certain: $d\_inf(present, 0) \leftarrow \top$.
certain: $\neg r\_inf(present, 0) \leftarrow \top$.
certain: $o(terminal\_insufficient\_funct, 0) \leftarrow \top$.
certain: $cs(stable, 0) \leftarrow \top$.

3.3 Semantics

In §3.2, we defined the syntax for any possibilistic disjunctive program. Now, in this section, we will study the semantics for these programs. Essentially, we have explored three approaches for capturing the semantics of possibilistic disjunctive programs. The first two are based on answer set models and the last one is based on pstable models. We start by presenting the approaches which are based on answer set models.

We will consider sets of atoms as interpretations; hence, before to define the possibilistic logic programming semantics, we will introduce two basic operations between sets of possibilistic atoms and a relation of order between them.

Definition 3.1 *Let $\mathcal{A}$ be a finite set of atoms and $(\mathcal{Q},\leq)$ be a lattice. Consider $\mathcal{PS} = 2^{\mathcal{A}\times\mathcal{Q}}$ the finite set of all the possibilistic atoms sets induced by $\mathcal{A}$ and $\mathcal{Q}$. $\forall A, B \in \mathcal{PS}$, we define.*

$$\begin{array}{ll} A \sqcap B & = \{(x, \mathcal{GLB}(\{q_1, q_2\}))|(x, q_1) \in A \wedge (x, q_2) \in B\} \\ A \sqcup B & = \{(x, q)|(x, q) \in A \textit{ and } x \notin B^*\} \cup \\ & \{(x, q)|x \notin A^* \textit{ and } (x, q) \in B\} \cup \\ & \{(x, \mathcal{LUB}(\{q_1, q_2\}))|(x, q_1) \in A \textit{ and } (x, q_2) \in B\}. \\ A \sqsubseteq B & \iff A^* \subseteq B^*, \textit{ and } \forall x, q_1, q_2, \\ & (x, q_1) \in A \wedge (x, q_2) \in B \textit{ then } q_1 \leq q_2. \end{array}$$

This definition is almost the same as Definition 7 presented in (76). The only difference is that in Nicolas *et al*'s definition, the operators *min* and *max* are used instead of the operators $\mathcal{GLB}$ and $\mathcal{LUB}$ for defining the operations $\sqcap$ and $\sqcup$[1]. The following proposition is a straightforward consequence of Proposition 6 of (76).

[1]The operators $\sqcap$ and $\sqcup$ were defined as follows in (76):

Proposition 3.1 $(\mathcal{PS}, \sqsubseteq)$ *is a complete lattice.*

3.3.1 A possibilistic semantics based on answer set models

We will define a possibilistic semantics which is strictly close to the proof theory of possibilistic logic and answer set models. Like answer set semantics' definition, our approach has as its base a syntactic reduction. In fact, this reduction is inspired in the Gelfond-Lifschitz reduction.

Definition 3.2 (Reduction P^M) *Let $P = \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic disjunctive logic program, M be a set of atoms. P reduced by M is the positive possibilistic disjunctive logic program:*

$$P^M := \quad \{(n(r) : \mathcal{A} \cap M \leftarrow \mathcal{B}^+) | r \in N, \mathcal{A} \cap M \neq \emptyset, \ \mathcal{B}^- \cap M = \emptyset, \mathcal{B}^+ \subseteq M\}$$

where r^* is of the form $\mathcal{A} \leftarrow \mathcal{B}^+, \ not\ \mathcal{B}^-$.

Notice that $(P^*)^M$ is not exactly the Gelfond-Lifschitz reduction. In fact, our reduction is stronger than Gelfond-Lifschitz reduction when P^* is a disjunctive program. For instance, let us consider the following table.

P	$P^{\{c,b\}}$	$(P^*)^{\{c,b\}}$
$\alpha_1 : a \vee b \leftarrow \top.$	$\alpha_1 : b \leftarrow \top.$	$a \vee b \leftarrow \top.$
$\alpha_2 : c \leftarrow \ not\ a.$	$\alpha_2 : c \leftarrow \top.$	$c \leftarrow \top.$
$\alpha_3 : c \leftarrow \ not\ b.$		

In the first column, the possibilistic program P is defined, in the second one the program P is reduced by $\{c, b\}$ (according to Definition 3.2) and in the third one the program P^* is reduced by $\{c, b\}$ (according to Gelfond-Lifschitz reduction). Observe that the reduction of Definition 3.2 removes from the head of the possibilistic disjunctive clauses any atom which does not belong to M. As we will see in Section 3.3.2, this property will be helpful for defining a possibilistic semantics for possibilistic disjunctive programs based on a fix-point operator.

Example 3.2 *In order to continue with our medical scenario described in the introduction, let P be a ground instance of the possibilistic program presented in Example 3.1:*

$A \sqcap B = \{(x, min(\{q_1, q_2\})|(x, q_1) \in A \wedge (x, q_2) \in B\}$ and
$A \sqcup B = \{(x, q)|(x, q) \in A \ and \ x \notin B^*\} \cup \{(x, q)|x \notin A^* \ and \ (x, q) \in B\} \cup \{(x, max(\{q_1, q_2\})|(x, q_1) \in A$ and $(x, q_2) \in B\}$.

probable: $r\_inf(present,1) \vee no\_r\_inf(present,1) \leftarrow action(transplant,0),$
$d\_inf(present,0).$
confirmed: $o(good\_graft\_funct,1) \vee o(delayed\_graft\_funct,1) \vee$
$o(terminal\_insufficient\_funct,1) \leftarrow action(transplant,0).$
confirmed: $action(transplant,0) \leftarrow o(terminal\_insufficient\_funct,0).$
plausible: $cs(stable,1) \leftarrow o(good\_graft\_funct,1).$
plausible: $cs(unstable,1) \leftarrow o(delayed\_graft\_funct,1).$
plausible: $cs(\mathit{0\text{-}urgency},1) \leftarrow o(terminal\_insufficient\_funct,1),$
$action(transplant,0).$
certain: $\bot \leftarrow action(transplant,0), action(wait,0).$
certain: $\bot \leftarrow action(transplant,0), cs(dead,0).$
certain: $d\_inf(present,0) \leftarrow \top.$
certain: $no\_r\_inf(present,0) \leftarrow \top.$
certain: $o(terminal\_insufficient\_funct,0) \leftarrow \top.$
certain: $cs(stable,0) \leftarrow \top.$

Observe that the variables of time T and $T2$ were instantiated with the values 0 and 1 respectively; moreover, observe that the atoms $\neg r\_inf(present,0)$ and $\neg r\_inf(present,1)$ were replaced by $no\_r\_inf(present,0)$ and $no\_r\_inf(present,1)$ respectively. This change was applied in order to manage the strong negation.

Now, let S be the following possibilistic set:

$$\begin{aligned} S = \; & \{(d\_inf(present,0), certain),\ (no\_r\_inf(present,0), certain),\\ & (o(terminal\_insufficient\_funct,0), certain),\ (cs(stable,0), certain),\\ & (action(transplant,0), confirmed),\ (o(good\_graft\_funct,1), confirmed),\\ & (cs(stable,1), plausible),\ (no\_r\_inf(present,1), probable)\}. \end{aligned}$$

We can see that P^{S^*} is:

probable: $no\_r\_inf(present,1) \leftarrow action(transplant,0), d\_inf(present,0).$
confirmed: $o(good\_graft\_funct,1) \leftarrow action(transplant,0).$
confirmed: $action(transplant,0) \leftarrow o(terminal\_insufficient\_funct,0).$
plausible: $cs(stable,1) \leftarrow o(good\_graft\_funct,1).$
plausible: $cs(unstable,1) \leftarrow o(delayed\_graft\_funct,1).$
plausible: $cs(\mathit{0\text{-}urgency},1) \leftarrow o(terminal\_insufficient\_funct,1),$
$action(transplant,0).$
certain: $\bot \leftarrow action(transplant,0), action(wait,0).$
certain: $\bot \leftarrow action(transplant,0), cs(dead,0).$
certain: $d\_inf(present,0) \leftarrow \top.$
certain: $no\_r\_inf(present,0) \leftarrow \top.$
certain: $o(terminal\_insufficient\_funct,0) \leftarrow \top.$

certain: $cs(stable, 0) \leftarrow \top$.

Once a possibilistic logic program P has been reduced by a set of possibilistic literals M, it is possible to test whether M is a possibilistic answer set of the program P by considering the following definition.

Definition 3.3 (Possibilistic answer set) *Let $P = \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic disjunctive logic program and M be a set of possibilistic atoms such that M^* is an answer set of P^*. M is a possibilistic answer set of P if and only if $P^{M^*} \vdash_{PL} M$ and $\nexists M' \in \mathcal{PS}$ such that $M' \neq M$, $P^{(M')^*} \vdash_{PL} M'$ and $M \sqsubseteq M'$.*

Example 3.3 *Let P be again the possibilistic program of Example 3.1 and S be the possibilistic set of atoms introduced in Example 3.2. First of all, we can see that S^* is an answer set of the extended disjunctive program P^*. Hence, in order to prove that S is a possibilistic answer set of P, we have to verify that $P^{S^*} \vdash_{PL} S$. This means that for each possibilistic atom $p \in S$, $P^{S^*} \vdash_{PL} p$. We can see that it is straightforward that*

$$\begin{aligned} P^{S^*} \vdash_{PL} \quad & \{(d\_inf(present, 0), certain),\ (no\_r\_inf(present, 0), certain), \\ & (o(terminal\_insufficient\_funct, 0), certain), \\ & (cs(stable, 0), certain)\} \end{aligned}$$

Now let us prove $(cs(stable, 1), plausible)$ from P^{S^*}.

Premises from P^{S^*}	
1. $o(terminal\_insufficient\_funct, 0)$	$certain$
2. $o(terminal\_insufficient\_funct, 0) \rightarrow action(transplant, 0)$	$confirmed$
3. $action(transplant, 0) \rightarrow o(good\_graft\_funct, 1)$	$confirmed$
4. $o(good\_graft\_funct, 1) \rightarrow cs(stable, 1)$.	$plausible$
From 1 and 2 by GMP	
5. $action(transplant, 0)$	$confirmed$
From 3 and 5 by GMP	
6. $o(good\_graft\_funct, 1)$	$confirmed$
From 4 and 6 by GMP	
7. $cs(stable, 1)$.	$plausible$

In this proof, we can also see the inference of the possibilistic atom $(action(transplant, 0), confirmed)$. The proof of the possibilistic atom $(no\_r\_inf(present, 1), probable)$ is similar to the proof of the possibilistic atom $(cs(stable, 1), plausible)$. Therefore, we can say that $P^{S^*} \vdash_{PL} S$ is true. Now, notice that there does not exists a possibilistic set S' such that $S' \neq S$, $P^{(S')^*} \vdash_{PL} S'$ and $S \sqsubseteq S'$; hence, we can conclude that S is a possibilistic answer set of P.

Now what can we say from S about our medical scenario? We can say that if it is confirmed *that a transplanting is done with a donor with an infection, it is* probable *that the recipient cannot be infected after the transplanting; moreover it is* plausible *that he can be stable. It is worth mentioning that this* optimistic *conclusion is just one of the possible scenarios that we can infer from the program P. In fact, the program P has six possibilistic answer sets where we can find pessimistic scenarios such as it is* probable *that the recipient is infected by the organ donor's infection and; moreover, it is* confirmed *that the recipient needs another transplanting.*

Now, let us study some properties of the possibilistic answer set semantics. First, observe that there is an important condition *w.r.t.* the definition of a *possibilistic answer set*. This is that a possibilistic set S cannot be a possibilistic answer set of a possibilistic logic program P if S^* is not an answer set of the extended logic program P^*. This condition guarantees that any clause of P^* is satisfied by M^*. For instance, let us consider the possibilistic logic program P:

$0.4 : a.$ $\qquad$ $0.6 : b.$

and the possibilistic set $S = \{(a, 0.4)\}$. We can see that $P^{S^*} \vdash_{PL} S$; however, S^* is not an answer set of P^*. Therefore, S could not be a possibilistic answer set of P. Hence, a straightforward relation between the possibilistic answer semantics and the answer set semantics is formalized by the following proposition.

Proposition 3.2 *Let P be a possibilistic disjunctive logic program. M is a possibilistic answer set of P iff M^* is an answer set of P^*.*

When all the possibilistic clauses of a possibilistic program P have as certain level the top of the lattice that was considered in P, the answer sets of P^* can be directly generalized to the possibilistic answer sets of P.

Proposition 3.3 *Let $P = \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic disjunctive logic program and $\mathcal{TOP}_{\mathcal{Q}}$ be the top of the lattice $(\mathcal{Q}, \leq)$. If $\forall r \in P$, $n(r) = \mathcal{TOP}_{\mathcal{Q}}$, and M' is an answer set of P^*, then $M := \{(l, \mathcal{TOP}_{\mathcal{Q}}) | l \in M'\}$ is a possibilistic answer set of P.*

For the class of possibilistic normal logic programs which are built under a totally ordered set, our definition of possibilistic answer set is closely related to the definition of a *possibilistic stable model* presented in (76). In fact, both semantics coincide.

Proposition 3.4 *Let $P := \langle(Q, \leq), N\rangle$ be a possibilistic normal program such that $(Q, \leq)$ is a totally ordered set and $\mathcal{L}_P$ has no extended atoms. M is a possibilistic answer set of P if and only if M is a possibilistic stable model of P.*

In order to prove that the possibilistic answer set semantics is computable, we will introduce a straightforward generalization of the possibilistic resolution rule introduced in (42):

(R) $(c_1\ \alpha_1)(c_2\ \alpha_2) \vdash (R(c_1, c_2)\ \mathcal{GLB}(\{\alpha_1, \alpha_2\}))$

where $R(c_1, c_2)$ is any classical resolvent of c_1 and c_2 such that c_1 and c_2 are disjunctions of literals. It is worth mentioning that it is easy to transform any possibilistic disjunctive logic program P into a set of possibilistic disjunctions $\mathcal{C}$. Indeed, $\mathcal{C}$ can be obtained as follows:

$$\mathcal{C} := \bigcup\{(a_1 \vee \ldots \vee a_m \vee \sim a_1 \vee \cdots \vee \sim a_j \vee a_{j+1} \vee \ldots, a_n\ \alpha)| \\ (\alpha : a_1 \vee \ldots \vee a_m \leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n) \in P\}$$

We remember to the reader that whenever we consider a possibilistic program as a theory, each negative literal *not* a is replaced by $\sim a$ such that $\sim$ is regarded as the negation in classic logic — in Example 3.4, the transformation of a possibilistic program into a set of possibilistic disjunctions is shown.

The following proposition shows that the resolution rule (R) is sound.

Proposition 3.5 *Let $\mathcal{C}$ be a set of possibilistic disjunctions, and $C = (c\ \alpha)$ be a possibilistic clause obtained by a finite number of successive application of* (R) *to $\mathcal{C}$; then $\mathcal{C} \vdash_{PL} C$.*

Like the possibilistic rule introduced in (42), (R) is complete for refutation. We will say that a possibilistic disjunctive program P is *consistent* if P has at least a possibilistic answer set. Otherwise P is said to be *inconsistent*. The inconsistency degree of a possibilistic logic program P is $Inc(P) = \mathcal{GLB}(\{\alpha | P_\alpha \text{ is consistent }\})$.

Proposition 3.6 *Let P be a set of possibilistic clauses and $\mathcal{C}$ be the set of possibilistic disjunctions obtained from P; then the valuation of the optimal refutation by resolution from $\mathcal{C}$ is the inconsistent degree of P.*

The main implication of Proposition 3.5 and Proposition 3.6 is that (R) suggests a method for inferring a possibilistic formula from a possibilistic knowledge base.

Corollary 3.1 *Let $P := \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic disjunctive logic program, φ be literal and $\mathcal{C}$ be a set of possibilistic disjunctions obtained from $N \cup \{(\sim \varphi\ \mathcal{TOP}_{\mathcal{Q}})\}$; then the valuation of the optimal refutation from $\mathcal{C}$ is $n(\varphi)$* i.e. *$P \vdash_{PL} (\varphi\ n(\varphi))$.*

Based on the fact that the resolution rule (R) suggests a method for inferring the necessity value of a possibilistic formula, we can define the following function for computing the possibilistic answer set models of an possibilistic program P.

```
Function $Poss\_Answer\_Sets(P)$
Let $ASP(P^*)$ be a function that computes the answer set models of the standard
logic program $P^*$ e.g., DLV (40).
    Poss-ASP := $\emptyset$
    For all $S \in ASP(P^*)$
        Let $\mathcal{C}$ be the set of possibilistic disjunctions obtained from $P^S$.
        $S' := \emptyset$
        for all $a \in S$
            $C' := \mathcal{C} \cup \{(\sim a\ \mathcal{TOP}_{\mathcal{Q}})\}$
            Search for a deduction of $(R(\Box)\ \alpha)$ by applying repeatedly
            the resolution rule (R) from $C'$, with $\alpha$ maximal.
            $S' := S' \cup \{(a\ \alpha)\}$
        endfor
        Poss-ASP := Poss-ASP $\cup\ S'$
    endfor
return(Poss-ASP).
```

The following proposition formalizes that the function *Poss_Answer_Sets* computes all the possibilistic answer sets of a possibilistic logic program.

Proposition 3.7 *Let $P := \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic logic program. The set Poss-ASP returned by Poss_Answer_Sets(P) is the set of all the possibilistic answer sets of P.*

In order to illustrate this algorithm, let us consider the following example:

Example 3.4 *Let $P := \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic program such that $\mathcal{Q} :=$ {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}, $\leq$ is the standard relation between rational numbers and N the following set of possibilistic clauses:*

$$\begin{array}{lrl}
0.7: & a \vee b & \leftarrow not\ c. \\
0.6: & c & \leftarrow not\ a, not\ b. \\
0.8: & a & \leftarrow b. \\
0.9: & e & \leftarrow b. \\
0.6: & b & \leftarrow a. \\
0.5: & b & \leftarrow a.
\end{array}$$

Firs of all, we can see that P^* has two answer sets: $S_1 := \{a, b, e\}$ and $S_2 := \{c\}$. This means that P has two possibilistic answer set models. Let us consider S_1 for our example. Then, one can see that P^{S_1} is:

$$\begin{array}{lrl} 0.7: & a \vee b & \leftarrow \top. \\ 0.8: & a & \leftarrow b. \\ 0.9: & e & \leftarrow b. \\ 0.6: & b & \leftarrow a. \\ 0.5: & b & \leftarrow a. \end{array}$$

Then $\mathcal{C} := \{(a \vee b\ 0.7), (a\vee \sim b\ 0.8), (e\vee \sim b\ 0.9), (b\vee \sim a\ 0.6), (b\vee \sim a\ 0.5)\}$. In order to infer the necessity value of the atom a, we add $(\sim a\ 1)$ to $\mathcal{C}$ and a search for finding an optimal refutation is applied. As we can see in Figure 3.2, there are three refutations, however the optimal refutation is $(\Box\ 0.7)$. This means that the best necessity value for the atom a is 0.7.

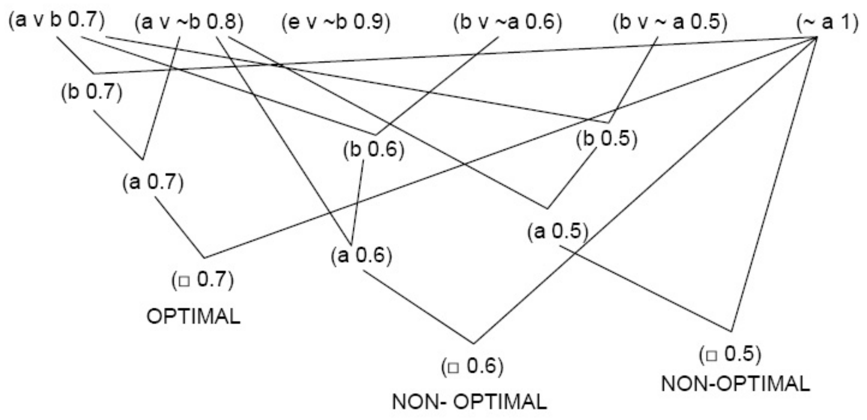

Figure 3.2: Possibilistic resolution: Search for an *optimal refutation* for the atom a.

In Figure 3.3, we can see the optimal refutation search for the atom b. As we can see the optimal refutation is $(\Box\ 0.6)$; hence the best necessity value for the atom b is 0.6.

In Figure 3.4, we can see that the best necessity value for the atom e is 0.6.

From the process of search, we can infer that a possibilistic answer set of the program P is : $\{(a, 0.7), (b, 0.6), (e, 0.6)\}$.

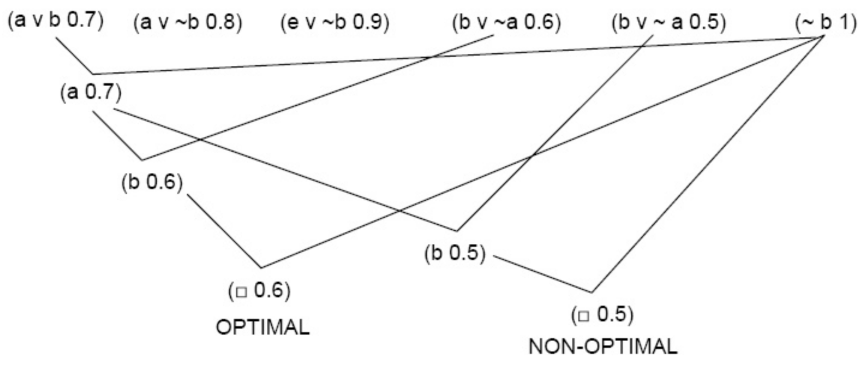

Figure 3.3: Possibilistic resolution: Search for an *optimal refutation* for the atom b.

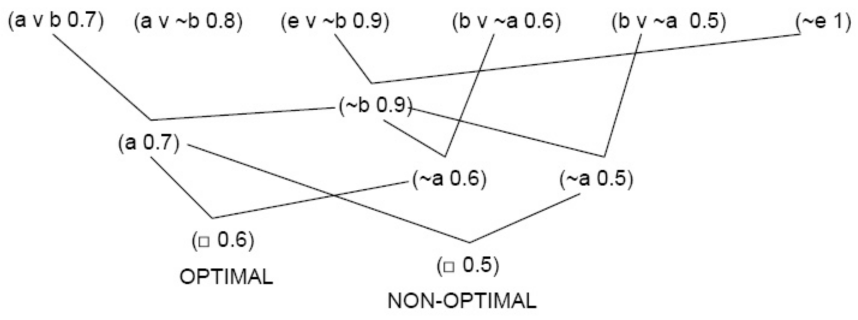

Figure 3.4: Possibilistic resolution: Search for an *optimal refutation* for the atom e.

3.3.2 Possibilistic answer sets based on partial evaluation

We have defined a possibilistic answer set semantics by considering the formal proof theory of possibilistic logic. However, in standard logic programming there are several frameworks for analyzing, defining and computing logic programming semantics (35; 36). One of these approaches is based on program transformations, in fact there are many studies around this approach *e.g.*, (19; 20; 21; 39). For the case of disjunctive logic program, one important transformation is *partial evaluation (also called unfolding)* (21).

In this section, we will see that it is also possible to define a possibilistic disjunctive semantics based on an operator which is a combination between partial evaluation for disjunctive logic programs and the infer rule GMP of possibilistic logic (see Section 2.5). This semantics has the same behavior to the semantics based on the proof theory of possibilistic logic.

We will start this section by defining a version of the general principle of partial evaluation (GPPE) for possibilistic positive disjunctive clauses.

Definition 3.4 (Grade-GPPE (G-GPPE)) *Let r_1 be a possibilistic clause of the form $\alpha : \mathcal{A} \leftarrow \mathcal{B}^+ \cup \{B\}$ and r_2 a possibilistic clause of the form $\alpha_1 : \mathcal{A}_1 \leftarrow \top$ such that $B \in \mathcal{A}_1$, then*

$$\textit{G-GPPE}(r_1, r_2) = (\mathcal{GLB}(\{\alpha, \alpha_1\}) : \mathcal{A} \cup (\mathcal{A}_1 \setminus \{B\}) \leftarrow \mathcal{B}^+)$$

Observe that one of the possibilistic clauses which is considered by G-GPPE has an empty body. For instance, let us consider the following two possibilistic clauses:

$$\begin{aligned} r_1 = {} & 0.7 : a \vee b \leftarrow \top. \\ r_2 = {} & 0.9 : e \leftarrow b. \end{aligned}$$

Then G-GPPE$(r_1, r_2) = (0.7 : e \vee a \leftarrow \top)$. Now, by considering G-GPPE, we will define the operator $\mathcal{T}$.

Definition 3.5 *Let P be a possibilistic positive logic program. The operator $\mathcal{T}(P)$ is defined as follows:*

$$\mathcal{T}(P) := P \cup \{\textit{G-GPPE}(r_1, r_2) | r_1, r_2 \in P\}$$

In order to illustrate the operator $\mathcal{T}$, let us consider the program P^{S_1} of Example 3.4.

$$\begin{array}{lrl} 0.7: & a \vee b & \leftarrow \top. \\ 0.8: & a & \leftarrow b. \\ 0.9: & e & \leftarrow b. \\ 0.6: & b & \leftarrow a. \\ 0.5: & b & \leftarrow a. \end{array}$$

Hence, $\mathcal{T}(P^{S_1})$ is:

$$\begin{array}{llll} 0.7: & a \vee b \leftarrow \top. & 0.7: & a \leftarrow \top. \\ 0.8: & a \leftarrow b. & 0.7: & e \vee a \leftarrow \top. \\ 0.9: & e \leftarrow b. & 0.6: & b \leftarrow \top. \\ 0.6: & b \leftarrow a. & 0.5: & b \leftarrow \top. \\ 0.5: & b \leftarrow a. & & \end{array}$$

Notice that by considering the possibilistic clauses that were added to P^{S_1} by $\mathcal{T}$, one can apply G-GPPE again. For instance, if we consider $0.6 : b \leftarrow \top$ and $0.9 : e \leftarrow b$ from $\mathcal{T}(P^{S_1})$, G-GPPE infers $0.6 : e \leftarrow \top$. Indeed, $\mathcal{T}(\mathcal{T}(P^{S_1}))$ is:

$$\begin{array}{llllll}
0.7: & a \vee b \leftarrow \top. & 0.7: & a \leftarrow \top. & 0.6: & a \leftarrow \top. \\
0.8: & a \leftarrow b. & 0.7: & e \vee a \leftarrow \top. & 0.5: & a \leftarrow \top. \\
0.9: & e \leftarrow b. & 0.6: & b \leftarrow \top. & 0.6: & e \leftarrow \top. \\
0.6: & b \leftarrow a. & 0.5: & b \leftarrow \top. & 0.5: & e \leftarrow \top. \\
0.5: & b \leftarrow a. & & & 0.6: & b \vee e \leftarrow \top. \\
 & & & & 0.5: & b \vee e \leftarrow \top.
\end{array}$$

An important property of the operator $\mathcal{T}$ is that it always reaches a fix-point.

Proposition 3.8 *Let P be a possibilistic disjunctive logic program. If $\Gamma_0 := \mathcal{T}(P)$ and $\Gamma_i := \mathcal{T}(\Gamma_{i-1})$ such that $i \in \mathcal{N}$, then $\exists\, n \in \mathcal{N}$ such that $\Gamma_n = \Gamma_{n-1}$. We denote Γ_n by $\Pi(P)$.*

Let us consider again the possibilistic program P^{S_1}. We can see that $\Pi(P^{S_1})$ is:

$$\begin{array}{llllllll}
0.7: & a \vee b \leftarrow \top. & 0.7: & a \leftarrow \top. & 0.6: & a \leftarrow \top. & 0.6 & a \vee e \leftarrow \top \\
0.8: & a \leftarrow b. & 0.7: & e \vee a \leftarrow \top. & 0.5: & a \leftarrow \top. & 0.5 & a \vee e \leftarrow \top \\
0.9: & e \leftarrow b. & 0.6: & b \leftarrow \top. & 0.6: & e \leftarrow \top. & & \\
0.6: & b \leftarrow a. & 0.5: & b \leftarrow \top. & 0.5: & e \leftarrow \top. & & \\
0.5: & b \leftarrow a. & & & 0.6: & b \vee e \leftarrow \top. & & \\
 & & & & 0.5: & b \vee e \leftarrow \top. & &
\end{array}$$

Observe that in $\Pi(PS_1)$ there are possibilistic facts (possibilistic clauses with empty bodies and one atom in their heads) with different necessity value. In order to infer the best necessity value of each possibilistic fact, one can consider the *least upper bound* of these values. For instance the best necessity value for the possibilistic atom a is $\mathcal{LUB}(\{0.7, 0.6, 0.5\}) = 0.7$. Based on this idea, we will define Sem_{min}.

Definition 3.6 *Let P be a possibilistic logic program and $Facts(P, a) := \{(\alpha : a \leftarrow \top) | (\alpha : a \leftarrow \top) \in P\}$. $Sem_{min}(P) := \{(x, \alpha) | Facts(P, x) \neq \emptyset$ and $\alpha := \mathcal{LUB}(\{n(r) | r \in Facts(P, x)\})\}$ where $x \in \mathcal{L}_P$.*

It is easy to see that $Sem_{min}(\Pi(P^{S_1}))$ is $\{(a, 0.7), (b, 0.6), (e, 0.6)\}$. Now by considering the operator $\mathcal{T}$ and Sem_{min}, we can define a semantics for possibilistic disjunctive logic programs that will be called possibilistic-$\mathcal{T}$ answer set semantics.

Definition 3.7 *Let P be a possibilistic disjunctive logic program and M be a set of possibilistic atoms such that M^* is an answer set of P^*. M is a possibilistic-$\mathcal{T}$ answer set of P if and only if $M = Sem_{min}(\Pi(P^{M^*}))$.*

In order to illustrate this definition, let us consider again the program P of Example 3.4 and $S = \{(a, 0.7), (b, 0.6), (e, 0.6)\}$. As commented in Example 3.4, S^* is an answer set of P^*. We have already seen that $Sem_{min}(\Pi(P^{S_1}))$ is $\{(a, 0.7), (b, 0.6), (e, 0.6)\}$, therefore we can say that S is a possibilistic-$\mathfrak{T}$ answer set of P. Observe that the possibilistic-$\mathfrak{T}$ answer set semantics and the possibilistic answer set semantics coincide. In fact the following proposition guaranties that both semantics are the same.

Proposition 3.9 *Let P be a possibilistic disjunctive logic program and M a set of possibilistic atoms. M is a possibilistic answer set of P if and only if M is a possibilistic-$\mathfrak{T}$ answer set of P.*

3.3.3 A possibilistic semantics based on pstable models

We have defined two semantics which capture the semantics for possibilistic disjunctive logic programs. Since these semantics were defined as extensions of the standard answer set semantics, there are some possibilistic logic programs which have no possibilistic answer sets. For instance, a simple possibilistic program as

$$\alpha : a \leftarrow not\ a$$

has no possibilistic answer set. In fact, the existence of one clause of this form will affect all the possibilistic knowledge base in such a way that the possibilistic knowledge base will not have a possibilistic answer set.

In this section, we present a second approach for capturing the semantics of possibilistic logic programs. This approach is based on *pstable semantics* (see Section 2.4.2). Pstable semantics emerges from the fusion of paraconsistent logics and ASP. This semantics is able to capture ASP; moreover, it is less sensitive (in the sense of inconsistency) than the answer set semantics.

We will start this section defining pstable semantics for *possibilistic normal programs* and after that we will show that this semantics is also able to capture the semantics of possibilistic disjunctive logic programs. Like possibilistic answer set semantics, possibilistic pstable semantics is defined in terms of a syntactic reduction. This reduction is based on Definition 2.3.

Definition 3.8 *Let P be a possibilistic normal program and M a set of atoms. We define*

$$PRED(P, M) := \{(\alpha : a \leftarrow \mathcal{B}^+,\ not\ (\mathcal{B}^- \cap M)) | (\alpha : a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-) \in P\}$$

Let us consider the following example in order to illustrate this definition.

Example 3.5 *First, let S be the set $\{(a, 0.6), (b, 0.7)\}$ and $P_1 := \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic program such that $(\mathcal{Q}, \geq)$ is the lattice of Example 3.4 and N the following set of possibilistic clauses:*

$0.7 : a \leftarrow not\ b, not\ c.$
$0.6 : a \leftarrow b.$
$0.8 : b \leftarrow a.$

Then, we can see that the program $PRED(P_1, S^*)$ is:

$0.7 : a \leftarrow not\ b.$
$0.6 : a \leftarrow b.$
$0.8 : b \leftarrow a.$

By considering the reduction PRED, we define the possibilistic pstable semantics as follows.

Definition 3.9 (Possibilistic Pstable Semantics) *Let P be a possibilistic normal logic program and M be a set of possibilistic atoms such that M^* is a pstable model of P^*. We say that M is a possibilistic pstable model of P if and only if $PRED(P, M^*) \vdash_{PL} M$ and $\nexists M'$ such that $M' \neq M$, $PRED(P, M^*) \vdash_{PL} M'$ and $M \sqsubseteq M'$.*

Example 3.6 *Let us continue with Example 3.5. We have already seen that $PRED(P_1, S^*)$ is:*

$0.7 : a \leftarrow not\ b.$
$0.6 : a \leftarrow b.$
$0.8 : b \leftarrow a.$

Then, we want to know if S is a possibilistic pstable models *of P_1. First of all, we have already seen in Section 2.4.2 that S^* is a pstable models of P_1^*. Hence, we have to construct a proof in possibilistic logic for $(a, 0.6)$ and $(b, 0.7)$. Let us consider the proof for the possibilistic atom $(a, 0.6)$:*

1. $(a \vee b) \rightarrow ((b \rightarrow a) \rightarrow a)$	*1*	***Tautology***
2. $\sim\ b \rightarrow a$	*0.7*	***Premise from*** $PRED(P_1, S)$
3. $a \vee b$	*0.7*	***From 2 by possibilistic logical equivalency***
4. $(b \rightarrow a) \rightarrow a)$	*0.7*	***From 1 and 3 by GMP***
5. $b \rightarrow a$	*0.6*	***Premise from*** $PRED(P_1, S)$
6. a	*0.6*	***From 4 and 5 by GMP***

Observe that the formula $\sim\ b \rightarrow a \quad 0.7$ *corresponds to the possibilistic normal clause* $0.7 : a \leftarrow \ not\ b$ *which belongs to the program* $RED(P_1, S^*)$*. The proof for* $(b, 0.7)$ *is similar to the proof of* $(a, 0.6)$*. Notice that* $\nexists\ S'$ *such that* $PRED(P_1, S^*) \vdash_{PL} S'$ *and* $S \sqsubseteq S'$*. Therefore, we can conclude that* S *is a* possibilistic pstable models *of* P_1*.*

Observe that the possibilistic program P_1 is an example where the possibilistic pstable semantics is different to both the possibilistic stable semantics (76) and the possibilistic answer set semantics (Definition 3.3). In fact, P_1 has no possibilistic stable model neither possibilistic answer set.

Even thought, there are programs where the possibilistic pstable semantics does not coincide with the possibilistic stable semantics neither with the possibilistic answer set semantics, we can identify a relationship between the possibilistic answer set semantics and the possibilistic pstable semantics.

Proposition 3.10 *Let* P *be a possibilistic normal program. If* M *is a possibilistic answer set of* P*, then the following conditions hold:*

a) M^* *is a pstable model of* P^**.*

b) *there exists a possibilistic pstable mode* M' *of* P *such that* $M \sqsubseteq M'$ *and* $M^* = M'^*$*.*

This proposition points out that whenever a possibilistic normal program has a possibilistic answer set M there exists a possibilistic pstable model M' such that the main differences between M and M' are the necessity-values of their elements. For instance, let us consider the following possibilistic program P:

$0.3 : a \leftarrow \top.$
$0.8 : a \leftarrow \ not\ a.$

One can see that P has the possibilistic answer set $M := \{(a, 0.3)\}$ and the possibilistic pstable model $M' := \{(a, 0.8)\}$. It is clear that $M^* = M'^*$; however, $M \sqsubseteq M'$.

Remark 3.1 *It is worth to comment that when* $P = \langle(\mathcal{Q}, \leq), N\rangle$ *is a possibilistic program which does not contain extended atoms* i.e. *atoms of form* $\neg a$*, and* $(\mathcal{Q}, \leq)$ *is a totally ordered set, it will be also true that: If* M *is a possibilistic stable model of* P*, then the following conditions hold:* $1.-$ M^* *is a pstable model of* P^* *and* $2.-$ *there exists a possibilistic pstable mode* M' *of* P *such that* $M \sqsubseteq M'$ *and* $M^* = M'^*$*.*

An interesting property of the possibilistic pstable semantics is that this semantics supports a kind of monotony *w.r.t.* the inference under possibilistic logic. In order to formalize this property, we will say that P is *equivalent* to P' *under the possibilistic pstable semantics* if and only if any possibilistic pstable model of P is also a possibilistic pstable model of P' and vice versa.

Proposition 3.11 *Let P be a possibilistic normal program. If $P \vdash_{PL} (x\ \alpha)$ then P is equivalent to $P \cup \{(x\ \alpha)\}$ under the possibilistic pstable semantics.*

Notice that neither the possibilistic answer set semantics nor the possibilistic stable semantics (76) satisfy Proposition 3.11 *i.e.* if $P \vdash_{PL} (x\ \alpha)$, then P is not equivalent to $P \cup \{(x\ \alpha)\}$ under possibilistic answer set semantics neither under the possibilistic stable semantics. In order to show this, let us consider a simple possibilistic logic program P:

$$\alpha : a \leftarrow not\ a$$

It is clear that $P \vdash_{PL} (a\ \alpha)$. P has no a possibilistic stable model neither a possibilistic answer set. However, $P \cup \{(a\ \alpha)\}$ has a possibilistic stable model and a possibilistic answer set which is the same in both cases and is $\{(a, \alpha)\}$.

The possibilistic answer set semantics was defined for the family of possibilistic disjunctive logic programs. This means that the possibilistic clauses could have a disjunction in their heads. Since the possibilistic pstable semantics was defined for possibilistic normal programs, one can think that the possibilistic pstable semantics is less expressive than the possibilistic answer semantics. However, by considering a simple mapping, one can extend the definition of possibilistic pstable models for possibilistic normal programs in order to define a semantics for possibilistic disjunctive logic programs.

An interesting result is that there exists a relationship between the possibilistic answer sets of a possibilistic disjunctive logic program P and the possibilistic pstable models of the possibilistic normal program $TRAD(P)$ (defined below). In order to formalize this result, we define some basic terms.

Given a possibilistic disjunctive logic program $P := \langle (\mathcal{Q}, \leq), N \rangle$, P_c will denote the set of possibilistic constraints which belong to P and P_N will denote the rest *i.e.* $P_N = P \setminus P_c$.

In order to see a possibilistic disjunctive logic program as a possibilistic normal logic program, we will define a simple mapping of a possibilistic disjunctive logic programs into a possibilistic normal logic programs.

Definition 3.10 *Let P be a possibilistic disjunctive logic program and $(\alpha : \mathcal{A} \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-) \in P_N$. We define*

$$R(\alpha : \mathcal{A} \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-) := \bigcup_{a \in \mathcal{A}} \{(\alpha : a \leftarrow \mathcal{B}^+,\ not\ (\mathcal{B}^- \cup (\mathcal{A} \setminus \{a\}))\}$$

The generalization of R over P is as follows: $R(P) := \bigcup_{C \in P} R(C)$.

For instance, $R(0.7 : a \vee b \leftarrow not\ c) := \{0.7 : a \leftarrow not\ c, not\ b; \quad 0.7 : b \leftarrow not\ c, not\ a\}$

By considering the mapping R, we define the function $TRAD$.

Definition 3.11 *Let P be a possibilistic disjunctive logic program. We define TRAD(P) as:*

$$TRAD(P) := R(P_N) \cup P_c$$

Now, by considering the function $TRAD$, we formalize that whenever a possibilistic disjunctive logic program P has a possibilistic answer set M, there exists a possibilistic pstable model M' of the possibilistic normal program $TRAD(P)$ such that the main differences between M and M' are the necessity-values of their elements.

Theorem 3.1 *Let P be a possibilistic disjunctive program. If M is a possibilistic answer set of P, then it implies that*

a) *M^* is a pstable model of $TRAD(P)^*$.*

b) *there exists a possibilistic pstable mode M' of $TRAD(P)$ such that $M \sqsubseteq M'$ and $M^* = M'^*$.*

Observe that this result is a generalization of the result of Proposition 3.10. In terms of computability, since there is an algorithm for inferring pstable models (69) and the possibilistic pstable semantics is based on the proof theory of possibilistic logic, the following proposition is a direct consequence of Proposition 3.7.

Proposition 3.12 *Given a possibilistic program $P := \langle(\mathcal{Q}, \leq), N\rangle$ there exists an algorithm that computes the set of possibilistic pstable models of P.*

3.4 Inconsistency in possibilistic logic programs

In this section, we will motivate the relevance of considering inconsistent possibilistic knowledge bases and we will introduce some criteria for managing inconsistent possibilistic logic programs.

3.4.1 Relevance of inconsistent possibilistic logic programs

Inconsistent knowledge bases usually are regarded as an *epistemic hell* that have to be avoided at all costs. However, many times it is difficult or impossible to stay away of managing inconsistent knowledge bases. There are approaches, as it is the case of Paraconsistent Logics, which allow to infer inconsistent pseudo-models. For instance, in (24), Bueno argues that to pursue inconsistent systems is a useful device for a number of reasons: (1) this is often the only way to explore inconsistent information without arbitrarily rejecting precious data. (2) pursuing inconsistent systems is sometimes the only way to obtain new information (particularly information that conflicts with deeply entrenched theories). As a result, (3) pursuing inconsistent belief systems allows us to make better *informed decisions* regarding which bits of information to accept or reject in the end.

In order to illustrate a small example, where to explore inconsistent information can be important to make a better informed decision, we will continue with the medical scenario described in Section 3.1. In Example 3.2, we have already presented the grounded program $P_{infections}$ of our medical scenario:

probable: $r\_inf(present,1) \vee no\_r\_inf(present,1) \leftarrow action(transplant,0), d\_inf(present,0)$.
confirmed: $o(good\_graft\_funct,1) \vee o(delayed\_graft\_funct,1) \vee o(terminal\_insufficient\_funct,1) \leftarrow action(transplant,0)$.
confirmed: $action(transplant,0) \leftarrow o(terminal\_insufficient\_funct,0)$.
plausible: $cs(stable,1) \leftarrow o(good\_graft\_funct,1)$.
plausible: $cs(unstable,1) \leftarrow o(delayed\_graft\_funct,1)$.
plausible: $cs(0\text{-urgency},1) \leftarrow o(terminal\_insufficient\_funct,1), action(transplant,0)$.
certain: $\bot \leftarrow action(transplant,0), action(wait,0)$.
certain: $\bot \leftarrow action(transplant,0), cs(dead,0)$.
certain: $d\_inf(present,0) \leftarrow \top$.
certain: $no\_r\_inf(present,0) \leftarrow \top$.
certain: $o(terminal\_insufficient\_funct,0) \leftarrow \top$.
certain: $cs(stable,0) \leftarrow \top$.

As commented in Example 3.2, in this program the atoms $\neg r\_inf(present,0)$ and

$\neg r\_inf(present, 1)$ were replaced by $no\_r\_inf(present, 0)$ and $no\_r\_inf(present, 1)$ respectively. Usually in standard answer set programming, the constraints

$\bot \leftarrow no\_r\_inf(present, 0), r\_inf(present, 0).$
$\bot \leftarrow no\_r\_inf(present, 1), no\_r\_inf(present, 1).$

must be added to the program for avoiding inconsistent answer sets. In order to comment the role of this kind of constraints, let C_1 be the following possibilistic constraints:

certain: $\bot \leftarrow no\_r\_inf(present, 0), r\_inf(present, 0).$
certain: $\bot \leftarrow no\_r\_inf(present, 1), no\_r\_inf(present, 1).$

Also let us consider three new possibilistic clauses (denoted by P_v):

confirmed: $v(kidney, 0) \leftarrow cs(stable, 1), action(transplant, 0).$
probable: $no\_v(kidney, 0) \leftarrow r\_inf(present, 1), action(transplant, 0).$
certain: $\bot \leftarrow \ not\ cs(stable, 1).$

The intended meaning of the predicate $v(t, T)$ is that the organ t is viable for transplanting and T denotes a moment at the time. Observe that we replaced the atom $\neg v(kidney, 0)$ by $no\_v(kidney, 0)$. The reading of the first clause is that if the clinical situation of the organ recipient is stable after the graft, then it is *confirmed* that the kidney is viable for transplanting. The reading of the second one is that if the organ recipient is infected after the graft, then it is *plausible* that the kidney is not viable for transplanting. The reading of the possibilistic constraint is that we do not want to consider scenarios where the clinical situation of the organ recipient is not stable. We will also consider the respective possibilistic constrain of the atoms $no\_v(kidney, 0)$ and $v(kidney, 0)$ (denoted by C_2):

certain: $\bot \leftarrow no\_v(kidney, 0), v(kidney, 0).$

Hence we define two programs

$$P := P_{infections} \cup P_v \text{ and } P_c := P_{infections} \cup P_v \cup C_1 \cup C_2$$

Basically, the difference between P and P_c is that P allows inconsistent possibilistic models and P_c does not allow inconsistent possibilistic models.

Now let us consider the possibilistic answer sets of the programs P and P_c. We can see that the program P_c has just one possibilistic answer set:

$\{(d\_inf(present, 0), certain)$, $(no\_r\_inf(present, 0), certain)$, $(o(terminal\_insufficient\_funct, 0), certain)$, $(cs(stable, 0), certain)$, $(action(transplant, 0), confirmed)$, $(o(good\_graft\_funct, 1), confirmed)$, **(cs(stable,1), plausible)**, **(no_r_inf(present,1), probable)**, **(v(kidney,0), plausible)**$\}$

This possibilistic answer set suggests that since it is plausible that recipient's clinical situation can be stable after the graft, it is plausible that the kidney is *viable* for transplanting. *Observe that the possibilistic answer sets of P do not warn that the organ recipient could be infected after the graft.*

Let us consider the possibilistic answer set of the program P:

$S_1 := \{(d\_inf(present, 0), certain)$, $(no\_r\_inf(present, 0), certain)$, $(o(terminal\_insufficient\_funct, 0), certain)$, $(cs(stable, 0), certain)$, $(action(transplant, 0), confirmed)$, $(o(good\_graft\_funct, 1), confirmed)$, **(cs(stable,1), plausible)**, **(no_r_inf(present,1), probable)**, **(v(kidney,0), plausible)**$\}$

$S_2 := \{(d\_inf(present, 0), certain)$, $(no\_r\_inf(present, 0), certain)$, $(o(terminal\_insufficient\_funct, 0), certain)$, $(cs(stable, 0), certain)$, $(action(transplant, 0), confirmed)$, $(o(good\_graft\_funct, 1), confirmed)$, **(cs(stable,1), plausible)**, **(r_inf(present,1), probable)**, **(v(kidney,0), plausible)**, **(no_v(kidney,0), probable)**$\}$

P has two possibilistic answer sets: S_1 and S_2. S_1 corresponds to the possibilistic answer set of the program P_c and S_2 is an inconsistent possibilistic answer set — because the atoms **(v(kidney,0), plausible)** and **(no_v(kidney,0), probable)** appear in S_2. Observe that although S_2 is an inconsistent possibilistic answer set, it contains important information *w.r.t.* the considerations of our scenario. S_2 suggests that even thought it is plausible that the clinical situation of the organ recipient can be stable after the graft, it is also probable that the organ recipient can be infected by the infection of the donor's organ.

Observe that essentially P_c is unable to infer the possibilistic answer set S_2 by the presence of the possibilistic constraint:

certain: $\perp \leftarrow no\_v(kidney, 0), v(kidney, 0).$

By defining this kind of constrains, we can guarantee that any possibilistic answer set inferred from P_c will be consistent; however, one can omit important considerations *w.r.t.* a decision-making problem. In fact, we agree with Bueno (24) that to consider inconsistent systems, as inconsistent possibilistic answer sets, some times is the only way to explore inconsistent information without arbitrarily rejecting precious data.

3.4.2 Inconsistency degrees of possibilistic sets

For managing inconsistent possibilistic answer set, it is necessary to define a criterion of preference between possibilistic answer sets. In order to define a criterion between possibilistic answer sets, we will define the concept of *inconsistency degree of a possibilistic set*. We say that a set of possibilistic atoms S is inconsistent (resp. consistent) if and only if S^* is inconsistent (resp. consistent) *i.e.* there exists atom a such that $a, \neg a \in S^*$.

Definition 3.12 *Let $\mathcal{A}$ be a finite set of atoms and extended atoms,$(\mathcal{Q},\leq)$ be a lattice and $S \in 2^{\mathcal{A}\times\mathcal{Q}}$. The inconsistent degree of S is defined as follows:*

$$InconsDegre(S) := \begin{cases} \mathcal{BOT}_{\mathcal{Q}} & \text{if } S^* \text{ is consistent} \\ \mathcal{GLB}(\{\alpha | S_\alpha \text{ is consistent}\}) & \text{otherwise} \end{cases}$$

where $\mathcal{BOT}_{\mathcal{Q}}$ is the bottom of the lattice $(\mathcal{Q},\leq)$ and $S_\alpha := \{(a,\alpha_1) \in S | \alpha_1 \geq \alpha\}$.

For instance, the possibilistic answer set S_2 of our example above has an inconsistency degree of *confirmed*. Based on the inconsistency degree of possibilistic sets, we can define a criterion of preference between possibilistic answer sets.

Definition 3.13 *Let $P = \langle(\mathcal{Q},\leq), N\rangle$ be a possibilistic program and M_1, M_2 two possibilistic answer set of P. We say that M_1 is weakest-inconsistent than M_2 if and only if $InconsDegre(M_1) < InconsDegre(M_2)$.*

For our example above, it is obvious that S_1 is weakest-inconsistent than S_2. In general terms, we will say that a possibilistic answer set M_1 is preferred than M_2 if and only if M_1 is weakest-inconsistent than M_2. This means that any consistent possibilistic answer set will be preferred than any inconsistent possibilistic answer set.

So far we have commented only the case of inconsistent possibilistic answer set. However, there are possibilistic programs that are inconsistent because they have no possibilistic answer sets neither possibilistic pstable models. For instance, let us consider the following possibilistic program P_{inc} (we are assuming the lattice of Example 3.4):

$$\begin{array}{ll} 0.3: & a \leftarrow \; not\ b. \\ 0.5: & b \leftarrow \; not\ c. \\ 0.6: & c \leftarrow \; not\ a. \end{array}$$

Observe that P^*_{inc} has no answer sets neither pstable models; hence, P_{inc} has no possibilistic answer sets neither possibilistic pstable models.

3.4.3 Restoring inconsistent possibilistic knowledge bases

In order to restore consistency of an inconsistent possibilistic knowledge base, possibilistic logic deletes the set of possibilistic formulæ which are lower than the inconsistent degree of the inconsistent knowledge base. By considering this idea, the authors of (76) defined the concept of α-cut for possibilistic logic programs. Based on Definition 14 of (76), we define its respective generalization for our approach.

Definition 3.14 *Let P be a possibilistic logic program*

- *the strict α-cut is the subprogram $P_{>\alpha} = \{r \in P | n(r) > \alpha\}$*

- *the consistency cut degree of P:*

$$ConsCutDeg(P) := \begin{cases} \mathcal{BOT}_{\mathcal{Q}} & \textit{if } P^* \textit{ is consistent} \\ \mathcal{GLB}(\{\alpha | P_\alpha \textit{ is consistent}\}) & \textit{otherwise} \end{cases}$$

where $\mathcal{BOT}_{\mathcal{Q}}$ is the bottom of the lattice ($\mathcal{Q}$,$\leq$).

Notice that the consistency cut degree of a possibilistic logic program identifies the minimum level of certainty for which a strict α-cut of P is consistent. As Nicolas *et al*, remarked in (76), by the non-monotonicity of the framework it is not ensure that a higher cut is necessarily consistent.

In order to illustrate these ideas, let us consider again the program P_{inc}. First, we can see that $ConsCutDeg(P_{inc}) = 0.3$; hence, the subprogram $P_{ConsCutDeg(P_{inc})}$ is:

$$\begin{array}{ll} 0.5: & b \leftarrow \textit{not } c. \\ 0.6: & c \leftarrow \textit{not } a. \end{array}$$

Observe that this program has a possibilistic answer set which is $\{(c, 0.6)\}$[1]. Hence thanks to the strict α-cut of P, one is able to infer information from P_{inc}

We have commented two kinds of inconsistency in our approach,

- one which arises from the presence of complementary atoms in a possibilistic answer set (or a possibilistic pstable model) and

- the other one which arises from the non-existence of possibilistic answer set (or possibilistic pstable models) of a possibilistic logic program.

[1]Remember that any possibilistic answer set is also a possibilistic pstable model.

For managing the inconsistency of possibilistic answer sets, we have defined a criterium of preference between possibilistic answer sets — of course that this criterium is also applied to possibilistic pstable models. For managing the non-existence of possibilistic answer set (or possibilistic pstable models) of a possibilistic logic program P, we have adopted the approach suggested by Nicolas *et al*, in (76) of cuts for getting subprograms of P which are consistent.

It worth to comment that in some cases, it is possible to apply α-cuts in order to avoid inconsistent possibilistic answers. For instance, let P be the following possibilistic program:

$$\begin{array}{ll} 0.9: & a \leftarrow \top. \\ 0.9: & \neg a \leftarrow \top. \\ 0.8: & b \leftarrow \top. \end{array}$$

We can see that $ConsCutDeg(P) = 0.9$; hence, if we apply a strict α-cut to P, we will get an empty program. On the other hand, if we allow an inconsistent possibilistic answer set, we get $\{(a, 0.9), (\neg a, 0.9), (b, 0.8)\}$. As one can see, $0.8 : b \leftarrow \top$ is not involved in the inconsistency of P. Hence, it is not necessary to loss this information. We believe that an inconsistent possibilistic answer set could be more informative answer than a null-possibilistic answer set for an expert.

3.5 Related Work

Logic programming with uncertainty is an extensively research area. In fact, it has proceeded along various research lines of logic logic programming. An interesting historical recollection in this topic was recently presented by V. S. Subrahmanian in (104). In this recollection he highlights some phases in the evolution of the topic from the viewpoint of a committed researcher.

Research on logic programming with uncertainty has dealt with various approaches of logic programming semantics, as well as different applications. Most of the approaches in the literature employ one of the following formalisms:

- annotated logic programming, *e.g.*, (63).
- probabilistic logic, *e.g.*, (62; 72; 75).
- fuzzy set theory, *e.g.*, (101; 109; 110).
- multi-valued logic, *e.g.*, (46; 66).
- evidence theoretic logic programming, *e.g.*, (10).
- possibilistic logic, *e.g.*, (2; 3; 4; 41; 76).

Basically, these approaches differ in the underlying notion of uncertainty and how uncertainty values, associated to clauses and facts, are managed.

As stated on §3.1, we are interested on modeling qualitative expressions such that these expressions could capture the available information especially when this information is incomplete, uncertain and inconsistent. As far of this chapter we have defined a logic programming approach with uncertainty which captures uncertain values by considering complete lattices. The use of lattices for capturing uncertain values is not new, maybe one of the most influential approach in this context was suggest by Fitting in (46). In (46), Fitting showed that interlaced bilattices provide a simple and elegant setting for the consideration of logic programming extensions allowing for incomplete or contradictory answers. On the theoretical level he showed that his approach is a considerable unification of several approaches.

An interesting observation of Fitting is that in the abstract level all interlaced bilattices are quite natural; however not all are appropriate for computer implementation. By Proposition 4.1 of (46), we know that given two complete lattices C and B, $\mathcal{B}(C, D)$ is an interlaced bilattice[1]. It is not difficult to see that essentially the semantics of a possibilisitic disjunctive logic program $P = \langle(\mathcal{Q}, \leq), N\rangle$ is defined in the domain of the interlaced bilattice $\mathcal{B}(\{0, 1\}, \mathcal{Q})$. Since the possibilisitic semantics defined in this chapter are computable, our approach is restricted to computable interlaced bilattices. Observe that by considering a complete lattice Q' different to $\{0, 1\}$, we can explore new logic programming semantics for our approach by considering multi-valued logics defined under Q' and Fitting's approach. Of course that this issue requires a deep analysis to understand how Fitting's approach and our approach are related. It is worth to comment that in (3), a possibilistic logic programming approach is defined over the many-valued *Gödel* logic. The syntax of this approach is restricted to a Horn-clause sublanguage of the many-valued *Gödel* logic; hence it is unable to capture default negation and disjunctive clauses.

To prioritize logic clauses, as it is done in our possibilistic approach, can be also regarded as a preference relation between rules. In fact, by considering the certainty degrees as *preferences*, it was defined two criteria for restoring inconsistent possibilistic knowledge bases in §3.4. Observe that these criteria are based on the notion of maximal consistent subsets of premises. In other words, we try to recover the maximal consistent subset of possibilistic clauses from an inconsistent possibilistic program to infer consistent information. The use of *qualitative preferences* in logic programming has been suggested by authors as G. Brewka in (23). The Brewka's approach is also motivated from the fact that a variety of applications numerical information is hard to obtain. To have a correct un-

[1]See (46) for details.

derstanding of the relationship between Brewka's approach and our approach requires a deep analysis.

3.6 Concluding remarks

In this chapter, we have introduced a possibilistic disjunctive logic programming approach. This approach introduces the use of possibilistic disjunctive clauses which are able to capture *incomplete information* and *incomplete states of a knowledge base* at the same time.

In particular, we have defined three approaches for capturing the semantics of the possibilistic disjunctive programs:

1. the first one is strictly close to the proof theory of possibilistic logic and answer set models;

2. the second one is based on partial evaluation, a fix-point operator and answer set models; and

3. the last one is also based on the proof theory of possibilistic logic and pstable semantics.

Based on the flexibility of possibilistic logic for defining degrees of uncertainty, we have illustrated in this chapter that it is possible to consider non-numerical degrees for capturing uncertain information. In particular, we have discussed the use of non-numerical degrees of uncertainty in a medical scenario.

For managing the inconsistency of possibilistic models, we have defined a criterium of preference between possibilistic answer sets. Also, for managing the non-existence of possibilistic answer set (or possibilistic pstable models) of a possibilistic logic program P, we have adopted the approach suggested by Nicolas *et al*, in (76) of cuts for getting subprograms of P which are consistent.

With the results of this chapter, we have achieved one of the main goals of this thesis: to define a non-monotonic approach able to perform reasoning under uncertain, incomplete and inconsistent information. It is worth comment that to the best of our knowledge, the approach presented in this chapter is the first one on dealing with possibilistic disjunctive logic programs.

Chapter 4

Studying abstract argumentation semantics based on logic programming semantics

In this chapter, we will present our study of abstract argumentation semantics based on logic programming semantics. We will see that by modeling argumentation frameworks as logic programs, it is possible to characterize and define new abstract argumentation semantics in terms of logic programming semantics.

4.1 Introduction

Argumentation theory, or argumentation, embraces the arts and sciences of civil debate, dialogue, conversation, and persuasion. It studies rules of inference, logic, and procedural rules in both artificial and real world settings. Argumentation is concerned primarily with reaching conclusions through logical reasoning, that is, claims based on premises. Although including debate and negotiation which are concerned with reaching mutually acceptable conclusions, argumentation theory also encompasses eristic dialog, the branch of social debate in which victory over an opponent is the primary goal. This art and science is often the means by which people protect their beliefs or self-interests in rational dialogue, in common parlance, and during the process of arguing.

Argumentation is also a formal discipline within artificial intelligence where the aim is to make a computer assist in or perform the act of argumentation. In fact, during the last years, argumentation has been gaining increasing importance in Multi-Agent Systems (MAS), mainly as a vehicle for facilitating *rational interaction* (*i.e.* interaction which involves the giving and receiving of reasons). A single agent may also use argumentation techniques to perform its individual reasoning because it needs to make decisions under complex preferences policies, in a highly dynamic environment.

Although several approaches have been proposed for argument theory, Dung's approach presented in (44), is a unifying framework which has played an influential role on argumentation research and Artificial Intelligence (AI). Dung's approach is regarded as an abstract model where the main concern is to find the set of arguments which are considered as acceptable. The strategy for inferring the set of acceptable arguments is based on abstract argumentation semantics. The kernel of Dung's framework is supported by four abstract argumentation semantics: *stable semantics*, *preferred semantics*, *grounded semantics*, and *complete semantics*.

Since Dung introduced his abstract argumentation approach, he proved that his approach can be regarded as a special form of logic programming with *negation as failure*. This result has at least two main implications:

1. It defines a general method for generating metainterpreters for argumentation systems and

2. it defines a general method for studying abstract argumentation semantics' properties in terms of logic programming semantics' properties.

Although the study of abstract argumentation semantics in terms of logic programming semantics has important implications, there are few efforts in order to characterize or to define new argumentation semantics based on logic programming semantics. In fact, the only argumentation semantics that have been characterized in terms of logic programming semantics (to the best of our knowledge) are the grounded semantics and the stable semantics (44).

In this chapter, we will present a study of abstract argumentation semantics in terms of logic programming semantics. We will start by presenting some basic conditions which will be considered when one wants to regard an argumentation framework as a logic program. The results of this chapter are mainly concerned on the argumentations semantics based on *admissible sets*.

Possibly, the two main challenges of studying Dung's approach in terms of logic programming are:

1. To find suitable logic programming codifications able to map an argumentation framework AF into a logic program P such that these codifications are polynomial time computable, and

2. To find suitable logic programming semantics able to capture the different patterns of inference of the argumentation semantics.

In (44), Dung defined a logic programming codification (P_{AF}, see §2.6) in order to map an argumentation framework into a logic program. In fact by considering P_{AF}, Dung showed that the well-founded semantics (§2.4.3) is a proper logic programming semantics to capture the grounded semantics and the answer set semantics (§2.4.1) is a proper logic programming semantics to capture the stable semantics. However, to the best of our knowledge, there does not exist a logic programming semantics able to capture the preferred semantics by using P_{AF}.

In this chapter will introduce the concept of *a suitable codification.* A suitable codification will be a mapping which at least is able to characterize the three well-accepted argumentation semantics: the grounded, preferred and stable semantics.

It is quite expected that a suitable codification of an argumentation framework should not only permit to characterize abstract argumentation semantics, but also it ought to permit to perform a deep study about an abstract argumentation semantics. Hence, by considering the flexibility of a suitable codification, we will present:

- a study of the preferred semantics in terms of minimal models and answer set sets models — this study will suggest some practical methods for implementing the preferred semantics and

- a study of the grounded semantics in order to define some intermediate argumentation semantics between the grounded and the preferred semantics.

Since we define a characterization of the preferred semantics in terms of answer set models and positive disjunctive logic program, we also outline how to perform any query *w.r.t. sceptical and credulous reasoning* under the preferred semantics by considering DLV system.

An interesting point of our study of the grounded semantics is that as we consider rewriting systems for defining extensions of the grounded semantics, we will see that by applying program transformation to a suitable codification one can describe the interactions of arguments of an argumentation framework.

The rest of chapter is divided as follows: In §4.2, we will define the concept of suitable codification for characterizing Dung's argumentation semantics. In §4.3 and §4.4, we introduce a suitable codification. In §4.5, we present our study of the preferred semantics. In §4.6, we define some extensions of the grounded semantics. Finally, in the last section, we present our concluding remarks.

4.2 Suitable codifications for arguing under admissible sets

In this section, we will define the concept of suitable codification for capturing argumentation theory in terms of logic programming.

The problem of finding suitable codifications for mapping argumentation theory into logic programming is close related to find suitable codifications of an argumentation framework as a logic program. This is because there is a strong relationship between the codification and the logic programming semantics which will be considered for characterizing an abstract argumentation semantics. For instance, Dung showed that by considering P_{AF} (see §2.6) and *WFS*, one can capture the grounded semantics; however any change of the codification or the logic programming semantics will induce a different candidate argumentation semantics.

What is a suitable codification for argumentation theory in terms of logic programming? Of course that the answer to this question will depend on the argumentation approach that one wants to capture in terms of logic programming. In this work, we are interested on Dung's argumentation style; hence, based on the fact that the grounded, preferred and stable semantics are the well-accepted argumentation semantics for the argumentation community (8; 16; 99), one can impose that a suitable codification at least must be able to characterize these semantics. It is worth mentioning that these conditions are imposed in order to follow an argumentation approach based on admissible sets.

Before to introduce the definition of a suitable codification, we will clarify what we understand when we say that a logic programming semantics characterizes an argumentation semantics. Formally, an argumentation semantics arg_SEM is a function from $\mathcal{AF}_{AR} \rightarrow 2^{AR}$. This means that an argumentation semantics can be regarded as a mapping from an argumentation framework into a set of sets of arguments (each set of arguments is called *extension*). Also remember that any logic programming semantics S can be regarded as a mapping from a logic program P into a set H of sets of literals, such that for each set of literals L in H, $P \cup L$ is consistent (in the strict sense of classical logic). We call each of the above sets of literals (such as L) partial models of the program P. By having these ideas in mind, we introduce the following definition:

Definition 4.1 *Given an argumentation framework $AF := \langle AR, attacks \rangle$, a logic program P, a logic programming semantics SEM, and an argumentation semantics arg_SEM. We say that SEM* of P characterizes arg_SEM of AF *if the following conditions hold:*

1. *for each model M inferred by SEM from P, there is an extension E inferred*

by arg_SEM from AF such that there exists a function f such that $f(M) = E$.

2. *for each extension E inferred by arg_SEM from AF, there is a model M inferred by SEM from P such that there exists a function g such that $g(E) = M$.*

Informally speaking, the first condition says that if M is a model of P in a given logic programming semantics, then M induces an extension of AF in a given argumentation semantics. In the same way, the second condition says that if E is an extension of AF in a given argumentation semantics, then E induces a model of P in a given logic programming semantics.

Notice that in this definition, there is a strict relation one to one between the models of P and the extensions of AF. This means that if we are characterizing a *skeptical argumentation semantics arg_SEM*, then it must be characterized by a *skeptical logic programming semantics SEM*. Also when we are charactering a *credulous argumentation semantics arg_SEM*, it must be characterized by a *credulous logic programming semantics*. By having it in mind, we introduce our definition of a suitable codification.

Definition 4.2 *Given an argumentation framework $AF := \langle AR, attacks\rangle$ and a logic program P, we will say that P is a suitable codification of AF if:*

1. *there is a logic programming semantics* SEM *such that* SEM *of P characterizes the grounded semantics of AF,*

2. *there is a logic programming semantics* SEM *such that* SEM *of P characterizes the stable semantics of AF,*

3. *there is a logic programming semantics* SEM *such that* SEM *of P characterizes the preferred semantics of AF,*

4. *the functions f and g of each characterization are polynomial time computable, and*

5. *there exists a polynomial time computable function μ such that $\mu(AF) = P$.*

It is worth mentioning that a suitable codification could be an useful tool for defining intermediate argumentation semantics between the grounded semantics and the preferred semantics. This means that it is possible to define an intermediate reasoning between the grounded semantics and the preferred semantics.

The problem of characterizing abstract argumentation semantics does not only depend on the codification but also in the logic programming semantics. In fact, to find a suitable logic programming semantic is as important as to find a suitable codification for characterizing a particular abstract argumentation semantics.

By Theorem 2.1, we have already seen that by using P_{AF}, *WFS* is a suitable logic programming semantics for characterizing the grounded semantics and answer set semantics for characterizing the stable semantics. However, to the best of our knowledge there is not a logic programming semantics which could characterize the preferred semantics by using P_{AF}. Hence we can not say that P_{AF} is a suitable codification.

In the following section, we will present a single mapping, which has some interesting properties in order to study argumentation semantics in terms of logic programming semantics. In fact, we will show that it satisfies the condition of a suitable codification.

4.3 Mapping an argumentation framework into a normal program

In this section, we will introduce a mapping in order to regard an argumentation framework as a logic program. We will see that this mapping has some interesting properties *w.r.t.* suitable codifications.

Our mapping is inspired in the conditions which make an argument to be defeated — this means that it is attacked by an acceptable argument (see Definition 2.8). Basically, it captures two basic conditions which make an argument to be defeated. In order to define our mapping, we introduce the predicate *d(x)*, where the intended meaning of *d(x)* is: "the argument x is *defeated*".

Definition 4.3 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework, then $\Psi(AF)$ is defined as follows:*

$$\Psi_{AF} := \bigcup_{a \in AR} \{ \bigcup_{b:(b,a) \in attacks} \{d(a) \leftarrow not\ d(b)\} \cup \bigcup_{b:(b,a) \in attacks} \{d(a) \leftarrow \bigwedge_{c:(c,b) \in attacks} d(c)\}\}$$

1. The first condition of Ψ_{AF}, $\bigcup_{b:(b,a) \in attacks} d(a) \leftarrow not\ d(b)$, suggests that the argument a is defeated when anyone of its adversaries is not defeated.

2. The second condition of Ψ_{AF}, $\bigcup_{b:(b,a) \in attacks} d(a) \leftarrow \bigwedge_{c:(c,b) \in attacks} d(c)$, suggests that the argument a is defeated when all the arguments that defend[1] a from one of its adversaries b are defeated.

[1]We say that c defends a if there exists b such that b attacks a and c attacks b.

The conditions captured by Ψ_{AF} are standard settings in argumentation for defining the status of an argument. In fact by considering different strength of the arguments, some approaches define different status for an argument as it is done by approached based on Defeasible Logic (54; 55).

Since Ψ_{AF} captures conditions which make an argument to be defeated, it is quite obvious that any argument which satisfies these conditions could not belong to an *admissible set*. Therefore these arguments also could not belong to a preferred/stable/grounded extension.

Notice that Ψ_{AF} is a *finite grounded program*, this means that it does not contain predicates with variables; hence, Ψ_{AF} is essentially a propositional formula (just considering the atoms like $d(a)$ as d_a) of propositional logic. In order to illustrate Ψ_{AF}, let us consider the following example.

Example 4.1 *Let $AF := \langle\{a,b,c\},\{(a,b),(b,c)\}\rangle$ be a argumentation framework — the graph representation of this argumentation framework is presented in Figure 2.1. We can see that Ψ_{AF} is:*

$$d(b) \leftarrow \mathit{not}\ d(a). \qquad d(b) \leftarrow \top. \qquad d(c) \leftarrow \mathit{not}\ d(b). \qquad d(c) \leftarrow d(a).$$

Observe that Ψ_{AF} has no normal clauses with the atom $d(a)$ in their head. This is essentially because Ψ_{AF} is capturing the arguments which could be defeated and the argument a will be always an acceptable argument.

It is worth mentioning that given an argumentation framework AF, Ψ_{AF} will have at most $2n^2$ normal clauses such that n is the number of arguments in AR and the maximum length[1] of each normal clause is $n+1$. Hence, we can say that Ψ_{AF} is quadratic size *w.r.t.* the number of arguments of AF.

In the following section, we will see that Ψ_{AF} is enough flexible for characterizing the stable, preferred and grounded semantics.

4.4 A suitable codification for arguing under admissible sets

In this section, we will show that the mapping Ψ_{AF} satisfies the conditions of a suitable codification. This means that it is able to characterize the grounded, stable and preferred semantics.

We start by defining some basic concepts. Given an argumentation framework $AF := \langle AR, Attacks\rangle$ and $E \subseteq AR$, we define the sets $f(E)$ and $compl(E)$ as follows:

[1]The length of our propositional clauses C is given by the number of atoms in the head of C plus the number of literals in the body of C

$$\begin{aligned} f(E) &= \{d(a) | a \in E\} \\ compl(E) &= \{d(a) | a \in AR \setminus E\} \end{aligned}$$

Observe that $f(E)$ essentially is embedding each argument a in the predicate $d(a)$ and $compl(E)$ essentially expresses the complement of E *w.r.t.* AR.

The first result that we want to introduce is that by considering the mapping Ψ_{AF} and the well-founded semantics, the grounded semantics can be captured.

Lemma 4.1 *Let $AF := \langle AR, attacks\rangle$ be an argumentation framework and $S \subseteq AR$. S is the grounded extension of AF if and only if $\exists\ D \subseteq AR$ such that $\langle f(D), f(S)\rangle$ is the well-founded model of Ψ_{AF}.*

In order to illustrate this lemma, let us consider the program Ψ_{AF} of Example 4.1. We can see that

$$WFS(\Psi_{AF}) := \langle \{d(b)\}, \{d(a), d(c)\}\rangle$$

Hence, by Lemma 4.1, this means that $\{a, c\}$ is the grounded extension of the argumentation framework $AF := \langle\{a, b, c\}, \{(a, b), (b, c)\}\rangle$.

We have seen that Ψ_{AF} is able to characterize the grounded semantics. Now we will see that by considering Ψ_{AF} and the answer set semantics the stable semantics can be captured.

Lemma 4.2 *Let AF be an argumentation framework and E a set of arguments. E is a stable extension of AF if and only if $compl(E)$ is a answer set of Ψ_{AF}.*

Let us consider the following example.

Example 4.2 *Let $AF := \langle AR, attacks\rangle$ be an argumentation framework, such that $AR := \{a, b, c, d, e\}$ and $attacks := \{(a, b), (b, a), (b, c), (c, d), (d, e), (e, c)\}$ (see Figure 4.1).*

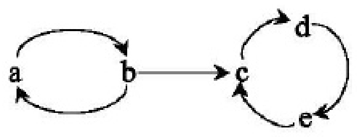

Figure 4.1: Graph representation of the argumentation framework $AF := \langle\ \{a, b, c, d, e\}, \{(a, b), (b, a), (b, c), (c, d), (d, e), (e, c)\}\rangle$.

In order to infer the stable semantics of the argumentation framework AF, we map AF into Ψ_{AF}:

$$\begin{array}{ll}
d(a) \leftarrow \; not\; d(b). & d(a) \leftarrow d(a).\\
d(b) \leftarrow \; not\; d(a). & d(b) \leftarrow d(b).\\
d(c) \leftarrow \; not\; d(b). & d(c) \leftarrow \; not\; d(e).\\
d(c) \leftarrow d(a). & d(c) \leftarrow d(d).\\
d(d) \leftarrow \; not\; d(c). & d(d) \leftarrow d(b), d(e).\\
d(e) \leftarrow \; not\; d(d). & d(e) \leftarrow d(c).
\end{array}$$

we can see that Ψ_{AF} has just one answer set model: $\{d(a), d(c), d(e)\}$. Then by Lemma 4.2, we can infer that AF has just one stable extension which is: $\{b, d\}$. Observe that for inferring the stable extension $\{b, d\}$, we are only considering the complement *of* $\{d(a), d(c), d(e)\}$ w.r.t. *AR.*

We have seen that Ψ_{AF} is able to characterize the grounded and stable semantics. Now we will see that by considering the pstable semantics and Ψ_{AF}, one can characterize the preferred semantics. In fact, we will see in the next section that the minimal models of Ψ_{AF} also characterize the preferred semantics.

Lemma 4.3 *Let AF be an argumentation framework and E a set of arguments. E is a preferred extension of AF if and only if compl(E) is a pstable model of Ψ_{AF}.*

Let us consider again the argumentation framework AF and the normal program Ψ_{AF} of Example 4.2. As we can see, Ψ_{AF} has two pstable models: $\{d(a), d(c), d(e)\}$ and $\{d(b), d(c), d(e), d(d))\}$. Then by Lemma 4.3, we can infer that AF has two preferred extensions: $\{b, d\}$ and $\{a\}$.

As immediate consequence, of Lemma 4.1, Lemma 4.2 and Lemma 4.3, we can say that the mapping Ψ_{AF} is a suitable codification

Theorem 4.1 *Ψ_{AF} is a suitable codification.*

The idea of identifying a suitable codification is not only to characterize the well-known argumentation semantics. We believe that it could be a tool for closing the *practical* and *theoretical* results of logic programming and argumentation theory.

In order to show the potentiality of a suitable codification, in the next two sections, we will present some important results *w.r.t.* the preferred and grounded semantics. First, we will present two new characterizations of the preferred semantics (by minimal models and answer sets). After that, we will present some extensions of the grounded semantics. These extensions can be regarded as intermediate argumentation semantics between the grounded and the preferred semantics.

4.5 Preferred semantics

In this section, we show that Ψ_{AF} offers some relevant properties in order to characterize the preferred semantics in terms of *minimal models* and *answer sets.*

4.5.1 Preferred semantics and minimal models

We will start our study of the preferred semantics by proving that the minimal models of Ψ_{AF} corresponds to the preferred extensions of AF. In fact, we will provide a method for computing preferred extensions. This method is based on model checking and Unsatisfiability (UNSAT). UNSAT is the complement of Satisfiability (SAT), a problem for which very efficient systems have been developed in AI during the last decade.

In order to characterize the preferred semantics in terms of minimal models, we will introduce some concepts.

Definition 4.4 *Let T be a theory with signature $\mathcal{L}$. We say that $\mathcal{L}'$ is a copy-signature of $\mathcal{L}$ if and only if*

- $\mathcal{L} \cap \mathcal{L}' = \emptyset$,
- *the cardinality of $\mathcal{L}'$ is the same to $\mathcal{L}$ and*
- *there is a bijective function f from $\mathcal{L}$ to $\mathcal{L}'$.*

It is well known that there exists a bijective function from one set to another if both sets have the same cardinality. Based on the concept of *copy-signature*, one can establish an important relationship between maximal and minimal models.

Proposition 4.1 *Let T be a theory with signature $\mathcal{L}$. Let $\mathcal{L}'$ be a copy-signature of $\mathcal{L}$. By $g(T)$ we denote the theory obtained from T by replacing every occurrence of an atom x in T by $\sim f(x)$. Then M is a maximal model of T if and only if $f(\mathcal{L} \setminus M)$ is a minimal model of $g(T)$.*

In order to regard Ψ_{AF} as a propositional formula, we will define the propositional formula $\alpha(AF)$ which is the same mapping to Ψ_{AF}. The only difference is that Ψ_{AF} is regarded as a normal program and α_{AF} is regarded as a propositional formula. Given an argumentation framework $AF := \langle AR, attacks \rangle$, $\alpha(AF)$ is defined as follows:

$$\alpha(AF) := \bigwedge_{a \in AR} ((\bigwedge_{b:(b,a) \in attacks} d(a) \leftarrow\sim d(b)) \wedge (\bigwedge_{b:(b,a) \in attacks} d(a) \leftarrow \bigwedge_{c:(c,b) \in attacks} d(c)))$$

For instance, let us consider the argumentation framework AF of Example 4.1. Hence, we can see that $\alpha(AF)$ is:

$$(d(b) \leftarrow \sim d(a)) \wedge (d(b) \leftarrow \top) \wedge (d(c) \leftarrow \sim d(b)) \wedge (d(c) \leftarrow d(a)) \tag{4.1}$$

Observe that $\alpha(AF)$ is the same to Ψ_{AF} of Example 4.1 (modulo notation). Like Ψ_{AF}, $\alpha(AF)$ is a finite grounded formula. The idea of regarding Ψ_{AF} as a propositional formula is for formalizing that the minimal models of Ψ_{AF} corresponds to the preferred extensions of AF. In order to formalize this property, let us consider the following proposition which was proved by Besnard and Doutre in (17).

Proposition 4.2 *(17) Let $AF := \langle AR, attacks\rangle$ be an argumentation framework. Let $\beta(AF)$ be the formula:*

$$\bigwedge_{a \in AR} ((a \rightarrow \bigwedge_{b:(b,a) \in attacks} \sim b) \wedge (a \rightarrow \bigwedge_{b:(b,a) \in attacks} (\bigvee_{c:(c,b) \in attacks} c)))$$

then, a set $S \subseteq AR$ is a preferred extension of AF if and only if S is a maximal model of the formula $\beta(AF)$.

In contrast with $\alpha(AF)$ which captures conditions which make an argument to be defeated, $\beta(AF)$ captures conditions which make an argument acceptable. However, we will prove that when the mapping $f(x)$ of the theory $g(\beta(AF))$ corresponds to $d(x)$ such that $x \in AF$, $\alpha(AF)$ is logically equivalent to $g(\beta(AF))$ (see the proof of Theorem 4.2). For instance, let us consider the argumentation framework AF of Example 4.1. The formula $\beta(AF)$ is:

$$(\sim a \leftarrow b) \wedge (\bot \leftarrow b) \wedge (\sim b \leftarrow c) \wedge (a \leftarrow c)$$

If we replace each atom x by the expression $\sim d(x)$, we get:

$$(\sim\sim d(a) \leftarrow \sim d(b)) \wedge (\bot \leftarrow \sim d(b)) \wedge (\sim\sim d(b) \leftarrow \sim d(c)) \wedge (\sim d(a) \leftarrow \sim d(c))$$

Now, if we apply transposition to each implication, we obtain:

$$(d(b) \leftarrow \sim d(a)) \wedge (d(b) \leftarrow \top) \wedge (d(c) \leftarrow \sim d(b)) \wedge (d(c) \leftarrow d(a))$$

The latter formula corresponds to $\alpha(AF)$. The following theorem is a straightforward consequence of Proposition 4.2 and Proposition 4.1.

Theorem 4.2 *Let* $AF := \langle AR, attacks \rangle$ *be an argumentation framework and* $S \subseteq AR$*. When the mapping* $f(x)$ *of the theory* $g(\beta(AF))$ *corresponds to* $d(x)$ *such that* $x \in AR$*, the following condition holds:* S *is a preferred extension of* AF *if and only if* $compl(S)$ *is a minimal model of* $\alpha(AF)$*.*

In order to illustrate Theorem 4.2, let us consider again $\alpha(AF)$ (4.1) *w.r.t.* the argumentation framework of Example 4.1. This formula has three models: $\{d(b)\}$, $\{d(b), d(c)\}$ and $\{d(a), d(b), d(c)\}$. Then, the only minimal model is $\{d(b)\}$, this implies that $\{a, c\}$ is the only preferred extension of AF. In fact, each model of $\alpha(AF)$ implies an admissible set of AF, this means that $\{a, c\}$, $\{a\}$ and $\{\}$ are the admissible sets of AF.

There is a well known relationship between minimal models and logical consequence, see (90). The following proposition is a direct consequence of such relationship. Let S be a set of well formed formulæ then we define

$$SetToFormula(S) := \bigwedge_{c \in S} c$$

Proposition 4.3 *Let* $AF := \langle AR, attacks \rangle$ *be an argumentation framework and* $S \subseteq AR$*.* S *is a preferred extension of* AF *if and only if* $compl(S)$ *is a model of* $\alpha(AF)$ *and*

$$\alpha(AF) \wedge SetToFormula(\sim \widetilde{compl(S)}) \models SetToFormula(compl(S))$$

There are several well-known approaches for inferring minimal models from a propositional formula (14; 34). For instance, it is possible to use UNSAT's algorithms for inferring minimal models. Hence, it is clear that we can use UNSAT's algorithms for computing the preferred extensions of an argumentation framework. This idea is formalized with the following proposition.

Theorem 4.3 *Let* $AF := \langle AR, attacks \rangle$ *be an argumentation framework and* $S \subseteq AR$*.* S *is a preferred extension of* AF *if and only if* $compl(S)$ *is a model of* $\alpha(AF)$ *and*

$$\alpha(AF) \wedge SetToFormula(\sim \widetilde{compl(S)}) \wedge \sim SetToFormula(compl(S))$$

is unsatisfiable.

In order to illustrate Theorem 4.3, let us consider again the argumentation framework AF of Example 4.1. Let $S = \{a\}$, then $compl(S) = \{d(b), d(c)\}$. We have already seen that $\{d(b), d(c)\}$ is a model of $\alpha(AF)$, hence the formula to verify its unsatisfiability is:

$$(d(b) \leftarrow \sim d(a)) \wedge (d(b) \leftarrow \top) \wedge (d(c) \leftarrow \sim d(b)) \wedge (d(c) \leftarrow d(a)) \wedge \\ \sim d(a) \wedge (\sim d(b) \vee \sim d(c))$$

However, this formula is satisfiable by the model $\{d(b)\}$, then $\{a\}$ is not a preferred extension. Now, let $S = \{a, c\}$, then $compl(S) = \{d(b)\}$. As seen before, $\{d(b)\}$ is also a model of $\alpha(AF)$, hence the formula to verify its unsatisfiability is:

$$(d(b) \leftarrow \sim d(a)) \wedge (d(b) \leftarrow \top) \wedge (d(c) \leftarrow \sim d(b)) \wedge (d(c) \leftarrow d(a)) \wedge \\ \sim d(a) \wedge \sim d(c) \wedge \sim d(b)$$

It is easy to see that this formula is unsatisfiable, therefore $\{a, c\}$ is a preferred extension of AF.

The relevance of Theorem 4.3 is that UNSAT is the prototypical and best-researched co-NP-complete problem. Hence, Theorem 4.3 opens the possibilities for using a wide variety of algorithms for inferring the preferred semantics.

4.5.2 Preferred semantics and answer set semantics

We have seen that the minimal models of Ψ_{AF} characterize the preferred extensions of AF. One interesting point of Ψ_{AF} is that Ψ_{AF} is logically equivalent to the positive logic program Γ_{AF} (defined below). It is well known that given a positive logic program P, all the minimal models of P correspond to the answer sets of P (see Section 2.4.1). This property will be enough for characterizing the preferred semantics by the answer set models of the positive disjunctive logic program Γ_{AF}. This characterization will suggest an other option for computing preferred extensions based on answer set solvers. This approach presents an easy-to-use form for inferring the preferred extensions of an argumentation framework. In this case, the kind of systems that we need for inferring the preferred extensions of an argumentation framework is any *disjunctive answer set solver e.g.*, DLV (40).

We start this section by defining a mapping function which is a variation of the mapping of Definition 4.3.

Definition 4.5 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $a \in AR$. We define the transformation function $\Gamma(a)$ as follows:*

$$\Gamma(a) := \{ \bigcup_{b:(b,a) \in attacks} \{d(a) \vee d(b)\}\} \cup \{ \bigcup_{b:(b,a) \in attacks} \{d(a) \leftarrow \bigwedge_{c:(c,b) \in attacks} d(c)\}\}$$

Now we define the function Γ in terms of an argumentation framework.

Definition 4.6 *Let $AF := \langle AR, attacks\rangle$ be an argumentation framework. We define its associated general program as follows:*

$$\Gamma_{AF} := \bigcup_{a\in AR} \Gamma(a)$$

Remark 4.1 *Notice that Ψ_{AF} (see Definition 4.3) is similar to Γ_{AF}. The main syntactic difference of Γ_{AF}* w.r.t. *Ψ_{AF} is the first part of Γ_{AF} which is $(\bigcup_{b:(b,a)\in attacks}(d(a) \vee d(b)))$; however this part is logically equivalent to the first part of Ψ_{AF} which is $(\bigcup_{b:(b,a)\in attacks} d(a) \leftarrow not\ d(b))$. In fact, the main difference is their behavior* w.r.t. *answer set semantics. In order to illustrate this difference, let us consider the argumentation framework $AF := \langle\{a\}, \{(a, a)\}\rangle$. We can see that*

$$\Gamma_{AF} := \{d(a) \vee d(a)\} \cup \{d(a) \leftarrow d(a)\}$$

and

$$\Psi_{AF} := \{d(a) \leftarrow not\ d(a)\} \cup \{d(a) \leftarrow d(a)\}$$

It is clear that both programs have a minimal model which is $\{d(a)\}$[1]; however Ψ_{AF} has no answer sets. This suggests that Ψ_{AF} is not a suitable representation for computing preferred extensions by using answer set solvers. Nonetheless we will see that the answer sets of Γ_{AF} characterize the preferred extensions of AF.

In the following theorem we formalize a characterization of the preferred semantics in terms of positive disjunctive logic programs and answer set semantics.

Theorem 4.4 *Let $AF := \langle AR, attacks\rangle$ be an argumentation framework and $S \subseteq AR$. S is a preferred extension of AF if and only if $compl(S)$ is an answer set of Γ_{AF}.*

Let us consider the following example.

Example 4.3 *Let $AF := \langle\{a, b, c, d, e\}, \{(a, b), (b, a), (b, c), (c, d), (d, e), (e, c)\}\rangle$ be an argumentation framework. Observe that AF corresponds to the argumentation framework of Example 4.2 — a graph representation of AF is presented in Figure 4.1. We can see that Γ_{AF} is*

[1]Notice that $\{d(a)\}$ suggests that AF has a preferred extensions which is $\{\}$.

$$\begin{array}{ll}
d(a) \vee d(b). & d(a) \leftarrow d(a). \\
d(b) \vee d(a). & d(b) \leftarrow d(b). \\
d(c) \vee d(b). & d(c) \vee d(e). \\
d(c) \leftarrow d(a). & d(c) \leftarrow d(d). \\
d(d) \vee d(c). & d(d) \leftarrow d(b), d(e). \\
d(e) \vee d(d). & d(e) \leftarrow d(c).
\end{array}$$

In example 4.2, we saw that Ψ_{AF} has just one answer set $\{d(a), d(c), d(e)\}$ which corresponds to the only stable extension $\{b, d\}$ of AF. Now, we can see that Γ_{AF} has two answer sets $\{d(a), d(c), d(e)\}$ and $\{d(b), d(c), d(e), d(d))\}$ that correspond to the preferred extensions of AF: $\{b, d\}$ and $\{a\}$.

Default negation

One of the advantages of characterizing the preferred semantics, by using a logic programming semantics with *default negation*, is that we can infer the acceptable arguments from the answer sets of Γ_{AF} in a straightforward form. For instance, let Λ_{AF} be the disjunctive logic program Γ_{AF} of Example 4.3 plus the following clauses:

$$\begin{array}{ll}
a \leftarrow \ not\ d(a). & b \leftarrow \ not\ d(b). \\
c \leftarrow \ not\ d(c). & d \leftarrow \ not\ d(d). \\
e \leftarrow \ not\ d(e). &
\end{array}$$

such that the intended meaning of each clause is: the argument x is acceptable if it is not defeated. Λ_{AF} has two answer sets which are $\{d(a), d(c), d(e), b, d\}$ and $\{d(b), d(c), d(e), d(d), a\}$. By taking the intersection of each model of Λ_{AF} with AR (the set of arguments of AF), we can see that $\{a, d\}$ and $\{a\}$ are the preferred extensions of AF. This idea is formalized by Proposition 4.4 below.

Definition 4.7 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework. We define its associated general program as follows:*

$$\Lambda_{AF} := \bigcup_{a \in AR} \{\Gamma(a) \cup \{a \leftarrow \ not\ d(a)\}\}$$

Notice that $\Gamma(a)$ and $\Lambda(a)$ are equivalent, the main difference between Γ_{AF} and Λ_{AF} is the rule $a \leftarrow \ not\ d(a)$ for each argument.

Proposition 4.4 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $S \subseteq AR$. S is a preferred extension of AF iff there is a stable model M of Λ_{AF} such that $S = M \cap AR$.*

4.5.3 DLV System: Sceptical and Credulous Reasoning

Whenever we use answer set solvers for computing the preferred extensions of an argumentation framework, we can compute all the preferred extensions of an argumentation framework. However, we can also perform any query with respect to *sceptical and credulous reasoning* under the preferred semantics.

For instance, let us consider the following argumentation framework $AF := \langle \{a, b, c, d\}, \{(a, b), (b, a), (a, c), (b, c), (c, d)\}\rangle$ (see Figure 4.2). We can see that AF has two preferred extensions: $\{a, c\}$ and $\{b, c\}$.

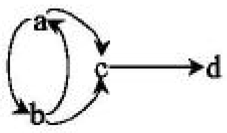

Figure 4.2: Graph representation of the argumentation framework $AF := \langle \{a, b, c, d\}, \{(a, b), (b, a), (a, c), (b, c), (c, d)\}\rangle$.

Now, we will illustrate how to perform queries by using our characterization of Section 4.5.2 and *DLV System* (40). Let `gamma-AF` be the file which contains Γ_{AF} (see Definition 4.7):

$$\begin{array}{ll}
d(a) \vee d(b). & d(a) \leftarrow d(a). \\
d(b) \vee d(a). & d(b) \leftarrow d(b). \\
d(c) \vee d(a). & d(c) \leftarrow d(b). \\
d(c) \vee d(b). & d(c) \leftarrow d(a). \\
d(d) \vee d(c). & d(d) \leftarrow d(a), d(b). \\
acc(a) \leftarrow not\ d(a). & acc(b) \leftarrow not\ d(b). \\
acc(c) \leftarrow not\ d(c). & acc(d) \leftarrow not\ d(d).
\end{array}$$

Let us now call DLV with the file `gamma-AF`:

```
$dlv gamma-AF
```

$\{d(b), d(c), acc(a), acc(d)\}$
$\{d(a), d(c), acc(b), acc(d)\}$

Hence, we can see that `gamma-AF` has two answer sets and each answer set represents a preferred extension of AF:

$\{d(b), d(c), acc(a), acc(d)\} \longrightarrow \{a, d\}$

$\{d(a), d(c), acc(b), acc(d)\} \longrightarrow \{b, d\}$

Now let us consider the traditional query with respect to *credulous reasoning*: Given an argumentation framework AF and argument x, we want to check if x is at least in one preferred extension of AF.

Let us suppose that we want to know which arguments belong to some preferred extensions. Hence, let `query-1` be the file:

$acc(X)?$

Let us call DLV with the brave/credulous reasoning front-end and `query-1`:

```
$ dlv -brave gamma-AF query-1
```

a
b
d

This means that the arguments a, b, d belong to some preferred extension. Now let us suppose that we want to ask for a particular argument. For instance, we want to know if argument a belongs to some preferred extension. Hence, let `query-2` be the file:

$acc(a)?$

Let us call DLV with the brave/credulous reasoning front-end and `query-2`:

```
$ dlv -brave gamma-AF query-2
```

$acc(a)$ `is bravely true, evidenced by` $\{d(b), d(d), acc(a), acc(d)\}$

This means that it is true that the argument a belongs to a preferred extension and even more we have a preferred extension which contains the argument a.

Now let us consider the traditional query with respect to *sceptical reasoning*: Given an argumentation framework AF and argument x, we want to check if x is in every preferred extension.

Let us suppose that we want to know if the argument d belongs to all the preferred extensions of AF. Hence let `query-3` be the file:

$acc(d)?$

Let us call DLV with the cautious/sceptical reasoning front-end and `query-3`:

```
$ dlv -cautious gamma-AF query-3
```

$acc(d)$ `is cautiously true.`

This means that it is true that the argument d belongs to all the preferred extensions of AF. Now let us suppose that we want to know if the argument a belongs to all the preferred extensions of AF. Hence let `query-4` be the file:

$acc(a)?$

Let us call DLV with the cautious/sceptical reasoning front-end and `query-4`.

```
$ dlv -cautious gamma-AF query-4
```

$acc(a)$ `is cautiously false, evidenced by` $\{d(a), d(d), acc(b), acc(d)\}$

This means that it is false that the argument a belongs to all the preferred extensions of AF. In fact, we have a counterexample.

Remark 4.2 *It is worth mentioning that we can use Ψ_{AF} (see Definition 4.3) and DLV system for performing any query with respect to* sceptical and credulous reasoning *under the stable semantics.*

4.6 Grounded semantics (GE_{AF})

In this section, we will show that the suitable codification Ψ_{AF} is not only able to characterize the grounded semantics but it is also able to define some extensions of the grounded semantics. Moreover, we will illustrate that by applying program transformation to Ψ_{AF}, one can describe the interaction of arguments of an argumentation framework.

We will start by presenting the extensions of the grounded semantics and after that we will present a small example where we will illustrate the use of rewriting systems for describe the interactions of arguments.

4.6.1 Extensions of GE_{AF} based on rewriting systems

The extensions of the grounded semantics that we will introduce are based on extensions of the well-founded semantics (WFS) (see Section 2.4.3). These WFS' extensions have as feature that they are defined in terms of *rewriting systems* (see Section 2.3). Hence, the strategy for defining the extensions of the grounded semantics will be based on a *calculus of logic program transformations*. The interesting point of this approach is that this calculus of logic program transformations will define a *declarative calculus of argumentation framework transformations*. We will see that when we apply a program transformation to Ψ_{AF}, this action will represent a *declarative operation* over the argumentation framework AF.

Before to introduce the extensions of the grounded semantics, we define some concepts. Since *WFS* is a 3-valued logic semantics, where any atom could be *true*, *false*, and *undefined*, we will define the concept of a *3-valued extension*, where any argument could be *accepted*, *defeated*, and *undecided*.

Definition 4.8 (3-valued extension) *Given an argumentation framework $AF := \langle AR, attacks\rangle$, and $S, D \subseteq AR$. A 3-valued extension is a tuple $\langle S, D\rangle$, where $S \cap D = \emptyset$ and S is a conflict-free set. We call an argument a acceptable if $a \in S$, an argument b defeated if $b \in D$, and an argument c undecided if $c \in AR \backslash \{S \cup D\}$.*

$WFS^{LLC'}$ semantics

The first extension of WFS that we will consider is the semantics $WFS^{LLC'}$. As seen in Section 2.4.3, this semantics is induced by the rewriting system $\mathcal{CS}_1$ which has the transformation rules: $\{RED^+$, RED^-, *Success*, *Failure*, *Loop*, $LLC'\}$. It worth mentioning that the transformation rules that induce WFS are $\{RED^+, RED^-, Success, Failure, Loop\}$. Hence the main difference between WFS and $WFS^{LLC'}$ is done by the behavior of the program transformation LLC'.

Since WFS and Ψ_{AF} characterize the grounded semantics, we will use Ψ_{AF} and $WFS^{LLC'}$ for defining an extension of the grounded semantics as follows:

Definition 4.9 *Let $AF := \langle AR, attacks\rangle$ be an argumentation framework and $S, D \subseteq AR$. $\langle S, D\rangle$ is the $WFS^{LLC'}$-extension of AF if and only if $\langle f(D), f(S)\rangle$ is the $WFS^{LLC'}$- model of Ψ_{AF}.*

As happens with WFS and $WFS^{LLC'}$, the main difference between the grounded extension and the $WFS^{LLC'}$-extension is done by the transformation rule **LLC′**. Based on Ψ_{AF}, intuitively we can say that LLC' first removes all the attacks of the argument a from AF; therefore it is reviewed by *Success* whether the argument a is defeated. In case that a appears as defeated, it will be assumed that the argument a is defeated. Notice that the only case that a could be defeated after removed its attacks is that a belongs to a cycle of attacks. Let us consider the following example.

Example 4.4 *Let $AF := \langle AR, attacks\rangle$ be an argumentation framework, where $AR := \{a,b,c\}$ and $attacks := \{(a,a),(a,b),(b,c),(c,b)\}$ (see Figure 4.3). Hence, Ψ_{AF} is:*

$$\begin{array}{lll} d(a) \leftarrow \ not\ d(a). & d(a) \leftarrow d(a). & d(c) \leftarrow \ not\ d(b). \\ d(b) \leftarrow \ not\ d(a). & d(b) \leftarrow \ not\ d(c). & d(c) \leftarrow d(c), d(a). \\ d(b) \leftarrow d(a). & d(b) \leftarrow d(c). & \end{array}$$

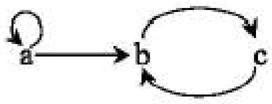

Figure 4.3: Graph representation of the argumentation framework $AF := \langle \{a,b,c\}, \{(a,a),(a,b),(b,c),(c,b)\}\rangle$.

To infer the AF's $WFS^{LLC'}$-extension, we need to get the Ψ_{AF}'s $WFS^{LLC'}$ model. Then, we apply $\mathcal{CS}_1$ to Ψ_{AF}. We can see in Figure 4.3 that the argument a *is a* controversial argument *since it is attacked by itself. Then the transformation rule LLC' will remove all the atoms of the form: $not\ d(a)$. This means that, LLC' will remove all the* a*'s attacks of AF. After that, it will view if* a *is defeated. Since* a *appears as defeated, it is assumed that the argument* a *is a defeated argument; hence, the assumption $d(a) \leftarrow \top$ is added to the program Ψ_{AF}:*

$$\begin{array}{lll} d(a) \leftarrow \ not\ d(a). & d(a) \leftarrow d(a). & d(c) \leftarrow \ not\ d(b). \\ d(b) \leftarrow \ not\ d(a). & d(b) \leftarrow \ not\ d(c). & d(c) \leftarrow d(c), d(a). \\ d(b) \leftarrow d(a). & d(b) \leftarrow d(c). & d(a) \leftarrow \top. \end{array}$$

If we assume that a *is a defeated argument, then RED^- will remove all its attacks (the clauses which will be removed are: $d(a) \leftarrow \ not\ d(a)$ and $d(b) \leftarrow \ not\ d(a)$) and Success will remove all its supports to other arguments (the clause $d(c) \leftarrow d(c), d(a)$ is reduced to $d(c) \leftarrow d(c)$).*

$$\begin{array}{lll} d(a) \leftarrow d(a). & d(b) \leftarrow \ not\ d(c). & d(c) \leftarrow \ not\ d(b). \\ d(b) \leftarrow d(a). & d(b) \leftarrow d(c). & d(c) \leftarrow d(c). \\ d(a) \leftarrow \top. & & \end{array}$$

Then applying Success, it is found that the argument b *is a defeated argument:*

$$\begin{array}{lll} d(a) \leftarrow \top. & d(b) \leftarrow \ not\ d(c). & d(c) \leftarrow \ not\ d(b). \\ d(b) \leftarrow \top. & d(b) \leftarrow d(c). & d(c) \leftarrow d(c). \end{array}$$

Therefore applying RED^{-}, it is removed all the attacks of the argument b*:*

$$\begin{array}{lll} d(a) \leftarrow \top. & d(b) \leftarrow \; not\; d(c). & \\ d(b) \leftarrow \top. & d(b) \leftarrow d(c). & d(c) \leftarrow d(c). \end{array}$$

Since the attack of the argument b *to* c *is removed, Loop will remove the clause $d(c) \leftarrow d(c)$. Then we get:*

$$d(b) \leftarrow \top. \qquad d(a) \leftarrow \top. \qquad d(b) \leftarrow \; not\; d(c).$$

Finally, since the argument b *was already fixed as a defeated argument, RED^{+} will remove the attack of the argument c to b which is represented by the clause: $d(b) \leftarrow \neg d(c)$. Then, the* normal form *of Ψ_{AF} is:*

$$d(b) \leftarrow \top. \qquad\qquad d(a) \leftarrow \top.$$

Therefore, $WFS^{LLC'}(\Psi_{AF}) := \langle \{d(a), d(b)\}, \{d(c)\}\rangle$, this means that $\langle \{c\}, \{a, b\}\rangle$ is the AF's $WFS^{LLC'}$- extension. We can conclude that the argument c *is an acceptable argument and* a, b *are defeated arguments. Notice that AF has an empty grounded extension, AF has no stable extensions and AF has only one preferred extension which is $\{c\}$. In fact, the set of acceptable arguments of the $WFS^{LLC'}$- extension corresponds to the only preferred extension of AF.*

In order to show the difference w.r.t. *the semantics $WFS^{LLC'}$ between the mapping Ψ_{AF} and the mapping P_{AF} (see Definition 2.13)suggested by Dung in (44). Let us consider the grounded instance of the program P_{AF}* w.r.t. *AF:*

$$\begin{array}{ll} acc(a) \leftarrow \; not\; d(a). & d(a) \leftarrow acc(a). \\ acc(b) \leftarrow \; not\; d(b). & d(b) \leftarrow acc(a). \\ acc(c) \leftarrow \; not\; d(c). & d(c) \leftarrow acc(b). \\ & d(b) \leftarrow acc(c). \end{array}$$

It is not difficult to see that $WFS^{LLC'}(P_{AF}) := \langle \{d(a)\}, \{acc(a)\}\rangle$. This means that the only thing that we can say w.r.t. *AF is that the argument a is defeated argument.*

WFS^{WK} semantics

Another interesting extension of WFS that we will consider for defining an extension of the grounded semantics is WFS^{WK}. As seen in Section 2.4.3, this semantics is induced by the rewriting system $\mathcal{CS}_2 := \{RED^{+}, RED^{-}, Success, Failure, Loop, Weak\text{-}Cases\}$.

By considering WFS^{WK} and Ψ_{AF}, we will define an extension of the grounded semantics as follows:

Definition 4.10 *Let $AF := \langle AR, attacks\rangle$ be an argumentation framework and $S, D \subseteq AR$. $\langle S, D\rangle$ is the WFS^{WK}-extension of AF if and only if $\langle f(D), f(S)\rangle$ is a WFS^{WK}- model of Ψ_{AF}.*

As we can see, the main difference between the characterizations of the grounded semantics and the WFS^{WK}-extension is done by the transformation rule *Weak-Cases*. Essentially the transformation rule *Weak-Cases* performs a reasoning by cases. Let us consider the following example.

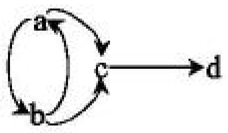

Figure 4.4: Graph representation of the argumentation framework $AF := \langle \{a, b, c, d\}, \{(a, b), (b, a), (a, c), (b, c), (c, d)\}\rangle$.

Example 4.5 *Let $AF := \langle AR, attacks\rangle$ be an argumentation framework, where $AR := \{a, b, c, d\}$ and $attacks := \{(a, b), (b, a), (a, c), (b, c), (c, d)\}$ (see Figure 4.4) — notice that this argumentation framework was introduced in Section 4.5.3, we repeat the figure for commodity of reading. We can see that Ψ_{AF} is:*

$$\begin{array}{lll}
d(a) \leftarrow\ not\ d(b). & d(a) \leftarrow d(a). & d(d) \leftarrow\ not\ d(c).\\
d(b) \leftarrow\ not\ d(a). & d(b) \leftarrow d(b). & d(d) \leftarrow d(b), d(a).\\
d(c) \leftarrow\ not\ d(b). & d(c) \leftarrow d(b). & \\
d(c) \leftarrow\ not\ d(a). & d(c) \leftarrow d(a). &
\end{array}$$

In order to infer the WFS^{WK}-extension of AF, it is applied $\mathcal{CS}_2$ to Ψ_{AF}. First of all, we can see that the argument a *is* controversial w.r.t. *the argument* c *because* a *is attacking to* c *($d(c) \leftarrow\ not\ d(a)$) and also* a *is defending to* c *($d(c) \leftarrow d(a)$). Therefore, if* a *is fixed as an acceptable argument, then* c *will be a defeated argument. Moreover, if* a *is fixed as a defeated argument, then* c *also will be a defeated argument. Under this situation, the transformation rule* Weak-Cases *will assume that the argument* c *is defeated, then it will remove the clauses $d(c) \leftarrow\ not\ d(a)$ and $d(c) \leftarrow d(a)$ from Ψ_{AF} and the clause $d(c) \leftarrow \top$ is added to Ψ_{AF}. Notice that the argument* b *is also controversial* w.r.t. c*. Then the clauses $d(c) \leftarrow\ not\ d(b)$ and $d(c) \leftarrow d(b)$ are removed from Ψ_{AF}:*

$$\begin{array}{lll}
d(a) \leftarrow\ not\ d(b). & d(a) \leftarrow d(a). & d(d) \leftarrow\ not\ d(c).\\
d(b) \leftarrow\ not\ d(a). & d(b) \leftarrow d(b). & d(d) \leftarrow d(b), d(a).\\
d(c) \leftarrow \top. & &
\end{array}$$

Since the argument c *was assumed as a defeated argument, the RED^{-} will remove* c*'s attacks. Hence, we get:*

$$\begin{array}{lll} d(a) \leftarrow \ not\ d(b). & d(a) \leftarrow d(a). & d(d) \leftarrow d(b), d(a). \\ d(b) \leftarrow \ not\ d(a). & d(b) \leftarrow d(b). & d(c) \leftarrow \top. \end{array}$$

Since this program is the normal form of Ψ_{AF}, $WFS^{WK}(\Psi_{AF}) := \langle \{d(c)\}, \{\} \rangle$. Hence $\langle \{\}, \{c\} \rangle$ is the WFS^{WK}-extension of AF. This means that the argument c is defeated.

Notice that the grounded extension of AF is the empty set, there are two stable extensions which are $\{a, d\}$ and $\{b, d\}$, and there are two preferred extensions which coincide with the stable extensions: $\{a, d\}$ and $\{b, d\}$. It is worth mentioning that usually any argument which does not belong to a preferred/stable extension is considered defeated. Then we can see that both preferred/stable extensions of AF coincide that the argument c is a defeated argument. Therefore we can appreciate that the WFS^{WK}-extension coincides with the preferred/stable extensions that the argument c is defeated.

Now let us consider again the mapping P_{AF} to see the behavior of P_{AF} w.r.t. *WFS^{WK}. The grounded instance of the program P_{AF}* w.r.t. *AF is:*

$$\begin{array}{ll} acc(a) \leftarrow \ not\ d(a). & d(b) \leftarrow acc(a). \\ acc(b) \leftarrow \ not\ d(b). & d(a) \leftarrow acc(b). \\ acc(c) \leftarrow \ not\ d(c). & d(c) \leftarrow acc(a). \\ acc(d) \leftarrow \ not\ d(d). & d(c) \leftarrow acc(b). \\ & d(d) \leftarrow acc(c). \end{array}$$

We can see that $WFS^{WK}(\Psi_{AF}) := \langle \{\}, \{\} \rangle$. This means that like the grounded extension, the argumentation semantics induced by WFS^{WK} and P_{AF} is empty.

$WFS^{WK+LCC'}$ semantics

We have defined two extensions of the grounded semantics based on two extensions of WFS, where the main support of these extensions is the use of the transformation rules: *Weak-Cases* and *LLC′*. Now the combination of these transformation rules also suggests another extension of the grounded semantics.

By considering $WFS^{WK+LLC'}$, we define a new extension of the grounded semantics as follows:

Definition 4.11 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $S, D \subseteq AR$. $\langle S, D \rangle$ is the $WFS^{WK+LLC'}$- extension of AF if and only if $\langle f(D), f(S) \rangle$ is a $WFS^{WK+LLC'}$- model of Ψ_{AF}.*

None of both $WFS^{LLC'}$ and WFS^{WK} extensions is the same to $WFS^{WK+LLC'}$-extension. In order to illustrate this difference let us consider the following example.

Example 4.6 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework, where $AR := \{a, b, c, d, e, f, m, n, p\}$ and $attacks := \{(a, b),\ (b, c),\ (c, a),\ (a, d),\ (d, e),\ (e, f),\ (m, e),\ (n, m),\ (n, p), (p, m),\ (p, n)\}$ (see Figure 4.5). It is not difficult to see that $WFS^{LLC'}$-extension $:= \langle \{\}, \{a, b, c, d, e\} \rangle$, WFS^{WK}-extension $:= \langle \{\}, \{m\} \rangle$, $WFS^{WK+LLC'}$-extension $:= \langle \{\}, \{a, b, c, d, e,\ m\} \rangle$, and the grounded extension is empty.*

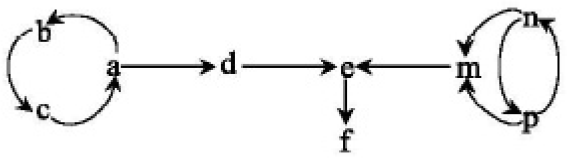

Figure 4.5: Graph representation of the argumentation framework $AF := \langle \{a, b, c, d, e, f, m, n, p\}, \{(a, b),\ (b, c),\ (c, a),\ (a, d),\ (d, e),\ (e, f),\ (m, e),\ (n, m),\ (n, p), (p, m),\ (p, n)\} \rangle$

This argumentation framework has no stable extensions and has tow preferred extensions: $\{n\}$ and $\{p\}$.

Formalizing the extensions of the grounded semantics

We have introduced three new abstract argumentation semantics, all of them have as common point a suitable codification which is Ψ_{AF} and the only difference between them is the logic programming semantics which it is applied to Ψ_{AF}.

Once we have defined a direct relationship between abstract argumentation semantics and logic programming semantics, it is possible to understand the behavior of some abstract argumentation semantics based on the properties of the logic programming semantics. For instance, since the grounded semantics is characterized by Ψ_{AF} and *WFS*, we can infer that the $WFS^{LLC'}$-extension, the WFS^{WK}-extension and the $WFS^{WK+LLC'}$-extension are extensions of the grounded semantics and are polynomial time computable. This is essentially because the semantics $WFS^{LLC'}$, WFS^{WK} and $WFS^{WK+LLC'}$ are extensions of *WFS* and are polynomial time computable. This result is formalized with the following theorem:

Theorem 4.5 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and E be the grounded extension of AF. Then*

a)
1. *If $\langle S, D \rangle$ is the $WFS^{LLC'}$-extension of AF then $E \subseteq S$.*
2. *If $\langle S, D \rangle$ is the WFS^{WK}-extension of AF then $E \subseteq S$.*
3. *If $\langle S, D \rangle$ is the $WFS^{WK+LLC'}$-extension of AF then $E \subseteq S$.*

b)
1. *The $WFS^{LLC'}$-extension of AF is polynomial time computable.*
2. *The WFS^{WK}-extension of AF is polynomial time computable.*
3. *The $WFS^{WK+LLC'}$-extension of AF is polynomial time computable.*

Another property that can be formalized *w.r.t.* the new argumentation semantics is that they are *intermediate semantics* between the grounded semantics and the preferred semantics. This is essentially because the semantics $WFS^{LLC'}$, WFS^{WK} and $WFS^{WK+LLC'}$ are strongest than WFS but they are weakest than the pstable semantics. Remember that the pstable models of Ψ_{AF} correspond to the preferred extensions of AF (see Lemma 4.3).

In order to show that our new argumentation semantics are intermediate semantics between the grounded semantics and the preferred semantics, we will show that they are contained in the preferred semantics.

Theorem 4.6 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework, E be a preferred extension of AF, and $E' := AR \setminus E$. Then,*

1. *If $\langle S, D \rangle$ is the $WFS^{LLC'}$-extension of AF then $S \subseteq E$ and $D \subseteq E'$.*
2. *If $\langle S, D \rangle$ is the WFS^{WK}-extension of AF then $S \subseteq E$ and $D \subseteq E'$.*
3. *If $\langle S, D \rangle$ is the $WFS^{WK+LLC'}$-extension of AF then $S \subseteq E$ and $D \subseteq E'$.*

Remark 4.3 *As final remark of this section, we want to point out that three extensions of the grounded semantics that were presented follow the approach of admissible sets. This means that none of the new argumentations semantics will infer an argument out of the scope of admissible sets.*

4.6.2 Rewriting systems and the interaction between arguments

The aim of this section is to *outline* an approach for describing the interaction of arguments based on rewiring systems and the suitable codification Ψ_{AF}. This approach allows to visualize the process of selecting acceptable arguments from an argumentation framework.

The general idea is that each time that it is applied a transformation rule to Ψ_{AF} the reduced program will suggest an approximation of the argumentation semantics that we are inferring. These approximations together with the reduced program will describe the interaction of the arguments of the argumentation framework.

In order to identify the status of an argument *i.e. acceptable*, *defeated* and *undecided*, we will define a mapping which follows the labeling style presented in (58).

Definition 4.12 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $SEM(\Psi_{AF}) = \langle \Psi_{AF}^{true}, \Psi_{AF}^{false} \rangle$. The labeling l_{SEM} is defined as follows: For all $a \in AR$, let:*

$$l_{SEM}(a) := \begin{cases} + & \textit{if } d(a) \in \Psi_{AF}^{false} \\ - & \textit{if } d(a) \in P\Psi_{AF}^{true} \\ ? & \textit{if } d(a) \notin \Psi_{AF}^{true} \cup \Psi_{AF}^{false} \end{cases}$$

Observe that the mapping l_{SEM} is defined in terms of a three valued logic programming semantics SEM (Definition 2.5) of the normal logic program Ψ_{AF}. By considering the concepts of a 3-valued extension of an argumentation framework (Definition 4.8), we will say that

- an argument $a \in AR$ is acceptable if $l_{SEM}(a) = +$
- an argument $a \in AR$ is defeated if $l_{SEM}(a) = -$
- an argument $a \in AR$ is undecided if $l_{SEM}(a) = ?$

Let us consider the argumentation framework of Figure 4.3. In Example 4.4, we saw that

$$WFS^{LLC'}(\Psi_{AF}) := \langle \{d(a), d(b)\}, \{d(c)\} \rangle$$

hence the graph representation of AF can be labeled as it is shown in Figure 4.6.

In order to illustrate that each time that one applies a transformation rule to Ψ_{AF}, $SEM(\Psi_{AF})$ will induce different labeling states of the graph representation of AF, we will present a medical scenario. In this medical scenario, a decision about whether an organ from a donor with endocarditis is viable or not for being transplanted should be made[1].

Let us assume that we have two transplant coordination units, one which is against the viability of the organ (UCT_D) and one which is in favour of the viability of the organ (UCT_R).

[1]The medical information was taken from (25).

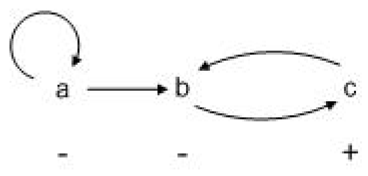

Figure 4.6: Graph representation of the argumentation framework $AF := \langle \{a, b, c\}, \{(a, a), (a, b), (b, c), (c, b)\}\rangle$ with labeling.

- UCT_D argues that the organ is not viable, since the organ donor had endocarditis due to *streptococcus viridans*, then the organ recipient could be infected by the same microorganism.
- In contrast, UCT_R argues that the organ is viable, because the organ presents correct function and correct structure and the infection could be prevented with a post-transplanted-treatment with penicillin, even if the organ recipient is allergic to penicillin, there is the option of a post-transplanted-treatment with teicoplanin.

Formally, we have an argumentation framework $AF := \langle AR, attacks\rangle$, where AR has the following arguments:

a.- organ is non viable.

b.- organ is viable.

c.- organ has correct function and correct structure.

d.- organ recipient could be infected with streptococcus viridans.

e.- post-transplanted-treatment with administer penicillin.

f.- post-transplanted-treatment with administer teicoplanin.

g.- recipient is allergic to penicillin.

and $attacks := \{(a, b), (b, a), (c, a), (d, b), (e, d), (f, d), (g, e)\}$ (The graphic representation of AF is shown in Figure 4.7). Then Ψ_{AF} is:

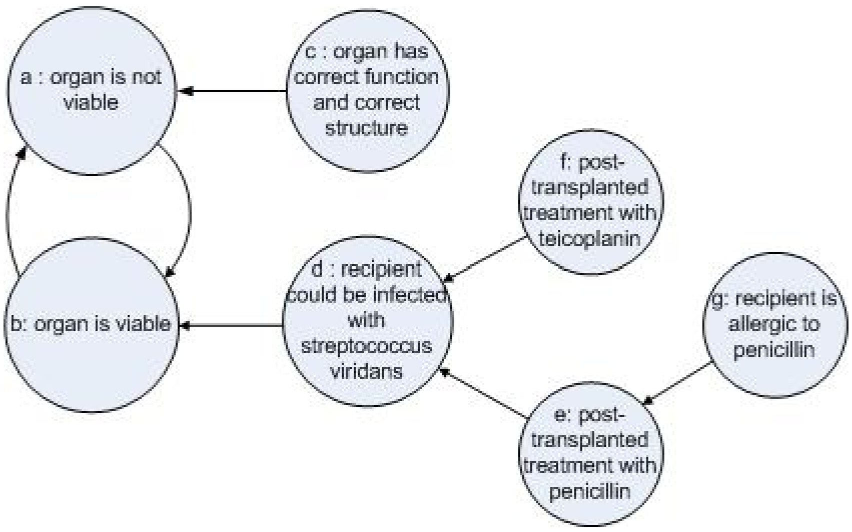
a : organ is not viable
c : organ has correct function and correct structure
f: post-transplanted treatment with teicoplanin
d : recipient could be infected with streptococcus viridans
b: organ is viable
g: recipient is allergic to penicillin
e: post-transplanted treatment with penicillin

Figure 4.7: A medical scenario where the decision about whether an organ from a donor with endocarditis is viable or not for being transplanted should be made.

$d(a) \leftarrow$ *not* $d(b)$.	$d(a) \leftarrow d(a), d(d)$.
$d(a) \leftarrow$ *not* $d(c)$.	$d(a) \leftarrow \top$.
$d(b) \leftarrow$ *not* $d(a)$.	$d(b) \leftarrow d(b), d(c)$.
$d(b) \leftarrow$ *not* $d(d)$.	$d(b) \leftarrow d(e), d(f)$.
$d(d) \leftarrow$ *not* $d(e)$.	$d(d) \leftarrow d(g)$.
$d(d) \leftarrow$ *not* $d(f)$.	$d(d) \leftarrow \top$.
$d(e) \leftarrow$ *not* $d(g)$.	$d(e) \leftarrow \top$.

A good question is: What can we infer from Ψ_{AF} and AF *w.r.t.* the medical scenario at this moment? We have not applied any transformation to Ψ_{AF} yet; however we can see that $SEM(\Psi_{AF}) := \langle \{d(a), d(d), d(e)\}, \{d(c), d(f), d(g)\} \rangle$. Hence:

$l_{SEM}(c) = +$	$l_{SEM}(a) = -$	$l_{SEM}(b) = ?$
$l_{SEM}(f) = +$	$l_{SEM}(d) = -$	
$l_{SEM}(g) = +$	$l_{SEM}(e) = -$	

This means that both UCT_D and UCT_R agree with regarding the arguments $\{c, f, g\}$ are acceptable. Hence the arguments that are attacked by them are defeated which are $\{a, d, e\}$. At this moment $SEM(\Psi_{AF})$ cannot say nothing about the argument b (see Figure 4.8).

Now, let us apply the transformation rule **Failure** to Ψ_{AF} as many times as we can, then we get the following reduced program:

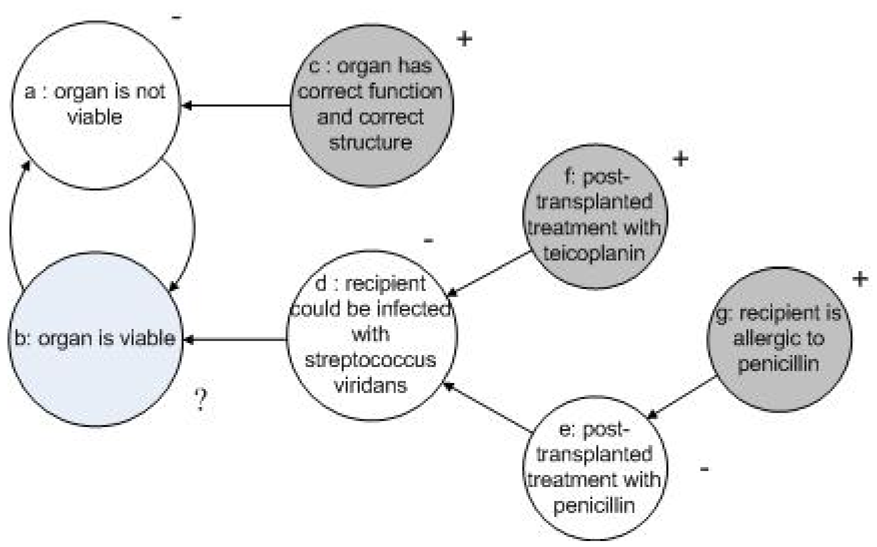
a : organ is not viable
-
c : organ has correct function and correct structure
+
f: post-transplanted treatment with teicoplanin
+
-
d : recipient could be infected with streptococcus viridans
g: recipient is allergic to penicillin
+
b: organ is viable
?
e: post-transplanted treatment with penicillin
-

Figure 4.8: Medical scenario — acceptable arguments: $\{c, f, g\}$, defeated arguments: {a, d, e }, undecided arguments: {b}.

$d(a) \leftarrow\ not\ d(b).$	$d(a) \leftarrow d(a), d(d).$
$d(a) \leftarrow\ not\ d(c).$	$d(a) \leftarrow \top.$
$d(b) \leftarrow\ not\ d(a).$	
$d(b) \leftarrow\ not\ d(d).$	
$d(d) \leftarrow\ not\ d(e).$	
$d(d) \leftarrow\ not\ d(f).$	$d(d) \leftarrow \top.$
$d(e) \leftarrow\ not\ d(g).$	$d(e) \leftarrow \top.$

Notice that the clauses that were removed by Failure were assuming that the argument b could be defeated in case that the arguments c and f were defeated, but we have already known that c and f are acceptable arguments, hence those clauses are irrelevant.

Now, let us apply the transformation rule **RED**$^-$ to the reduced program as many times as we can, then the new reduced program Ψ'_{AF} is:

$d(a) \leftarrow\ not\ d(b).$	$d(a) \leftarrow d(a), d(d).$
$d(a) \leftarrow\ not\ d(c).$	$d(a) \leftarrow \top.$
$d(d) \leftarrow\ not\ d(f).$	$d(d) \leftarrow \top.$
$d(e) \leftarrow\ not\ d(g).$	$d(e) \leftarrow \top.$

Now notice that the clauses which were removed from the program were the clauses which were representing the attackers of the arguments $\{a, d, e\}$. At this moment $SEM(\Psi'_{AF}) := \langle\{d(a), d(d), d(e)\}, \{d(c), d(f), d(g), d(b)\}\rangle$, this means that $SEM(\Psi_{AF})$ is able to suggest that the argument b is acceptable. Applying the transformation rule *Success* as many times as we can, we get the following new reduced program.

$$\begin{array}{lll} d(a) \leftarrow & not\ d(b). & \\ d(a) \leftarrow & not\ d(c). & d(a) \leftarrow \top. \\ d(d) \leftarrow & not\ d(f). & d(d) \leftarrow \top. \\ d(e) \leftarrow & not\ d(g). & d(e) \leftarrow \top. \end{array}$$

We can observe that in this reduced program, all the attackers of the defeated arguments were removed from the program. We can visualize this status as is shown in Figure 4.9. Finally, applying the transformation rule Red^+, we get the normal form of Ψ_{AF} which is:

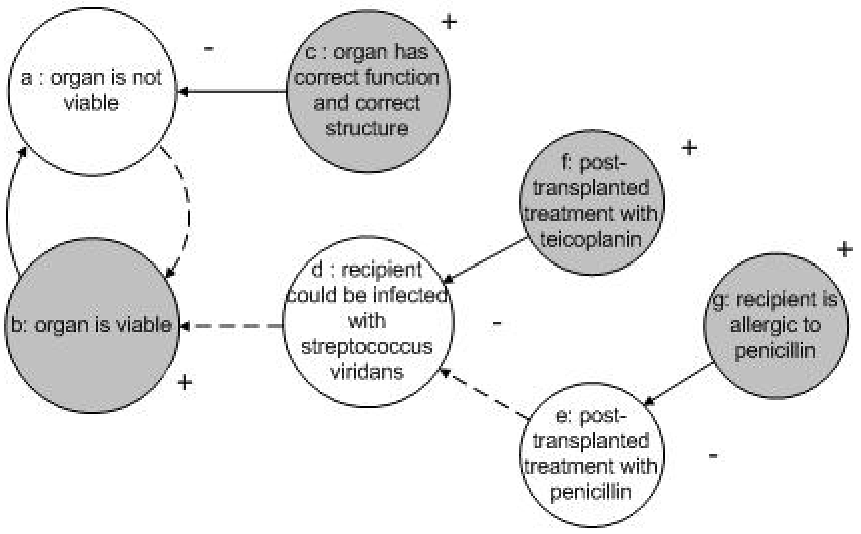
a : organ is not viable
c : organ has correct function and correct structure
f: post-transplanted treatment with teicoplanin
d : recipient could be infected with streptococcus viridans
b: organ is viable
g: recipient is allergic to penicillin
e: post-transplanted treatment with penicillin
+
-
+
+
-
+
-

Figure 4.9: Medical scenario — acceptable arguments: $\{c, f, g\}$, defeated arguments: {a, d, e, d }, undecided arguments: $\emptyset$.

$$d(a) \leftarrow \top. \qquad d(d) \leftarrow \top. \qquad d(e) \leftarrow \top.$$

This means that $WFS(\Psi_{AF}) = \langle\{d(a), d(d), d(e)\}, \{d(c), d(f), d(g), d(b)\}\rangle$; hence, we have the 3-valued extension $\langle\{c, f, g, b\}, \{a, d, e\}\rangle$ and the following final mapping of the each argument of AF:

$l_{SEM}(c) = +$	$l_{SEM}(a) = -$
$l_{SEM}(f) = +$	$l_{SEM}(d) = -$
$l_{SEM}(g) = +$	$l_{SEM}(e) = -$
$l_{SEM}(b) = +$	

Since the argument b is acceptable, one can conclude that the organ is viable for transplanting and the support of this conclusion are the arguments c, f, and g. Therefore, one can say that the transplant coordination unit UCT_R is the winner of the disagreement.

Notice that the grounded extension of the argumentation framework AF is: $\{b, c, f, g\}$. As all the new semantics presented in Section 4.6.1 are extensions of the grounded semantics, then all of them have the behavior described in this section.

In general terms, we can say that the interaction between rewriting systems and normal programs which represent an argumentation framework could describe the interaction of arguments of an argumentation framework. This allows to visualize the process of selecting acceptable arguments from an argumentation framework.

4.7 Concluding remarks

In this chapter, we concentrate our attention in some well-accepted patterns (argumentation semantics) for inferring arguments which are part of a conflict.

When Dung introduced his abstract argumentation approach, he proved that it can be regarded as a special form of logic programming with *negation as failure*. In fact, he showed that the grounded and stable semantics can be characterized by the well-founded and answer set semantics respectively. This result has at least two main implications:

1. It defines a general method for generating metainterpreters for argumentation systems and
2. it defines a general method for studying abstract argumentation semantics' properties in terms of logic programming semantics' properties.

Concerning this issue, Dung did not give any characterization of the preferred semantics in terms of logic programming semantics.

The first result of this chapter is a reconsideration of Theorem 17 of (44). We show that there exists a suitable codification (Ψ_{AF}) of an argumentation framework in terms of a logic program such that

- the well-founded model of Ψ_{AF} characterizes the grounded extension of AF (Lemma 4.1);

- the answer sets of Ψ_{AF} characterize the stable extensions of AF (Lemma 4.2); and
- the pstable models of Ψ_{AF} characterize the preferred extensions of AF (Lemma 4.3).

This result unifies, in logic programming with negation as failure, the three principal argumentation semantics of Dung's approach. Also this means that the main patterns of inference in argumentation semantics can be totally captured by logic programming with *negation as failure*. These results suggest that one can explore argumentation semantics by considering the results of logic programming with *negation as failure*. For instance, since the pstable semantics can be characterized by paraconsistent logics (as the Cw and G_3' logics (89)) or modal logics (as the S5 modal logic (91)), one can explore argumentation constructions in terms of paraconsistent logics or modal logics.

The existence of codifications as Ψ_{AF} suggests a class of suitable codifications for exploring the interaction of arguments by considering logic programming semantics. To find suitable codifications for argumentation theory based on logic programming could help to close the wide separation between argumentation theory and argumentation systems. It is quite obvious that a suitable codification of an argumentation framework should not only permit to compute abstract argumentation semantics, but also it ought to permit to perform a deep study about an abstract argumentation semantics. In this sense, we show that Ψ_{AF} allows to characterize the preferred semantics in terms of

- minimal models (see Theorem 4.2) and
- the answer set semantics (see Theorem 4.4).

These characterizations have as main result the definition of a direct relationship between one of the most satisfactory argumentation semantics and may be the most successful approach of non-monotonic reasoning of the last two decades *i.e.* logic programming with the answer set semantics. Based on this fact, we introduce a novel and easy-to-use method for implementing argumentation systems which are based on the preferred semantics. It is quite obvious that our method will take advantage of the platform that has been developed under stable model semantics for generating argumentation systems. For instance, we can implement the preferred semantics inside Object-Oriented programs based on our characterization (Theorem 4.4, Proposition 4.4) and DLV JAVA Wrapper (100).

Our experience in the interaction between argumentation semantics and logic programming semantics suggests that the correct understanding of the behavior of one side helps to understand the behavior of the other side. For instance, thanks to the deep study that there is on the well-founded semantics, we define three extensions of the grounded semantics:

- the $WFS^{LLC'}$-extension,
- the WFS^{WK}-extension and
- the $WFS^{WK+LLC'}$- extension.

The $WFS^{LLC'}$-extension is an interesting extension of the grounded extension which is able to deal with the problem of *self-defeated arguments* (99). The construction of the $WFS^{LLC'}$-extension is based on the transformation rule LLC' that it is nothing else than the application of *the local logical consequence* in classical logic to a logic normal program. In classical logic, we know that

$$\sim a \rightarrow a \vdash_c a \tag{4.2}$$

Also, we can see that if $AF := \langle \{a\}, \{(a,a)\} \rangle$, then Ψ_{AF} is:

$$d(a) \leftarrow \neg d(a). \qquad d(a) \leftarrow d(a).$$

by applying LLC' to Ψ_{AF}, we get the program Ψ'_{AF}:

$$d(a). \qquad d(a) \leftarrow d(a).$$

Hence $SEM(\Psi'_{AF}) = \langle \{d(a)\}, \{\} \rangle$. This means that the argument a is fixed as a defeated argument. Since WFS is a modular semantics (36), therefore we can guarantee by (4.2) that any self-defeated argument will be fixed as a defeated argument in the $WFS^{LLC'}$-extension. In fact this property is also satisfied by the $WFS^{WK+LLC'}$- extension.

The WFS^{WK}-extension is another interesting extension of the grounded semantics which provides to the grounded semantics with the property of *reasoning by cases*.

The last extension of the grounded semantics presented, the $WFS^{WK+LLC'}$-extension, is the stronger extension of the grounded semantics introduced in this chapter. The $WFS^{WK+LLC'}$- extension concentrates the properties of the grounded semantics, the $WFS^{LLC'}$-extension and the WFS^{WK}-extension.

Observe that the extensions of the grounded semantics presented in this chapter show that by considering an argumentation framework as a logic program one can construct argumentation semantics which could satisfy some particular properties. These properties will be supported by formal logic foundations.

It is worth mentioning that the extensions of the grounded semantics presented in this chapter are intermediate semantics between the grounded and preferred semantics and they have the property that they are polynomial time computable (Theorem 4.5, Theorem 4.6).

As we know, WFS is a three valued semantics and all the extensions of the grounded semantics presented here are three-valued extensions. Hence, by considering this relationship, in the last section of this chapter, we *outline* a labeling process for identifying the status of an argument in the process of inferring three valued argumentation semantics. This approach has as aim to describe the interaction of the arguments of an argumentation framework based on rewiring systems and Ψ_{AF}. This approach is close related to the approach presented in (58); however, a deep study is needed in order to identify the possible relationships between the argumentation semantics presented in (58) and some possible logic programming semantics.

Chapter 5

Beyond of admissible sets

In this chapter, by considering the idea that argumentation semantics can be viewed as a special form of logic programming semantics with negation as failure, we show that any logic programming semantics as the answer set semantics, the minimal models, the pstable semantics etc.*, can define candidate argumentation semantics. These new argumentation semantics will overcome some of the problems of the Dung's argumentation semantics that have been discussed in the literature.*

5.1 Introduction

As commented in Section 1.2, according to Bench-Capon and Dunne, the three principal abstract argumentation semantics introduced by Dung are the grounded, preferred and stable semantics. However, these semantics exhibit a variety of problems which have illustrated in the literature (13; 16; 26; 27; 99). Authors as P. Baroni *et al*, have suggested that in order to overcome Dung's abstract argumentation semantics problems, it is necessary to define flexible argumentation semantics which are not necessarily based on admissible sets (13).

According to Baroni *et al*, in (13) the preferred semantics is regarded as the most satisfactory approach; however, they have also pointed out that the preferred semantics produces some questionable results in some cases concerning cyclic attack relations (13). For instance, let us consider the argumentation framework that appears in Figure 5.1[1]. In this argumentation framework there are two arguments: a and b. We can see that the argument a is attacked by itself and the argument b is attacked by the argument a. Intuitively, some authors as

[1]This argumentation framework has received special attention in the literature in order to commented the problem of the self-defeated arguments (98; 99) and to point out some of the problems of the Dung's argumentation semantics (13).

Prakken and Vreeswijk (99) suggest that one can expect that the argument b can be considered as an acceptable argument since it is attacked by the argument a which is attacked by itself. However, the preferred semantics is unable to infer the argument b as an acceptable argument — the only preferred extension of the argumentation framework of Figure 5.1 is the empty set. In fact, none of the argumentation semantics suggested by Dung is able to infer the argument b as acceptable.

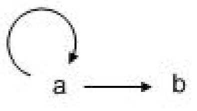

Figure 5.1: Graph representation of the argumentation framework $AF := \langle \{a, b\}, \{(a, a), (a, b)\}\rangle$.

Another interesting argumentation framework which has been commented on literature (13; 99) is presented in Figure 5.2. The preferred semantics *w.r.t.* this argumentation framework is only able to infer the empty set. Some authors, as Prakken and Vreeswijk (99), Baroni *et al*(13), suggest that the argument e can be considered as an acceptable argument since it is attacked by the argument d which is attacked by three arguments: a, b, c. Observe that the arguments a, b and c form a cyclic of attacks.

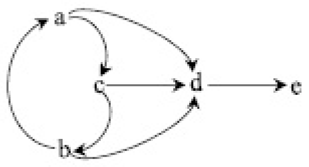

Figure 5.2: Graph representation of the argumentation framework $AF := \langle \{a, b, c, d, e\}, \{(a, c), (c, b), (b, a), (a, d), (c, d), (b, d), (d, e)\}\rangle$.

The stable argumentation semantics defined by Dung in (44) is also considered as another proper argumentation semantics. However, this semantics has been criticized by some authors as Bench-Capon and Dunne (16), Caminada (27) because frequently this semantics is undefined . For instance, the argumentation framework of Figure 5.3 has no stable extensions; however, it has a preferred extension, $\{c\}$ — in fact the argumentation frameworks of Figure 5.1 and Figure 5.2 are two examples where the stable argumentation semantics is also undefined.

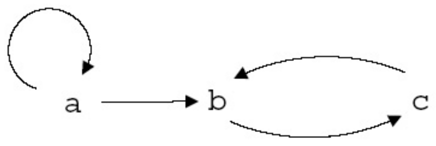

Figure 5.3: Graph representation of the argumentation framework $AF = \langle \{a, b, c\}, \{(a, a), (a, b), (b, c), (c, b)\}\rangle$.

The solutions to the problems of the argumentation semantics suggested by Dung are really diverse some researchers have focused on improving the stable argumentation semantics (27), some other researchers have focused on improving the preferred semantics (86), and still other researchers have focused on improving the concept of *admissible set* which is the basis of the argumentation semantics suggested by Dung (13; 58).

We can recognize two major branches for improving Dung's approach. On the one hand, we can take advantage of graph theory; on the other hand, we can take advantage of logic programming with negation as failure.

With respect to graph theory, the approach suggested by Baroni *et al*, in (13) is maybe the most general solution defined until now for improving Dung's approach. This approach is based on a solid concept in graph theory which is a *strongly connected component* (SCC). Based on this concept, Baroni *et al*, describe a recursive approach for generating new argumentation semantics. For instance, the argumentation semantics CF2 suggested in (13) is able to infer the argument b as an acceptable argument of the argumentation framework of Figure 5.1. Also CF2 is able to infer the extensions: $\{a, e\}$, $\{b, e\}$, $\{c, e\}$ from the argumentation framework of Figure 5.2. This means that CF2 regards the argument e as an acceptable argument.

As we saw in Chapter 4, argumentation theory can be viewed as a special form of logic programming with *negation as failure*. In particular, we introduced a mapping in order to map an argumentation framework into a logic program. When we have a logic program which represents an argumentation framework, it is natural to think that we can split this program into subprograms where each subprogram could represent a part of an argumentation framework. For instance, let us consider a single version of the mapping introduced in Definition 4.3 in order to represent the argumentation framework of Figure 5.1 as the logic program P:

$$
\begin{array}{l}
d(a) \leftarrow \ not\ d(a). \\
d(b) \leftarrow \ not\ d(a). \\
acc(a) \leftarrow \ not\ d(a). \\
acc(b) \leftarrow \ not\ d(b).
\end{array}
$$

Observe that we are only considering the negative clauses of the mapping Ψ_{AF} and two clauses more in order to infer the acceptable arguments by *negation as failure*. Moreover observe that this program can be also inferred from Dung's mapping P_{AF} (see Definition 2.13) by considering the grounding instance of P_{AF} and applying the well-known principle of partial evaluation to P_{AF} (this process is illustrated in the proof of Lemma 4.2). This codification can be regarded as the common point between the mappings Ψ_{AF} and P_{AF}.

As we saw in §4.3, the intended meaning of the first clause of P says that the argument a is defeated if the argument a is not defeated. The second clause of P says that the argument b is defeated if the argument a is not defeated. The third clause of P says that the argument a is acceptable if the argument a is not defeated and the last clause says that the argument b is acceptable if the argument b is not defeated.

Notice that the program P can be split in three subprograms, *i.e.* P_1, P_2 and P_3, where P_1 is:

$$d(a) \leftarrow \ not\ d(a).$$

P_2 is:

$$
\begin{array}{l}
d(b) \leftarrow \ not\ d(a). \\
acc(a) \leftarrow \ not\ d(a).
\end{array}
$$

and P_3 is:

$$acc(b) \leftarrow \ not\ d(b).$$

We can see that P_2 depends on P_1 because the atom $d(a)$ is defined in the program P_1. In the same way, P_3 depends on P_1 and P_2. Hence, in order to infer the semantics of P_2, we have to infer the semantics of P_1 before. For instance, let us consider the minimal models of P_1. It is easy to see that the only minimal model of P_1 is: $\{d(a)\}$. Hence, in order to infer the semantics of P_2 based on the minimal models of P_1, we can remove from P_2 any clause that contains $not\ \ d(a)$ in their bodies — let P_2' be the reduced program. Notice that P_2' is an empty program; hence, the only minimal model of P_2' is the empty model *i.e.* the atoms $d(b)$ and $acc(a)$ are considered as false. Now, for inferring the semantics of P_3, we consider the minimal models of P_1 union the minimal models of P_2'. We can infer the semantics of P_3 based on the model $\{d(a)\}$ — let P'_3 be the reduced program by considering $d(a)$ as true and $d(b)$ as false. It is easy to see that the only minimal model of P'_3 is: $\{acc(b)\}$. Therefore, the semantics of P will be the union of the minimal models of P_1 ($\{d(a)\}$) union the minimal models of P'_2 ($\emptyset$) union the minimal models of P'_3 ($\{acc(b)\}$). Hence, we have an unique model for P which is $\{acc(b), d(a)\}$. This model suggests that we can consider the argument b as acceptable and the argument a as defeated.

The idea of spitting a logic program into its component, in order to define logic programming semantics, has been explored by some authors in logic programming (37). For instance, by splitting a logic program, Dix and Müller in (37) combine ideas of the answer set semantics and the well-founded semantics in order to define a skeptical logic programming semantics which satisfies the property of relevance and the general principle of partial evaluation.

In the first part of this chapter, we will formalize a recursive general schema for constructing new logic programming semantics. This recursive general schema is based on the idea of splitting a logic program into its components. The new logic programming semantics have as main properties that they are always defined for any logic program and they satisfy the property of relevance.

In the second part of this chapter, by considering the idea that argumentation semantics can be viewed as a special form of logic programming semantics with negation as failure and the schema presented in the first part of the chapter, we show that any logic programming semantics as the answer set semantics semantics, the minimal models, *etc.*, can define candidate argumentation semantics. These new argumentation semantics will overcome some of the problems of the Dung's argumentation semantics that have been discussed in the literature. In fact, we will see that some of our new argumentation semantics have similar behavior to the argumentation semantics defined in terms *strongly connected components* (13).

The rest of the chapter is divided as follows: In §5.2, we define some extra concept *w.r.t.* logic programming. In §5.3, we introduce our new recursive general schema for defining new logic programming semantics. In §5.4, we define how to construct new argumentation semantics based on the approach presented in §5.3. Finally in the last section, we present our concluding remarks.

5.2 Preliminaries

In this section, we will define some important concepts in order to split a logic program into its components. We start presenting some basic relations.

Given a normal program P, we will call *non-trivial tautology* $C \in P$ if C is at one of the following forms:

$$C = a \leftarrow (\mathcal{B}^{+} \cup \{a\}),\ \mathit{not}\ \mathcal{B}^{-} \text{ such that } \mathcal{B}^{+} \neq \emptyset \text{ or } \mathcal{B}^{-} \neq \emptyset \tag{5.1}$$
$$C = a \leftarrow (\mathcal{B}^{+} \cup \{x\}),\ \mathit{not}\ (\mathcal{B}^{-} \cup \{x\}) \text{ such that } x \in \mathcal{L}_P \tag{5.2}$$

For instance, the clauses $a \leftarrow a,\ \mathit{not}\ b$ and $b \leftarrow a,\ \mathit{not}\ a$ are two non-trivial tautologies. The clauses of the form $x \leftarrow x$ will be called *trivial tautologies*.

A program P induces a notion of *dependency* between atoms from $\mathcal{L}_P$. We say that *a depends immediately on b* if and only if b appears in the body of a clause in P, such that a appears in its head. The two place relation *depends on* is the transitive closure of *depends immediately on*. The *dependencies of an atom x* is defined by:

$\mathit{dependencies}\text{-}\mathit{of}(x)$ is the set $\{a | x\ \mathit{depends\ on}\ a\}$

In order to illustrate this definition, let us consider the following normal program, denoted by RE (Running Example):

$e \leftarrow e.$
$c \leftarrow c.$
$a \leftarrow\ \mathit{not}\ b, c.$
$b \leftarrow\ \mathit{not}\ a,\ \mathit{not}\ e.$
$d \leftarrow b.$

We can see that $\mathcal{L}_P = \{a, b, c, d, e\}$. Now let us infer the dependency relations between the atoms of $\mathcal{L}_P$:

$\mathit{dependencies}\text{-}\mathit{of}(a) = \{a, b, c, e\}$
$\mathit{dependencies}\text{-}\mathit{of}(b) = \{a, b, c, e\}$
$\mathit{dependencies}\text{-}\mathit{of}(c) = \{c\}$
$\mathit{dependencies}\text{-}\mathit{of}(d) = \{a, b, c, e\}$
$\mathit{dependencies}\text{-}\mathit{of}(e) = \{e\}$

We define an equivalence relation $\equiv$ between atoms of $\mathcal{L}_P$ as follows: $a \equiv b$ if and only if $a = b$ or (a *depends-on* b and b *depends-on* a). We write $[a]$ to denote the equivalent class induced by the atom a. By considering again the normal program RE, we can see that:

$$\begin{array}{ll} [a] = [b] = \{a, b\} & [d] = \{d\} \\ [c] = \{c\} & [e] = \{e\} \end{array}$$

We take $\leq_P$ to denote the partial order induced by $\equiv_P$ on its equivalent classes. Hence, $[a] \leq [b]$ if and only if b *depends-on* a. For instance, by considering the equivalent classes of the program RE, the following relations hold: $\{c, e\} \leq \{a, b\} \leq \{d\}$. By considering the relation $\leq_P$, each atom of $\mathcal{L}_P$ is assigned an order as follows:

- An atom a is *of order* 0, if $[a]$ is minimal in $\leq_P$.
- An atom a is *of order* $n+1$, if n is the maximal order of the atoms of which a depends such that n is the order of the atom b and $b \neq a$.

We say that a program P is of order n if n is the maximum order of its atoms. By considering again the normal program RE, we can see that:

a is of order 1	d is of order 2
b is of order 1	e is of order 0
c is of order 0	

this means that RE is a program of order 2.

We can also break a program P (of order n) into the disjoint union of programs P_i ($0 \leq i \leq n$) such that P_i is the set of rules such that the head of each atom is of order i (with respect to P). We say that $P_0, \ldots, P_n$ are the relevant modules of P. In order to illustrate these ideas, let us consider the following table, where RE_0, RE_1, RE_2 are the respective relevant modules of the normal program RE:

RE	RE_0	RE_1	RE_2
$e \leftarrow e.$	$e \leftarrow e.$		
$c \leftarrow c.$	$c \leftarrow c.$		
$a \leftarrow$ *not* $b, c.$		$a \leftarrow$ *not* $b, c.$	
$b \leftarrow$ *not* a, *not* $e.$		$b \leftarrow$ *not* a, *not* $e.$	
$d \leftarrow b.$			$d \leftarrow b.$

There is an interesting class of normal programs that is called *stratified programs*. This class of normal logic programs satisfies certain syntactic conditions *w.r.t.* the occurrence of their positive and negative literals. The stratified logic programs have gained a lot of importance in connection with the search for nice declarative semantics for logic programs and the treatment of negative information in logic programming — the interested reader can find some interesting results *w.r.t.* stratified programs in (11; 68). The formal definition of a stratified logic program is defined as follows:

Definition 5.1 *(11) Let P be a normal logic program. P is called a stratified logic program if for any clause $a \leftarrow l_1, \ldots, l_m$, not $l_{m+1}, \ldots,$ not l_{nn} in P_i, $0 \leq i \leq n$, then*

1. *for every l_j, $1 \leq j \leq m$ there is a P_q such that $l_j \in \mathcal{L}_{P_q}$ and $q \leq i$, and*

2. for every l_j, $m+1 \leq j \leq nn$, there is a P_q such that $l_j \in \mathcal{L}_{P_q}$ and $q < i$.

Observe that the first condition only says that the positive literals of the clause can appear in any component lower than or equal to P_i, and the second condition says that the negative literals must appear in a clause which belongs to a component strictly lower than P_i. For instance, one can see that the logic program RE, introduced above, is not a stratified normal programs. However, the component RE_0 is a stratified logic program by itself.

Now we introduce a single reduction for any normal program. The idea of this reduction is to remove from a normal program any atom which has already fixed to some true value. In fact this reduction is based on a pair of sets of atoms $\langle T; F\rangle$ such that the set T contains the atoms which can be considered as true and the set F contains the atoms which can be considered as false. Formally, this reduction is defined as follows:

Let $\langle T; F\rangle$ be a pair of sets of atoms. The reduction $R(P, \langle T; F\rangle)$ is obtained by 4 steps:

1. We replace every atom x that occurs in the bodies of P by 1 if $x \in T$ as well as we replace every atom x that occurs in the bodies of P by 0 if $x \in F$;
2. We replace every occurrence of *not* 1 by 0 and *not* 0 by 1;
3. Every clause with a 0 in its body is removed;
4. Finally, we remove every occurrence of 1 in the body of the clauses.

Note that this is not the Gelfond-Lifschitz reduction presented in Section 2.4.1. For instance, let P be the normal program $RE \setminus RE_0$. This means that P is:

$a \leftarrow \ not\ b, c.$
$b \leftarrow \ not\ a,\ not\ e.$
$d \leftarrow b.$

Then, $R(P, \langle\{c\}; \{e\}\rangle)$ is:

$a \leftarrow \ not\ b.$
$b \leftarrow \ not\ a.$
$d \leftarrow b.$

Given a set of interpretations Q and a signature $\mathcal{L}$, we define Q *restricted to* $\mathcal{L}$ as $\{M \cap \mathcal{L} | M \in Q\}$. For instance, let Q be $\{\{a, c\}, \{c, d\}\}$ and $\mathcal{L}$ be $\{c, d, e\}$, hence Q *restricted to* $\mathcal{L}$ is $\{\{c\}, \{c, d\}\}$.

Let P be a program and $P_0, \ldots, P_n$ its relevant modules. We say that a semantics S satisfies the property of relevance if for every i, $0 \leq i \leq n$, $S(P_0 \cup \cdots \cup P_i) = S(P)$ restricted to $\mathcal{L}_{P_0 \cup \cdots \cup P_i}$.

Some logic programming semantics that we will consider in this chapter are *the minimal model semantics* (denoted by MM)[1], *the answer set semantics* (53) (denoted by $AnswerSets$), *the revised stable semantics* (97) (denoted by $RevStable$) and *the pstable semantics* (92) (denoted by $Pstable$).

5.3 Construction of new logic programming semantics

In this section we construct elaborated logic programming semantics based on a simpler logic programming semantics. We have in mind that our new logic programming semantics satisfy the following suitable properties:

1. *They should be always defined.* In both approaches logic programming and argumentation theory, it is appreciated that logic programming semantics (resp. argumentation semantics) can be always defined (16; 28; 37; 97). Hence, our new logic programming semantics will be always defined.

2. *They should satisfy the relevance property.* The relevance property is an important property in order to define well-behaved semantics in logic programming (36; 37).

3. *They should agree with $AnswerSets$ for the class of stratified programs.* On the one hand, answer set semantics is probably the most accepted logic programming semantics in the last two decades, and on the other hand, the class of stratified logic programs is the class of logic programs where the most accepted logic programming semantics agree (11; 68).

4. *They should be useful to model argumentation problems.* As we commented in §5.1, one of the main objectives of this chapter is to suggest a general schema for constructing argumentation semantics based on logic programming semantics.

We assume that every semantics S satisfies the following trivial property:

> *For every program P such that every atom that appears in P also occurs as a fact of P then $S(P)$ is defined.*

Every well known semantics satisfies this basic property such as $AnswerSets$, MM, $RevStable$, and $Pstable$. In fact in all these semantics there is exactly one

[1]It is worth mentioning that the minimal model semantics of a logic program P is given by the minimal models of P.

intended model that is $\mathcal{L}_P$. For instance, let P be the following program:

$$\begin{array}{ll} a \leftarrow \top. & b \leftarrow \top. \\ a \leftarrow b. & b \leftarrow \ not\ a. \end{array}$$

Observe that all the atoms of the program P appear as facts in P; hence P will always have the model: $\{a, b\}$.

Remark 5.1 *For the construction of our semantics we assume that our programs do not include* non-trivial tautologies *(see §5.2) . If a program includes them, we simply remove them in the very first stage of our construction of our semantics.*

5.3.1 Semantics always defined

It is sometimes desirable that a semantics of a normal program is always defined; for instance, the case when a program is modeling an argumentation problem. Given a particular semantics S, we show how to construct a semantics based on S that is always defined.

First of all, we will define some concepts *w.r.t.* the notion of generalized S model.

Definition 5.2 *Let S be a logic programming semantics, P be a logic program and A be a set of atoms (called abductives) such that $A \subseteq \mathcal{L}_P$.*

- *We say that M_B is a generalized S model of P* w.r.t. *A if $M \in S(P \cup B)$ where $B \subseteq A$ and $M \subseteq \mathcal{L}_P$.*
- *We define a partial order between generalized S models (*w.r.t. *A) of a program according to the set inclusion* w.r.t. *the subindex B. We say that M is a minimal generalized S model of P* w.r.t. *A if there exists a set of atoms B, such that M_B is a generalized S model of P* w.r.t. *A and M_B is minimal* w.r.t. *the partial order just defined.*
- *We write S^* to denote the minimal generalized S semantics, where $A = \mathcal{L}_P$. Namely $S^*(P)$ is the collection of minimal generalized S models of P* w.r.t. *$\mathcal{L}_P$.*

Observe that in our definition we are not instantiating the definition to a particular logic programming semantics.

The concept of generalized S model is closely related to the semantics of *abductive logic programming* (60; 61), in particular to the concept of *generalized answer set*. It has also been explored for different logic programming approaches as in (9; 93). For instance, the authors in (93) consider two partial order relations between the generalized models for defining minimal generalized models, one by considering the set inclusion *w.r.t.* the subindexes of the generalized models (as in Definition 5.2) and another one *w.r.t.* the cardinality of the subindexes of the generalized models[1].

In order to illustrate Definition 5.2, take the program C:

$$p \leftarrow \ not\ p.$$

We can see that $AnswerSets(C)$ is undefined, however $AnswerSets^*(C) = \{\{p\}\}$. Note that $\{p\}_{\{p\}}$ is the unique generalized answer set of C. This is because $\{p\}$ is an answer set of $C \cup \{p\}$. Moreover, $\{p\}$ is the unique minimal generalized model of C. Observe also that $Pstable(C) = Pstable^*(C) = \{\{p\}\}$.

Take the program D:

$$\begin{aligned} a &\leftarrow \ not\ b.\\ b &\leftarrow \ not\ c.\\ c &\leftarrow \ not\ a. \end{aligned}$$

Observe that $Pstable(D)$ is undefined; however, one can see that $\{a, b\}$ is a pstable model of $P \cup \{a\}$, $\{b, c\}$ is a pstable model of $P \cup \{b\}$ and $\{c, a\}$ is a pstable model of $P \cup \{c\}$. Since the models $\{a, b\}_{\{a\}}$, $\{b, c\}_{\{b\}}$, $\{c, a\}_{\{c\}}$ are the three minimized generalized pstable models of D, $Pstable^*(D) = \{\{a, b\}, \{b, c\}, \{a, c\}\}$. The same situation happens to $AnswerSets$. $AnswerSets(D)$ is undefined; however, $AnswerSets^*(D) = \{\{a, b\}, \{b, c\}, \{a, c\}\}$.

The following lemma insures that any semantics induced by Definition 5.2 will be defined — the proof of the lemma is immediate thanks to the basic property defined at the beginning of this section.

Lemma 5.1 *For every semantics S and program P, $S^*(P)$ is defined.*

One important property of the semantics induced by Definition 5.2 is that the concept of generalized model will be important only in the case that the initial semantics S is undefined.

Lemma 5.2 *For every semantics S and program P. If $S(P)$ is defined then $S^*(P) = S(P)$.*

Proof: If $S(P)$ is defined then it is clear that $M \in S(P)$ iff $M_{\{\}}$ is a minimal generalized S model of P. ■

This lemma insures that whenever a semantics S is defined, it will be the same to S^* *e.g.*, $Pstable(C) = Pstable^*(C) = \{\{p\}\}$. The following lemma makes some observations *w.r.t.* MM, $RevStable$, $AnswerSets$, $Pstable$ and their induced semantics based on the concept of generalized model.

[1]In (93), a generalized model is called a $\mathcal{L}$-completion.

Lemma 5.3 *MM and MM^* are the same semantics. $RevStable$ and $RevStable^*$ are the same semantics. $AnswerSets$ is different from $AnswerSets^*$. $Pstable$ is different from $Pstable^*$. $AnswerSets^*$, $Pstable^*$, and MM^* are 3 different semantics.*

Proof: MM is the same as MM^* follows because MM is always defined (Lemma 5.2). For the same reason $RevStable$ and $RevStable^*$ are the same semantics. Note that program C already defined shows that $AnswerSets$ is different from $AnswerSets^*$. Program D shows that $Pstable$ is different from $Pstable^*$.

Consider Program E:

$$\begin{aligned} a &\leftarrow \ not\ b.\\ b &\leftarrow \ not\ a.\\ p &\leftarrow \ not\ b.\\ p &\leftarrow \ not\ p.\end{aligned}$$

$AnswerSets^*(E) = AnswerSets(E) = \{\{p,a\}\}$. $Pstable^*(E) = Pstable(E) = \{\{p,a\},\{p,b\}\}$. $MM^*(E) = MM(E) = \{\{p,a\},\{p,b\}\}$. This program shows that $Pstable^*$ is different from $AnswerSets^*$ and that MM^* is different from $AnswerSets^*$

Consider Program F:

$$\begin{aligned} a &\leftarrow \ not\ b.\\ b &\leftarrow \ not\ a.\\ u &\leftarrow a.\\ x &\leftarrow \ not\ y, u.\\ y &\leftarrow \ not\ z, u.\\ z &\leftarrow \ not\ x, u.\end{aligned}$$

One can see that $Pstable^*(F) = \{\{b\}\}$ and $MM^*(F) = \{\{b\}, \{a,u,x,y\}, \{a,u,x,z\}, \{a,u,y,z\}\}$. Hence, this program shows that $Pstable^*$ is different from MM^*. ■

5.3.2 Constructing relevant semantics

It is sometimes desirable that a semantics satisfies the relevance property. Relevance is a fundamental property when we are interested in defining the semantics of a split logic program. Given a semantics S we show how to construct a relevant semantics based on S that we denote by S^r.

We assume in this subsection the following property:

For every logic program, if an atom occurs in a program then it should also occur in the head of some rule.

Later, we show how to deal with programs that do not satisfy this property. Given Q and R both sets of interpretations, we define

$$Q * R := \{M_1 \cup M_2 | M_1 \in Q, M_2 \in R\}$$

Definition 5.3 *Let S be a logic programming semantics that is always defined. We define the associate S^r semantics recursively as follow: Given a program P of order 0, $S^r(P) = S(P)$. For a program P of order $n > 0$ we define*

$$S^r(P) = \bigcup_{M \in S(P_0)} \{M\} * S^r(R(P \setminus P_0, \langle M; \mathcal{L}_{P_0} \setminus M\rangle))$$

For the reader who knows $STABLE^{rel}$'s definition presented in (37), we want to point out that S^r is similar to $STABLE^{rel}$ *w.r.t.* the relevance property; however, $STABLE^{rel}$ has a skeptical construction and S^r has a construction by scenarios.

Consider the program E defined before. We illustrate our definition to compute $AnswerSets^{*^r}(E)$. Recall that E is:

$$\begin{aligned} a &\leftarrow\ not\ b.\\ b &\leftarrow\ not\ a.\\ p &\leftarrow\ not\ b.\\ p &\leftarrow\ not\ p.\end{aligned}$$

Then E_0 is:

$$\begin{aligned} a &\leftarrow\ not\ b.\\ b &\leftarrow\ not\ a.\end{aligned}$$

$AnswerSets(E_0) = \{\{a\}, \{b\}\}$. Since $AnswerSets(E_0)$ has two models, there are two cases to consider:

1. First, consider M be $\{a\}$. We can see that $E \setminus E_0$ is:

$$\begin{aligned} p &\leftarrow\ not\ b.\\ p &\leftarrow\ not\ p.\end{aligned}$$

Moreover, we can see that $R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle)$ is:

$$\begin{array}{l} p. \\ p \leftarrow \; not\; p. \end{array}$$

Hence, $S^r(R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle))$ is: $\{\{p\}\}$. So, $\{M\} * S^r(R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle))$ is: $\{\{p, a\}\}$.

2. Now, consider M be $\{b\}$. In this case, we can see that $R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle)$ is:

$$p \leftarrow \; not\; p.$$

Observe that $AnswerSets^*$ of this reduced program is $\{\{p\}\}$. Hence, $S^r(R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle))$ is: $\{\{p\}\}$. So, $\{M\} * S^r(R(E \setminus E_0, < M; \mathcal{L}_{E_0} \setminus M >))$ is: $\{\{p, b\}\}$.

Therefore $AnswerSets^{*^r}(E) = \{\{p, a\}, \{p, b\}\}$.

The general case.

We have shown how to construct a relevant semantics for programs where every atom in the signature of the program occurs in the head of some rule of that program. Now, why is it important that every atom in the signature of the program must occur in the head of some rule of that program? In order to answer this question, let us consider the program J:

$$a \leftarrow \; not\; b.$$

and let S be a logic programming semantics. Now, let us suppose we want to infer $S^r(J)$. Hence, the first step, in order to infer $S^r(J)$, is to split J into its components. Since a depends on b, but b does not depend on a, $[b] \leq_P [a]$. This means that J has to be split into two components: J_0 and J_1. Remember that J_0 will contain all the clauses whose head is the atom b and J_1 will contain all the clauses whose head is the atom a. It is obvious that J_0 is an empty component, but this is a problem because we cannot apply Definition 5.3 in order to infer $S^r(J)$. A simple way to avoid this problem is to add the tautology $b \leftarrow b$ to J. For instance, let K be the program:

$$\begin{array}{l} b \leftarrow b. \\ a \leftarrow \; not\; b. \end{array}$$

Observe that like J, K has also two components; however, none of them is empty. This idea will suggest a natural generalization of Definition 5.3 in order to infer S^r for any logic program.

Definition 5.4 *Let P be a normal logic program. We define*

$$S^r(P) = S^r(P \cup \{x \leftarrow x : x \in \mathcal{L}_P \setminus HEAD(P)\})$$

This completes our construction of relevant semantics. Consider again the program J:

$$a \leftarrow\ not\ b.$$

Since b does not occur in the head of some rule, we add the tautology $b \leftarrow b$, obtaining K:

$$\begin{aligned} b &\leftarrow b. \\ a &\leftarrow\ not\ b. \end{aligned}$$

Now we can proceed as before to obtain our intended semantics. $MM^{*^r}(K) = AnswerSets^{*^r}(K) = Pstable^{*^r}(K) = \{\{a\}\}$. Hence, $MM^{*^r}(J) = AnswerSets^{*^r}(J) = Pstable^{*^r}(J) = \{\{a\}\}$.

Observe that the logic programming semantics induced by Definition 5.4 are different *w.r.t.* the logic programming semantics induced by Definition 5.2. For instance, the following lemmas formalize the differences *w.r.t.* some semantics.

Lemma 5.4 *$AnswerSets^*$ is different from $AnswerSets^{*^r}$. $Pstable^*$ is different from $Pstable^{*^r}$.*

Proof: Program E shows that $AnswerSets^*$ is different from $AnswerSets^{*^r}$. Program F shows that $Pstable^*$ is different from $Pstable^{*^r}$.

■

Lemma 5.5 *$AnswerSets^{*^r}$, $Pstable^{*^r}$, and MM^{*^r} are 3 different semantics.*

Proof:

Consider Program L:

$$\begin{aligned} a &\leftarrow\ not\ b. \\ b &\leftarrow\ not\ a. \\ p &\leftarrow\ not\ b. \\ p &\leftarrow\ not\ p. \\ b &\leftarrow\ not\ p. \end{aligned}$$

Then $AnswerSets^{*^r}(L) = \{\{p, a\}\}$. However $Pstable^{*^r}(L) = \{\{p, a\}, \{p, b\}\}$. Hence $AnswerSets^{*^r}$ is different from $Ptable^{*^r}$. This example also show that $AnswerSets^{*^r}$ is different from MM^{*^r}.

Consider Program R:

$$\begin{aligned} x &\leftarrow \mathit{not}\ y.\\ y &\leftarrow \mathit{not}\ z.\\ z &\leftarrow \mathit{not}\ x.\\ x &\leftarrow \mathit{not}\ u.\\ d &\leftarrow \mathit{not}\ z.\\ u &\leftarrow \mathit{not}\ d. \end{aligned}$$

This example shows that $AnswerSets^{*^r}$ is different from MM^{*^r}. $AnswerSets^{*^r}(R) = \{\{x, y, d\}\}$. However $\{u, x, z\} \in MM^{*^r}(R)$. This example also shows that $Pstable^{*^r}$ is different from MM^{*^r}, since $Pstable^{*^r}(R) = AnswerSets^{*^r}(R)$.

■

As we commented in Section 5.2, the stratified logic programs define a class of logic programs which have interesting properties *w.r.t.* the search for nice declarative semantics. In particular we can observe that *AnswerSets*, *Pstable*, MM^{*^r}, as well as all our refined versions for *AnswerSets* and *Pstable* agree with respect to the class of stratified programs.

Lemma 5.6 *Let P be a stratified program. Hence,*
$AnswerSets(P) = Pstable(P) = MM^{*^r}(P) = AnswerSets^{*^r}(P) = Pstable^{*^r}(P)$

Proof:

The proof is by induction on the number of components of P *e.g.*, $P = P_0 \cup \cdots \cup P_n$.

Base Step By definition of stratified program, one can see that if $P = P_0$, then P is a *definite program i.e.* P does not have negative literals. Hence, $MM(P) = AnswerSets(P) = Pstable(P)$. Since $MM(P)$ is always defined, $AnswerSets(P) = Pstable(P) = MM^{*^r}(P) = AnswerSets^{*^r}(P) = Pstable^{*^r}(P)$.

Inductive Step Now, let us suppose that P has n components.

Let us suppose that it is true that $AnswerSets(P_k) = Pstable(P_k) = MM^{*^r}(P_k) = AnswerSets^{*^r}(P_k) = Pstable^{*^r}(P_k)$ for some $k < n$.

By definition, $S^r(P) = \bigcup_{M \in S(P_0)} \{M\} * S^r(R(P \setminus P_0, \langle M; \mathcal{L}_{P_0} \setminus M\rangle))$

Observe that since P is a stratified program, hence for any $M \in S^r(P_k)$, $R(P_{k+1} \setminus P_k, \langle M; \mathcal{L}_{P_{k+1}} \setminus M\rangle)$ is a *definite program*. Therefore by inductive hypothesis, $AnswerSets(P) = Pstable(P) = MM^{*^r}(P) = AnswerSets^{*^r}(P) = Pstable^{*^r}(P)$.

■

As final result of this subsection, we can introduce the following lemma whose proof is immediate by Lemma 5.1.

Lemma 5.7 *For every semantics S and program P, S^{*^r} is defined.*

This lemma insures that any logic programming semantics induced by Definition 5.2 and Definition 5.4 will be defined for any logic program.

Before to finish this section, we want to remember to the reader that there is an important preprocessing that must be applied to any logic program in order to apply the semantics introduced in this section. In Remark 5.1, we pointed out that we have assumed that our programs do not include *non-trivial tautologies*. This preprocessing has an important role in logic programming semantics as MM^{*^r}. For instance, let P be the following logic program:

$$\begin{array}{ll} a \leftarrow \ not\ b. & \\ b \leftarrow a,\ not\ a. & \text{(non-trivial tautology)} \end{array}$$

Observe that the atom b is implied by the formula $a \wedge \ not\ a$; hence, one can expect not to infer the atom b from P. However, one can see that P has a minimal model which contains the atom b. This means that $MM^{*^r} = \{\{a, b\}\}$; therefore, MM^{*^r} is inferring an unexpect model. It is worth to comment that this problem does not happen with semantics as $AnswerSets$ and $Pstable$ — these semantics only infer from P the model $\{a\}$.

In order to avoid some kind of unexpect models, one can remove from P any *non-trivial tautology*. Observe that the clause $b \leftarrow a,\ not\ a$ is a non-trivial tautology. Hence by removing this non-trivial tautology from P, we can get the logic program P':

$$a \leftarrow \ not\ b.$$

In this case like the semantics $AnswerSets$ and $Pstable$, MM^{*^r} only infers the model $\{a\}$ from P'.

It worth to comment that the constructive definition of relevant semantics presented in Definition 5.4 only considers trivial tautologies in order to insure that any atom of a given program appears in the head of some clause.

5.4 Construction of abstract argumentation semantics

In the previous section, we have defined a recursive general schema for constructing new logic programming semantics. As we mentioned in the introduction, we are interested in constructing new abstract argumentation semantics based on logic programming semantics. In this section, we will show how to take advantages of our approach for building new abstract argumentation semantics.

It is quite obvious that in order to regard an argumentation framework as a logic program, we require a function mapping which constructs a logic program from an argumentation framework. Hence let $\mathcal{M}$ be a mapping from the class of argumentation frameworks ($\mathcal{AF}_{AR}$) to the class of logic programs ($Prog_{\mathcal{L}}$). $\mathcal{M}$ assigns to every argumentation framework AF a logic program P. We are going to denote the image of AF under $\mathcal{M}$ as $\mathcal{P}_{AF}$.

Now, we will define how any logic programming semantics can induce a *candidate* abstract argumentation semantics under a particular mapping.

Definition 5.5 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and S be any logic programming semantics. The semantics S induces the candidate abstract argumentation semantics $S^{\mathcal{M}}_{Arg}$ as follows:*

$$S^{\mathcal{M}}_{Arg}(AF) := \mathcal{F}(S^{*^r}(\mathcal{P}_{AF}))$$

such that $\mathcal{F}$ is a mapping from $2^{\mathcal{L}_{\mathcal{P}_{AF}}}$ to 2^{AR}.

Observe that for each model of $\mathcal{P}_{AF}$ the mapping $\mathcal{F}$ is defining an extension for the argumentation framework AF. Informally speaking we can say that $S^{\mathcal{M}}_{Arg}$ is the *candidate abstract argumentation semantics* induced by the logic programming semantics S under the mapping $\mathcal{M}$.

The following lemma is immediate thanks to Lemma 5.1.

Lemma 5.8 *For every logic programming semantics S, the candidate abstract argumentation semantics $S^{\mathcal{M}}_{Arg}$ is always defined*

In order to illustrate our general schema for constructing abstract argumentation semantics, let us introduce a simple mapping to regard an argumentation framework as a normal logic program. In this mapping, we are only considering the negative clauses of the mapping Ψ_{AF} (see Definition 4.3). Remember that the intended meaning of the predicate *d(x)* is: "the argument x is defeated". This means that the argument x is attacked by an acceptable argument.

Definition 5.6 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework. We define:*

$$\mathcal{P}_{AF} := \bigcup_{a \in AR} \Big(\bigcup_{b:(b,a) \in attacks} d(a) \leftarrow \; not \; d(b) \Big)$$

The only condition which is captured by this program is that any argument will be defeated when anyone of its adversaries is not defeated. Observe that essentially $\mathcal{P}_{AF}$ is capturing the basic principle of *conflict-freeness* (see Definition

2.7); hence, one can insure that any candidate abstract argumentation semantics induced by $\mathcal{P}_{AF}$ at least will satisfy the principle of conflict-freeness. It worth to comment that according to Baroni and Giacomin in (12), the principle of conflict-freeness is the minimal requirement to be satisfied by any argumentation semantics. Two relevant properties of the mapping P_{AF} are:

1. The stable models of $\mathcal{P}_{AF}$ characterize the stable argumentation semantics of AF (Lemma 4.2) and
2. The well founded model of $\mathcal{P}_{AF}$ characterizes the grounded semantics of AF (Lemma 4.1).

In order to illustrate the mapping $\mathcal{P}_{AF}$, let AF be the argumentation framework of Figure 5.2. We can see that $\mathcal{P}_{AF}$ is:

$$\begin{array}{lll} d(a) \leftarrow \ not \ d(b). & d(d) \leftarrow \ not \ d(a). & d(e) \leftarrow \ not \ d(d). \\ d(b) \leftarrow \ not \ d(c). & d(d) \leftarrow \ not \ d(b). & \\ d(c) \leftarrow \ not \ d(a). & d(d) \leftarrow \ not \ d(c). & \end{array}$$

In order to construct our abstract argumentation semantics induced by any logic programming semantics and the mapping $\mathcal{P}_{AF}$. Let us point out that $\mathcal{L}_{\mathcal{P}_{AF}} := \{d(a) | a \in AR\}$. Hence given any logic programming semantics S and an argumentation framework AF:

$$\mathcal{F}(S(\mathcal{P}_{AF})) := \bigcup_{M \in S(\mathcal{P}_{AF})} \{a | d(a) \in \mathcal{L}_{\mathcal{P}_{AF}} \setminus M\}$$

We have already commented that the answer sets of $\mathcal{P}_{AF}$ characterize the stable extensions of AF. For instance, we can see that the program $\mathcal{P}_{AF}$ of the argumentation framework of Figure 5.2 has no answer sets, this means that this argumentation framework has no stable extensions. However, let us consider the argumentation semantics $AnswerSets^{*^r}_{Arg}$ *w.r.t.* AF.

First of all, we have to compute the semantics $AnswerSets^{*^r}$ *w.r.t.* $\mathcal{P}_{AF}$. It is not difficult to see that

$$AnswerSets^{*^r}(\mathcal{P}_{AF}) := \{\{d(a), d(b), d(d)\}, \{d(b), d(c), d(d)\}, \{d(a), d(c), d(d)\}\}$$

Then we can see that

$$\mathcal{F}(AnswerSets^{*^r}(\mathcal{P}_{AF})) := \{\{a, e\}, \{b, e\}, \{c, e\}\}$$

This means that

$$AnswerSets^{*^r}_{Arg}(\mathcal{P}_{AF}) := \{\{a, e\}, \{b, e\}, \{c, e\}\}$$

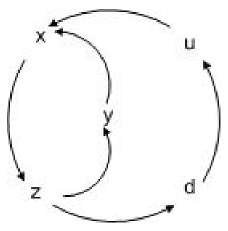

Figure 5.4: Graph representation of the argumentation framework $AF := \langle \{x, y, z, u, d\}, \{(x, z), (z, y), (y, x), (u, x), (z, d), (d, u)\}\rangle$.

Observe that the extensions suggested by the argumentation semantics $AnswerSets^{*^r}_{Arg}$ are the same to the extensions suggested by the semantics $CF2$ introduced by P. Baroni *et al*, in (13). In fact $MM^{*^r}_{Arg}$ and $Pstable^{*^r}_{Arg}$ also coincide with CF2 in this example.

Now let us consider the argumentation framework of Figure 5.4. It is not difficult to see that $\mathcal{P}_{AF}$ *w.r.t.* the argumentation framework of Figure 5.4 is:

$$\begin{array}{rl}
d(x) \leftarrow & not\ d(y). \\
d(y) \leftarrow & not\ d(z). \\
d(z) \leftarrow & not\ d(x). \\
d(x) \leftarrow & not\ d(u). \\
d(d) \leftarrow & not\ d(z). \\
d(u) \leftarrow & not\ d(d).
\end{array}$$

Observe that $\mathcal{P}_{AF}$ is exactly (modulo notation) the program R of the proof of Lemma 5.5. Since $AnswerSets^{*^r}(\mathcal{P}_{AF}) = \{\{d(x), d(y), d(d)\}\}$, therefore $AnswerSets^{*^r}_{Arg} = \{\{u, z\}\}$. We can see that $AnswerSets^{*^r}_{Arg}$ coincides with the preferred semantics, the stable argumentation semantics and $Pstable^{*^r}_{Arg}$ in this example. Now let us compute the argumentation semantics $MM^{*^r}_{Arg}$. Since $MM^{*^r}(\mathcal{P}_{AF}) :=$

$$\{\{d(y), d(z), d(u)\}, \{d(x), d(z), d(d)\}, \{d(u), d(x), d(z)\}, \{d(x), d(y), d(d)\}\}$$

then $MM^{*^r}_{Arg}(\mathcal{P}_{AF}) = \{\{x, d\}, \{y, u\}, \{y, d\}, \{z, u\}\}$. Notice that $MM^{*^r}_{Arg}$ coincides again with the argumentation semantics CF2.

We want to remark that the argumentation semantics $MM^{*^r}_{Arg}$ is really close to the argumentation semantics CF2. Let us consider another example where $MM^{*^r}_{Arg}$ and CF2 coincide. Let AF be the argumentation framework of Figure 5.5[1].

In order to compute $MM^{*^r}_{Arg}(AF)$, first of all we have to map AF to $\mathcal{P}_{AF}$:

[1]This argumentation framework was introduced in (13).

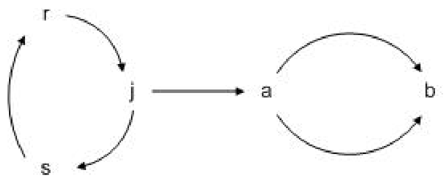

Figure 5.5: Graph representation of the argumentation framework $AF := \langle \{r, s, j, a, b\}, \{(r, j), (j, s), (s, r), (j, a), (a, b), (b, a)\}\rangle$.

$$\begin{aligned} d(r) &\leftarrow not\ d(s). \\ d(s) &\leftarrow not\ d(j). \\ d(j) &\leftarrow not\ d(r). \\ d(a) &\leftarrow not\ d(j). \\ d(a) &\leftarrow not\ d(b). \\ d(b) &\leftarrow not\ d(a). \end{aligned}$$

As we can see, $MM^{*^r}(\mathcal{P}_{AF})$ has five models which are:

$$\{\{d(r), d(s), d(a)\}, \{d(j), d(r), d(a)\}, \{d(j), d(r), d(b)\}, \\ \{d(s), d(j), d(a)\}, \{d(s), d(j), d(b)\}\}$$

This means that:

$$MM^{*^r}_{Arg}(AF) := \{\{j, b\}, \{s, b\}, \{s, a\}, \{r, b\}, \{r, a\}\}$$

As commented before, $MM^{*^r}_{Arg}(AF) = CF2(AF)$.

5.5 Related work

As far this chapter, we have commented that our approach for building new argumentation semantics is close related to Baroni *et al*'s approach presented in (13). In this section, we will point out some of the main common points between our approach and Baroni *et al*'s approach.

As we know, Baroni *et al*'s approach is based on a solid concept in graph theory which is a *strongly connected component* (SCC)[1]. Based on the fact that any argumentation framework AF can be represented by a directed graph, Baroni

[1]A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex. The strongly connected components (SCC) of a directed graph are its maximal strongly connected subgraphs (31).

et al's approach consider the strongly connected components of AF in order to introduce a recursive definition for argumentation semantics. The behavior of the argumentation semantics will be mainly influenced by

- a basic argumentation semantics-specific function. This function is a pattern of inference of arguments *e.g.*, conflict-free sets, admissible sets, *etc.*

In §5.2, we saw that any logic program induces a notion of *dependency* between atoms from $\mathcal{L}_P$. In fact, we saw that this dependency can define classes of atoms such that theses classes of atoms form a partial order which encodes the dependencies existing among the atoms. By considering theses classes of atoms, one can break any logic program into a disjoint union of subprograms. Based on this disjoint union of subprograms, we introduce a recursive definition for logic programming semantics.

In §5.4, we saw that by considering an argumentation framework as a logic program, one can use the recursive definition for logic programming semantics presented in §5.3 in order to induce candidate argumentation semantics. The behavior of the induce candidate argumentation semantics will be mainly influenced by two variables:

- the representation of an argumentation framework into a logic program and
- the basic logic programming semantics function *e.g.*, the minimal model semantics, the answer set semantics, the pstable semantics, *etc.*

Unlike to Baroni *et al*'s approach which always starts with the same representation of an argumentation framework into a directed graph, our approach allows to use different representations of an argumentation framework in terms of logic programs. These different representations could induce different dependencies of atoms of the given logic program; therefore we can break a logic program into different disjoint subprograms. This means that we can break an argumentation framework into different components.

Even though one can consider different representations of an argumentation framework in terms of logic programs, we can consider single mappings of an argumentation framework into a logic program in order to split an argumentation framework similarly as it is done by Baroni *et al*'s approach. For instance, let us consider the argumentation framework AF of Figure 5.2. Observe that the graph representation of AF has three strongly connected components:

$$\begin{aligned} SCC^0_{AF} &= \{a, b, c\} \\ SCC^1_{AF} &= \{d\} \\ SCC^2_{AF} &= \{e\} \end{aligned}$$

By considering the relation of attack between sets of arguments and the so called *directionality principle* (13), Baroni *et al*'s approach defines a partial order between theses strongly connected components (let us denote this relation as $\leq_{SCC}$). $\leq_{SCC}$ defines the following relations between the above strongly connected components:

$$\{a, b, c\} \leq_{SCC} \{d\} \leq_{SCC} \{e\} \tag{5.3}$$

Essentially, these relations capture that SCC^0_{AF} attacks SCC^1_{AF} and SCC^1_{AF} attacks SCC^2_{AF}. Since there is not a strongly connected component that attacks SCC^0_{AF}, SCC^0_{AF} is called *initial.*

Now, let us consider the representation of AF in terms of normal logic programs according to the mapping presented in Definition 5.6. We can see that $\mathcal{P}_{AF}$ is:

$$\begin{array}{lll} d(a) \leftarrow \; not\; d(b). & d(d) \leftarrow \; not\; d(a). & d(e) \leftarrow \; not\; d(d). \\ d(b) \leftarrow \; not\; d(c). & d(d) \leftarrow \; not\; d(b). & \\ d(c) \leftarrow \; not\; d(a). & d(d) \leftarrow \; not\; d(c). & \end{array}$$

Observe that this program induces the following classes of atoms:

$$\begin{array}{lll} [d(a)] = \{d(a), d(b), d(c)\} & [d(d)] = \{d(d)\} & [d(e)] = \{d(e)\} \\ [d(b)] = \{d(a), d(b), d(c)\} & & \\ [d(c)] = \{d(a), d(b), d(c)\} & & \end{array}$$

such that

$$\begin{array}{lll} d(a) \text{ is of order } 0 & d(d) \text{ is order } 1 & d(e) \text{ is of order } 2 \\ d(b) \text{ is of order } 0 & & \\ d(c) \text{ is of order } 0 & & \end{array}$$

This means that we have the following relations of atoms:

$$\{d(a), d(b), d(c)\} \leq_p \{d(d)\} \leq_p \{d(e)\} \tag{5.4}$$

Observe that we have very similar relations between the partial orders: $\leq_{SCC}$ (5.3) and $\leq_p$ (5.4). In Baroni *et al*'s approach, the partial order $\leq_{SCC}$ is used for splitting an argumentation framework in order to define a recursive construction of argumentation semantics. In our approach, the partial order $\leq_p$ is used for splitting a logic program in order to define a recursive construction of logic programming semantics. If the given logic program captures an argumentation framework AF (as it is done by $\mathcal{P}_{AF}$), $\leq_p$ will support the construction of an recursive construction of argumentation semantics as well.

In Baroni *et al*'s approach, once an argumentation framework is partitioned into its strongly connected components and the relation between them is defined, the possible choices for extensions within each *initial strongly connected component* are determined using *a basic argumentation semantics-specific function e.g.*, conflict-free sets, admissible sets, *etc.*, which returns the extensions of the argumentation frameworks consisting of a single strongly connected components. For instance, let us consider the initial strongly connected component ($SCC^0_{AF} = \{a, b, c\}$) of the argumentation framework AF of Figure 5.2. SCC^0_{AF} will define the following subargumentation framework:

$$AF_{SCC^0_{AF}} = \langle \{a, b, c\}, \{(a, c), (c, b), (b, a)\} \rangle$$

Now let us consider, as basic argumentation semantics-specific function, the function which returns all the possible conflict-free sets ($\mathcal{CF}$) of an argumentation framework. Hence we can see that $\mathcal{CF}(AF_{SCC^0_{AF}}) = \{\{a\}, \{b\}, \{c\}\}$.

Once a basic argumentation semantics-specific function S_{arg} was applied to all the initial strongly connected component of an argumentation framework, for each choice determined by S_{arg}

- the nodes directly attacked within subsequent strongly connected components are suppressed and
- a distinction between defended and undefended arguments is taken into account.

This changes are done according to the *reinstatement principle*. This basic principle prescribes that the arguments defeated by an extension E play no role in the selection of the arguments to be included in E (13).

In our approach, once the partial order $\leq_p$ is defined in terms of $\mathcal{L}_{\mathcal{P}_{AF}}$, the program $\mathcal{P}_{AF}$ is split into its components. For instance, $\mathcal{P}_{AF}$ is splint as follows:

P_0	P_1	P_2
$d(a) \leftarrow$ *not* $d(b)$.	$d(d) \leftarrow$ *not* $d(a)$.	$d(e) \leftarrow$ *not* $d(d)$.
$d(b) \leftarrow$ *not* $d(c)$.	$d(d) \leftarrow$ *not* $d(b)$.	
$d(c) \leftarrow$ *not* $d(a)$.	$d(d) \leftarrow$ *not* $d(c)$.	

Observe that P_0 can be constructed from $AF_{SCC^0_{AF}}$. In fact $P_0 = \mathcal{P}_{AF_{SCC^0_{AF}}}$. Let us consider the minimal model semantics MM as basic logic programming semantics function in order to illustrate the relations of our approach with Baroni *et al*'s approach. The basic logic programming semantics function is applied recursively as it is presented in Definition 5.3. Observe that

$$MM(P_0) = \{\{d(a), d(b)\}, \{d(b), d(c)\}, \{d(a), d(c)\}\}$$

therefore $\mathcal{F}(MM(P_0)) = \{\{a\}, \{b\}, \{c\}\}$. This means that

$$\mathcal{F}(MM(P_0)) = \mathcal{CF}(AF_{SCC^0_{AF}})$$

Like Baroni *et al*'s approach which reduces recursively the original argumentation framework by considering the results of the basic argumentation semantics-specific function, our approach applies a reduction to a logic program.

As we can see there are several key points that we can identified in common between Baroni *et al*'s approach and our approach, some them are:

1. Both approach consider an partial order for splitting the representation of an argumentation framework *i.e.* graph representation or logic program.

2. Both approach use a recursive definition for constructing the desired semantics (argumentation semantics / logic programming semantics).

A deep analysis is required in order to understand more about the relation between Baroni *et al*'s approach and our approach. In fact some interesting questions that we are going to consider in our future work are: Which is the class of argumentation semantics that can be characterized in both approaches? Which are the logic programming semantics that are more useful for building argumentation semantics? Which mappings of an argumentation framework into a logic program defines useful argumentation semantics?

With respect to the above questions, we know, by the moment, that the mapping of Definition 5.6 is a practical mapping that by considering the logic programming MM^{*^r} is able to suggest an argumentation semantics similar to CF2. It is worth to comment that CF2 is one of the most accepted argumentation semantics builded under the Baroni *et al*'s approach (12).

As final comment, we want to comment that any of the candidate argumentation semantics S_{arg} suggested under our approach can be explored their non-monotonic properties in terms of the logic programming semantics which induces S_{arg}. This is a relevant feature of our approach since many of the new argumentation semantics are only motivated by particular examples; hence, the identification of the non-monotonic reasoning properties, that a particular argumentation semantics satisfies, takes relevance in order to support the well-behaviour of an argumentation semantics.

5.6 Concluding remarks

Authors as Baroni *et al*, have suggested that in order to overcome Dung's abstract argumentation semantics problems, it is necessary to define flexible argumentation semantics which are not necessarily based on admissible sets (13). For instance, Baroni *et al*, have pointed out that in any admissibility-based semantics odd-length cycles admit only the empty extension. Based on the fact that logic programming offers a wide liberty for modeling knowledge, we can construct abstract argumentation semantics by specifying the basic conditions that our new argumentation semantics must satisfy. For instance, the mapping introduced in Definition 5.6 only captures the restriction that any argument will be defeated when anyone of its adversaries is not defeated. This single mapping is enough for characterizing argumentation semantics as the grounded semantics and the stable argumentation semantics. In fact, we also defined the abstract argumentation semantics $MM^{*^r}_{Arg}$ which is similar to CF2 — the semantics $MM^{*^r}_{Arg}$ is constructed under the mapping of Definition 5.6 and the logic programming semantics MM^{*^r}.

By considering the idea that argumentation semantics can be viewed as a special form of logic programming semantics with negation as failure, we introduce a general recursive approach for defining a new family of logic programming semantics which induce a new family of abstract argumentation semantics.

We have in mind that our new logic programming semantics satisfy the following suitable properties:

1. They should be always defined.
2. They should satisfy the relevance property.
3. They should agree with answer set semantics for the class of stratified programs.
4. They should be useful to model argumentation problems.

To explore the properties of the family of the abstract argumentation semantics which are induced by our approach is an issue for argumentation research. In fact, it is part of our future research. It is worth mentioning that thanks to the properties that the logic programming semantics hold, we can study the argumentation semantics that are constructed under these logic programming semantics *e.g.*, Lemma 5.8.

Chapter 6

A non-monotonic possibilistic-based argumentation approach

In this chapter, we introduce a possibilistic-based argumentation approach which is based on the possibilistic logic programming approach introduced in Chapter 3 and the argumentation semantics explored in Chapter 4 and Chapter 5.

6.1 Introduction

One of the purposes of argumentation theory is to provide tools for supporting decisions. For instance, argumentation theory is able to suggest arguments in favour a decision. Usually argumentation theory is adequate for supporting decisions in scenarios where the information is inconsistent and incomplete. Indeed, an interesting feature of argumentation theory inference is that it is able to manage inconsistent information in a natural way.

In Chapter 3, we defined an approach for modeling uncertain, incomplete and inconsistent information. In fact, in §3.4, we defined some criteria for dealing with two kinds of inconsistency of a possibilistic program

- one which arises from the presence of complementary atoms in a possibilistic answer set (or a possibilistic pstable model) and
- the other one which arises from the non-existence of possibilistic answer set (or possibilistic pstable models) of a possibilistic logic program.

In order to manage these problems of inconsistency, we define two different approaches in §3.4: One based on a preference criterion between possibilistic atoms and the other one based on cuts as it is done Possibilistic Logic.

As we saw in §3.4, many times to consider inconsistent information is often the only way to explore inconsistent information without arbitrarily rejecting precious data. Also pursuing inconsistent systems is sometimes the only way to obtain new information. As a result, pursuing inconsistent belief systems allows us to make better informed decisions regarding which bits of information to accept or reject in the end.

In order to extend our approach presented in Chapter 3, in this chapter, we will define a possibilistic-based argumentation approach. This approach will be based on the concept of *possibilistic argument* and the argumentation semantics explored in Chapter 4 and Chapter 5.

To consider arguments in order to support conclusions provides a natural process for supporting decision-making in possibilistic knowledge bases. For instance, one can require to support/justify each of the possibilistic atoms which belongs to a possibilistic answer set (or possibilistic pstable models). For example, let us consider the program P of Section 3.4:

probable: $r\_inf(present, 1) \vee no\_r\_inf(present, 1) \leftarrow action(transplant, 0),$ $d\_inf(present, 0).$
confirmed: $o(good\_graft\_funct, 1) \vee o(delayed\_graft\_funct, 1) \vee$ $o(terminal\_insufficient\_funct, 1) \leftarrow action(transplant, 0).$
confirmed: $action(transplant, 0) \leftarrow o(terminal\_insufficient\_funct, 0).$
plausible: $cs(stable, 1) \leftarrow o(good\_graft\_funct, 1).$
plausible: $cs(unstable, 1) \leftarrow o(delayed\_graft\_funct, 1).$
plausible: $cs(0\text{-urgency}, 1) \leftarrow o(terminal\_insufficient\_funct, 1),$ $action(transplant, 0).$
certain: $\perp \leftarrow action(transplant, 0), action(wait, 0).$
certain: $\perp \leftarrow action(transplant, 0), cs(dead, 0).$
certain: $d\_inf(present, 0) \leftarrow \top.$
certain: $no\_r\_inf(present, 0) \leftarrow \top.$
certain: $o(terminal\_insufficient\_funct, 0) \leftarrow \top.$
certain: $cs(stable, 0) \leftarrow \top.$
confirmed: $v(kidney, 0) \leftarrow cs(stable, 1), action(transplant, 0).$
probable: $no\_v(kidney, 0) \leftarrow r\_inf(present, 1), action(transplant, 0).$
certain: $\perp \leftarrow \ not\ cs(stable, 1).$

Remember that this program captures the possible situations where an organ recipient can be found after a graft. In particular, we are considering a donor's organ with a possible infection (see the introduction of Chapter 3.1 and Example 3.1 for a full description).

Now based on the knowledge base of the program P, we want to know if a kidney from a donor with an infection is viable for transplanting. In Section 3.4, we saw that this program has two possibilistic answer sets:

$$\begin{aligned} S_1 := \quad & \{(d\_inf(present, 0), certain),\ (no\_r\_inf(present, 0), certain), \\ & (o(terminal\_insufficient\_funct, 0), certain),\ (cs(stable, 0), certain), \\ & (action(transplant, 0), confirmed),\ (o(good\_graft\_funct, 1), confirmed), \\ & \mathbf{(cs(stable,1),plausible)},\ \mathbf{(no\_r\_inf(present,1),\ probable)}, \\ & \mathbf{(v(kidney,0),\ plausible)}\} \end{aligned}$$

$$\begin{aligned} S_2 := \quad & \{(d\_inf(present, 0), certain),(no\_r\_inf(present, 0), certain), \\ & (o(terminal\_insufficient\_funct, 0), certain),(cs(stable, 0), certain), \\ & (action(transplant, 0), confirmed),\ (o(good\_graft\_funct, 1), confirmed), \\ & \mathbf{(cs(stable,1),\ plausible)},\mathbf{(r\_inf(present,1),\ probable)}, \\ & \mathbf{(v(kidney,0),\ plausible)},\ \mathbf{(no\_v(kidney,0),\ probable)}\} \end{aligned}$$

By considering the possibilistic answer set S_1, we can see that there is a possibilistic atom $(v(kidney, 0), plausible)$ that suggests that the kidney is viable for transplanting, but a good question is: *what is it the information supporting this conclusion?* By definition of possibilistic answer set (see Definition 3.3), we know that $P^{S_1} \vdash_{PL} (v(kidney, 0)\ plausible)$. This means that there is a set of possibilistic clauses $Support \subseteq P^{S_1^*}$, such that $Support \vdash_{PL} (v(kidney, 0)\ plausible)$. In fact, we can restrict to $Support$ in order to be minimal *w.r.t.* set inclusion. This ideas of considering the minimal information in order to support a conclusion is captured by the concept of argument in argumentation theory. By considering a standard form of an argument, in argumentation literature (1; 64), we can say that the following structure is an argument that suggests that the kidney is plausible to be transplanted:

$$\begin{aligned} & Arg_1 = \langle v(kidney, 0), \\ & \\ & \{certain : o(terminal\_insufficient\_funct, 0) \leftarrow \top; \\ & confirmed : action(transplant, 0) \leftarrow o(terminal\_insufficient\_funct, 0); \\ & confirmed : o(good\_graft\_funct, 1) \vee o(delayed\_graft\_funct, 1) \vee \\ & \qquad\qquad\quad o(terminal\_insufficient\_funct, 1) \leftarrow action(transplant, 0); \\ & plausible : cs(stable, 1) \leftarrow o(good\_graft\_funct, 1); \\ & confirmed : v(kidney, 0) \leftarrow cs(stable, 1), action(transplant, 0); \\ & certain : \bot \leftarrow\ not\ cs(stable, 1)\}, \\ & \\ & plausible\ \rangle \end{aligned}$$

Essentially, one can identify three components in the structure of this argument: *a conclusion*, *a support* and *a degree of uncertainty*. Observe that when a possibilistic answer set is inconsistent, as in S_2, one can construct arguments which attack each other. For instance, by considering S_2 one cannot only construct an argument in favor to consider the kidney viable to be transplanted; but it is also possible to construct an arguments with oppositive conclusion:

$$
\begin{array}{l}
Arg_2 = \langle no\_v(kidney, 0), \\
\\
\{certain : o(terminal\_insufficient\_funct, 0) \leftarrow \top; \\
certain : d\_inf(present, 0) \leftarrow \top; \\
confirmed : action(transplant, 0) \leftarrow o(terminal\_insufficient\_funct, 0); \\
probable : r\_inf(present, 1) \vee no\_r\_inf(present, 1) \leftarrow action(transplant, 0), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad d\_inf(present, 0). \\
probable : no\_v(kidney, 0) \leftarrow r\_inf(present, 1), action(transplant, 0) \\
certain : \bot \leftarrow \; not \; cs(stable, 1)\}, \\
\\
probable \; \rangle
\end{array}
$$

In this case, we have to identify some criteria for selecting possibilistic arguments which are in conflict.

As we saw, in Chapter 4, the selection of arguments can be done by considering special patterns of selection of arguments *i.e.* the Dung's argumentation semantics and the new argumentation semantics presented in Chapter 4 and Chapter 5. However, the selection of arguments is just one of the steps in the inference in argumentation theory.

The inference in argumentation theory usually can be regarded by 4 steps (48):

1. Argument construction. This argument construction is based on a knowledge base. In our case, we will consider possibilistic disjunctive logic programs.

2. Argument valuation. In this step, it is assigned a weight to arguments. This weight will be assigned by possibilistic answer sets/possibilistic pstable models in our approach.

3. Argumentation interaction. In this step, the conflicts between the arguments are identified.

4. Argumentation status evaluation. For determining the winning or justified arguments.

In this chapter, we will formalize these 4 steps in the context of our possibilistic logic programming approach presented in Chapter 3. Essentially, by considering a possibilistic logic program, we will define a strategy for building possibilistic arguments. A possibilistic argument will have a standard structure (as in argumentation theory) and will be inferred by possibilistic logic programming semantics, as the possibilistic answer set semantics or the possibilistic pstable semantics, of the given possibilistic knowledge base.

In order to manage the conflict that could appear between possibilistic arguments, we will instantiate the argumentation framework structure introduce by Dung in terms of possibilistic arguments. This instantiation will allow us to use the argumentation semantics presented in Chapter 4 and Chapter 5 in order to infer justified arguments.

The rest of this chapter is divided as follows: In §6.2, we will start presenting the definition of a possibilistic argument. After that, in §6.3, we formalize how to manage the conflicts between possibilistic arguments. In the last section of this chapter, we present our concluding remarks.

6.2 Building possibilistic arguments

As we commented in the previous section, the first step in the inference in argumentation theory is the construction of arguments. Hence, in this section, we shall start by defining how to build possibilistic arguments from a possibilistic program.

A possibilistic argument can be constructed by considering any possibilistic logic programming semantics *i.e.* the possibilistic answer set semantics, the possibilistic pstable semantics. Since one can consider the skeptical and credulous versions of possibilistic semantics as the possibilistic answer set semantics and the possibilistic pstable semantics, we will define two kinds of possibilistic arguments: *brave possibilistic arguments* and *cautious possibilistic arguments*.

Definition 6.1 (Possibilistic Arguments) *Let $P = \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic logic program. A possibilistic argument Arg* w.r.t. *P is a tuple of the form $Arg = \langle Claim, Support, \alpha\rangle$ such that the following conditions hold:*

1. *$Support \subseteq P$.*
2. *$Support$ is minimal* w.r.t. *set inclusion.*
3. *$\exists\ M \in PSEM(Support)$ such that $(Claim, \alpha) \in M$ (in this case the possibilistic arguments Arg is called* brave. *When the existent quantified $\exists$ is changed by the for all quantified $\forall$, the possibilistic arguments Arg is called* cautious*).*

$PSEM$ is any possibilistic logic programming semantics i.e. *possibilistic stable semantics, possibilistic answer set semantics, possibilistic pstable semantics. Brave-$\mathcal{ARG}_P^{PSEM}$ gathers all the brave possibilistic arguments which can be constructed from P and the possibilistic logic programming semantics $PSEM$. Cautious-$\mathcal{ARG}_P^{PSEM}$ gathers all the cautious possibilistic arguments which can be constructed from P and the possibilistic logic programming semantics $PSEM$.*

In order to simplify the following definition, $\mathcal{ARG}_P$ will denote any set of possibilistic arguments constructed from P.

The arguments Arg_1 and Arg_2 presented in the previous section are two examples of two brave possibilistic arguments *w.r.t.* the possibilistic answer set semantics. In order to illustrate some features of this definition, let us consider the following two programs:

$$\begin{array}{llll} P_1: & 0.5 : a \leftarrow \top. & P_2: & 0.5 : a \leftarrow \top. \\ & 0.6 : b \leftarrow \neg a & & 0.6 : b \leftarrow \; not\ a \end{array}$$

First of all, observe that the only difference between P_1 and P_2 is the kind of negation that is used in the second clause of each program. In P_1, the atom a is negated by the strong negation ($\neg$) and in P_2, the atom a is negated by the *negation as failure* (*not*). This simple syntactic difference makes strong differences *w.r.t.* the semantics of each program, and therefore to the set of possibilistic arguments which can be inferred from each program. It is easy to see that in both programs P_1 and P_1, one can construct an argument for the atom a which is:

$$Arg_a = \langle a, \{0.5 : a \leftarrow \top\}, 0.5\rangle$$

Now the question is: is there an argument for the atom b from P_1 and P_2? On the one hand, observe that there does not exist a subprogram P_1' of P_1 such that $P_1' \vdash_{PL} b$, this means that one cannot build an possibilistic argument for the atom b from the program P_1. On the other hand, by considering the subprogram $P_2' \subseteq P_2$ such that $P_2' = \{0.6 : b \leftarrow \; not\ a\}$, one can easily see that any reasonable possibilistic argumentation semantics $PSEM$ that captures negation as failure will inter that $\exists M \in PSEM(P_2')$ such that $(b, 0.6) \in M$. This means that by considering any reasonable possibilistic semantic $PSEM$, one can construct the possibilistic argument:

$$Arg_b = \langle b, \{0.6 : b \leftarrow \; not\ a\}, 0.6\rangle$$

from the program P_2.

Before to follow on, we want to point out that whenever a possibilistic program P is positive *i.e.* it has no negative literals. The construction of possibilistic arguments will depend on the inference of Possibilistic Logic. Hence, in this case we can ensure that $Brave\text{-}\mathcal{ARG}_P^{PSEM} = Cautious\text{-}\mathcal{ARG}_P^{PSEM}$.

6.3 Conflicts between possibilistic arguments

Once we have defined how to build possibilistic arguments, we require to define how the possibilistic arguments will interact. In other words, we will define the cases when two possibilistic arguments will be in a conflict and then to define which arguments will be considered accepted (according to an argumentation semantics).

Usually the relation of *attack* between arguments is defined in terms of complementary atoms *i.e.* a and $\neg a$. For instance, if we have the possibilistic program:

$$\begin{array}{ll} P: & 0.6 : a \leftarrow \top. \\ & 0.5 : \neg a \leftarrow \top. \end{array}$$

one can construct two possibilistic arguments: $Arg_1 = \langle a, \{0.6 : a \leftarrow \top\}, 0.6\rangle$ and $Arg_2 = \langle \neg a, \{0.5 : \neg a \leftarrow \top\}, 0.5\rangle$. It is clear that both arguments Arg_1 and Arg_2 are in a conflict since Arg_1 and Arg_2 have complementary conclusions. In this case, we will say that the arguments Arg_1 and Arg_2 attack each other. This kind of relations are standard in argumentation theory. Since Arg_1 has a possibilistic degree greatest than Arg_2's possibilistic degree, Arg_1 will be preferred than Arg_2. Hence, in order to define which argument can be considered accepted according to an argumentation semantics, we will can *omit* that Arg_2 attacks Arg_1.

But now, how will a possibilistic argument be affected by the presence of negative literals in its support in the interaction with other possibilistic arguments? Let us consider the arguments Arg_a and Arg_b constructed in the previous section *w.r.t.* the possibilistic program P_2. We can see that Arg_b has as conclusion the atom b and as support the possibilistic program:

$$0.6 : b \leftarrow \ not\ a$$

This clause suggests that b is inferred if *there does not exist evidence* of a. However, there is a possibilistic argument Arg_a which suggests that *there exists evidence* to believe in a. Hence, even though Arg_a has a possibilistic degree lower than the Arg_b's possibilistic degree, we cannot say that Arg_b is preferred than Arg_a. Therefore, by the fact that *negation as failure* captures no evidence, we will assume that Arg_a *attacks* Arg_b.

It is worth to remember that the clause $0.6 : b \leftarrow \ not\ a$ has a different intended meaning of the clause:

$$0.6 : b \leftarrow \neg a$$

In this case, one intended meaning of this clauses is that b is inferred if $\neg a$ is true, in other words there is a possibilistic theory Γ such that $\Gamma \vdash_{PL} (\neg a, \alpha)$.

By having in mind the previous ideas, we will define the relation of *attack* between possibilistic arguments.

Definition 6.2 *Let Arg_1 and Arg_2 be two possibilistic arguments such that $Arg_1 = \langle Claim_1,\ Support_1,\ \alpha_1\rangle$ and $Arg_2 = \langle Claim_2,\ Support_2,\ \alpha_2\rangle$. We say that Arg_1 attacks Arg_2 if one of the following conditions hold:*

i) *$Claim_1 = l$, $Claim_2 = \widetilde{l}$ and $\alpha_1 \geq \alpha_2$.*

ii) *$\exists(q : l \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-) \in Support_2$ such that $\widetilde{Claim_1} \in \mathcal{B}^+$ and $\alpha_1 \geq \alpha_2$.*

iii) *$\exists(q : l \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-) \in Support_2$ and $Claim_1 \in \mathcal{B}^-$.*

In order to illustrate a little bit more this definition, let us consider the following example.

Example 6.1 *Let P be the following possibilistic program:*

$$\begin{array}{ll} 0.1 : b \leftarrow \top. & 0.8 : \neg b \leftarrow \top. \\ 0.9 : a \leftarrow \ not\ b. & 0.5 : c \leftarrow \neg a. \end{array}$$

As we can see, one can construct four possibilistic arguments by considering any reasonable possibilistic argumentation semantics:

$$\begin{array}{rl} Arg_1 = & \langle b, \{0.1 : b \leftarrow \top\}, 0.1\rangle \\ Arg_2 = & \langle \neg b, \{0.8 : \neg b \leftarrow \top\}, 0.8\rangle \\ Arg_3 = & \langle a, \{0.9 : a \leftarrow \ not\ b\}, 0.9\rangle \\ Arg_4 = & \langle c, \{0.5 : c \leftarrow \neg a\}, 0.5\rangle \end{array}$$

By instantiating Definition 6.2, one identify the following conflicts between these possibilistic arguments:

Arg_2 attacks Arg_1 by condition i).
Arg_3 attacks Arg_4 by condition ii).
Arg_1 attacks Arg_3 by condition iii).

Observe that Arg_1 does not attack Arg_2 because Arg_1's possibilistic degree is less than Arg_2's possibilistic degree.

In order to define which possibilistic arguments will be considered accepted according to an argumentation semantics, we will use Dung's argumentation style. Therefore, we will define the basic concept of *possibilistic argumentation framework* in terms of Dung's argumentation framework.

Definition 6.3 (Possibilistic Argumentation Framework) *Given a possibilistic logic program, a possibilistic argumentation framework AF* w.r.t. *P is the tuple $AF = \langle \mathcal{ARG}_P,\ Attacks \rangle$, where attacks contains the relations of attack between the arguments of $\mathcal{ARG}_P$.*

Observe that essentially, we are instantiating the Dung's argumentation approach into possibilistic arguments. Hence, we can take advantage of all the results introduced in Chapter 4 and Chapter 5 in order to manage the interaction of possibilistic arguments.

In order to illustrate the application of argumentation semantics into possibilistic argumentation frameworks, let us consider the following examples.

Example 6.1 *Let us continue with Example 6.1. As we saw, we have the following relations: Arg_2 attacks Arg_1, Arg_3 attacks Arg_4, Arg_1 attacks Arg_3. Hence, we have the following possibilistic argumentation framework:*

$$AF = \langle \{Arg_1, Arg_2, Arg_3\}, \{(Arg_2, Arg_1), (Arg_3, Arg_4), (Arg_1, Arg_3)\} \rangle$$

It is easy to see that AF has an unique preferred extensions which is: $\{Arg_2, Arg_3\}$. Hence, we can consider as accepted arguments:

$$\begin{array}{ll} Arg_2 = & \langle \neg b, \{0.8 : \neg b \leftarrow \top\}, 0.8 \rangle \\ Arg_3 = & \langle a, \{0.9 : a \leftarrow \ not\ b\}, 0.9 \rangle \end{array}$$

This set of possibilistic arguments suggests that one can infer from the possibilistic program P the set of possibilistic atoms: $\{(\neg b, 0.8), (a, 0.9)\}$. Observe that the only possibilistic answer set of P is the inconsistent set: $\{(b, 0.1), (\neg b, 0.8)\}$.

Now, let us consider another example.

Example 6.2 *Let $P = \langle (\mathcal{Q}, \leq), N \rangle$ be the following possibilistic logic program:*

$0.5 : a \leftarrow \ not\ b.$
$0.5 : b \leftarrow \ not\ a.$
$0.6 : p \leftarrow \ not\ p.$
$0.7 : p \leftarrow \ not\ a.$

By considering the possibilistic answer set semantics (PASP), we can see that:

$$\begin{array}{lll} Brave\text{-}\mathcal{ARG}_P^{PASP} = \{ & Arg_1 = \langle a, \{0.5 : a \leftarrow \ not\ b\}, 0.5 \rangle, & \\ & Arg_2 = \langle b, \{0.5 : b \leftarrow \ not\ a\}, 0.5 \rangle, & \\ & Arg_3 = \langle p, \{0.7 : p \leftarrow \ not\ a\}, 0.7 \rangle & \} \end{array}$$

Hence, the possibilistic argumentation framework AF_P w.r.t. *$Brave\text{-}\mathcal{ARG}_P^{PASP}$ is $\langle \{Arg_1, Arg_2, Arg_3\}, \{(Arg_1, Arg_2), (Arg_2, Arg_1), (Arg_1, Arg_3)\} \rangle$. Once we have* *constructed the possibilistic argumentation frame AF_P, we can apply to AF_P any argumentation semantics in order to infer extensions (sets) of possibilistic arguments from AF_P. For example, we can see that AF_P has tow stable extensions which are: $\{Arg_2, Arg_3\}$ and $\{Arg_1\}$. In this example, the stable and preferred semantics coincide.*

Observe that the two stable extensions of the possibilistic argumentation framework AF_P of Example 6.2 suggest that one can infer the following two sets of possibilistic atoms: $\{(b, 0.5), (p, 0.7)\}$, $\{(a, 0.5)\}$ from the possibilistic program P. It is worth to mentioning, that the only possibilistic answer of the possibilistic program P is: $\{(b, 0.5), (p, 0.7)\}$. However, one can see that $\{(a, 0.5)\}$ can be also inferred from P as it is done by our possibilistic-based argumentation approach.

An other interesting point of considering our possibilistic-based argumentation approach is that it could help to manege inconsistent possibilistic programs *w.r.t.* the non-existent of possibilistic answer sets (or possibilistic pstable models) of a possibilistic program. For instance, we saw that the program P_{inc}:

$$\begin{array}{ll} 0.3: & a \leftarrow \ not\ b. \\ 0.5: & b \leftarrow \ not\ c. \\ 0.6: & c \leftarrow \ not\ a. \end{array}$$

presented in §3.4 has neither possibilistic answer set nor possibilistic pstable models. However, one can see that by considering either the possibilistic answer set semantics or the possibilistic pstable semantics, we have the following set of possibilistic brave possibilistic arguments:

$$\begin{array}{lll} Brave\text{-}\mathcal{ARG}_{P_{inc}} = \{ & Arg_1 = \langle a, \{0.3 : a \leftarrow \ not\ b\}, 0.3\rangle, & \\ & Arg_2 = \langle b, \{0.5 : b \leftarrow \ not\ c\}, 0.5\rangle, & \\ & Arg_3 = \langle c, \{0.6 : c \leftarrow \ not\ a\}, 0.6\rangle & \} \end{array}$$

Hence, we can define the following possibilistic argumentation framework $AF_{P_{inc}} = \langle\{Arg_1, Arg_2, Arg_3\}, \{(Arg_1, Arg_3), (Arg_3, Arg_2), (Arg_2, Arg_1)\}$. It is easy to see that $AF_{P_{inc}}$ has no stable extensions, the only preferred extension of $AF_{P_{inc}}$ is empty and the grounded extension of $AF_{P_{inc}}$ is empty as well. However, if we consider the argumentation semantics $MM^{*^r}_{Arg}$ presented in §5.4 (or the argumentation semantics CF2 (13)), we can see that $MM^{*^r}_{Arg}(AF_{P_{inc}})$ has three extensions: $\{\{Arg_1\}, \{Arg_2\}, \{Arg_3\}\}$. This means that $MM^{*^r}_{Arg}$ is suggesting that one can infer the following three sets of possibilistic atoms from P_{inc}:$\{(a, 0.3)\}$, $\{(b, 0.5)\}$ and $\{(c, 0.6)\}$. Remember in §3.4, by considering the consistency cut degree of P_{inc}, we could infer $\{(c, 0.6)\}$ as possibilistic answer set of $P_{ConsCutDeg(P_{inc})}$.

6.4 Some Properties

In this section, we identify some properties *w.r.t.* our possibilistic-based argumentation approach.

Since the possibilistic arguments are based on possibilistic models (as possibilistic answer sets or possibilistic pstable models), the possibilistic arguments satisfy some properties which are inherited of Possibilistic Logic. Possibilistic logic has a basic principle that is:

> The strength of a conclusion is the strength of the weakest argument used in its proof.

According to Dubois and Prade (43), the contribution of possibilistic logic setting is to relate this principle (measuring the validity of an inference chain by its weakest link) to fuzzy set-based necessity measures in the framework of Zadeh's possibilistic theory.

This basic principle of possibilistic logic will give an interesting property to any possibilistic argument: The strength of a possibilistic argument $Arg = \langle Claim, Support, q\rangle$ will be the strength of the weakest possibilistic clause of $Support$.

In order to prove this property, let us introduce the following lemma:

Lemma 6.1 *Let P be a possibilistic logic program and $Arg = \langle Claim, Support, q\rangle$ be a possibilistic argument* w.r.t. *P. Then*

$$Support \subseteq P_q$$

<u>Proof</u>: The result follows from the followings two observations:

1. By Definition 6.1, we know that $Support^M \vdash_{PL} (Claim, q)$ and $Support$ is minimal *w.r.t.* set inclusion.

2. By Proposition 11 of (42), it is true that $\Gamma \vdash_{PL} (Claim, q)$ if and only if $\Gamma_q \vdash_{PL} (Claim, q)$.

■

By considering this lemma, we formalize the following result.

Proposition 6.1 (Weakest link) *Let P be a possibilistic logic program an $Arg = \langle Claim, Support, q\rangle$ be a possibilistic argument* w.r.t. *P. Then*

$$q = \mathcal{GLB}\{n(r) | r \in Support\}$$

Proof: The result is direct by Lemma 6.1. ■

The property of *the weakest link* has been discussed by some authors (5; 7) as an important property because it could help

- to allow an agent to compare different arguments in order to select the best one and
- to determine the acceptable arguments among the conflicting ones.

An important property of the possibilistic-based argumentation inference is that it will infer consistent sets of possibilistic atoms from any possibilistic knowledge base. For instance, by considering Dung's argumentation semantics we can ensure the following theorem

Theorem 6.1 (Consistency Information) *Let $AF = \langle \mathcal{ARG}_P, Attacks \rangle$ be a possibilistic argumentation framework and $S \subseteq \mathcal{ARG}$. If S is either a preferred extension, a stable extension or the grounded extension of AF, then the following condition holds: If $Cs = \{Claim | \langle Claim, Support, \alpha \rangle \in S\}$, then Cs is a consistent set of literals.*

Proof: It is straightforward by the fact that any admissible set is a conflict-free set. ■

6.5 A Possibilistic Argumentation Engine

As we know, one of the main objectives of argumentation theory is to explore mechanics in order to implement argumentation engines into argumentation systems. These argumentation systems can be intelligence systems as agents, a tool for supporting dialogue or a tool for supporting decision making based on argumentation reasoning.

So far this thesis, we have presented results *w.r.t*:

- how to modeling uncertain, incomplete and inconsistent information and
- how to characterize argumentation semantics as the preferred semantics in order to be implemented into argumentation systems.

In fact, in this chapter, we have defined an possibilistic-based argumentation inference which is based on possibilistic knowledge bases, possibilistic logic programming semantics, and argumentation semantics.

In order to outline that by considering some of the results presented in thesis can be implemented into real argumentation systems. In Figure 6.5, we present a general architecture of a possibilistic argumentation engine. This architecture has three main components for storing data:

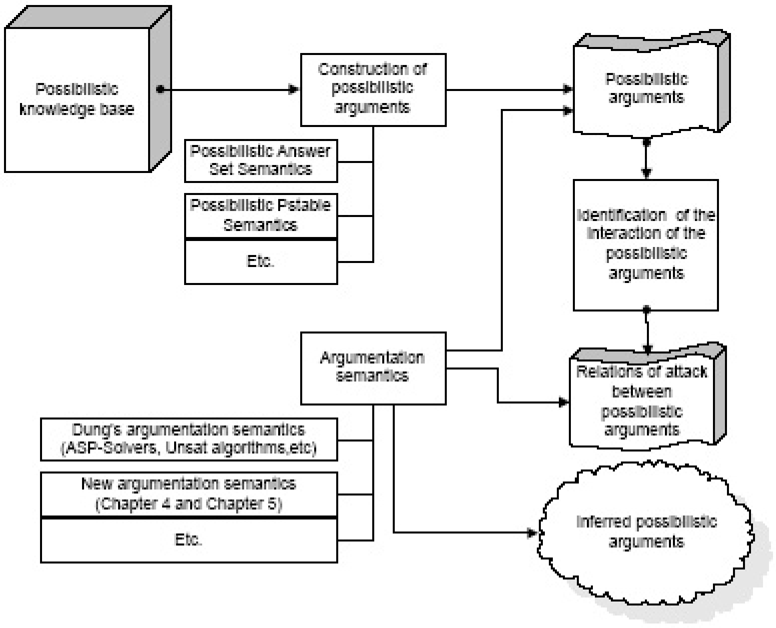
Possibilistic knowledge base
Construction of possibilistic arguments
Possibilistic arguments
Possibilistic Answer Set Semantics
Possibilistic Pstable Semantics
Etc.
Identification of the interaction of the possibilistic arguments
Argumentation semantics
Relations of attack between possibilistic arguments
Dung's argumentation semantics (ASP-Solvers, Unsat algorithms,etc)
New argumentation semantics (Chapter 4 and Chapter 5)
Etc.
Inferred possibilistic arguments

Figure 6.1: A Possibilistic Argumentation Engine Architecture.

- Possibilistic knowledge base (PK-Storing data),
- Possibilistic arguments (PA-Storing data) and
- Relations of attack between possibilistic arguments (RA-Storing data).

Obviously the initial data will be a possibilistic logic program which will be into PKB-Storing data. PA-Storing data and RA-Storing data have information which is inferred from PK-Storing data. PA-Storing data and RA-Storing data are updated each time that PK-Storing data is updated.

In the possibilistic argumentation engine architecture, there are three main functions:

- The construction of possibilistic arguments,
- the identification of the interaction (attacks) of the possibilistic arguments, and
- the inference of accepted arguments.

From these functions, the construction of the possibilistic arguments and the inferences of accepted arguments can be considered as dynamic functions in the sense that they can change their behavior by changing the possibilistic logic programming semantics and the argumentation semantics respectively.

Observe that the function which construct possibilistic arguments can be implemented as a frontend of an answer set solver (40; 103) or a pstable solver (69). Remember that the algorithms that we suggested for inferring possibilistic answer sets and possibilistic pstable models are based on answer set solvers and pstable solvers respectively.

As a final comment, observe that if we consider the Dung's argumentation semantics for inferring accepted arguments (as the stable, preferred and grounded semantics), we can implemented as a frontend of answer set solvers as DLV (40). In fact, for the case of the preferred semantics, it can be implemented by using some UNSAT solver.

6.6 Related work

In §3.5, we have already commented the related work *w.r.t.* the possibilistic logic programming approach used in the possibilistic-based argumentation approach defined in this chapter. Therefore, in this section, we will only concentrate our attention to possibilistic approaches in the context of argumentation theory.

Maybe the two main representative argumentation approaches which are close related to our approach are the proposals of Leila and Prade in (6) and Alsinet *et al*in (2).

In the proposal of Leila and Prade, they define an argumentation approach for supporting decision making. The specification language is based on the syntax of possibilistic logic. Unlike to our possibilistic arguments which are based on the inference of possibilistic logic (for the case of positive possibilistic programs[1]), in Leila-Prade's approach, the arguments are based on the inference of classical logic. Indeed, the certain level of each argument is inferred by a secondary definition to argumentation's definition. In Leila-Prade's approach, the inference over conflicting arguments is managed essentially by the grounded semantics.

The proposal of Alsinet *et al*, in (2), is an argumentation approach which combine features from argumentation theory and logic programming (without negation as failure). The specification language of this approach is based on a possibilistic logic programming approach. It is worth mentioning that this possibilistic logic programming approach is defined over the possibilistic *Gödel* logic (3). Indeed, the construction of arguments in this approach is based on the

[1]Remember that a positive possibilistic program is a program which does not have clauses with literals negated by negation as failure.

inference of the possibilistic *Gödel* logic. The inference over conflicting arguments is based on a dialectical analysis which defines a skeptical reasoning process.

6.7 Concluding remarks

In this chapter, we defined a possibilistic-based argumentation approach based on:

- the inference of possibilistic logic programming semantics and
- Dung's argumentation semantics style.

This approach inherits all the expressiveness of the possibilistic disjunctive logic programs (presented in Chapter 3) and offers some natural mechanisms for dealing with reasoning under inconsistent information. In fact, this approach does not requite to apply cuts to an inconsistent possibilistic knowledge base, as it is done in possibilistic logic programming, in order to manage the non-existence of possibilistic models. Another interesting property of our approach is that any set of possibilistic atoms inferred by the possibilistic-based argumentation inference will be consistent.

We also outline a possibilistic argumentation engine architecture in order to outline that by considering the results presented in thesis, we can implement real argumentation systems. In fact this architecture is a good picture where theoretical results in argumentation theory and argumentation systems can converge.

Chapter 7

Discussion

In this final chapter, we present a summary of the work carried out in this thesis. We will start given an overview of this thesis. After that, we comment the major contributions presented in this research.

7.1 Thesis overview

One of our first objectives of this thesis was to explore a specification language for modeling medical information in order to support medical decision-making. To support medical decision-making is often complicated by the need to integrate uncertain, incomplete and potentially conflicting information from several sources (65). There are researchers in medical decision-making (47; 48) who have pointed out that clinicians frequently have difficulty acquiring or estimating the numbers required to model decisions *e.g.*, probabilities. Hence based on this evidence and the fact that probability has some problems for performing symbolic reasoning (§1.1), we decided to explore a non-probabilistic approach for capturing uncertain information. Moreover since there are also cognitive researchers (59) who have observed that people usually appeal to their experience or commonsense for supporting their decisions, we decided to explore a specification language able to perform reasoning under non-numerical values of uncertainty.

In Chapter 3, we defined a possibilistic disjunctive logic programming approach. This approach introduces the use of possibilistic disjunctive clauses which are able to capture *incomplete information* and *incomplete states of a knowledge base* at the same time (§3.2). Based on the flexibility of possibilistic logic for defining degrees of uncertainty, we have illustrated in Chapter 3 that it is possible to consider non-numerical degrees for capturing uncertain information. In particular, we have discussed the use of non-numerical degrees of uncertainty in a medical scenario.

In order to define the semantics of a possibilistic disjunctive logic program, we define three approaches:

- the first is strictly close to the proof theory of possibilistic logic and *answer set models* (§3.3.1);
- the second is based on *partial evaluation*, a fix-point operator and *answer set models* (§3.3.2); and
- the last is also based on the proof theory of possibilistic logic and *pstable semantics* (§3.3.3).

As many times it is difficult or impossible to keep away of managing inconsistent knowledge bases, we also explore some criteria for managing inconsistent possibilistic logic programs (§3.4). In particular, we define a preference criterion between inconsistent possibilistic models and adopt the approach of cuts for restoring consistency of an inconsistent possibilistic knowledge base.

Strictly speaking, we can accept that when we consider *non-numerical values* for capturing uncertain information, our possibilistic clauses are not at all possibilistic formulæ in the sense of possibilistic logic. However, since our possibilistic semantics are based on the proof theory of possibilistic logic, we can insure that our approach satisfies the basic principle of possibilistic logic that is:

> *The strength of a conclusion is the strength of the weakest argument used in its proof.*

In fact, based on this principle we define possibilistic arguments as base of a non-monotonic possibilistic-based argumentation approach (see Chapter 6).

Once we defined a specification language for capturing uncertain information, we decided to explore another important step in the decision-making process which is how to choose one decision over anther and how to explain a decision. Some researchers (16; 30; 47; 48; 99) argues that from a practical perspective, argumentation provides a versatile computation model for developing advanced services for decision support and for computer human dialogues in which explanation and rationale play role.

In Chapter 4, we explored the well-known abstract argumentation semantics which have as main function to find the set of arguments which are considered as acceptable. In particular, we explore *how* to model abstract argumentation semantics from a point of view of non-monotonic logic programming semantics. In order to consider an argumentation framework as a logic program, we define some basic conditions for studying abstract argumentation semantics which are based on admissible sets. In fact, we introduce the concept of *suitable codification* (§4.2). By considering a suitable codification (§4.3,§4.4), we present:

- a study of the preferred semantics in terms of minimal models and answer set models —this study will suggest some practical methods for implementing the preferred semantics (§4.5) — and
- a study of the grounded semantics in order to define some intermediate argumentation semantics between the grounded and the preferred semantics (§4.6).

An interesting point of our study of the grounded semantics is that we outline an approach for describing the interaction of arguments based on rewriting systems and our suitable codification (§4.6.2).

Despite the abstract argumentation semantics built in terms of admissible sets are well-accepted (Dung's approach §2.6), these semantics exhibit a variety of problems which have illustrated in literature (§1.2). In the process of studying the Dung's abstract argumentation semantics in terms of logic programming semantics, we could recognize that one can define new abstract argumentation semantics in terms of logic programming semantics. By considering the idea that argumentation semantics can be viewed as a special form of logic programming semantics with *negation as failure*, we introduce a general recursive approach for defining a new family of logic programming semantics which induces a new family of abstract argumentation semantics (Chapter 5). We have in mind that our new logic programming semantics satisfy the following suitable properties (§5.3):

1. They should be always defined.
2. They should satisfy the relevance property.
3. They should agree with answer set semantics for the class of stratified programs.
4. They should be useful to model argumentation problems.

Based on the fact that logic programming offers a wide liberty for modeling knowledge, we can construct abstract argumentation semantics by specifying the basic conditions that our new argumentation semantics must satisfy. For instance, the mapping introduced in Definition 5.6 only captures the restriction that any argument will be defeated when anyone of its adversaries is not defeated. This single mapping is enough for characterizing argumentation semantics as the grounded semantics and the stable argumentation semantics. In fact, we also defined the abstract argumentation semantics $MM^{*^r}_{Arg}$ which is similar to CF2 — the semantics $MM^{*^r}_{Arg}$ is constructed under the mapping of Definition 5.6 and the logic programming semantics MM^{*^r} (§5.4).

In Chapter 6, we defined an approach for building possibilistic arguments from possibilistic logic programs. We argue that this argumentation approach is adequate for supporting decision-making in scenarios where the information is uncertain and incomplete. One of the motivations for defining this possibilistic argumentation approach is that once it is inferred the possibilistic answer set (or the possibilistic pstable models) of a possibilistic knowledge base, one can require to support/justify each of the possibilistic atoms which belongs to a possibilistic answer set (or possibilistic pstable models). Another important motivation for considering the construction of possibilistic arguments is the flexibility of the inference in argumentation theory for managing inconsistent information. For instance, although a possibilistic logic programming semantics could infer inconsistent possibilistic models from a possibilistic knowledge Σ, by considering the possibilistic arguments built form Σ, one can infer conclusions from Σ supported by consistent subset of Σ.

7.2 Impact of the main contributions

In this section, we highlight four of the major contribution of this thesis. The first one is the development of a possibilistic disjunctive logic programming approach. The second one is the study of three of the Dung's argumentation semantics in terms of logic programming semantics. The third one is the definition of general schema for constructing argumentations semantics based on logic programming semantics. The last one is the definition of a possibilistic-based argumentation approach based on the possibilistic disjunctive approach introduced in this thesis.

7.2.1 Possibilistic Disjunctive Logic Programming

In Chapter 3, we define a possibilistic disjunctive logic programming approach in order to capture incomplete information and incomplete states of a knowledge base at the same time and also to capture inconsistent information. This approach could be of interest for researchers who are interested on capturing uncertain information where non-numerical values are not easy available. In fact, our approach means to capture qualitative knowledge that usually people use for supporting their decisions. In this sense in Chapter 3, we illustrate the use of our approach in the medical domain for capturing qualitative knowledge.

The results presented in Chapter 3 also contribute to the intensive research that have been done during the last years around of the answer set semantics. For the answer set semantics community, we introduce the first logic programming semantics which are able to capture possibilistic disjunctive logic programs and an approach for capturing *qualitative preferences* between clauses. These qualitative preferences are formalized by partial orders and ideas from possibilistic logic. An important property of our approach is that it is computable. Hence, the construction of real applications based on our approach are feasible. In fact, we proposed some algorithms for implementing possibilistic disjunctive interpreters as frontends of both answer set solvers and pstable model solvers.

7.2.2 Study of the Dung's argumentation semantics

In Chapter 4, we present novel results *w.r.t.* the close relationship between Dung's argumentation semantics and logic programming semantics. We show that one can capture the grounded, stable and preferred semantics by one logic program and three logic programming semantics (the well-founded semantics, the answer set semantics and the pstable semantics). These results open the possibilistic of studying the non-monotonic features of argumentation semantics as the preferred semantics in terms of logic foundations. For instance, since the preferred semantics can be characterized by the pstable semantics, one can define the preferred semantics in terms of paraconsistent logics (as the Cw and G3 logics) or modal logics (as the S5 modal logic).

An important concern in abstract argumentation theory is the computational complexity of the decision problems that has been shown to range from linear to $\Pi_2^{(p)}$-complete. A summary of this is given in Table 7.1 (practically all this table was taken from (45))

As we can see in Table 7.1, the computational complexity of the decision problem of argumentation semantics as the preferred semantics is hard. In this issue, we showed that by considering an argumentation framework AF as a propositional formula $\alpha(AF)$, one can characterize the preferred extensions of AF as the minimal models of $\alpha(AF)$. By using this result, we showed that one can use any UNSAT's algorithm for inferring the preferred extensions of AF. Remember that UNSAT is the complement of Satisfiability (SAT), a problem for which very efficient systems have been developed in AI during the last decade. This result suggests that one can implement *argumentation-components* based on UNSAT's algorithm in order to capture the preferred semantics in argumentation systems.

Another important result *w.r.t.* the preferred semantics was the formalization of the preferred semantics in terms of answer sets of a positive disjunctive logic program. This result opens the possibilistic of using any disjunctive answer set solver for computing the preferred extensions of an argumentation framework. In fact, by considering the codifications presented in Chapter 4 and any disjunctive answer solver, one can perform any of the decision queries presented in Table 7.1 (excepting by query h). Hence, the implementation of *argumentation-components* based on answer set solvers is a feasible way for supporting the implementation of argumentation systems. It is worth mentioning that nowadays the answer set

	Instance	**Decision Question**	**Complexity**
(a)	$AF = \langle AR, attacks \rangle$, $x \in AR$	Is x in the *grounded* extension of AF?	linear
(b)	$AF = \langle AR, attacks \rangle$, $x \in AR$	Is x in any *preferred* extension?	NP-complete
(c)	$AF = \langle AR, attacks \rangle$, $x \in AR$	Is x in any *stable* extension?	NP-complete
(d)	$AF = \langle AR, attacks \rangle$	Does AF have a *non-empty* preferred extension?	NP-complete
(e)	$AF = \langle AR, attacks \rangle$	Does AF have any stable extension?	NP-complete
(f)	$AF = \langle AR, attacks \rangle$, $x \in AR$	Is x in *every* stable extension?	CO-NP-Complete
(g)	$AF = \langle AR, attacks \rangle$, $x \in AR$	Is x in *every* preferred extension?	$\Pi_2^{(p)}$-complete
(h)	$AF = \langle AR, attacks \rangle$	Is AF a coherent argumentation framework?	$\Pi_2^{(p)}$-complete

Table 7.1: Decision problems in finite argumentation frameworks

solvers as DLV are faster algorithms able to give answer to problems of hard complexity.

By capturing Dung's argumentation semantics as logic programming semantics, we do not only identify general methods for generating argumentation components. This approach allows to explore the definition of new argumentation semantics in order to overcome some of their limitations. For instance, we define three extensions of the grounded semantics based on *local logical consequence in classical logic* and *reasoning by cases* in order to overcome some of the cases where the grounded semantics is empty. These extensions are able to manage, in a natural way, problems as *the self-defeated arguments*.

7.2.3 Construction of argumentations semantics and logic programming semantics

Since argumentation semantics can be viewed as a special form of logic programming semantics with negation as failure, in Chapter 5, we show that any logic programming semantics as the answer set semantics, the minimal models, the pstable semantics *etc.*, can define *candidate* argumentation semantics. This approach suggests a systematic process for constructing new argumentation semantics and new logic programming semantics.

Nowadays it has increased the number of new argumentation semantics in the context of Dung's argumentation approach; however, many of these new argumentation semantics are only motivated by particular examples. Hence, the definition of argumentation semantics with logical foundations takes relevance. Indeed, the identification of the non-monotonic reasoning properties, that a particular argumentation semantics satisfies, takes relevance in order to support the well-behaviour of an argumentation semantics.

Our approach is open enough for constructing new argumentation semantics based on the following considerations:

- the basic principles that we want that our new argumentation semantics must satisfy *i.e.* conflict-freeness, reinstatement *etc.*

- the basic logic programming semantics function *e.g.*, the minimal model semantics, the answer set semantics, the pstable semantics, *etc.*

The basic principles that we want that a new argumentation semantics has to satisfy are captures by the codification of an argumentation framework in terms of a logic program. Observe that this approach opens the possibilities of constructing new argumentation semantics under well accepted principles *i.e.* conflict-freeness.

We want to point out that also our approach motivates argumentation research by having in mind results of the non-monotonic reasoning area and the logic programming area. For instance, in order to define our approach for constructing argumentation semantics, we first introduce a recursive schema for constructing logic programming semantics which are always defined. Hence, this property ensures that any argumentation semantics induced by the new logic programming semantics will be always defined as well.

7.2.4 A possibilistic-based argumentation approach

In Chapter 6, we introduce the argumentation-based approach which is based on the syntaxis and semantics of possibilistic disjunctive logic programs. This approach tries to converge ideas from our possibilistic disjunctive logic programming approach and our results *w.r.t.* argumentation semantics.

Unlike to standard argumentation approach which are based on the monotonic inference of classical logic for building arguments, our approach is based on the non-monotonic inferences of logic programming semantics as the possibilistic answer set semantics and the possibilistic pstable semantics for building arguments.

The possibilistic-based argumentation inference defined for this approach can be regarded as an adapted tool for

- automatic reasoning based on possibilistic logic programming semantics where the available information is pervaded with inconsistencies.
- supporting decision making under a possibilistic knowledge base.

7.3 Future work

Research in logic programming with uncertainty has been extensively studies; however, the study of logic programming with qualitative degrees in the context of ASP is still lacking. The results of Chapter 3 can be considered into this direction. However, there are several issues open to explore and investigate *e.g.*, qualitative optimization, qualitative planning. It is worth mentioning that Brewka in (23) has motivated an approach for qualitative optimization. In (85), we have defined a possible extension of our possibilistic approach in order to develop qualitative planning.

As young area, in argumentation there are many challenges in order to build real intelligence systems based on fundamental mechanisms of argumentation. The study of argumentation semantics in terms of logic programming semantics is promised. For instance, as it has increased the number of new argumentation semantics in the context of Dung's argumentation style, the identification of the non-monotonic reasoning properties, that a particular argumentation semantics satisfies, will take relevance in order to support the well-behaviour of an argumentation semantics. In (12), it was defined a first set of basic principles in order to evaluate argumentation semantics. We believe that the set of principles described in (12) can be enriched by the identification of the non-monotonic reasoning properties that must satisfy any argumentation semantics. Of course, that this study could be explored by the characterization of argumentation semantics in terms of logic programming semantics.

In Chapter 5, a novel strategy for constructing argumentation semantics in terms of logic programming semantics was defined. As we commented, this approach is related to Baroni *et al*'s approach presented in (13). Hence a deep analysis is required in order to understand more about the relation between Baroni *et al*'s approach and our approach. In fact some interesting questions that we are going to consider in our future work are: Which is the class of argumentation semantics that can be characterized in both approaches? Which are the logic programming semantics that are more useful for building argumentation semantics? Which mappings of an argumentation framework into a logic program define useful argumentation semantics?

In general, we believe that a good study of the relationship between argumentation semantics and logic programming semantics with negation as failure could contribute to develop of prominent non-monotonic reasoning approaches.

Bibliography

[1] T. Alsinet, C. I. Chesñevar, L. Godo, S. Sadri, and G. R. Simari. Formalizing argumentative reasoning in a possibilistic logic programming setting with fuzzy unification. *International Journal of Approximate Reasoning*, page in press, 2008. 121

[2] T. Alsinet, C. I. Chesñevar, L. Godo, and G. R. Simari. A logic programming framework for possibilistic argumentation: Formalization and logical properties. *Fuzzy Sets and Systems*, 159(10):1208–1228, 2008. 55, 132

[3] T. Alsinet and L. Godo. A Ccomplete Calculus for Possibilistic Logic Programming with Fuzzy Propositional Variable. In *Proceedings of the Sixteen Conference on Uncertainty in Artificial Intelligence*, 1-10, 2000. ACM Press. 55, 56, 132

[4] T. Alsinet and L. Godo. Towards an automated deduction system for first-order possibilistic logic programming with fuzzy constants. *Int. J. Intell. Syst.*, 17(9):887–924, 2002. 55

[5] L. Amgoud and C. Cayrol. Inferring from inconsistency in preference-based argumentation frameworks. *J. Autom. Reasoning*, 29(2):125–169, 2002. 130

[6] L. Amgoud and H. Prade. Using arguments for making decisions: A possibilistic logic approach. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 10–17, Arlington, Virginia, 2004. AUAI Press. 4, 132

[7] ASPIC:Project. *Deliverable D2.1: Theoretical framework for argumentation.* Argumentation Service Plarform with Integrated Components, 2004. 130

[8] ASPIC:Project. *Deliverable D2.2:Formal semantics for inference and decision-making.* Argumentation Service Plarform with Integrated Components, 2005. 62

[9] M. Balduccini and M. Gelfond. Logic Programs with Consistency-Restoring Rules. In P. Doherty, J. McCarthy, and M.-A. Williams, editors, *International Symposium on Logical Formalization of Commonsense Reasoning*, AAAI 2003 Spring Symposium Series, Mar 2003. 102

[10] J. F. Baldwin. Evidential support logic programming. *Fuzzy Sets and Systems*, 24(1):1–26, October 1987. 55

[11] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, 2003. 9, 10, 15, 27, 99, 101

[12] P. Baroni and M. Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artificial Intelligence.*, 171(10-15):675–700, 2007. 111, 117, 142

[13] P. Baroni, M. Giacomin, and G. Guida. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168:162–210, October 2005. 5, 93, 94, 95, 97, 112, 113, 115, 116, 117, 128, 142

[14] R. Ben-Eliyahu-Zohary. An incremental algorithm for generating all minimal models. *Artificial Intelligence*, 169(1):1–22, 2005. 70

[15] T. Bench-Capon. Value-based argumentation frameworks. In *Proceedings of Non Monotonic Reasoning*, pages 444–453, 2002. 4

[16] T. J. M. Bench-Capon and P. E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10-15):619–641, 2007. 1, 4, 5, 62, 93, 94, 101, 136

[17] P. Besnard and S. Doutre. Checking the acceptability of a set of arguments. In *Tenth International Workshop on Non-Monotonic Reasoning (NMR 2004),*, pages 59–64, June 2004. 69

[18] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997. 4

[19] S. Brass and J. Dix. Characterizations of the disjunctive stable semantics by partial evaluation. *J. Log. Program.*, 32(3):207–228, 1997. 42

[20] S. Brass and J. Dix. Characterizations of the disjunctive well-founded semantics: Confluent calculi and iterated gcwa. *J. Autom. Reasoning*, 20(1):143–165, 1998. 42

[21] S. Brass and J. Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 38(3):167–213, 1999. 42

[22] S. Brass, U. Zukowski, and B. Freitag. Transformation-based bottom-up computation of the well-founded model. In *NMELP*, pages 171–201, 1996. 12, 13, 17, 18

[23] G. Brewka. Answer sets: From constraint programming towards qualitative optimization. In V. Lifschitz and I. Niemelä, editors, *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, volume 2923 of *Lecture Notes in Computer Science*, pages 34–46. Springer, 2004. 56, 142

[24] O. Bueno. *Knowledge and Inquiry : Essays on the Pragmatism of Isaac Levi*, chapter Why Inconsistency Is Not Hell: Making Room for Inconsistency in Science, pages 70–86. Cambridge Studies in Probability, Induction and Decision Theory. CAMBRIDGE UNIVERSITY PRESS, 2006. 50, 52

[25] F. Caballero, A. López-Navidad, M. Perea, C. Cabrer, L. Guirado, and R. Solá. Successful liver and kidney transplantation from cadaveric donor with left-sided bacterial endocarditis. *American Journal of Transplantation*, 5:781–787, 2005. 84

[26] M. Caminada. Contamination in formal argumentation systems. In *BNAIC 2005 - Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence, Brussels, Belgium, October 17-18*, pages 59–65, 2005. 5, 93

[27] M. Caminada. Semi-Stable semantics. In P. E. Dunne and T. J. Bench-Capon, editors, *Proceedings of COMMA*, volume 144, pages 121–130. IOS Press, 2006. 5, 93, 94, 95

[28] M. Caminada and C. Sakama. On the existence of answer sets in normal extended logic programs. In *ECAI*, pages 743–744, 2006. 101

[29] J. L. Carballido, J. C. Nieves, and M. Osorio. Inferring Preferred Extensions by Pstable Semantics. *Iberoamerican Journal of Artificial Intelligence (Inteligencia Artificial)ISSN: 1137-3601*, to appear. 7, 162, 164

[30] C. I. Chesñevar, A. G. Maguitman, and R. P. Loui. Logical models of argument. *ACM Comput. Surv.*, 32(4):337–383, 2000. 4, 136

[31] T. H. Cormen, C. E. Leiserson, R. L. Riverst, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001. 113

[32] B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order*. Cambridge University Press, second edition, 2002. 9, 24

[33] N. Dershowitz and D. A. Plaisted. *Handbook of Automated Reasoning*, chapter Rewriting. Elsevier Science Publishers, 2001. 12

[34] Y. Dimopoulos and A. Torres. Graph theoretical structures in logic programs and default theories. *Theor. Comput. Sci.*, 170(1-2):209–244, 1996. 70

[35] J. Dix. A classification theory of semantics of normal logic programs: I. strong properties. *Fundam. Inform.*, 22(3):227–255, 1995. 42

[36] J. Dix. A classification theory of semantics of normal logic programs: II. weak properties. *Fundam. Inform.*, 22(3):257–288, 1995. 17, 42, 91, 101, 163

[37] J. Dix and M. Müller. Partial evaluation and relevance for approximations of stable semantics. In *ISMIS*, volume 869 of *Lecture Notes in Computer Science*, pages 511–520. Springer, 1994. 97, 101, 105

[38] J. Dix, M. Osorio, and C. Zepeda. A General Theory of Confluent Rewriting Systems for Logic Programming and its applications. *Annals of Pure and Applied Logic*, 108(1–3):153–188, 2001. 12, 13

[39] J. Dix, M. Osorio, and C. Zepeda. A general theory of confluent rewriting systems for logic programming and its applications. *Ann. Pure Appl. Logic*, 108(1-3):153–188, 2001. 13, 18, 42, 168

[40] S. DLV. Vienna University of Technology. http://www.dbai.tuwien.ac.at/proj/dlv/, 1996. 16, 40, 71, 74, 132

[41] D. Dubois, J. Lang, and H. Prad. Towards possibilistic logic programming. In K. Furukawa, editor, *ICLP*, pages 581–595. The MIT Press, 1991. 55

[42] D. Dubois, J. Lang, and H. Prade. Possibilistic logic. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, pages 439–513. Oxford University Press, Oxford, 1994. 2, 3, 19, 20, 27, 39, 129, 156, 158

[43] D. Dubois and H. Prade. Possibilistic logic: a retrospective and prospective view. *Fuzzy Sets and Systems*, 144(1):3–23, 2004. 2, 20, 129

[44] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995. 4, 20, 22, 23, 60, 61, 79, 89, 94

[45] P. E. Dunne. Computational properties of argument systems satisfying graph-theoretic constraints. *Artificial Intelligence*, 171(10-15):701–729, 2007. 139

[46] M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11(1&2):91–116, 1991. 55, 56

[47] J. Fox and S. Das. *Safe and Sound: Artificial Intelligence in Hazardous Applications*. AAAI Press/ The MIT Press, 2000. 1, 135, 136

[48] J. Fox, D. Glasspool, D. Grecu, S. Modgil, M. South, and V. Patkar. Argumentation-Based inference and decision making — A Medical Perspective. *IEEE Intelligence Systems*, 22(6):34–41, November/December 2007. 122, 135, 136

[49] J. Fox and S. Modgil. From arguments to decisions: Extending the Toulmin view. In D. Hitchcock and B. Verheij, editors, *Arguing on the Toulmin model: New essays on argument analysis and evaluation*, pages 273–287. Springer Netherlands, 2006. 25

[50] A. V. Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991. 17

[51] M. Gelfond. *Handbook of Knowledge Representation*, chapter Answer Sets, pages 285–316. Elsevier, 2008. 27

[52] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988. 15

[53] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991. 15, 27, 101

[54] G. Governatori and M. J. Maher. An argumentation-theoretic characterization of defeasible logic. In *ECAI*, pages 469–473. IOS Press, 2000. 65

[55] G. Governatori, M. J. Maher, G. Antoniou, and D. Billington. Argumentation semantics for defeasible logic. *J. Log. Comput.*, 14(5):675–702, 2004. 65

[56] J. Y. Halpern. *Reasoning about uncertainty.* The MIT Press, 2005. 1, 2

[57] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation, 3/E.* Addison Wesley Higher Education, 2007. 29

[58] H. Jakobovits and D. Vermeir. Robust semantics for argumentation frameworks. *Journal of logic and computation*, 9(2):215–261, 1999. 4, 84, 92, 95

[59] D. Kahneman, P. Slovic, and A. Tversky. *Judgment under uncertainty:Heuristics and biases.* Cambridge Univertisy Press, 1982. 3, 135

[60] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook in Artificial Intelligence and Logic Programming, Volume 5*, pages 235–324. Oxford University Press, Oxford, 1998. 102

[61] A. C. Kakas and P. Mancarella. Generalized stable models: A semantics for abduction. In *ECAI*, pages 385–391, 1990. 102

[62] G. Kern-Isberner and T. Lukasiewicz. Combining probabilistic logic programming with the power of maximum entropy. *Artificial Intelligence*, 157(1-2):139–202, 2004. 55

[63] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. Log. Program.*, 12(3&4):335–367, 1992. 55

[64] P. Krause, S. Ambler, M. Elvang-Gøransson, and J. Fox. A logic of argumentation for reasoning under uncertainty. *Computational Intelligence*, 11:113–131, 1995. 121

[65] A. W. Kushniruk. Analysis of Complex Decision Process in Health Care: Cognitive Approaches to Health Informatics. *Journal of Biomedical Informatics*, 34:365–376, 2001. 135

[66] L. V. S. Lakshmanan. An epistemic foundation for logic programming with uncertainty. In *FSTTCS*, pages 89–100, 1994. 55

[67] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Trans. Comput. Log.*, 2(4):526–541, 2001. 162

[68] J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1987. 9, 99, 101

[69] A. López. Implementing pstable. In R. Dávila, M. Osorio, and C. Zepeda, editors, *Workshop in Logic, Language and Computation*, volume 220. CEUR Workshop Proceedings, 2006. 49, 132, 160

[70] A. López-Navidad and F. Caballero. Extended criteria for organ acceptance: Strategies for achieving organ safety and for increasing organ pool. *Clinical Transplantation, Blackwell Munksgaard*, 17:308–324, 2003. 28, 30

[71] A. López-Navidad, P. Domingo, and M. A. Viedma. Professional characteristics of the transplant coordinator. In *XVI International Congress of the Transplantation Society*, volume 29 of *Transplantation Proceedings*, pages 1607–1613. Elsevier Science Inc, February-March 1997. 28, 31

[72] T. Lukasiewicz. Probabilistic logic programming. In *ECAI*, pages 388–392, 1998. 55

[73] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. reprinted in McC90. 2

[74] E. Mendelson. *Introduction to Mathematical Logic*. Chapman and Hall/CRC, Fourth edition 1997. 9

[75] R. T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Inf. Comput.*, 101(2):150–201, 1992. 55

[76] P. Nicolas, L. Garcia, I. Stéphan, and C. Lafèvre. Possibilistic Uncertainty Handling for Answer Set Programming. *Annal of Mathematics and Artificial Intelligence*, 47(1-2):139–181, June 2006. 27, 31, 34, 38, 47, 48, 54, 55, 57, 155

[77] J. C. Nieves and U. Cortés. Modality argumentation programming. In *Proceedings of the conference Modeling Decisions for Artificial Intelligence (MDAI 2006)*, volume 3885 of *Lecture Notes in Computer Science*, pages 295–306. Springer, 2006. 8

[78] J. C. Nieves and M. Osorio. Inferring Preferred Extensions by Pstable Semantics. In *Proceedings of the Latin-American Workshop on Non-Monotonic Reasoning (LA-NMR07) Workshop*, volume 286 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007. 7

[79] J. C. Nieves, M. Osorio, and U. Cortés. Inferring Preferred Extensions by Minimal Models. In G. Simari and P. Torroni, editors, *Argumentation and Non-Monotonic Reasoning (LPNMR-07 Workshop)*, pages 114–124, Arizona, USA, 2007. 7

[80] J. C. Nieves, M. Osorio, and U. Cortés. Modality-based argumentation using possibilistic stable models. In R. Kibble, F. Grasso, and C. Reed, editors, *7th Workshop on Computational Models of Natural Argument (CMNA VII)*, pages 35–41, Hyderabad, India, January 2007. 8

[81] J. C. Nieves, M. Osorio, and U. Cortés. Semantics for possibilistic disjunctive programs. In S. Costantini and R. Watson, editors, *Answer Set Programming: Advances in Theory and Implementation (ICLP-07 Workshop)*, pages 271–284, 2007. 7

[82] J. C. Nieves, M. Osorio, and U. Cortés. Semantics for possibilistic disjunctive programs (poster). In C. Baral, G. Brewka, and J. Schlipf, editors, *Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-07)*, number 4483 in LNAI, pages 315–320. Springer-Verlag, 2007. 7

[83] J. C. Nieves, M. Osorio, and U. Cortés. Possibilistic-based argumentation: An answer set programming approach. In *In Proc. of the Mexican International Conference on Computer Science*, page to appear. IEEE Computer Science Press, 2008. 8

[84] J. C. Nieves, M. Osorio, and U. Cortés. Preferred Extensions as Stable Models. *Theory and Practice of Logic Programming*, 8(4):527–543, July 2008. 7

[85] J. C. Nieves, M. Osorio, U. Cortés, F. Caballero, and A. López-Navidad. Reasoning about actions under uncertainty: A possibilistic approach. In C. Angulo and L. Godo, editors, *In proceedings of CCIA*, 2007. 142

[86] J. C. Nieves, M. Osorio, U. Cortés, I. Olmos, and J. A. Gonzalez. Defining new argumentation-based semantics by minimal models. In *Seventh Mexican International Conference on Computer Science (ENC 2006)*, pages 210–220. IEEE Computer Science Press, September 2006. 7, 95

[87] J. C. Nieves, M. Osorio, C. Zepeda, and U. Cortés. Inferring acceptable arguments with answer set programming. In *Sixth Mexican International Conference on Computer Science (ENC 2005)*, pages 198–205. IEEE Computer Science Press, September 2005. 7

[88] M. Osorio, J. R. Arrazola, and J. L. Carballido. Logical Weak Completions of Paraconsistent Logics. *Journal of Logic and Computation*, doi: 10.1093/logcom/exn015, 2008. 16, 159

[89] M. Osorio and J. L. Carballido. Brief study of G'3 logic. *Journal of Applied Non-Classical Logics*, 18(4):79–103, 2008. 90

[90] M. Osorio, J. A. Navarro, and J. Arrazola. Applications of Intuitionistic Logic in Answer Set Programming. *Theory and Practice of Logic Programming (TPLP)*, 4(3):225–354, May 2004. 70, 167

[91] M. Osorio, J. A. Navarro, J. R. Arrazola, and V. Borja. Ground Nonmonotonic Modal Logic S5: New Results. *Journal of Logic and Computation*, 15(5):787–813, 2005. 90, 158

[92] M. Osorio, J. A. Navarro, J. R. Arrazola, and V. Borja. Logics with Common Weak Completions. *Journal of Logic and Computation*, 16(6):867–890, 2006. 16, 101, 157, 158

[93] M. Osorio and J. C. Nieves. $W_{s,c}$-Stable Semantics for Propositional Theories. In *International Conferences of CIC'2001*, pages 319–328, México, 2001. 102, 103

[94] M. Osorio and J. C. Nieves. Pstable semantics for possibilistic logic programs. In *MICAI 2007: Advances in Artificial Intelligence, 6th Mexican International Conference on Artificial Intelligence*, number 4827 in LNAI, pages 294–304. Springer-Verlag, 2007. 7

[95] M. Osorio, J. C. Nieves, and C. Giannella. Useful Transformations in Answer Set Programming. In A. Provetti and S. T. Cao, editors, *Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning (AAAI Spring 2001 Symposium)*, Stanford, USA, March 2001. 13

[96] F. J. Pelletier and R. Elio. *Scope of Logic, Methodology and Philosophy of Science*, volume 1 of *Synthese Library*, chapter Logic and Computation, pages 137–156. Dordrecht: Kluwer Academic Press, 2002. 3

[97] L. M. Pereira and A. M. Pinto. Revised stable models - a semantics for logic programs. In *EPIA*, pages 29–42, 2005. 101

[98] J. L. Pollock. Justification and defeat. *Artif. Intell.*, 67(2):377–407, 1994. 93

[99] H. Prakken and G. A. W. Vreeswijk. Logics for defeasible argumentation. In D. Gabbay and F. Günthner, editors, *Handbook of Philosophical Logic*, volume 4, pages 219–318. Kluwer Academic Publishers, Dordrecht/Boston/London, second edition, 2002. 4, 5, 21, 62, 91, 93, 94, 136

[100] F. Ricca. The dlv java wrapper. In *2003 Joint Conference on Declarative Programming, AGP-2003, Reggio Calabria, Italy, September 3-5, 2003*, pages 263–274, 2003. 90

[101] M. Rodríguez-Artalejo and C. A. Romero-Díaz. Quantitative Logic Programming revisited. In J. Garrigue and M. Hermenegildo, editors, *9th International Symposium, FLOPS*, volume 4989 of *LNCS*, pages 272–288. Springer-Verlag Berlin Heidelberg, 2008. 55

[102] S. Rusell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pretice Hall Series in Artificial Intelligence, 2003. 1

[103] S. SMODELS. Helsinki University of Technology. http://www.tcs.hut.fi/Software/smodels/, 1995. 132

[104] V. S. Subrahmanian. Uncertainty in logic programming. *Association for Logic Programming (ALP), Newsletter*, 20(2), May/June 2007. 55

[105] P. Szolovits. *Artificial Intelligence and Medicine*. Westview Press, Boulder, Colorado, 1982. 1

[106] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955. 157

[107] A. Tversky and D. Kahneman. *Judgment under uncertainty:Heuristics and biases*, chapter Judgment under uncertainty:Heuristics and biases, pages 3–20. Cambridge Univertisy Press, 1982. 3, 30

[108] D. van Dalen. *Logic and structure*. Springer-Verlag, Berlin, 3rd., aumented edition edition, 1994. 11

[109] M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986. 55

[110] D. Van-Nieuwenborgh, M. D. Cock, and D. Vermeir. An introduction to fuzzy answer set programming. *Ann. Math. Artif. Intell.*, 50(3-4):363–388, 2007. 55

[111] G. Vreeswijk. Abstract argumentation systems. *Artificial Intelligence*, 90(1-2):225–279, 1997. 4

Appendix A

Proofs of Chapter 3

In this annex we give the proof of most of the results given in Chapter 3.

Proposition 3.2 *Let P be a possibilistic disjunctive logic program. M is a possibilistic answer set of P iff M^* is an answer set of P^*.*
Proof: The proof is straightforward by the possibilistic answer set's definition. ■

Proposition 3.3 *Let $P = \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic disjunctive logic program and $\mathcal{TOP}_{\mathcal{Q}}$ be the top of the lattice $(\mathcal{Q}, \leq)$. If $\forall r \in P$, $n(r) = \mathcal{TOP}_{\mathcal{Q}}$, and M' is an answer set of P^*, then $M := \{(l, \mathcal{TOP}_{\mathcal{Q}}) | l \in M'\}$ is a possibilistic answer set of P.*
Proof: We know that if M is a possibilistic answer set of P, then M^* is an answer set of P^* (by Proposition 3.2) and $N^{M^*} \vdash_{PL} M$. Now, since (GMP) $(\varphi\ \mathcal{TOP}_{\mathcal{Q}}), (\varphi \rightarrow \psi\ \mathcal{TOP}_{\mathcal{Q}}) \vdash_{PL} (\psi\ \mathcal{TOP}_{\mathcal{Q}})$, then any formula inferred from P by GMP will have $\mathcal{TOP}_{\mathcal{Q}}$ as necessity-value. Then, if M' is an answer set of P^*, then $\{(l, \mathcal{TOP}_{\mathcal{Q}}) | l \in M'\}$ will be a possibilistic answer set of P. ■

Proposition 3.4 *Let $P := \langle(Q, \leq), N\rangle$ be a possibilistic normal program such that $(Q, \leq)$ is a totally ordered set and $\mathcal{L}_P$ has no extended atoms. M is a possibilistic answer set of P if and only if M is a possibilistic stable model of P.*
Proof: (Sketch) It is not difficult to see that when P is a possibilistic normal program, then the syntactic reduction of Definition 3.2 and the syntactic reduction of Definition 10 from (76) coincide. Then the proof is reduced to possibilistic definite programs. But, this case is straightforward, since essentially GMP is applied for inferring the possibilistic models of the program in both approaches.
■

Proposition 3.5 *Let $\mathcal{C}$ be a set of possibilistic disjunctions, and $C = (c\ \alpha)$ be a possibilistic clause obtained by a finite number of successive application of* (R) *to $\mathcal{C}$; then $\mathcal{C} \vdash_{PL} C$.*

<u>Proof</u>: (The proof is similar to the proof of Proposition 3.8.2 of (42)) Let us consider two possibilistic clauses: $C_1 = (c_1\ \alpha_1)$ and $C_2 = (c_2\ \alpha_2)$, the application of R yields $C' = (R(c_1, c_2)\ \mathcal{GLB}(\{\alpha_1, \alpha_2\}))$. By classic logic, we known that $R(c_1, c_2)$ is sound; hence the key point of the proof is to show that $n(R(c_1, c_2)) \geq \mathcal{GLB}(\{\alpha_1, \alpha_2\})$.

By definition of necessity-valued clause, $n(c_1) \geq \alpha_1$ and $n(c_2) \geq \alpha_2$, then $n(c_1 \wedge c_2) = \mathcal{GLB}(\{n(c_1), n(c_2)\}) \geq \mathcal{GLB}(\{\alpha_1, \alpha_2\})$. Since $c_1 \wedge c_2 \vdash_C R(c_1, c_2)$, then $n(R(c_1, c_2)) \geq n(c_1 \wedge c_2)$ (because if $\varphi \vdash_{PL} \psi$ then $N(\psi) \geq N(\varphi)$). Thus $n(R(c_1, c_2)) \geq \mathcal{GLB}(\{\alpha_1, \alpha_2\})$; therefore (R) is sound. Then by induction any possibilistic formula inferred by a finite number of successive applications of (R) to $\mathcal{C}$ is a logical consequence of $\mathcal{C}$. ■

Proposition 3.6 *Let P be a set of possibilistic clauses and $\mathcal{C}$ be the set of possibilistic disjunctions obtained from P; then the valuation of the optimal refutation by resolution from $\mathcal{C}$ is the inconsistent degree of P.*

<u>Proof</u>: (The proof is similar to the proof of Proposition 3.8.3 of (42)) By possibilistic logic, we know that $\mathcal{C} \vdash_{PL} (\perp\ \alpha)$ if and only if $(\mathcal{C}_\alpha)^*$ is inconsistent in the sense of classic logic. Since (R) is complete in classic logic, then there exists a refutation $R(\Box)$ from $(\mathcal{C}_\alpha)^*$. Thus considering the valuation of the refutation $R(\Box)$, we obtain a refutation from $\mathcal{C}_\alpha$ such that $n(R(\Box)) \geq \alpha$. Then $n(R(\Box)) \geq Inc(\mathcal{C})$. Since (R) is sound then $n(R(\Box))$ cannot be strictly greater than $Inc(\mathcal{C})$. Thus $n(R(\Box))$ is equal to $Inc(\mathcal{C})$. According to Proposition 3.8.1 of (42), $Inc(\mathcal{C}) = Inc(P)$, thus $n(R(\Box))$ is also equal to $Inc(P)$. ■

Proposition 3.7 *Let $P := \langle(\mathcal{Q}, \leq), N\rangle$ be a possibilistic logic program. The set Poss-ASP returned by Poss_Answer_Sets(P) is the set of all the possibilistic answer sets of P.*

<u>Proof</u>: The result follows from the following facts:

1. The function ASP computes all the answer set of P^*.
2. If M is a possibilistic answer set of P iff M^* is an answer set of P^* (Proposition 3.2).
3. By Corollary 3.1, we know that the possibilistic resolution rule R is sound and complete for computing optimal possibilistic degrees.

■

Proposition 3.8 *Let P be a possibilistic disjunctive logic program. If $\Gamma_0 := \mathcal{T}(P)$ and $\Gamma_i := \mathcal{T}(\Gamma_{i-1})$ such that $i \in \mathcal{N}$, then $\exists\ n \in \mathcal{N}$ such that $\Gamma_n = \Gamma_{n-1}$. We denote Γ_n by $\Pi(P)$.*

Proof: It is not difficult to see that the operator $\mathcal{T}$ is monotonic, then the proof is direct by Tarski's Lattice-Theoretical Fixpoint Theorem (106). ■

Proposition 3.9 *Let P be a possibilistic disjunctive logic program and M a set of possibilistic atoms. M is a possibilistic answer set of P if and only if M is a possibilistic-$\mathcal{T}$ answer set of P.*
Proof: Two observations:

1. By definition, it is straightforward that if M_1 is a possibilistic answer set of P, then there exists a possibilistic-$\mathcal{T}$ answer set M_2 of P such that $M_1^* = M_2^*$ and viceversa.

2. Since G-GPPE can be regarded as a macro of the possibilistic rule (R), we can conclude by Proposition 3.5 that G-GPPE is sound.

Let M_1 be a possibilistic answer set of P and M_2 be a possibilistic-$\mathcal{T}$ answer set of P. By Observation 1, the central point of the proof is to prove that if $(a, \alpha_1) \in M_1$ and $(a, \alpha_2) \in M_2$ such that $M_1^* = M_2^*$, then $\alpha_1 = \alpha_2$.

The proof is by contradiction. Let us suppose that $(a, \alpha_1) \in M_1$ and $(a, \alpha_2) \in M_2$ such that $M_1^* = M_2^*$ and $\alpha_1 \neq \alpha_2$. Then there are two cases $\alpha_1 < \alpha_2$ or $\alpha_1 > \alpha_2$

$\alpha_1 < \alpha_2$: Since G-GPPE is sound (Observation 2), then α_1 is not the optimal necessity-value for the atom a, but this is false by Corollary 3.1.

$\alpha_1 > \alpha_2$: If $\alpha_1 > \alpha_2$ then there exists a possibilistic claus $\alpha_1 : \mathcal{A} \leftarrow \mathcal{B}^+ \in P^{(M_1)^*}$ that belongs to the optimal refutation of the atom a and it was not reduced by G-GPPE. But this is false because G-GPPE is a macro of the resolution rule (R).

■

Proposition 3.10 *Let P be a possibilistic normal program. If M is a possibilistic answer set of P, then the following conditions hold:*

a) *M^* is a pstable model of P^*.*

b) *there exists a possibilistic pstable mode M' of P such that $M \sqsubseteq M'$ and $M^* = M'^*$.*

Proof:

a) The proof is straightforward by Theorem 4.4 of (92) (The Theorem 4.4 of (92) says that given a normal logic program P and a set of atoms M, if M is an answer set of P then M is a pstable model of P).

b) First of all observe that the following relation is true:

$$P^{M^*} \subseteq PRED(P, M) \tag{A.1}$$

By a), it is direct that if M is a possibilistic answer set of P, then there exists a possibilistic pstable model M' such that $M^* = M'^*$. Hence, if $(a, \alpha_1) \in M$, then $a \in M'^*$ such that $(a, \alpha_2) \in M'$. Therefore, the relevant part of prove is to show that $\alpha_1 \leq \alpha_2$.

The proof is by contradiction: Let us suppose that $\alpha_2 < \alpha_1$, by definition of possibilistic answer set and pstable model, $P^{M^*} \vdash_{PL} (a, \alpha_1)$ and $PRED(P, M) \vdash_{PL} (a, \alpha_2)$ such that α_1 and α_2 are optimal. Since $P^{M^*} \subseteq PRED(P, M)$, hence then $PRED(P, M) \vdash_{PL} (a, \alpha_1)$. Therefore, α_2 is not the optimal value of a *w.r.t.* $PRED(P, M)$. This is a contradiction, because α_2 is the optimal value of a *w.r.t.* $PRED(P, M)$ by definition of the possibilistic pstable semantics.

■

Proposition 3.11 *Let P be a possibilistic normal program. If $P \vdash_{PL} (x\ \alpha)$ then P is equivalent to $P \cup \{(x\ \alpha)\}$ under the possibilistic pstable semantics.*
Proof: Some observations;

a) By Theorem 7.11 of (91) and Theorem 5.1 of (92), we can see that: If $P^* \vdash_C x$ then P^* is equivalent to $P^* \cup \{x\}$ under the pstable semantics *i.e.* $Pstable(P^*) = Pstable(P^* \cup \{x\})$

b) By definition of the possibilistic pstable semantics: M is a possibilistic pstable model of P then M^* is a pstable model of P^*.

c) By definition of the syntactic reduction $PRED$, it is easy to see that: Given a possibilistic normal program P and a set of atoms M : $PRED(P \cup (a, \alpha), M) = PRED(P, M) \cup \{(a, \alpha)\}$.

d) In (42), it was proved that: $P \vdash_{PL} (x\ \alpha)$ iff $P_\alpha \vdash_{PL} (x\ \alpha)$.

We use *poss_Pstable* to denote the semantics operator of the possibilistic pstable semantics. Then we have to prove that

$$poss\_Pstable(P) = poss\_Pstable(P \cup \{(x\ \alpha)\})$$

=> We have to prove that if $M \in poss\_Pstable(P)$ then $M \in poss\_Pstable(P \cup \{(x\ \alpha)\})$.

Proof: $M \in poss\_Pstable(P)$ iff $M^* \in Pstable(P^*)$ (by b) iff $M^* \in Pstable(P^* \cup \{x\})$ (by a). Hence, there exists $M' \in poss\_Pstable(P \cup \{(x\ \alpha)\})$ such that $M^* = M'^*$.

Let us suppose that $M \neq M'$, this means that there exists $(a, \alpha_1) \in M$ and $(a, \alpha_2) \in M'$ such that $\alpha_1 \neq \alpha_2$.

If $\alpha_1 \neq \alpha_2$, then there two cases:

$\alpha_1 > \alpha_2$: If $\alpha_1 > \alpha_2$, then $PRED(P, M^*)_{\alpha_1} \subset PRED(P \cup \{(a, \alpha)\}, M'^*)_{\alpha_2}$ (remember that $PRED(P \cup (a, \alpha), M) = PRED(P, M) \cup \{(a, \alpha)\}$). Since $PRED(P, M^*)_{\alpha_1} \vdash_{PL} (a\ \alpha_1)$ and $PRED(P \cup (a, \alpha), M'^*)_{\alpha_2} \vdash_{PL} (a\ \alpha_2)$, hence α_2 is not the optimal necessity value of a inferred from $PRED(P \cup \{(a, \alpha)\}, M'^*)_{\alpha_2}$. This is a contradiction, because M' is a possibilistic Pstable model of $P \cup (a, \alpha)$.

$\alpha_1 < \alpha_2$: If $\alpha_1 < \alpha_2$, then $PRED(P, M'^*)_{\alpha_2} \subset PRED(P, M^*)_{\alpha_1}$ and $\alpha < \alpha_1$. Hence $\alpha_2 = \alpha$. Then $P_{\alpha_2} \subset P_{\alpha_1}$. Since $P_{\alpha_2} \vdash_{PL} (x\ \alpha_2)$, then $P_{\alpha_2} \vdash_{PL} (a\ \alpha_2)$. Therefore $P_{\alpha_1} \vdash_{PL} (a\ \alpha_2)$. Then α_1 is not the optimal necessity value for a inferred from $PRED(P, M)_{\alpha_1}$. This is a contradiction, because M is a possibilistic Pstable model of P.

Therefore $\alpha_1 = \alpha_2$, this means that $M = M'$.

<= We have to prove that if $M \in poss\_Pstable(P \cup \{(x\ \alpha)\})$ then $M \in poss\_Pstable(P)$. The proof is similar to the previous case.

■

Theorem 3.1 *Let P be a possibilistic disjunctive program. If M is a possibilistic answer set of P, then it implies that*

a) *M^* is a pstable model of $TRAD(P)^*$.*

b) *there exists a possibilistic pstable mode M' of $TRAD(P)$ such that $M \sqsubseteq M'$ and $M^* = M'^*$.*

Proof:

a) The proof is straightforward by Theorem 5.3 of (88) (The Theorem 5.3 of (88) says that if M is an answer set of P^* then M is a pstable model of $(TRAD(P))^*$. The authors of (88) use the concept of "closed under d-shift"; but this concept is nothing else that the consideration of the mapping $TRAD$ without the possibilistic values).

b) Direct by a) and Proposition 3.10.

■

Proposition 3.12 *Given a possibilistic program $P := \langle(\mathcal{Q}, \leq), N\rangle$ there exists an algorithm that computes the set of possibilistic pstable models of P.*
Proof: The algorithm is the same to the algorithm presented in the proof of Proposition 3.7. The only difference is that instead of using an algorithm for computing the answer sets of P^*, it is used an algorithm for computing the pstable models of P^* *e.g.*, the algorithm presented in (69). ■

Appendix B

Proofs of Chapter 4

In this annex we give the proof of most of the results given in Chapter 4.

Lemma 4.1 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $S \subseteq AR$. S is the grounded extension of AF if and only if $\exists\, D \subseteq AR$ such that $\langle f(D), f(S) \rangle$ is the well-founded model of Ψ_{AF}.*
Proof: The proof is by induction on the minimum number of steps N to get the Ψ_{AF}'s normal form. Let F_{AF} be the characteristic function of AF. By Lemma 2.1, we known that $WFS(P) := SEM(norm_{CS_0}(P))$. So, let $\langle f(D), f(S) \rangle$ be the well-founded model of Ψ_{AF}.

Base Step If $N = 0$, then Ψ_{AF} is in its normal form. By the definition of Ψ_{AF}, if $a \in AR$ and $\nexists b \in AR$ such that $(b, a) \in attacks$, then $d(a) \notin HEAD(\Psi_{AF})$. This means that $d(a) \in f(S)$ and $a \in F^0_{AF}$. It is easy to see that F^0_{AF} is the fix point of F_{AF} and $f(F^0_{AF}) = f(S)$. Therefore, S is the grounded extension of AF.

Inductive step Now, let us suppose that Ψ_{AF} is not in its normal form, so we need N steps to get its normal form. Let $\langle f(D'), f(S') \rangle$ be $SEM(\Psi_{AF})$ and $\langle f(D''), f(S'') \rangle$ be $SEM(\Psi''_{AF})$ such that $\Psi_{AF} \rightarrow^T \Psi''_{AF}$ and $T \in CS_0$. If $a \in AR$ and $\nexists b \in AR$ such that $(b, a) \in attacks$, then $d(A) \notin HEAD(\Psi_{AF})$, $d(A) \in f(S')$, $A \in F^m_{AF}$, and $m \geq 0$. There are two relevant cases *w.r.t.* the argument a:

1. If $b \in AR$ such that b is attacked by a, then there is a rule $r_1 \in \Psi_{AF}$ of the form $r_1 : d(b) \leftarrow \; not\ d(a)$; therefore, if $T = RED^+$, then $d(b) \leftarrow \top \in \Psi''_{AF}$ and $d(b) \in f(D'')$. This means b is a defeated argument and $b \notin F^m_{AF}$.
2. If b is defended by a, then there is a rule $r_2 \in \Psi_{AF}$ of the form $r_2 : d(b) \leftarrow d(x_1), \ldots, d(a), \ldots, d(x_n)$, where $x_i \in AR$ such that x_i defends

b; therefore, r_2 is deleted by $Failure$. This means, if $T = Failure$, then $r_2 \notin \Psi''_{AF}$. Notice that, if $d(b) \notin HEAD(\Psi''_{AF})$, then $d(b) \in f(S'')$ and $B \in F^m_{AF}$.

One can see that the application of CS_0 over Ψ_{AF} will remove from Ψ_{AF} any rule $r \in \Psi_{AF}$ such that r's head is an atom of the form $d(a)$ and a is an acceptable argument. So, by inductive hypothesis, it is easy to see that if $\langle f(D), f(S)\rangle$ is the well-founded model of Ψ_{AF} then S is the grounded extension of AF.

■

Proof of Lemma 4.2

In order to prove Lemma 4.2, let us introduce the following two lemmas *w.r.t. answer set equivalence*[1].

According to (67), P_1 is answer set equivalent to P_2 if P_1 and P_2 have the same answer sets. Also we can say that P_1 is *strongly equivalent* to P_2 if and only if for every logic program P, $P_1 \cup P$ and $P_2 \cup P$ have the same answer sets. The intuitive idea of stable equivalence is that for any two programs which are stable equivalent we can replace one by the other in any large program without changing the stable models (declarative semantics). By having in mind this idea, we present the following two lemmas:

Lemma B.1 *(29) Let P be a normal program that includes the clause:*

$$x \leftarrow not\ y$$

Suppose in addition that all the rules in P with head equal to y are:
$y \leftarrow not\ y_1.\quad \ldots\quad y \leftarrow not\ y_m.$
Then P is answer set equivalent to $P \cup \{x \leftarrow y_1, \ldots, y_m\}$.

Lemma B.2 *(29) Let P be a normal program, let $r_1, r_2, ..., r_n$ be definite rules such that the two programs P and $P \cup r_i$ have the same answer sets for any $i \in \{1, 2..., n\}$. Then P and $P \cup r_1 \cup r_2 \cup ... \cup r_n$ have the same answer sets.*

The following lemma is a straightforward result by the properties of *negation as failure* of the answer set semantics.

[1]The proofs of Lemma B.1 and Lemma B.2 were done in collaboration with José Luis Carballido in (29).

Lemma B.3 *Let P be a normal program such that there exist P_1 and P_2 satisfying:*

1. *$P = P_1 \cup P_2$ and $P_1 \cap P_2 = \{\}$.*
2. *The atoms in the head of P_1 do not occur in P_2.*
3. *The atoms in the body of P_1 do not occur in the head of P_1.*

Then M is an answer set of P if and only if there exist M_1 and M_2 such that $M = M_1 \cup M_2$, M_2 is an answer set of P_2 and $M_1 = \{x : x \leftarrow \alpha \in P_1, M_2(\alpha) = 1\}$.

By considering Lemma B.1, Lemma B.2 and Lemma B.3, we present the proof of Lemma 4.2.

Lemma 4.2 *Let AF be an argumentation framework and E a set of arguments. E is a stable extension of AF if and only if $compl(E)$ is a answer set of Ψ_{AF}.*
Proof: Let P be the grounding of the program P_{AF} (see Definition 2.13). Hence, P is of the form:

$$P = \bigcup_{b:(b,a) \in attacks} \{d(a) \leftarrow acc(b)\} \cup \bigcup_{a \in AR} \{acc(a) \leftarrow \ not\ d(a)\}$$

Now let P' be the program obtained from P by applying the well-know principle of partial evaluation (PPE) (36) to P:

$$P' = \bigcup_{b:(b,a) \in attacks} \{d(a) \leftarrow \ not\ d(b)\} \cup \bigcup_{a \in AR} \{acc(a) \leftarrow \ not\ d(a)\}$$

Let $P' = P'_2 \cup P'_1$ such that $P'_1 = \bigcup_{a \in AR} \{acc(a) \leftarrow \ not\ d(a)\}$ and
$P'_2 = \bigcup_{b:(b,a) \in attacks} \{d(a) \leftarrow \ not\ d(b)\}$. Since the answer set semantics is closed under PPE (36), M is an answer set of P if and only if M is answer set of P'. By Lemma B.3, M is answer set of P' if and only if $M = M_2 \cup M_1$ such that M_2 is an answer set of P'_2 and $M_1 = \{acc(a) | acc(a) \leftarrow \ not\ d(a) \in P'_1$ and $d(a) \notin M_2\}$. Hence, by Theorem 2.1, E is a stable extension of AF if and only if $compl(E)$ is an answer set of P'_2 (observe that $compl(E) = M_2$).

By Lemma B.1 and Lemma B.2, M_2 is an answer set of P'_2 if and only M_2 is an answer set of $P'_2 \cup \bigcup_{b:(b,a) \in attacks} \{d(a) \leftarrow \bigwedge_{c:(c,b) \in attacks} d(c)\}$. Therefore, E is a stable extension of AF iff $compl(E)$ is an answer set of P'_2iff $compl(E)$ is an answer set of Ψ_{AF}. ■

Proof of Lemma 4.3

Since our formalization of the characterization of the preferred semantics by the pstable semantics will consider Theorem 4.4 and this theorem is based on positive disjunctive logic programs and the answer set semantics, we will introduce a lemma which will show how to recover the answer sets of a positive disjunctive logic program by considering pstable models.

As we know by Definition 2.4, the pstable semantics is defined for normal programs. Hence, for a disjunctive positive rule

$$r = a_1 \vee a_2 \vee ... \vee a_s \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

we define the closure of r as the union of the s rules:

$$a_i \leftarrow \wedge b_k \bigwedge (\wedge \; not \; a_j)_{\{j \neq i\}}$$

for $i : 1, 2, \ldots, s$ and $k : 1, 2, \ldots, n$. In the case in which the rule r is normal, its closure is the same r. The closure $CL(P)$, of a disjunctive program P, is the union of the closures of its rules. Observe that this program is normal. Our lemma then says[1]:

Lemma B.4 *(29) For a positive disjunctive program P, one can recover its stable models from the pstable models of $CL(P)$, specifically, the stable models of P are exactly the pstable models of $CL(P)$.*

By considering Lemma B.4, we will prove our Lemma 4.3.

Lemma 4.3 *Let AF be an argumentation framework and E a set of arguments. E is a preferred extension of AF if and only if $compl(E)$ is a pstable model of Ψ_{AF}.*

Proof: Let P be Ψ_{AF} and $CLO(P)$ be $P \cup \{x \leftarrow \; not \; y : y \leftarrow \; not \; x \in P\}$. By Theorem 4.4 and Lemma B.4, one can immediately see that E is a preferred extension of AF iff *compl*(E) is a pstable model of $CLO(P)$ (observe that $CL(\Gamma_{AF}) = CLO(P)$). We will now see that M is a pstable model of P iff M is a pstable model of $CLO(P)$.

Suppose that M is a pstable of P. It is immediate to see that M is a pstable model of $CLO(P)$.

The interesting case is the converse. Suppose that M is a pstable model of $CLO(P)$ and let us try to prove that M is a pstable model of P. By our assumption and the definition of pstable model, M is a model of $CLO(P)$ and

[1]The proof of Lemma B.4 was done in collaboration with José Luis Carballido in (29).

$RED(CLO(P), M) \models M$. Clearly M is a model of P. We only need to prove that $RED(P, M) \models M$. This follows since $RED(P, M) \models RED(CLO(P), M)$ and this finishes this first part of the proof. It is worth to explain the last step in some more detail.

In order to prove that $RED(P, M) \models RED(CLO(P), M)$, let r_1 be any rule that belongs to $CLO(P)$ such that it does not belong to P. Then r_1 must be of the form $x \leftarrow$ *not* y. Clearly r_2 (of the form $y \leftarrow$ *not* x) belongs to both P and $CLO(P)$. Let s_1 be the result of r_1 after the reduction RED. Similarly, let s_2 be the result of r_2 after the reduction RED. So, $s_2 \in (RED(P, M) \cap RED(CLO(P), M))$ and $s_1 \in RED(CLO(P), M)$. We have two cases:
A) $r_1 = s_1$. Then clearly $s_2 \models s_1$. Hence $RED(P, M) \models s_1$.
B) Suppose that r_1 is different from s_1. Then $r_2 = s_2$. In addition s_1 should be the fact x which belongs to $RED(CLO(P), M)$. Necessarily $y \notin M$. Let $x \leftarrow$ *not* z_1, ...,$x \leftarrow$ *not* z_m be all the non definite rules of P such that x is in the head. Then $y \leftarrow z_1, ..., z_m \in P$. Since M is a model of P then there exists $z \in \{z_1, ..., z_m\}$ such that $z \notin M$. Hence $x \leftarrow$ *not* $z \in P$ and so x is a fact of $RED(P, M)$. Hence, $x \models s_1$ and so $RED(P, M) \models s_1$. From both cases $RED(P, M) \models s_1$, so it follows that $RED(P, M) \models RED(CLO(P), M)$.

As a consequence of the above reasoning E is a preferred extension of AF iff $compl(E)$ is a pstable model of P. ■

Theorem 4.1 *Ψ_{AF} is a suitable codification.*
Proof: The theorem follows of Lemma 4.1, Lemma 4.2, Lemma 4.3, and the fact that Ψ_{AF} is polynomial time computable function. ■

Proposition 4.1 *Let T be a theory with signature $\mathcal{L}$. Let $\mathcal{L}'$ be a copy-signature of $\mathcal{L}$. By $g(T)$ we denote the theory obtained from T by replacing every occurrence of an atom x in T by $\sim f(x)$. Then M is a maximal model of T if and only if $f(\mathcal{L} \setminus M)$ is a minimal model of $g(T)$.*
Proof: First of all two observations:

1. Given $M_1, M_2 \subseteq \mathcal{L}_T$, it is true that $M_1 \subset M_2$ iff $f(\mathcal{L}_T \setminus M_2) \subset f(\mathcal{L}_T \setminus M_1)$.

2. Given a propositional formula A, an interpretation M from $\mathcal{L}_T$ to $\{0, 1\}$ and $x \in \{0, 1\}$. Then it is not difficult to prove by induction on A's length[1] that $M(A) = x$ iff $f(\mathcal{L}_T \setminus M)(g(A)) = x$.

=> To prove that if M is a maximal model of T then $f(\mathcal{L} \setminus M)$ is a minimal model of $g(T)$. The proof is by contradiction. Let us suppose that M is a

[1]Since A is a disjunctive clause, the length of A is given by the number of atoms in the head of A plus the number of literals in the body of A.

maximal model of T but $f(\mathfrak{L} \setminus M)$ is a model of $g(T)$ and is not minimal. Then if $f(\mathfrak{L} \setminus M)$ is not minimal then there exists M_2 such that $f(\mathfrak{L} \setminus M_2)$ is a model of $g(T)$ and $f(\mathfrak{L} \setminus M_2) \subset f(\mathfrak{L} \setminus M)$. Then by observation 2, if $f(\mathfrak{L} \setminus M_2)$ is a model of $g(T)$ then M_2 is a model of T. By observation 1, if $f(\mathfrak{L} \setminus M_2) \subset f(\mathfrak{L} \setminus M)$ then $M \subset M_2$. But this is a contradiction because M is a maximal model of T.

<= To prove that if $f(\mathfrak{L} \setminus M)$ is a minimal model of $g(T)$ then M is a maximal model of T. The proof is also by contradiction. Let us suppose that $f(\mathfrak{L} \setminus M)$ is a minimal model of $g(T)$ but M is model of T and is not maximal. If M is not maximal, then exists a model M_2 of T such that $M \subset M_2$. Then by observation 2, if M_2 is a model of T then $f(\mathfrak{L} \setminus M_2)$ is a model of $g(T)$. By observation 1, if $M \subset M_2$ then $f(\mathfrak{L} \setminus M_2) \subset f(\mathfrak{L} \setminus M)$. But this is a contradiction because $f(\mathfrak{L} \setminus M)$ is a minimal model of $g(T)$.

■

Theorem 4.2 *Let $AF := \langle AR, attacks\rangle$ be an argumentation framework and $S \subseteq AR$. When the mapping $f(x)$ of the theory $g(\beta(AF))$ corresponds to $d(x)$ such that $x \in AR$, the following condition holds: S is a preferred extension of AF if and only if compl(S) is a minimal model of $\alpha(AF)$.*

Proof: Two observations:

1. Since the mapping $f(x)$ corresponds to $d(x)$, then *compl*$(S) = f(AR \setminus S)$ because *compl*$(S) := \{d(a) | a \in AR \backslash S\}$ and $f(AR \backslash S) := \{f(a) | a \in AR \backslash S\}$.

2. $\alpha(AF)$ is logically equivalent to $g(\beta(AF))$:

$$g(\beta(AF)) = \bigwedge_{a\in AR}((\sim d(a) \rightarrow \bigwedge_{b:(b,a)\in attacks} d(b)) \wedge (\sim d(a) \rightarrow \bigwedge_{b:(b,a)\in attacks}(\bigvee_{c:(c,b)\in attacks} \sim d(c))))$$

Since $a \rightarrow \bigwedge_{b\in S} b \equiv \bigwedge_{b\in S}(a \rightarrow b)$, we get:

$$\bigwedge_{a\in AR} (\bigwedge_{b:(b,a)\in attacks} (\sim d(a) \rightarrow d(b)) \wedge (\bigwedge_{b:(b,a)\in attacks} (\sim d(a) \rightarrow \bigvee_{c:(c,b)\in attacks} \sim d(c)))$$

By applying transposition and cancelation of double negation in both implications, we get:

$$\bigwedge_{a\in AR} (\bigwedge_{b:(b,a)\in attacks} (\sim d(b) \rightarrow d(a)) \wedge (\bigwedge_{b:(b,a)\in attacks} (\sim \bigvee_{c:(c,b)\in attacks} \sim d(c) \rightarrow d(a)))$$

Now, for the right hand side of the formula we need to apply Morgan laws:

$$\bigwedge_{a \in AR} (\bigwedge_{b:(b,a) \in attacks} (\sim d(b) \rightarrow d(a)) \wedge (\bigwedge_{b:(b,a) \in attacks} (\bigwedge_{c:(c,b) \in attacks} d(c) \rightarrow d(a))))$$

Finally by changing $\rightarrow$ by $\leftarrow$, we get $\alpha(AF)$.

$$\bigwedge_{a \in AR} (\bigwedge_{b:(b,a) \in attacks} (d(a) \leftarrow \sim d(b)) \wedge (\bigwedge_{b:(b,a) \in attacks} (d(a) \leftarrow \bigwedge_{c:(c,b) \in attacks} d(c)))) = \alpha(AF)$$

Now the main proof: S is a preferred extension of AF iff (by Proposition 4.2) S is a maximal model of $\beta(AF)$ iff (by Proposition 4.1) $f(AR \setminus S)$ is a minimal model of $g(\beta(AF))$ iff (by observations 1 and 2) $compl(S)$ is a minimal model of $\alpha(AF)$.

■

Proposition 4.3 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $S \subseteq AR$. S is a preferred extension of AF if and only if $compl(S)$ is a model of $\alpha(AF)$ and*

$$\alpha(AF) \wedge SetToFormula(\sim \widetilde{compl(S)}) \models SetToFormula(compl(S))$$

Proof: First of all, let us introduce the following relationship between minimal models and logic consequence.

Lemma B.5 *(90) For a given general program P, M is a model of P and $P \cup \neg\widetilde{M}) \models M$ iff M is a minimal model of P.*

This lemma was introduced in terms of augmented programs. Since a general program is a particular case of an augmented program, we write the lemma in terms of general programs (see (90) for more details about augmented programs). Main Proof: S is a preferred extension of AF iff (by Theorem 4.2) $compl(S)$ is a minimal model of $\alpha(AF)$ iff (by lemma B.5) $compl(S)$ is a model of $\alpha(AF)$ and $\alpha(AF) \wedge SetToFormula(\neg\widetilde{compl(S)}) \models SetToFormula(compl(S))$. ■

Theorem 4.3 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $S \subseteq AR$. S is a preferred extension of AF if and only if $compl(S)$ is a model of $\alpha(AF)$ and*

$$\alpha(AF) \wedge SetToFormula(\sim \widetilde{compl(S)}) \wedge \neg SetToFormula(compl(S))$$

is unsatisfiable.
Proof: Directly, by Proposition 4.3. ■

Theorem 4.4 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $S \subseteq AR$. S is a preferred extension of AF if and only if $compl(S)$ is an answer set of Γ_{AF}.*
Proof: S is a preferred extension of AF iff *compl(S)* is a minimal model of $\alpha(AF)$ (by Theorem 4.2) iff $compl(S)$ is a minimal model of Γ_{AF} (since Γ_{AF} is logically equivalent to $\alpha(AF)$ in classical logic) iff *compl(S)* is an answer set of Γ_{AF} (since Γ_{AF} is a positive disjunctive logic program and for every positive disjunctive logic program P, M is an answer set of P iff M is a minimal model of P). ■

Proposition 4.4 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $S \subseteq AR$. S is a preferred extension of AF iff there is a stable model M of Λ_{AF} such that $S = M \cap AR$.*
Proof: The proof is straightforward from Theorem 4.4 and the semantics of default negation. ■

Theorem 4.5 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and E be the grounded extension of AF. Then*

a)
1. *If $\langle S, D \rangle$ is the $WFS^{LLC'}$-extension of AF then $E \subseteq S$.*
2. *If $\langle S, D \rangle$ is the WFS^{WK}-extension of AF then $E \subseteq S$.*
3. *If $\langle S, D \rangle$ is the $WFS^{WK+LLC'}$-extension of AF then $E \subseteq S$.*

b)
1. *The $WFS^{LLC'}$-extension of AF is polynomial time computable.*
2. *The WFS^{WK}-extension of AF is polynomial time computable.*
3. *The $WFS^{WK+LLC'}$-extension of AF is polynomial time computable.*

Proof:

a) It is direct by Lemma 4.1.

b) The result follows from the facts that: 1.- Thee mapping Ψ_{AF} is a polynomial time computable function, and 2.- The rewriting systems CS_1, CS_2, and CS_3 are polynomial time computable (39).

■

Theorem 4.6 *Let $AF := \langle AR, attacks \rangle$ be an argumentation framework, E be a preferred extension of AF, and $E' := AR \setminus E$. Then,*

1. *If $\langle S, D\rangle$ is the $WFS^{LLC'}$-extension of AF then $S \subseteq E$ and $D \subseteq E'$.*

2. *If $\langle S, D\rangle$ is the WFS^{WK}-extension of AF then $S \subseteq E$ and $D \subseteq E'$.*

3. *If $\langle S, D\rangle$ is the $WFS^{WK+LLC'}$-extension of AF then $S \subseteq E$ and $D \subseteq E'$.*

Proof: (sketch)
This theorem follows from the facts that

1. The pstable semantics satisfies the basic transformation RED^{+}, RED^{-}, $Success$, $Failure$, $Loop$, WK, and LLC' (this means that any atom inferred by these transformations is also inferred by Pstable semantics. This fact is essentially because these transformation are closed under classic logic),

2. The pstable models of Ψ_{AF} correspond to the preferred extensions of AF (Lemma 4.3).

■