UNIVERSITAT POLITÈCNICA DE CATALUNYA

DEPT. LLENGUATGES I SISTEMES INFORMÀTICS (LSI)

Ph.D. Program: Artificial Intelligence

The Role of Preferences in Logic Programming: Nonmonotonic Reasoning, User Preferences, Decision under Uncertainty

Ph.D. Candidate:
ROBERTO CONFALONIERI

Advisor:
Dr. JAVIER VÁZQUEZ-SALCEDA

Co-advisor:
Dr. JUAN CARLOS NIEVES SÁNCHEZ

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

September 2011

To my family

Acknowledgements

I would like to thank all the people who have supported me during this period of my life.

First of all, I would like to thank my supervisors Dr. Javier Vázquez-Salceda and Dr. Juan Carlos Nieves for their valuable suggestions.

I would like to thank Dr. Henri Prade of the Institut de Recherche Informatique de Toulouse (IRIT) of the Universitatè Paul Sabatier in Toulouse (France) for his guidance during my research stage at IRIT.

I would like to thank all the people I met in Spain and France for their friendship. Studying, working, and living abroad have been part of a unique and challenging experience that has enriched me from a professional and a human point of view.

Last but not least, a special thank to Karol. I would like to thank her for her constant love and support. I am indebted to her for her tireless encouragement.

This work has been partially funded by the European Commission through the IST-034418 CONTRACT project and the FP7-215890 ALIVE project.

Abstract

Intelligent systems that assist users in fulfilling complex tasks need a concise and processable representation of incomplete and uncertain information. In order to be able to choose among different options, these systems also need a compact and processable representation of the concept of preference.

Preferences can provide an effective way to choose the best solutions to a given problem. These solutions can represent the most plausible states of the world when we model incomplete information, the most satisfactory states of the world when we express user preferences, or optimal decisions when we make decisions under uncertainty.

Several domains, such as, reasoning under incomplete and uncertain information, user preference modeling, and qualitative decision making under uncertainty, have benefited from advances on preference representation. In the literature, several symbolic approaches of nonclassical reasoning have been proposed. Among them, logic programming under answer set semantics offers a good compromise between symbolic representation and computation of knowledge and several extensions for handling preferences.

Nevertheless, there are still some open issues to be considered in logic programming. In nonmonotonic reasoning, first, most approaches assume that exceptions to logic program rules are already specified. However, sometimes, it is possible to consider implicit preferences based on the specificity of the rules to handle incomplete information. Secondly, the joint handling of exceptions and uncertainty has received little attention: when information is uncertain, the selection of default rules can be a matter of explicit preferences and uncertainty. In user preference modeling, although existing logic programming specifications allow to express user preferences which depend both on incomplete and contextual information, in some applications, some preferences in some context may be more important than others. Furthermore, more complex preference expressions need to be supported. In qualitative decision making under uncertainty, existing logic programming-based methodologies for making decisions seem to lack a satisfactory handling of preferences and uncertainty.

The aim of this dissertation is twofold: 1) to tackle the role played by preferences in logic programming from different perspectives, and 2) to contribute to this novel field by proposing several frameworks and methods able to address the above issues. To this end, we will first show how preferences can be used to select default rules in logic programs in an implicit and explicit way. In particular, we propose (i) a method for selecting logic

program rules based on specificity, and (ii) a framework for selecting uncertain default rules based on explicit preferences and the certainty of the rules. Then, we will see how user preferences can be modeled and processed in terms of a logic program (iii) in order to manage user profiles in a context-aware system and (iv) in order to propose a framework for the specification of nested (non-flat) preference expressions. Finally, in the attempt to bridge the gap between logic programming and qualitative decision under uncertainty, (v) we propose a classical- and a possibilistic-based logic programming methodology to compute an optimal decision when uncertainty and preferences are matters of degrees.

Resum

Els sistemes intel·ligents que assisteixen a usuaris en la realització de tasques complexes necessiten una representació concisa i formal de la informació que permeti un raonament no-monòton en condicions d'incertesa. Per a poder escollir entre les diferents opcions, aquests sistemes solen necessitar una representació del concepte de preferència.

Les preferències poden proporcionar una manera efectiva de triar entre les millors solucions a un problema. Aquestes solucions poden representar els estats del món més plausibles quan es tracta de modelar informació incompleta, els estats del món més satisfactori quan expressem preferències de l'usuari, o decisions òptimes quan estem parlant de presa de decisió incorporant incertesa.

L'ús de les preferències ha beneficiat diferents dominis, com, el raonament en presència d'informació incompleta i incerta, el modelat de preferències d'usuari, i la presa de decisió sota incertesa. En la literatura, s'hi troben diferents aproximacions al raonament no clàssic basades en una representació simbòlica de la informació. Entre elles, l'enfocament de programació lògica, utilitzant la semàntica de *answer set*, ofereix una bona aproximació entre representació i processament simbòlic del coneixement, i diferents extensions per gestionar les preferències.

No obstant això, en programació lògica es poden identificar diferents problemes pel que fa a la gestió de les preferències. Per exemple, en la majoria d'enfocaments de raonament no-monòton s'assumeix que les excepcions a *default rules* d'un programa lògic ja estan expressades. Però de vegades es poden considerar preferències implícites basades en l'especificitat de les regles per gestionar la informació incompleta. A més, quan la informació és també incerta, la selecció de *default rules* pot dependre de preferències explícites i de la incertesa. En el modelatge de preferències del usuari, encara que els formalismes existents basats en programació lògica permetin expressar preferències que depenen d'informació contextual i incompleta, en algunes aplicacions, donat un context, algunes preferències poden ser més importants que unes altres. Per tant, resulta d'interès un llenguatge que permeti capturar preferències més complexes. En la presa de decisions sota incertesa, les metodologies basades en programació lògica creades fins ara no ofereixen una solució del tot satisfactòria pel que fa a la gestió de les preferències i la incertesa.

L'objectiu d'aquesta tesi és doble: 1) estudiar el paper de les preferències en la programació lògica des de diferents perspectives, i 2) contribuir a aquesta jove àrea d'investigació proposant diferents marcs teòrics i mètodes per abordar els problemes anteriorment citats.

Per a aquest propòsit veurem com les preferències es poden utilitzar de manera implícita i explícita per a la selecció de default rules proposant: (i) un mètode basat en l'especificitat de les regles, que permeti seleccionar regles en un programa lògic; (ii) un marc teòric per a la selecció de *default rules* incertes basat en preferències explícites i la incertesa de les regles. També veurem com les preferències de l'usuari poden ser modelades i processades usant un enfocament de programació lògica (iii) que suporti la creació d'un mecanisme de gestió dels perfils dels usuaris en un sistema amb reconeixement del context; (iv) que permeti proposar un marc teòric capaç d'expressar preferències amb fòrmules imbricades. Per últim, amb l'objectiu de disminuir la distància entre programació lògica i la presa de decisió amb incertesa proposem (v) una metodologia basada en programació lògica clàssica i en una extensió de la programació lògica que incorpora lògica possibilística per modelar un problema de presa de decisions i per inferir una decisió òptima.

Resumen

Los sistemas inteligentes que asisten a usuarios en tareas complejas necesitan una representación concisa y procesable de la información que permita un razonamiento no-monótono e incierto. Para poder escoger entre las diferentes opciones, estos sistemas suelen necesitar una representación del concepto de preferencia.

Las preferencias pueden proporcionar una manera efectiva para elegir entre las mejores soluciones a un problema. Dichas soluciones pueden representar los estados del mundo más plausibles cuando hablamos de representación de información incompleta, los estados del mundo más satisfactorios cuando hablamos de preferencias del usuario, o decisiones óptimas cuando estamos hablando de toma de decisión con incertidumbre.

El uso de las preferencias ha beneficiado diferentes dominios, como, razonamiento en presencia de información incompleta e incierta, modelado de preferencias de usuario, y toma de decisión con incertidumbre. En la literatura, distintos enfoques simbólicos de razonamiento no clásico han sido creados. Entre ellos, la programación lógica con la semántica de *answer set* ofrece un buen acercamiento entre representación y procesamiento simbólico del conocimiento, y diferentes extensiones para manejar las preferencias.

Sin embargo, en programación lógica se pueden identificar diferentes problemas con respecto al manejo de las preferencias. Por ejemplo, en la mayoría de enfoques de razonamiento no-monótono se asume que las excepciones a *default rules* de un programa lógico ya están expresadas. Pero, a veces se pueden considerar preferencias implícitas basadas en la especificidad de las reglas para manejar la información incompleta. Además, cuando la información es también incierta, la selección de *default rules* pueden depender de preferencias explícitas y de la incertidumbre. En el modelado de preferencias, aunque los formalismos existentes basados en programación lógica permitan expresar preferencias que dependen de información contextual e incompleta, in algunas aplicaciones, algunas preferencias en un contexto puede ser más importantes que otras. Por lo tanto, un lenguaje que permita capturar preferencias más complejas es deseable. En la toma de decisiones con incertidumbre, las metodologías basadas en programación lógica creadas hasta ahora no ofrecen una solución del todo satisfactoria al manejo de las preferencias y la incertidumbre.

El objectivo de esta tesis es doble: 1) estudiar el rol de las preferencias en programación lógica desde diferentes perspectivas, y 2) contribuir a esta joven área de investigación proponiendo diferentes marcos teóricos y métodos para abordar los problemas anteriormente citados. Para este propósito veremos como las preferencias pueden ser usadas de man-

era implícita y explícita para la selección de *default rules* proponiendo: (i) un método para seleccionar reglas en un programa basado en la especificad de las reglas; (ii) un marco teórico para la selección de default rules basado en preferencias explícitas y incertidumbre. También veremos como las preferencias del usuario pueden ser modeladas y procesadas usando un enfoque de programación lógica (iii) para crear un mecanismo de manejo de los perfiles de los usuarios en un sistema con reconocimiento del contexto; (iv) para crear un marco teórico capaz de expresar preferencias con formulas anidadas. Por último, con el objetivo de disminuir la distancia entre programación lógica y la toma de decisión con incertidumbre proponemos (v) una metodología para modelar un problema de toma de decisiones y para inferir una decisión óptima usando un enfoque de programación lógica clásica y uno de programación lógica extendida con lógica posibilística.

Sommario

Sistemi intelligenti, destinati a fornire supporto agli utenti in processi decisionali complessi, richiedono una rappresentazione dell'informazione concisa, formale e che permetta di ragionare in maniera non monotona e incerta. Per poter scegliere tra le diverse opzioni, tali sistemi hanno bisogno di disporre di una rappresentazione del concetto di preferenza altrettanto concisa e formale.

Le preferenze offrono una maniera efficace per scegliere le miglior soluzioni di un problema. Tali soluzioni possono rappresentare gli stati del mondo più credibili quando si tratta di ragionamento non monotono, gli stati del mondo più soddisfacenti quando si tratta delle preferenze degli utenti, o le decisioni migliori quando prendiamo una decisione in condizioni di incertezza.

Diversi domini come ad esempio il ragionamento non monotono e incerto, la strutturazione del profilo utente, e i modelli di decisione in condizioni d'incertezza hanno tratto beneficio dalla rappresentazione delle preferenze. Nella bibliografia disponibile si possono incontrare diversi approcci simbolici al ragionamento non classico. Tra questi, la programmazione logica con *answer set semantics* offre un buon compromesso tra rappresentazione simbolica e processamento dell'informazione, e diversi estensioni per la gestione delle preferenze sono state proposti in tal senso.

Nonostante ció, nella programmazione logica esistono ancora delle problematiche aperte. Prima di tutto, nella maggior parte degli approcci al ragionamento non monotono, si suppone che nel programma le eccezioni alle regole siano già specificate. Tuttavia, a volte per trattare l'informazione incompleta è possibile prendere in considerazione preferenze implicite basate sulla specificità delle regole. In secondo luogo, la gestione congiunta di eccezioni e incertezza ha avuto scarsa attenzione: quando l'informazione è incerta, la scelta di *default rule* può essere una questione di preferenze esplicite e d'incertezza allo stesso tempo. Nella creazione di preferenze dell'utente, anche se le specifiche di programmazione logica esistenti permettono di esprimere preferenze che dipendono sia da un'informazione incompleta che da una contestuale, in alcune applicazioni talune preferenze possono essere più importanti di altre, o espressioni più complesse devono essere supportate. In un processo decisionale con incertezza, le metodologie basate sulla programmazione logica viste sinora, non offrono una gestione soddisfacente delle preferenze e dell'incertezza.

Lo scopo di questa dissertazione è doppio: 1) chiarire il ruolo che le preferenze giocano nella programmazione logica da diverse prospettive e 2) contribuire proponendo in

questo nuovo settore di ricerca, diversi *framework* e metodi in grado di affrontare le citate problematiche. Per prima cosa, dimostreremo come le preferenze possono essere usate per selezionare *default rule* in un programma in maniera implicita ed esplicita. In particolare proporremo: (i) un metodo per la selezione delle regole di un programma logico basato sulla specificità dell'informazione; (ii) un *framework* per la selezione di *default rule* basato sulle preferenze esplicite e sull'incertezza associata alle regole del programma. Poi, vedremo come le preferenze degli utenti possono essere modellate attraverso un programma logico, (iii) per creare il profilo dell'utente in un sistema *context-aware*, e (iv) per proporre un *framework* che supporti la definizione di preferenze complesse. Infine, per colmare le lacune in programmazione logica applicata a un processo di decisione con incertezza (v) proporremo una metodologia basata sulla programmazione logica classica e una metodologia basata su un'estensione della programmazione logica con logica possibilistica.

Contents

List of Figures XXI

List of Tables XXIII

Glossary XXV

1 Introduction 1
- 1.1 Preferences in Nonmonotonic Reasoning and Uncertainty 2
- 1.2 User Preference Modeling 5
- 1.3 Preferences in Decision under Uncertainty 6
- 1.4 Thesis Objectives 8
  - 1.4.1 Preferences in Nonmonotonic Reasoning and Uncertainty 8
  - 1.4.2 User Preference Modeling 10
  - 1.4.3 Preferences in Decision under Uncertainty 11
- 1.5 Thesis Structure 13

2 Background, Notation and Classes of Logic Programs Considered 17
- 2.1 Answer Set Programming 17
  - 2.1.1 Syntax 17
  - 2.1.2 Interpretations and Models 19
  - 2.1.3 Answer Set Semantics 19
- 2.2 Extensions 21
  - 2.2.1 Disjunctive Logic Programs 21
  - 2.2.2 Generalized Disjunctive Logic Programs 22
  - 2.2.3 Nested Logic Programs 22
  - 2.2.4 Logic Programs with Ordered Disjunction 23
- 2.3 Rewriting Systems 25
- 2.4 Possibilistic Logic 26
- 2.5 Possibilistic Answer Set Programming 28
- 2.6 Classes of Logic Programs Considered in this Document 29

Part I Preferences in Nonmonotonic Reasoning and Uncertainty

3 Handling Incomplete and Uncertain Information . . . 33
3.1 Seminal Approaches to Nonmonotonic Reasoning . . . 33
3.1.1 Default Logic . . . 34
3.1.2 Circumscription . . . 35
3.1.3 Autoepistemic Logic . . . 36
3.2 Dealing with Incomplete Information in Possibility Theory . . . 37
3.2.1 Handling Default Rules in Possibilistic Logic . . . 37
3.2.2 Other Approaches . . . 38
3.3 Dealing with Incomplete Information in Logic Programming . . . 39
3.3.1 Implicit Preferences in Logic Programs . . . 40
3.3.2 Explicit Preferences in Logic Programs . . . 41
3.3.3 LPODs: An Approach for Context-dependent Preferences Among Literals . . . 42
3.4 Dealing with Incomplete Information and Uncertainty . . . 43
3.4.1 Fuzzy Answer Set Programming . . . 43
3.4.2 Probabilistic Answer Set Programming . . . 44
3.4.3 Possibilistic Answer Set Programming . . . 44
3.5 Discussion . . . 45

4 An Approach to Nonmonotonic Reasoning using Implicit Preferences . . . 47
4.1 Preliminaries . . . 48
4.2 Handling Default Reasoning . . . 49
4.2.1 Ordering Rules by Specificity . . . 50
4.2.2 Rules Rewriting . . . 51
4.2.3 Consistency Test . . . 54
4.2.4 Minimal Model Computation . . . 55
4.3 Related Work . . . 57
4.4 Discussion and Concluding Remarks . . . 59

5 LPPODs: a Framework for Dealing with Explicit Preferences and Uncertainty . . . 63
5.1 LPPODs Syntax . . . 66
5.2 LPPODs Semantics . . . 67
5.2.1 Possibilistic Answer Sets of LPPODs by Fix-Point . . . 67
5.2.2 Possibilistic Answer Sets of LPPODs by Possibility Distribution . . . 70
5.3 Transformation Rules for LPPODs . . . 73
5.4 Possibilistic Answer Sets Selection . . . 76
5.5 Computing the LPPODs Semantics . . . 79
5.6 Related Work . . . 80
5.6.1 Uncertainty in Logic Programming . . . 81
5.6.2 Preferences in Logic Programming . . . 82
5.7 Discussion and Concluding Remarks . . . 83

Part II User Preference Modeling

6 User Preference Handling ... 89
6.1 User Preferences in Qualitative Choice Logic ... 89
6.2 User Preferences in Possibilistic Logic ... 91
6.3 User Preferences in Conditional Preference Networks ... 92
6.4 User Preferences in Logic Programming ... 93
6.4.1 Logic Programs with Ordered Disjunction ... 94
6.4.2 Logic Programs with Ordered and Unordered Disjunction ... 95
6.4.3 Answer Set Optimization Programs ... 96
6.5 Context-aware Systems ... 97
6.5.1 User Profile ... 98
6.5.2 Context ... 99
6.6 Discussion ... 100

7 Handling User Preferences in a Context-Aware System ... 103
7.1 Use Case and Requirements ... 104
7.2 User Profile Adaptation Process and Context-Dependent Preferences ... 105
7.3 User Preferences and Profile Ontology ... 108
7.4 Mapping User Profiles to LPPODs ... 108
7.5 User Preference Representation ... 109
7.6 User Preference Reasoning ... 112
7.6.1 LPPODs Solver ... 112
7.6.2 User Preference Computation and Ordering ... 113
7.7 Retrieving Personalized Suggestions ... 115
7.8 Related Work ... 117
7.8.1 Database Preference Queries ... 118
7.8.2 Preference Queries in Possibilistic Databases ... 119
7.9 Discussion and Concluding Remarks ... 121

8 LPODs$^+$: A Framework for Handling Nested Preferences ... 123
8.1 LPODs$^+$ Syntax ... 124
8.2 LPODs$^+$ Semantics ... 126
8.3 LPODs$^+$ Answer Sets Selection ... 127
8.4 Implementation Design ... 129
8.4.1 Mapping LPODs$^+$ to Disjunctive Logic Programs ... 129
8.4.2 Optimal Answer Set Computation ... 132
8.5 Related Work ... 134
8.6 Discussion and Concluding Remarks ... 136

Part III Preferences in Decision under Uncertainty

9 Qualitative Decision Making Under Uncertainty ... 139
9.1 Classical Decision Theory ... 139
9.2 Qualitative Decision Making ... 140
9.2.1 Possibilistic Semantics of Qualitative Decision ... 141

9.2.2 Other Approaches . . . 142
9.3 Modeling Qualitative Decision in Possibilistic Logic . . . 142
9.3.1 Qualitative Decision in Propositional Bases . . . 142
9.3.2 Qualitative Decision in Stratified Propositional Bases . . . 143
9.4 Modeling Qualitative Decision in Logic Programming . . . 144
9.4.1 Answer Sets and Qualitative Decision Making . . . 144
9.4.2 Qualitative Model of Decision Making . . . 144
9.5 Discussion . . . 145

10 An ASP-based Methodology for Computing Decision Under Uncertainty . . . 147
10.1 Modeling Decision Making in ASP . . . 148
10.1.1 Abduction and LPODs . . . 148
10.1.2 Fully Certain Knowledge and All-or-nothing Preferences . . . 149
10.1.3 Computation of Decisions . . . 150
10.2 Modeling Decision Making Under Uncertainty in ASP . . . 153
10.2.1 Uncertain Knowledge and Prioritized Preferences . . . 153
10.2.2 Classical ASP-based Computation . . . 154
10.2.3 Possibilistic ASP-based Computation . . . 158
10.3 Related Work . . . 163
10.4 Discussion and Concluding Remarks . . . 164

Part IV Conclusion and Future Perspectives

11 Conclusion . . . 169
11.1 Summary . . . 169
11.2 Possible Extensions and Future Work . . . 175

Publications of the Author . . . 177

A Proofs . . . 179
A.1 Proofs of Chapter 4 . . . 179
A.2 Proofs of Chapter 5 . . . 180
A.3 Proofs of Chapter 8 . . . 186
A.4 Proofs of Chapter 10 . . . 189

References . . . 191

Index . . . 207

List of Figures

2.1 Logic Programs Considered 30

6.1 Evening Dressing Example [Boutilier et al., 2004] 92

7.1 Overview of User Profile Adaptation Process 106
7.2 Preference Ontology 107
7.3 Ontology for Rules containing User Preferences related to Context 108
7.4 Ontology for LPPODs 109
7.5 MDD applied to User Preference Modeling 110
7.6 Examples of User Profiles (*Citizen* and *Tourist*) encoded by LPPODs in which Red boxes denote *Requirements*, Orange boxes denote *Preferences* and Yellow boxes denote *Context*. 111
7.7 Overview of the *posPsmodels* system 112
7.8 Final GUIs showing several suggestions 117

List of Tables

5.1 Example of Rule Satisfaction Degrees 77

7.1 Relations R (top-left), S (top-right), and T (bottom) (example taken from [Bosc et al., 2010]) 120

8.1 Example of rules captured by the LPODs$^+$ syntax 125

11.1 Summary of the results in the first part 171
11.2 Summary of the results in the second part 173
11.3 Summary of the results in the third part 174

Glossary

ASP - Answer Set Programming
ASO Programs - Answer Set Optimization Programs
CP-nets - Conditional Preference Networks
DLPs - Disjunctive Logic Programs
DLPODs - Disjunctive Logic Programs with Ordered Disjunction
DM - Decision Making
DMU - Decision Making under Uncertainty
GDLPs - Generalized Disjunctive Logic Programs
ICDs - Interactive Community Displays
LPODs - Logic Programs with Ordered Disjunction
LPODs$^+$ - Nested Logic Programs with Ordered Disjunction
LPPODs - Logic Programs with Possibilistic Ordered Disjunction
OD$^+$-program - Nested Ordered Disjunction Program
NLPs - Nested Logic Programs
PASP - Possibilistic Answer Set Programming

1

Introduction

Preferences are a crucial notion guiding our choices and actions. Choosing the color of a car or choosing among different mortgage loans are examples in which preferences can guide humans from simple up to very important decisions. Preferences are a multi-disciplinary topic and they have been extensively been studied in economics, psychology, philosophy, logics and other human-centered disciplines. Nevertheless, they are a relative new topic in Artificial Intelligence (AI) [Kaci, 2011; Domshlak et al., 2011] and they have become of great interest for the development of reasoning mechanisms in intelligent systems [Brafman and Domshlak, 2009; Chen and Pu, 2004].

Intelligent systems are usually studied and developed to assist users in fulfilling complex tasks. To act autonomously and to intelligently support users, such systems require a concise and processable *representation of incomplete* and *uncertain information*. To understand user choices and to choose among different options in a desirable way, such system also need a concise and processable *representation of preferences*.

In fact, preferences can be used to select the most preferred pieces of knowledge or beliefs under which we build our reasoning in the presence of incomplete and uncertain information, *e.g., normally if it is cloudy, it is almost certain to assume that it will rain rather than it will not*; or they can represent preferred choices between several alternatives, *e.g., if I am hungry I prefer to eat in an Italian rather than a Mexican restaurant*; or they can be involved in the choice of an optimal decision when knowledge is not fully certain, *e.g., shall I take a bus or a taxi knowing that there might be a bus-strike, and that I do not like to pay extra money, but I prefer to arrive quickly to my destination*.

These simple examples suggest how preferences are intrinsically related to several issues which can be typically encountered in domains such as *reasoning under incomplete and uncertain information*, *user preference modeling*, and *qualitative decision under uncertainty*. Indeed, preferences provide an effective way to choose among outcomes, whether these are the most plausible states of the world according to pieces of default knowledge, such as in nonmonotonic reasoning [Delgrande et al., 2004; Brewka et al., 2008], or the most satisfactory states when expressing user preferences in a given context [Brewka et al., 2004a; Boutilier et al., 2004; Benferhat et al., 2001], or decisions under uncertainty, such as in qualitative decision making [Dubois et al., 2001; Dubois and Prade, 1995; Boutilier, 1994; Brafman and Tennenholtz, 1996].

In nonmonotonic reasoning, preferences can be implicit or explicit and they are used to select default rules in order to draw the most plausible conclusions. In this respect, one is interested in the most preferred (logical) models which are in agreement with a set of default rules. Moreover, when information is not fully certain, a formal way to reason about uncertain default rules is required and the selection of default rules can be a matter of certainty degrees as well. In the modeling of user preferences, preferences have to be compactly represented for encoding alternative options which can depend both on contextual and incomplete information. In decision making scenarios, preferences (and uncertainty) have to be taken into account to select an optimal choice according to a risk-prone or a risk-averse attitude.

When these decision problems want to be properly formalized and used in practice, a formalism able to represent and to reason under incomplete and uncertain knowledge, to model preferences and qualitative decision making, and that offers efficient computation must be taken into account. In the literature, several symbolic approaches of non classical reasoning have been proposed. Among them, logic programming under answer set semantics or *Answer Set Programming* (ASP) offers a good compromise between symbolic representation and computation of knowledge.

ASP is regarded to be the computational incarnation of nonmonotonic reasoning [Baral and Gelfond, 1994; Gelfond, 1994; Baral, 2003]. Indeed, its expressivity allows to reason about incomplete information and its syntax is restrictive enough for the implementation of several efficient answer set solvers [Niemelä and Simons, 1997; Leone et al., 2006; Gebser et al., 2011]. Moreover, several extensions have been proposed to deal with preferences [Delgrande et al., 2004; Van Nieuwenborgh and Vermeir, 2006], with uncertainty [Nicolas et al., 2006; Nieves et al., 2011; Baral et al., 2009; Bauters et al., 2010a] and with decision making [Brewka, 2005; Grabos, 2004]. Therefore, it is not surprising to want to apply a logic programming approach in the development of reasoning mechanisms needed by intelligent systems that support decision tasks.

Although logic programming is usually considered a powerful knowledge representation and reasoning tool and several proposals for dealing with incomplete and uncertain information, user preference modeling, and qualitative decision making have been suggested in the literature, there are still some open issues. We will describe them in the following sections.

1.1 Preferences in Nonmonotonic Reasoning and Uncertainty

Reasoning with incomplete information and reasoning with uncertainty are two important types of reasoning that have been actively investigated in order to design inference mechanisms able to draw conclusions from available information.

Reasoning with incomplete information amounts to formalize default reasoning which represents the implicit capacity of humans to reason when the available information is incomplete. Default reasoning is *nonmonotonic*, since previously drawn inferences might be withdrawn by the introduction of new information [Benferhat et al., 1997; Sombé, 1990; Brewka et al., 1993].

Several formalisms have been proposed in AI to account for default or nonmonotonic reasoning. Each formalism models *default rules*, *i.e.,* statements of the form *normally p's are q's*, in order to capture incomplete knowledge. The representation of default rules depends on the formalism considered.

In the literature, three main trends can be identified:[1]

- seminal approaches based on nonmonotonic logics such as *default logic* [Reiter, 1980], *circumscription* [McCarthy, 1980], and *autoepistemic logic* [Moore, 1985],
- approaches based on nonmonotonic inference relations of System P [Kraus et al., 1990; Benferhat et al., 1997; Lukasiewicz, 2005], and, more recently,
- approaches based on logic programming with negation as failure [Baral and Gelfond, 1994; Gelfond, 1994; Baral, 2003],
- approaches based on argumentation inference [Rahwan and Simari, 2009].

In practice, reasoning with incomplete information amounts to select default rules. In such nonmonotonic reasoning formalisms, the notion of *preference* is closely tied with the selection of default rules. Indeed, one is interested in selecting the *preferred models* (in a logical sense) which are in agreement with a set of defaults. This is usually achieved using two types of preferences: *explicit* and *implicit*.

Explicit preferences among default rules have been proposed in extended versions of nonmonotonic logics, such as *prioritized default logic* [Brewka, 1994; Delgrande and Schaub, 1997a, 2000; Rintanen, 1995, 1998], *prioritized circumscription* [Lifschitz, 1985], and *prioritized autoepistemic logic* [Konolige, 1989; Rintanen, 1994]. In these approaches, a partial order is explicitly assigned to the default rules. On the other hand, implicit preferences can be obtained from the information described by the rules themselves such as in System Z [Pearl, 1990]. This approach lies on *specificity* according to which more specific rules are always preferred to less specific ones and they are given a higher priority in the reasoning process [Pearl, 1990]. A similar ranking is used in rational closure of System P [Kraus et al., 1990]. Possibilistic logic [Dubois et al., 1994] also rank-orders default rules by assigning constraints to the least specific possibilistic distribution associated to the classical models of a possibilistic knowledge base [Benferhat et al., 1998].

Concerning logic programming, ASP models incomplete information in default rules by means of negation as failure. In ASP, the preferred answer sets of a logic program are those in which everything is as normal as possible [Baral and Gelfond, 1994; Baral, 2003]. However, when rules are in conflict, a criterion for the selection of rules is needed. To this end, several ASP extensions for handling preferences have been developed including [Gelfond and Son, 1998; Zhang and Foo, 1997; Brewka and Eiter, 1999; Sakama and Inoue, 2000; Delgrande et al., 2003; Brewka et al., 2004b; Van Nieuwenborgh and Vermeir, 2006]. Preferences can be literals-oriented such as in [Sakama and Inoue, 2000; Brewka et al., 2004b], rules-oriented such as in [Zhang and Foo, 1997; Brewka and Eiter, 1999; Delgrande et al., 2003; Van Nieuwenborgh and Vermeir, 2006], or context-dependent such as in Logic Pro-

[1] This is not a comprehensive survey. Further references can be found in [Sombé, 1990; Brewka et al., 1993; Benferhat et al., 1997; Lukasiewicz, 2005].

grams with Ordered Disjunction (LPODs) [Brewka et al., 2004b].[2] In these approaches, preferences are used to select rules in a explicit way. On the other hand, implicit preferences can provide an effective mechanism for selecting default rules and for handling incomplete information in logic programming (this will be one of our thesis objectives, see Section 1.4.1).

Besides the fact that information can be incomplete, information can also be uncertain. Consequently, the design of intelligent systems also requires a proper representation and management of uncertain information.

Reasoning with uncertain information requires a theory of uncertainty in order to represent uncertainty and to draw inferences using uncertain data. To this end, several theories have been proposed such as probability theory [Reichenbach, 1949; Walley, 1991], fuzzy set theory [Zadeh, 1975], evidence theory [Shafer, 1976], and possibility theory [Zadeh, 1978; Dubois and Prade, 1988]. Their combination with classical logic has converged in logical frameworks such as probabilistic logic [Nilsson, 1994; Lukasiewicz, 2005] and possibilistic logic [Dubois et al., 1994]. Further on, the integration of these formalisms into logic programming have made it possible to extend logic programming semantics to handle uncertain information. Several types of logic programming frameworks have been created. They range on frameworks based on *annotated logics* [Kifer and Subrahmanian, 1992], *probabilistic logic* [Ng and Subrahmanian, 1992; Lukasiewicz, 1998; Kern-Isberner and Lukasiewicz, 2004; Saad and Pontelli, 2005; Baral et al., 2009], *fuzzy logic* [Alsinet and Godo, 2002; Nieuwenborgh et al., 2007; Bauters et al., 2010b], *bilattices* [Fitting, 1991], *evidence theory* [Baldwin, 1987; Wan, 2009], to *possibilistic logic* [Dubois et al., 1991; Nicolas et al., 2006; Nieves et al., 2011; Bauters et al., 2010a]. However, the joint handling of incomplete and uncertain information has received little attention both in logical- and in logic programming-based approaches with the exception of some works including [Saad and Pontelli, 2005; Nicolas et al., 2006; Nieves et al., 2011; Baral et al., 2009; Bauters et al., 2010a].

Dealing with incomplete and uncertain information together amounts to handle uncertain default rules. Handling uncertain default rules basically involves two tasks, that are, the selection of default rules and the management of the uncertainty. This can be done in different ways which depend on the setting considered. For instance, in logical-based settings such as possibilistic and probabilistic logic [Lukasiewicz, 2005; Dupin de Saint-Cyr and Prade, 2008], the selection of default rules is independent from the uncertainty management. This is due to the fact that, in these logics, incomplete information is not explicitly represented. As such, default rules need to be transformed into strict rules before to process the uncertainty associated to them. Instead, in ASP, such transformation is not required and the selection of uncertain default rules can be a matter of incomplete and uncertain information at the same time. In this respect, an approach able to express explicit preferences for the selection of exceptions to default rules when information is uncertain has not been studied yet (this will be one of our thesis objectives, see Section 1.4.1).

[2] From a nonmonotonic reasoning point of view, LPODs is a logic programming specification which supports context-dependent explicit preference statements such as *in context c it is more reasonable to assume a rather than b* (encoded as $a \times b \leftarrow c$) to select the preferred exceptions to be used in the inference process. LPODs can also be used to model user preferences, as we will mention in Section 1.2. Both views will be discussed in more detail in Chapter 3 and Chapter 6 respectively.

1.2 User Preference Modeling

Preferences are not only a matter of (uncertain) default rule selection. Preferences are mainly understood as a means for representing user choices. As such, they are an essential aspect in any intelligent system that supplies personalized content to users and that assists users with decisions [Chen and Pu, 2004; Brafman and Domshlak, 2009]. A natural question regarding the former view is how user preferences should be modeled and processed.

One simple approach would be to ask users to explicitly describe their preference models. This method is clearly ineffective, since it is not adequate for supporting scenarios in which the set of possible choices is too large to be described in an explicit way.

The above concern has fostered the study of *compact preference representation languages* [Brewka et al., 2004a; Boutilier et al., 2004; Benferhat et al., 2001; Brewka et al., 2004b, 2003; Kärger et al., 2008]. Generally speaking, these specifications define a preference model in a compact way and a preference relation to rank the outcomes of the model. The way in which preferences are encoded and the ranking is obtained mainly depends on the specification used. Preference representation languages can be categorized into:[3]

- approaches which extend classical logic such as Qualitative Choice Logic (QCL) [Brewka et al., 2004a] and Possibilistic Logic [Benferhat et al., 2001];
- approaches based on Conditional Preference Networks (CP-nets) [Boutilier et al., 2004] and CP-nets extensions such as TCP-nets [Brafman et al., 2006], CP-theories [Wilson, 2004], and Conditional Importance Networks (CI-nets) [Bouveret et al., 2009];
- approaches based on logic programming such as LPODs [Brewka, 2002; Brewka et al., 2004b], Answer Set Optimization (ASOs) programs [Brewka et al., 2003], Logic Programs with Ordered and Unordered Disjunction (DLPODs) [Kärger et al., 2008], CR-Prolog with Ordered Disjunction [Balduccini and Mellarkod, 2003], and Resourced ASP [Costantini and Formisano, 2009].

In logical-based approaches such as QCL and Possibilistic Logic, preferences are modeled in a logical way by means of non-standard propositional formulas. A preference order is obtained by ranking valid interpretations according to setting-dependent preferential entailments. For instance, in QCL, preferences are modeled using a non-standard logic connective $\overrightarrow{\times}$ able to capture *ordered disjunctions* [Brewka et al., 2004a]. An ordering between valid interpretations is achieved by means of preferential inference relations which define the *preferred models*. In possibilistic logic, preferences are modeled by a set of more or less imperative goals [Benferhat et al., 2001] and an ordering is defined in terms of preferred interpretations.

[3] These approaches belong to the AI stream of work. However, the studies of the process that support the construction of preferences have historically deep roots in Philosophy and Logic. In 1957, the pioneering work of [Halldén, 1957] initiated a line of research that was subsequently systematized in [von Wright, 1963] which is usually taken to be the seminal work in *preference logics*. This line of research continues nowadays. For instance, the work of [van Benthem et al., 2009] and [Liu, 2010], develop new modal preference logics that improve over [Halldén, 1957] in several directions. Other studies about preferences can be found in [Hansson, 2001; Lichtenstein and Slovic, 2006; Bienvenu et al., 2010]. In [Bienvenu et al., 2010], a prototypical preference logic is proposed which makes a first step to bridge the gap between preference logics and AI preference languages.

A more intuitive way for modeling conditional (context-dependent) preferences and preference relations is offered by CP-nets [Boutilier et al., 2004]. A CP-net consists of a directed graph in which nodes represent attributes (over a given domain) and edges express preference links between them. CP-nets model conditional preferences based on the expression of *ceteris paribus* preference statements. A partial order among the outcomes of a CP-net is obtained by means of a conditional preference table associated to each node in the graph.

Although QCL, Possibilistic Logic, and CP-nets own some nice features w.r.t. preference handling, they seem to miss an important aspect which is commonly encountered in knowledge representation, that is, the modeling of incomplete information. In fact, beside the fact that preferences are context-dependent, they may also depend on incomplete knowledge. To this end, it is interesting to look at logic programming approaches based on answer set semantics. For instance, LPODs [Brewka, 2002; Brewka et al., 2004b], DLPODs [Kärger et al., 2008], and ASOs [Brewka et al., 2003] allow to express incomplete knowledge through the use of negation as failure.

LPODs combines some of the ideas behind QCL with logic programming with negation as failure. In this way, LPODs can model basic qualitative preference statements such as *At night, I prefer going to a pub over going to a bar* by means of ordered disjunction rules of the form $pub \times bar \leftarrow night$. DLPODs basically follows the same idea behind LPODs, but they have a less restricted syntax able to combine literals connected by $\vee$ in ordered disjunction rules. ASO programs decouple the answer set generation from the preference specification and they allow more complex preference formulas.

On the other hand, new sources of data are changing the way in which people relate to information. For instance, the appearance of context-aware devices or systems, such as iPhone or Android, allows users to obtain personalized information anytime from (almost) everywhere. Now, a user can obtain recommendations about restaurants based on his or her current location at lunchtime, about a personalized city tour in the afternoon, or about movies in the evening, *etc*. To provide personalized content, these systems require an effective preference model for handling context-dependent user preferences.

The use of logic programming specifications for modeling user preferences in context-aware systems is appealing. In particular, they can be used for representing *user profiles* (this will be one of our thesis objectives, see Section 1.4.2). In this respect, there may exist more complex preference statements that these specifications are not able to express. This issue suggests that studying a less restricted syntax can be valuable (this will be one of our thesis objectives, see Section 1.4.2).

1.3 Preferences in Decision under Uncertainty

Preferences are also important for selecting an optimal decision in decision making under uncertainty (DMU). In DMU, preferences are used to mark the states of the world that a user or decision maker prefers to achieve when taking a decision. Making an optimal decision amounts to find a decision (if it exists) that satisfies as many preferences as possible taking into account that knowledge about the world is not fully certain. Choosing an op-

timal decision is not an easy task. It requires a theory able to represent preferences and uncertainty and to rank decisions according to decision rules. The way in which preferences and uncertainty are represented depend on the framework considered.

Historically, DMU emanates from economics and game theory with the expected utility theory of von Neumann and Morgenstern [von Neumann and Morgenstern, 1944] and with the subjective expected utility of Savage [Savage, 1972]. According to these classical decision theories, the ranking of decisions is done according to an *expected utility function* which combines probability values associated to states of the world with utility values associated to the consequences produced by the decisions. These approaches assume that a probability distribution, encoding the uncertainty, and an utility function, encoding preferences, are fully specified and available.

However, such functions and distributions are not always accessible or they are difficult to obtain [Doyle and Thomason, 1999]. For instance, a user may be reluctant to provide numerical utilities for all possible consequences, or only a partial description of the uncertainty can be retrieved. Since an expected utility approach needs numerical probabilities for each state and numerical utilities for all possible consequences, the automatization of a decision making process rises some limitations of classical decision theory approaches. Moreover, since preferences and knowledge can be human-oriented, a qualitative representation of information can be more adequate than a quantitative one. These limitations fostered the appearance of a new decision paradigm in AI called *qualitative decision theory* [Doyle and Thomason, 1999].

The terms qualitative decision theory refers to more than one kind of representation and several qualitative decision making frameworks have been proposed in the literature [Boutilier, 1994; Tan and Pearl, 1994a,b; Dubois et al., 2001; Dubois and Prade, 1995; Bonet and Geffner, 1996]. Among them, possibilistic theory has shown to provide a convenient setting for modeling qualitative DMU [Dubois et al., 2001; Dubois and Prade, 1995]. According to this approach, the available knowledge is described by formulas which are more or less certainly true and the preferences are described in a separate prioritized propositional base. Then, by assuming a commensurateness hypothesis between the level of certainty and the preference priority, pessimistic and optimistic criteria justified on the basis of postulates have been proposed for the selection of an optimal decision [Dubois et al., 2001, 1999].

Although the formulation of qualitative DMU has proved to be compatible with the symbolic representation of uncertainty and preferences in the possibilistic logic setting, the computation of the pessimistic and optimistic criteria has remained in the background. On the other hand, despite the nice computational features of logic programming, there is not too much work in the logic programming literature that addresses qualitative DMU. This opens the possibility to investigate whether it is possible to cast the possibilistic logic formulation of DMU into a logic programming setting and to provide the computation of an optimal decision according to pessimistic and optimistic criteria (this will be one of our thesis objectives, see Section 1.4.3).

1.4 Thesis Objectives

The main aim of this dissertation is to tackle the role of preferences in logic programming from different perspectives. So far, we have seen how the concept of preference is related to (uncertain) nonmonotonic reasoning, user preference modeling, and qualitative DMU. With this research work, we aim at showing that, also in logic programming, preferences can offer an effective way to select (possibly uncertain) default rules, to express user preferences, and to choose among different decisions under uncertainty.

The objectives of this thesis are presented according to these three perspectives. For each of them, we discuss the main motivations of our work and we describe in detail the objectives pursued.

1.4.1 Preferences in Nonmonotonic Reasoning and Uncertainty

Preferences are an important aspect in logic programming for selecting default rules and several extensions for handling preferences have been proposed in the literature. Most extensions assume that preferences are used in an explicit way to select default rules in which exceptions are captured by means of negation as failure. Selecting default rules explicitly is not always possible. For instance, when different pieces of knowledge encoded by default rules are integrated, possible exceptions must be made explicit before to be able to select default rules. When such rules contain different levels of specificity, an implicit preference order among the rules can be drawn in order to identify exceptions to default rules and to deal with incomplete information. Only few approaches able to consider implicit preferences in logic programming have been proposed [You et al., 1999; Garcia et al., 2000, 2009]. Most approaches advocate for the use of negation as failure. However, nonmonotonic reasoning can be sometimes handled with preferences rather than negation as failure and it is interesting to look for modeling exceptions by means of strong negation. Moreover, in the methodology for qualitative decision making that we will propose later in the document, we require that knowledge about the world is encoded by means of extended definite logic programs, *i.e.*, negation as failure free. It may be the case that such knowledge contains a set of rules which involve an implicit specificity and which can have exceptions that must be properly handled. Therefore, the first objectives we will pursue are:

Obj1: to identify and to generate exceptions to default rules
Obj2: to manage incomplete information without *negation as failure*

To fulfill the above objectives, we propose *an approach to nonmonotonic reasoning using implicit preferences in logic programming*. The approach consists in a rewriting procedure which considers the specificity of the rules in order to identify conflicting rules and to generate exceptions. Specificity is obtained by adapting the Z-ordering for ranking the default rules in a logic program. The ranking allows to identify those default rules which are in an exceptional situation w.r.t. other more specific rules. Once such rules are identified, they are rewritten into strict rules in which exceptions are explicitly captured by strong negated atoms. At the same time, a set of completion rules[4] are created. The approach turns to have

[4] As we will see in Chapter 4, completion rules are rules that complete contextual and incomplete information.

some nice features. First, it provides a methodology to restore the consistency of inconsistent logic programs that implicitly involve specificity between the rules. This is particularly useful in knowledge integration and decision making scenarios.[5] Secondly, it can provide an operational counter-part to negation as failure based approaches for handling exceptions in logic programming,

While in the first contribution we propose to handle nonmonotonic reasoning using implicit preferences rather than negation as failure, in the second one, we handle incomplete information using negation as failure and we focus on a different problem: the *selection of default rules when information is uncertain*. In particular, most of the approaches in logic programming able to deal with uncertainty do not consider nonmonotonic reasoning or, if they do, no notion of explicit preferences for selecting uncertain default rules is considered. In the case of LPODs, explicit preferences allow to select exceptions to default rules to be used in the reasoning process. But, what happens when such rules are uncertain? Is it possible to consider the certainty of the rules for selecting the most preferred and plausible beliefs to be used in the reasoning? Therefore, the next objectives we will pursue are:

Obj3: to represent and reason about preferences with incomplete and uncertain information

Obj4: to select preferred sets of beliefs in an explicit way when information is uncertain

The second contribution in the first part of this thesis tackles the objectives above by introducing a *logic programming framework able to deal with explicit context-dependent preferences and uncertainty*. Such framework is called *Logic Programs with Possibilistic Ordered Disjunction* (LPPODs). The LPPODs framework extends two existing frameworks: LPODs and possibilistic normal programs [Nicolas et al., 2006]. On one hand, from LPODs, the framework inherits the distinctive feature of expressing explicit context-dependent preferences among different exceptions to default rules (modeled as atoms of a logic program) and, in general, of modeling context-dependent preferences. On the other hand, possibilistic normal logic programs allow to capture qualitative certainty measures about the certainty of rules (modeled as necessity values according to possibilistic logic [Dubois et al., 1994]). In this way, the LPPODs framework supports a reasoning which is nonmonotonic, preference- and uncertainty-aware. The possibilistic management of uncertain information can be motivated in several ways. First, as stated by several authors [Dubois and Prade, 2004; McCarthy and Hayes, 1969] probabilistic-based approaches do not fit very well with logical entailment, while fuzzy set-based logic programming approaches are essentially a generalization of logic programming to continuous domains, rather than a form of approximate reasoning [Bauters et al., 2010b]. Secondly, by using possibilistic logic, we have been able to reuse an existing implementation of the possibilistic solver *posSmodels*[6] for possibilistic normal programs and the LPODs solver *psmodels*[7] in order to provide a straightforward computation of the LPPODs framework.

[5] Indeed, in qualitative decision making, a consistency between knowledge and a decision is assumed.

[6] `http://www.info.univ-angers.fr/pub/pn/Softwares/posSmodels.html`

[7] `http://www.tcs.hut.fi/Software/smodels/priority/`

The LPPODs framework is not only a matter of uncertain nonmonotonic reasoning. Indeed, its capability to express explicit preferences and to model necessity values makes it possible to adopt the framework a) for modeling and computing user profiles in a context-aware system (as we will discuss in Section 1.4.2); b) for modeling qualitative DMU according to its possibilistic logic reading and for computing an optimal decision according to pessimistic and optimistic criteria (as we will discuss in Section 1.4.3).

1.4.2 User Preference Modeling

Logic programming can be an effective tool for building a preference model. Indeed, several logic programming proposals, such as LPODs, DLPODs, or ASO programs, support the specification of preferences which can depend both on contextual and incomplete information. Moreover, they can draw an order between the model outcomes. On the other hand, context-aware systems have a limited sensibility towards user specific desires when preference models or user profiles are not taken into account. As such, user profile handling is a nice feature to add to context-aware systems.

A possible way to represent user profiles is in terms of a logic programming-based preference model. The use of a logic programming-based approach can be motivated in several ways. First, preferences can depend on contextual information. For instance, asking for a restaurant suggestion is clearly different from asking for a cinema suggestion. Secondly, preferences can depend on incomplete information: normally one can have some preferences unless some exceptional conditions are met. Thirdly, the compact symbolic logic programming representation and its appealing computational features advocate for the use of logic programming in a system that has to process user preference models in a quick way. Finally, in context-aware systems, users can have many preferences and an approach that considers all preferences as equally important at the moment of selecting the content to suggest can be too rigid. Preferences in some contexts can be more important than others or some preferences can become obsolete over the time. As such, a way to measure preference importance should be considered. While incomplete and contextual information is already contemplated by existing logic programming specifications, importance weights are not. Therefore, our next objectives are:

Obj5: to build a preference model which can depend on importance weights, contextual, and incomplete information
Obj6: to integrate such preference model in an existing context-aware system
Obj7: to compute the preference model in a practical way

To tackle the objectives above, we propose to *handle user preferences in context-aware systems* by means of LPPODs. In describing our proposal, we take into account a specific instance of context-aware systems, Interactive Community Displays (ICDs).[8] Although we have introduced LPPODs in the context of uncertain nonmonotonic reasoning, its features seem to fit particularly well in the representation of the required preference model. Indeed,

[8] An Interactive Community Displays (ICD) is a multimedia information point which offers interactive services on the public thoroughfare and provides information to people living in or visiting a city [Ceccaroni et al., 2009].

contextual and incomplete preferences can be easily captured by ordered disjunction rules and necessity values associated to the rules can be used to handle preference weights. In this way, LPPODs can be used to describe user profiles in the system and solutions of an LPPOD, *i.e.,* its possibilistic answer sets, can provide an ordered list of preference items. Such ordered list can be used to enhance the personalization capabilities of such systems (*e.g.,* in the content filtering). As far as integration is concerned, it is unlikely that end-users will specify their preferences at symbolic level, *i.e.,* according to the LPPODs syntax. Therefore, a more abstract way for specifying user preferences is required. To this end, a preference ontology is proposed to bridge the gap between the ontological management of information, on which new information systems typically rely, and the symbolic level representation of information used in the LPPODs inference process. The computation of the preference model is straightforwardly obtained by implementing the LPPODs semantics in an ASP-based solver that we called *posPsmodels*.

Another important aspect in the modeling of user preferences in context-aware systems is to have enough flexibility for capturing preference statements. For instance, in the above approach, since LPPODs is based on LPODs, only singleton preference literals can be specified in each preference rule. However, there are cases in which more complex expressions need to be specified. Preferences statements which can express equalities (*e.g., I prefer cinema to pub or tv*) such as in DLPODs, or combinations of more generic formulas, such as in ASO programs, may allow to capture richer preferences rules. But, is it possible to allow a less restricted syntax for expressing nested preferences in logic program rules by means of boolean connectives $\{not, \neg, \wedge, \vee, \times\}$? In this respect, the next objectives are:

Obj8: to provide a specification for nested (or non-flat) preference expressions in logic programs

Obj9: to define and compute the semantics of this new specification

To fulfill the above objectives, we propose an extension of LPODs, called *Nested Logic Programs with Ordered Disjunction* (LPODs$^+$), which allows the writing of nested preference formulas built by means of connectives $\{not, \neg, \wedge, \vee, \times\}$. To capture the LPODs$^+$ syntax and semantics, we extend Lifschitz's proposal of nested logic programs [Lifschitz et al., 1999] to account for the ordered disjunction connective $\times$. Consequently, the LPODs$^+$ semantics can be defined in terms of a program reduction and the semantics of nested logic programs in an easy way. Since the syntax of LPODs$^+$ is too complex to be handled by existent answer set solvers, we design a translation procedure to map any LPOD$^+$ to an equivalent disjunctive logic program.

1.4.3 Preferences in Decision under Uncertainty

Despite the nice computational features offered by logic programming, the use of a logic programming approach for modeling qualitative DMU and for computing an optimal decision has remained almost unexplored. Existing proposals [Brewka, 2005; Grabos, 2004] suggest a general methodology for modeling qualitative decision problems by means of LPODs. The common idea is to encode a decision problem by means of an LPOD and to select an optimal decision by means of comparison criteria based on the LPODs semantics.

However, these approaches are not satisfactory w.r.t. the handling of preferences and uncertainty for several reasons. First, preferences are always considered to be part of the world description, whereas, in qualitative DMU, preferences and knowledge about the world are modeled in two separate knowledge bases. Secondly, these approaches do not deal with uncertainty, or, if they do, they do not treat uncertain information in a formal way. For instance, in [Brewka, 2005], no explicit notion of uncertainty is contemplated. The plausibility of a state (of the decision problem at hand) is considered in terms of answer set semantics, that is, a state is plausible whenever it is not excluded. However, a further discrimination between the plausibility of states is not possible. In [Grabos, 2004], the plausibility of states is represented through ordered disjunction rules, called belief rules, which capture a plausibility order between states. Although this approach specifies more advanced decision rules, it still seems to miss a satisfactory handling of qualitative DMU. In particular, it lacks a formal justification of the proposed decision rules. Therefore, it is interesting to look for a methodology able to capture qualitative DMU in logic programming (in a way closer to its possibilistic logic formulation) and to compute the pessimistic and optimistic criteria. In the attempt to bridge the gap between logic programming and qualitative DMU, the following objectives are considered:

Obj10: to model qualitative decision making in logic programming
Obj11: to model qualitative decision making under uncertainty in logic programming
Obj12: to compute an optimal decision according to pessimistic and optimistic criteria in logic programming

The computation of an optimal decision according to possibilistic pessimistic and optimistic criteria requires a suitable encoding of qualitative DMU in logic programming. This means that knowledge about the world and preferences need to be represented in two separate knowledge bases. To this end, we can observe that the similarity between a decision and an abduction problem is striking (as already stressed in [Sabbadin, 1998]). In this respect, logic programs have already been used to model abduction problems [Eiter et al., 1999; Brewka, 2006]. In particular, in [Brewka, 2006], Brewka shows how LPODs can model abduction in a concise and elegant way. Therefore, the use of LPODs looks reasonable in order to *model qualitative decision making* (in terms of abduction) and to compute an optimal decision when knowledge is fully certain and preferences are all-or-nothing.[9] Instead, when decisions are a matter of certainty and preference degrees, the representation of knowledge and preferences must be extended to capture certainty levels and preference priorities. For this purpose, the possibilistic extension of LPODs, LPPODs, allows to associate certainty and preference degrees to the rules of a logic program. In this way, the LPPODs framework can be used *to model decision making problem under uncertainty*. Then, the logic programming counterpart of the computation of the possibilistic decision criteria can be obtained in two ways by applying both a classical (based on LPODs) and a possibilistic (based on LPPODs) logic programming procedure.

[9] To avoid misunderstandings, it is worthy to clarify that the LPODs-based modeling we are referring to here is not the one used in the approaches to decision making discussed above.

1.5 Thesis Structure

This thesis consists of 11 chapters (including the introduction and a background) and an appendix. The main chapters are organized into three parts which correspond to the different perspectives to which preferences are related as described in the previous sections.

Chapter 2 provides the technical background related to the thesis. Basic notions about Answer Set Programming, Possibilistic Logic, and Possibilistic Answer Set Programming are provided. For a better readability of the document, the classes of logic programs considered are briefly outlined. The content in this chapter is properly referenced from different parts of the document when needed.

Part I is dedicated to the multifaceted relation preferences have in nonmonotonic reasoning and in nonmonotonic reasoning under uncertainty. This part is organized into three chapters.

Chapter 3 surveys different formalizations of the concept of default rule and preference extensions that have been proposed in the AI literature. In particular, we overview several approaches based on nonmonotonic logics, nonmonotonic inference relations of System P, and logic programming, as well as some proposals for handling preferences in an implicit and explicit way. Then, we present some of the research focused on nonmonotonic reasoning under uncertainty from a logic programming point of view. This chapter is concluded by a discussion motivating and introducing the following two chapters.

Chapter 4 describes a procedure able to consider implicit preferences in extended logic programs to handle nonmonotonic reasoning without negation as failure. Some significant results are achieved. This procedure can be used to restore the consistency of inconsistent programs which involve a specificity ordering between the rules. Nonmonotonicity can be obtained in extended definite logic programs by means of a set of completion rules that add pieces of knowledge in a proper way. The relations between this non-standard nonmonotonic reasoning and the standard one based on negation as failure are also depicted. We conclude this chapter by discussing the representation capabilities of this approach in terms of preference representation.

Chapter 5 deals with Logic Programs with Possibilistic Ordered Disjunction (LPPODs), a possibilistic extension of LPODs. LPPODs provides a logic programming framework which embeds in a unified way several aspects of common-sense reasoning, such as explicit preferences, nonmonotonicity and uncertainty. Indeed, the LPPODs framework captures uncertain information by means of necessity values according to possibilistic logic and it supports the specification of explicit preferences among exceptions to program rules. In this way, apart of representing and reasoning about uncertain and incomplete information, LPPODs can be used for the selection of preferred beliefs sets when program rules are uncertain. The chapter is concluded by a discussion of related work from the nonmonotonic reasoning and uncertainty handling points of view. We also discuss the representation capabilities of this approach in terms of preference representation.

Part II is dedicated to user preference modeling. This part is organized into three chapters.

Chapter 6 first reviews several preference handling methods. Then, some notions about user profiles and context-awareness are presented in order to provide a basic understand-

ing of context-aware systems. This chapter is ended by a discussion motivating and introducing the following two chapters.

Chapter 7 describes how preferences have been integrated into a context-aware system for enhancing the personalization capability. Starting from a simple use case description, we show how user profiles and preferences can be modeled at knowledge level by means of ontologies. This abstract representation is then converted at symbolic level to logic program rules according to the LPPODs syntax. LPPODs can be used to model user preferences which can depend both on importance weights and on contextual information (possibly incomplete). This symbolic representation can be processed in a practical way by means of a solver implementing the LPPODs semantics. The chapter is concluded by a discussion about related work about preference queries to classical and possibilistic databases.

Chapter 8 presents Nested Logic Programs with Ordered Disjunction (LPODs$^+$), a proposal for representing nested preference expressions in logic programming. Nested expressions are expressed as boolean combinations of formulas built by means of $\{not, \neg, \vee, \wedge, \times\}$ connectives. We show how the LPODs$^+$ syntax and semantics can be easily captured extending some results of nested logic programs. Then, we design a translation procedure for mapping LPODs$^+$ into disjunctive logic programs. We conclude the chapter by depicting the relationship between LPODs$^+$ and other existing logic programming specification for preference modeling.

Part III is dedicated to preferences in decision making under uncertainty. This part consists of two chapters.

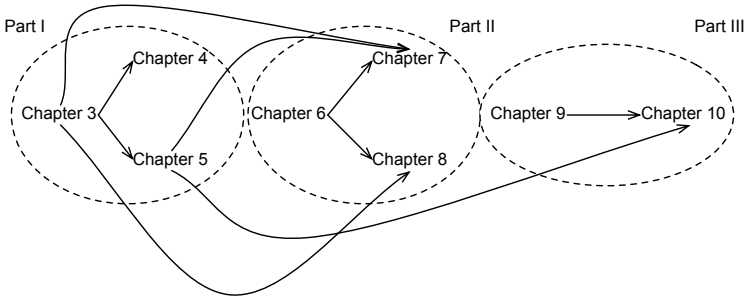
Chapter 9 first overviews classical decision theory and introduces qualitative decision theory especially from the possibilistic logic point of view. In particular, a pessimistic and optimistic decision criterion are presented. It also shows how the computation of these criteria can be reformulated in terms of a prioritized propositional base. Some of existing ASP methodologies are presented, and discussing their unsatisfactory treatment of uncertainty, we introduce the next chapter.

Chapter 10 presents the modeling of a basic decision making problem, *i.e.,* when knowledge is certain and preferences are all-or-nothing, in terms of LPODs. Then, this result is extended to the case in which certainty and preferences are matters of degrees. We show how it can be captured by means of LPPODs. The computation of the possibilistic decision criteria is provided by two procedures. In the first procedure, we reduce the computation of an optimal decision to a standard ASP computation (based on LPODs). In the second one, we provide an alternative procedure, closer to possibilistic logic, based on the LPPODs semantics. The chapter is concluded by a discussion about related work of qualitative decision making in logic programming.

Part IV consists of a single chapter (**Chapter 11**) which provides a summary of the thesis and points out some lines of future research.

Appendix A contains the technical proofs for all the results obtained in the thesis.

The following figure depicts the reading dependency of the chapters. An arrow from a chapter to another one suggests that the understanding of the latter depends on the reading of the former.


Part I
Part II
Part III
Chapter 3
Chapter 4
Chapter 5
Chapter 6
Chapter 7
Chapter 8
Chapter 9
Chapter 10

2

Background, Notation and Classes of Logic Programs Considered

In this chapter we provide the basics of Answer Set Programming (ASP), some concepts about Rewriting Systems, Possibilistic Logic, and Possibilistic Answer Set Programming (PASP). The sections in this part are properly referenced from several parts of the document when needed. This is intended to be a basic introduction on the concepts used in this thesis. More specific knowledge, will be presented at the beginning of each chapter when needed.

The chapter is organized as follows. In Section 2.1, several concepts of logic programming under answer set semantics are presented. Section 2.2 introduces several classes of logic programs such as Disjunctive Logic Programs, Generalized Disjunctive Logic Programs, Nested Logic Programs, and LPODs. In Section 2.3, rewriting systems and program transformations for logic programs are briefly described. These notions are particularly useful for the understanding of some sections in Chapter 5. Possibilistic Logic and PASP (Section 2.4 and 2.5) are important concepts which are extensively used in several part of the document. Finally, Section 2.6 overviews the classes of logic programs which are considered and defined in this document.

2.1 Answer Set Programming

In this section, we provide the formal definition of the syntax and semantics of answer sets for logic programs. We assume that the reader is familiar with logic programming which has been extensively studied in [Lloyd, 1987; Lifschitz, 1996; Baral, 2003].

2.1.1 Syntax

The language for a logic program consists of

- an enumerable set $\mathcal{A}$ of elements called *atoms* (denoted $a, b, c, \ldots$)
- *connectives* $\wedge, \vee, \leftarrow, \neg, not, \bot$
- *auxiliary symbols* "(", ")", ",", "."

in which $\{\wedge, \vee, \leftarrow\}$, $\{not, \neg\}$, $\{\bot, \top\}$ are 2-place, 1-place and 0-place connectives respectively. Atoms negated by $\neg$ are known as *extended atoms*. A *literal* is an atom a or an extended atom $\neg a$.

Remark 2.1. In this document, we use the concept of atom without paying attention whether it is an extended atom or not. As a consequence, the concept of atom and literal are used as interchangeable concepts.

We consider two types of negation in our logic programs: strong negation $\neg$ and negation as failure *not*.[1] Intuitively, *not* a is true whenever there is no reason to believe a, whereas $\neg a$ requires a proof of the negated atom.

Given a finite set of atoms $\mathcal{A}$, a *extended normal rule* r is an expression of the form

$$a \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_{m+n} \tag{2.1}$$

in which a and each b_i ($1 \leq i \leq m+n$) are atoms. Given an extended normal rule r, the left part of r is known as the head ($head(r) = a$) and the right part is known as the body ($body(r)$= $\{b_1, \ldots, b_n, not\ b_{m+1}, \ldots, not\ b_{m+n}\}$). Rules with empty body ($m+n=0$), that is, $a \leftarrow \top$, are referred to as *facts*. Rules with empty head, that is, $\bot \leftarrow b_1, \ldots, b_m$, $not\ b_{m+1}, \ldots, not\ b_{m+n}$, are referred to as *constraints*. The managing of constraints in ASP is done by replacing each rule of the form $\leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_{m+n}$ by a new rule of the from $f \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_{m+n}, not\ f$ in which f is a new atom symbol not belonging to the signature of the program. When $n = 0$, *i.e.,* there are not negated-by-failure atoms, the rule is an *extended definite rule*. To distinguish between the positive and the negative part of the body, we define $\mathcal{B}^+ = \{b_1, \ldots, b_m\}$ for the positive body and $\mathcal{B}^- = \{b_{m+1}, \ldots, b_{m+n}\}$ for the negative body. The intuitive reading of a rule r is as follows: if all atoms in $\mathcal{B}^+$ are derived and no atom in $\mathcal{B}^-$ is derivable, then $head(r)$ can be derived.

An *extended normal logic program* P is a finite set of extended normal rules. If all the rules in P are definite rules, then P is called a *definite logic program*. By $\mathcal{L}_P$, we denote the signature of P, *i.e.,* set of atoms that appear in the rules of P.

In our programs, we will manage strong negation $\neg$, as it is done in ASP [Baral, 2003]. Basically, each extended atom $\neg a$ is replaced by a new atom symbol a' which does not appear in the language of the program $\mathcal{L}_P$.

Example 2.1. Let P be the following logic program:

$$P = \left\{ \begin{array}{c} a \leftarrow b, not\ d \\ \neg b \leftarrow c, not\ \neg d \\ b \leftarrow \top \\ c \leftarrow \top \end{array} \right\}$$

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

Example 2.2. Let P' be the following logic program:

$$P' = \left\{ \begin{array}{c} a \leftarrow b, not\ d \\ b' \leftarrow c, not\ d' \\ b \leftarrow \top \\ c \leftarrow \top \end{array} \right\}$$

[1] In this document, we sometimes consider only strong negation in our logic programs. In such cases, we explicitly state it.

In order to disallow models with complementary atoms, such as b and $\neg b$ for instance, a constraint of the form $\bot \leftarrow b, b'$ is added to the logic program. We will omit this constraint in order to allow models with complementary atoms. However, this constraint can be added without loss of generality.

2.1.2 Interpretations and Models

The declarative semantics of a logic program is given by the usual (model-theoretic) semantics of formulas in classical logic. In the following, we will introduce the basic concepts to define the minimal model of a logic program.

In order to be able to define a model of a logic program, we need to define an *interpretation*. Let us first remind that a logic program can be treated as a theory. In such a case, each negated-by-failure atom *not* a is replaced by $\sim a$ such that $\sim$ is regarded as the classical negation in classical logic. Formulas are constructed as usual in classical logic by the connectives $\vee, \wedge, \leftarrow, \sim, \bot$. A theory T is a finite set of formulas. By $\mathcal{L}_T$, we denote the signature of T, namely the set of atoms that occur in T. Generally, given a set of proposition symbols S and a theory T in a logic X, $T \vdash_X S$ if and only if $\forall s \in S, T \vdash_X s$.

Formally, an interpretation of a formula F is a truth value assignment such that:

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

- $I(a \wedge b) = \min\{I(a), I(b)\}$
- $I(a \vee b) = \max\{I(a), I(b)\}$
- $I(a \leftarrow b) = 0$ *iff* $I(b) = 1$ *and* $I(a) = 0$
- $I(\sim a) = 1 - I(a)$
- $I(\bot) = 0$

The above mapping is unique [van Dalen, 1994]. 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-value) *model* of a logic program P if and only if, for each rule $r \in P$, $I(c) = 1$, in which c is the clause obtained by r. A theory is consistent if it admits a model, otherwise it is inconsistent. Given a theory T and a formula α, α is a logical consequence of T, denoted by $T \models \alpha$, if, for every model I of T, it holds that $I(\alpha) = 1$. A model I of a theory T is a *minimal model*, if it does not exist a model I' of T, such that $I' \neq I$ and $I' \subset I$.

The concept of minimal model is used in the *answer set semantics* definition.

2.1.3 Answer Set Semantics

One of the semantics which characterizes an extended normal logic program (normal logic program from now on) is the so-called *answer set semantics*. The answer set semantics was originally defined in [Gelfond and Lifschitz, 1988, 1991]. Answer sets as such are defined via a reduction to definite logic programs. Let us remember that a logic program is called definite if $\mathcal{B}^- = \emptyset$ for all its rules.

Let us introduce some terminology first. Given a set of atoms $M \subseteq \mathcal{L}_P$, a rule r is applicable in M if $\mathcal{B}^+ \subseteq M$. The set of applicable rules in a program P w.r.t. M is denoted by $App(P, M)$. A rule r is *satisfied* by M, denoted by $M \vDash r$, if when r is applicable in M, then $head(r) \in M$. Therefore, the satisfaction of a rule r is based on its applicability w.r.t. M. As a consequence, a rule r is not satisfied by M ($M \nvDash r$) if and only if $\mathcal{B}^+ \subseteq M$ and $head(r) \notin M$. Finally, a set of atoms M is closed under a logic program P, if $\forall r \in P$, $M \vDash r$ (or in another words if for any $r \in P$, $head(r) \in M$ whenever $\mathcal{B}^+ \in M$.).

With these formalities at hand, the smallest set of atoms which is closed under a definite logic program is called the *minimal model* of P and it is denoted by $Cn(P)$.

On the other hand, in the case of a normal logic program P, the semantics is defined in terms of the Gelfond-Lifschitz reduction [Gelfond and Lifschitz, 1991]. The Gelfond-Lifschitz reduct of a normal logic program w.r.t. a set of atoms M is defined as:

$$P^M = \{head(r) \leftarrow \mathcal{B}^+ \mid r \in P,\ \mathcal{B}^- \cap M = \emptyset\} \tag{2.2}$$

Observe that all rules in P^M do not contain *not*, *i.e.*, P it is a definite logic program. The reduction is the core of the *answer set semantics* definition.

Definition 2.2. *A set M of atoms is an answer set of a normal logic program P if $Cn(P^M) = M$.*

An alternative characterization of Cn can be obtained through an immediate *consequence operator* [Lloyd, 1987]. Let P be a definite logic program and M a set of atoms. The operator T_P is defined as:

$$T_P(M) = \{head(r) \mid r \in P,\ body(r) \subseteq M\} \tag{2.3}$$

Thus, T_P computes the set of atoms deducible from M by means of P. It allows to define the sequence $T_P^0 = T_P(\emptyset)$, $T_P^{k+1} = T_P(T_P^k)$, $\forall k \geq 0$. For any definite logic program P, $Cn(P)$ is the least fix-point of T_P and $Cn(P) = \bigcup_{k \geq 0} T_P^k(\emptyset)$ contains all the consequences of the program P. More generally, for any answer set M of a normal logic program P it holds that $M = \bigcup_{k \geq 0} T_{P^M}^k(\emptyset)$.

Example 2.3. For illustration, let us consider the following programs

$$P_1 = \begin{Bmatrix} a \leftarrow not\ b \\ b \leftarrow not\ a \end{Bmatrix} \qquad\qquad P_2 = \left\{ a \leftarrow not\ a \right\}$$

Regarding P_1, among the four candidate sets, we find two answer sets, *i.e.*, $\{a\}$ and $\{b\}$, as it can be checked by means of the following table.

M	P_1^S	$Cn(P_1^M)$
$\emptyset$	$a \leftarrow \top$ $b \leftarrow \top$	$\{a, b\}$
$\{a\}$	$a \leftarrow$	$\{a\}$
$\{b\}$	$b \leftarrow$	$\{b\}$
$\{a, b\}$		$\emptyset$

Unlike the previous example, the program P_2 does not admit any answer set.[2]

M	P_2^S	$Cn(P_2^M)$
$\emptyset$	$a \leftarrow \top$	$\{a\}$
$\{a\}$		$\emptyset$

The credulous inference relation under answer set semantics, denoted as $\models_c$, is defined as:

Definition 2.3. *Given a logic program P and a set of atoms S, $P \models_c S$ holds, if and only if there exists an answer set M of P such that $S \subseteq M$.*

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

$$P = \left\{ \begin{array}{l} a \leftarrow \top \\ \neg a \leftarrow \top \\ b \leftarrow \top \end{array} \right\}$$

the only answer set of P is $\{a, \neg a, b\}$. It is worth mentioning that in the ASP literature there are several forms for handling an inconsistent program. For instance, by applying the original definition [Gelfond and Lifschitz, 1991], the only answer set of P is $\{a, b\}$.

Regarding complexity, we have the following: given a normal logic program P and an atom p, deciding whether p is true in some answer sets of P is **NP**-complete. Given a normal logic program P and an atom p, deciding whether p is true in all answer sets of P is co-**NP**-complete. Given a normal logic program P and a set of atoms M, deciding whether M is an answer set of P is **P**.

Answer sets of normal logic programs can be computed using the DLV[3], SMODELS[4], or GRINGO[5] answer set solvers.

2.2 Extensions

In this section, several classes of logic programs which extend normal logic programs are presented.

2.2.1 Disjunctive Logic Programs

Disjunctive logic programs extend logic programs to account for disjunctive information in the head of a rule [Gelfond and Lifschitz, 1991].

[2] A semantics which is able to handle such situation is the *pstable semantics* [Osorio et al., 2006].
[3] `http://www.dbai.tuwien.ac.at/proj/dlv/`
[4] `http://www.tcs.hut.fi/Software/smodels/`
[5] `http://sourceforge.net/projects/potassco/files/gringo/`

More precisely, a Disjunctive Logic Program (DLP) is a collection of disjunctive rules of the form

$$a_0 \vee \ldots \vee a_k \leftarrow \mathcal{B}^+, not\ \mathcal{B}^- \tag{2.4}$$

in which the head of a disjunctive rule is a disjunction $a_0 \vee \ldots \vee a_k$ of atoms a_i (where $0 \leq i \leq k$). $p \vee q$ expresses that p *is true* or q *is true*. Letting $head(r) = \{a_0, \ldots, a_k\}$, a set of atoms M is closed under a definite logic program P if for any $r \in P$, $head(r) \cap M \neq \emptyset$ whenever $\mathcal{B}^+ \subseteq m$.

The definition of P^M carries over from normal programs. An answer set M of a disjunctive logic program is a minimal model being closed under P^M. For instance, the disjunctive logic program $P = \{a \vee b\}$ has two answer sets, $\{a\}$ and $\{b\}$. The set $\{a, b\}$ is closed under $P^{\{a,b\}}$ but it is not an answer set since it is not minimal.

The usage of disjunction raises the complexity of the underlying decision problems. Deciding whether there exists an answer set M for a given atom a such that $a \in M$ is Σ_2^P-complete [Eiter and Gottlob, 1995]. Answer sets of disjunctive logic programs can be computed using the DLV system. DLPs are considered in Chapter 8.

2.2.2 Generalized Disjunctive Logic Programs

Generalized Disjunctive logic programs extend disjunctive logic programs by allowing negation as failure in the head of rules.

A Generalized Disjunctive Logic Program (GDLP) is a collection of rules of the form

$$a_0 \vee \ldots \vee a_k \vee not\ a_{k+1} \vee \ldots \vee not\ a_{k+l} \leftarrow \mathcal{B}^+, not\ \mathcal{B}^- \tag{2.5}$$

in which a_i's ($0 \leq i \leq k + l$) are atoms. Generalized disjunctive logic programs were first considered by Lifschitz and Woo in [Lifschitz and Woo, 1992] and coined generalized disjunctive logic programs by Inoue and Sakama [Inoue and Sakama, 1998].

The computation of the semantics of a GDLP P is reduced to the computation of the semantics of an equivalent disjunctive logic program P', obtained by P by means of a translation procedure defined in [Janhunen, 2001]. The procedure basically eliminates negated by failure literals from the heads of the rules. GDLPs are considered in Chapter 8.

2.2.3 Nested Logic Programs

Nested Logic Programs (NLPs) under the stable model semantics was introduced by Lifschitz *et al.* in [Lifschitz et al., 1999]. NLPs are characterized by bodies and heads of rules which are comprised of arbitrary expressions, *i.e.,* formulas composed of $\wedge$, $\vee$, and *not*.

It is worth observing that NLPs properly generalize, normal logic programs, DLPs, and GLPs. A nested logic program can be always transformed into an equivalent disjunctive logic program by means of several transformation rules [Lifschitz et al., 1999]. NLPs are considered in Chapter 8.

2.2.4 Logic Programs with Ordered Disjunction

Concerning preferences in ASP, *Logic Programs with Ordered Disjunction* (LPODs) are extended logic programs augmented by an ordered disjunction connector $\times$ which allows to capture qualitative preferences in the head of rules [Brewka et al., 2004b]. An LPOD is a finite collection of rules of the form

$$c_1 \times \ldots \times c_k \leftarrow \mathcal{B}^+, not\ \mathcal{B}^- \tag{2.6}$$

in which c_i (for $1 \leq i \leq k$) are atoms. The rule r states that if the body is satisfied then some c_i must be in an answer set, if possible c_1, if c_1 is not possible then c_2, and so on. Each of the c_i represents alternative, ranked options for problem solutions the user specifies according to a desired order. If $k = 1$ then the rule is an extended normal rule. The semantics of LPODs is based on the semantics of split programs which represent every option of the ordered disjunction. An alternative characterization of the LPODs semantics is given in terms of the following reduction.

Definition 2.4 ($\times$-reduction). [Brewka et al., 2004b] *Let $r = c_1 \times \ldots \times c_k \leftarrow \mathcal{B}^+, not\ \mathcal{B}^-$ be an ordered disjunction rule and M be a set of atoms. The $\times$-reduct $r^M_\times$ is defined as*

$$r^M_\times = \{c_i \leftarrow \mathcal{B}^+ | c_i \in M \wedge M \cap (\{c_1, \ldots, c_{i-1}\} \cup \mathcal{B}^-) = \emptyset\} \tag{2.7}$$

Let P be an LPOD and M be a set of atoms. The $\times$-reduct $P^M_\times$ is defined as

$$P^M_\times = \bigcup_{r \in P} r^M_\times \tag{2.8}$$

Then, the LPODs semantics is captured by the following definition.

Definition 2.5 (SEM_{LPOD}). [Brewka et al., 2004b] *Let P be an LPOD and M be a set of atoms. Then, M is an answer set of P if and only if M is closed under all the rules in P and M is the minimal model of $P^M_\times$.*

Example 2.4. Let P be the following LPOD:

$$P = \begin{Bmatrix} r_1 : a \times b \leftarrow not\ c. \\ r_2 : b \times c \leftarrow not\ d. \end{Bmatrix}$$

Then, if we consider the set of atoms $M_1 = \{a, b\}$, $M_2 = \{c\}$, and $M_3 = \{b\}$, it can be checked that:

$$P^{M_1}_\times = \begin{Bmatrix} r_1 : a \leftarrow \top. \\ r_2 : b \leftarrow \top. \end{Bmatrix} \qquad P^{M_2}_\times = \{r_2 : c \leftarrow \top.\} \qquad P^{M_3}_\times = \begin{Bmatrix} r_1 : b \leftarrow \top. \\ r_2 : b \leftarrow \top. \end{Bmatrix}$$

Since M_1, M_2, and M_3 are closed under the rules in P and are minimal models of $P^{M_1}_\times$, $P^{M_2}_\times$, and $P^{M_3}_\times$ respectively, they are valid answer sets of P. Moreover, the answer set $\{a, b\}$ gives us the best option for both rules in P, whereas $\{c\}$ gives us only the second best option for rule r_2 and $\{b\}$ the second best option for rule r_1. Thus, intuitively, $\{a, b\}$ should be

the preferred answer set, since it gives the best option for every rule containing ordered disjunction.

Following the intuition behind Example 2.4, degrees of satisfaction of a rule are defined in [Brewka et al., 2004b] and they are used to determine preferred answer sets. Formally:

Definition 2.6 (Rule Satisfaction Degree). [Brewka et al., 2004b] *Let M be an answer set of an LPOD P. Then the satisfaction degree M w.r.t. a rule $r = c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m$, not $b_{m+1} \ldots$, not b_{m+n}, denoted by $deg_M(r)$, is*

- *1 if $b_j \notin M$ for some j $(1 \leq j \leq m)$, or $b_i \in M$ for some i $(m+1 \leq i \leq m+n)$,*
- *j $(1 \leq j \leq k)$ if all $b_l \in M$ $(1 \leq l \leq m)$, $b_i \notin M$ $(m+1 \leq i \leq m+n)$, and $j = min\{r \mid c_r \in M, 1 \leq r \leq k\}$.*

The degrees can be viewed as penalties. In fact, a higher degree expresses a lesser satisfaction. Therefore, if the body of a rule is not satisfied, then there is no reason to be dissatisfied and the best possible degree 1 is obtained [Brewka et al., 2004b]. Based on the satisfaction degree of single rules, a global preference ordering on answer sets can be specified through different combination strategies.

Given a set of atoms M and $M^i(P) = \{r \in P \mid deg_M(r) = i\}$, the following conditions for defining that an answer set M_1 is strictly preferred to an answer set M_2 have been specified [Brewka et al., 2004b]:

- *cardinality-based*: at the smallest degree j where $|M_1^i(P)| \neq |M_2^i(P)|$, $|M_1^i(P)| > |M_2^i(P)|$
- *inclusion-based*: at the smallest degree j where $M_1^i(P) \neq M_2^i(P)$, $M_1^i(P) \supset M_2^i(P)$
- *pareto-based*: there is a rule $r \in P$ such that $deg_{M_1}(r) < deg_{M_2}(r)$ and for no $r' \in P$ $deg_{M_2}(r) < deg_{M_1}(r)$

Pareto entails inclusion, and inclusion entails cardinality. Cardinality and inclusion-based criteria have been proved not to be general enough [Brewka et al., 2004b] and we will not consider them any further.

Example 2.5. Reconsidering the LPOD in Example 2.4, it can be checked that $\{a, b\}$ is the single preferred answer set w.r.t. Pareto, inclusion and cardinality. On the other hand, answer sets $\{b\}$ and $\{c\}$ are not comparable.

The LPODs semantics is implemented in an efficient ASP-based solver *psmodels*[6] [Brewka et al., 2002].

Concerning LPODs complexity, we have the following [Brewka et al., 2004b]: deciding whether an LPOD P has a preferred answer set is **NP**-complete. Given an answer set M of P, deciding whether M is preferred is **co-NP**-complete. Given an atom a appearing in P, deciding whether there exists a preferred answer set M such that $a \in M$ is Σ_2^P-complete.

The LPODs specification is considered in several parts of this thesis. In Chapter 5, LPODs is one of the building blocks of LPPODs. In Chapter 6, LPODs is exemplified as a logic programming specification for user preference modeling. In Chapter 8, LPODs is extended for supporting non-flat preference expressions. Finally, in Chapter 10, LPODs is used to compute an optimal decision in qualitative decision making.

[6] `http://www.tcs.hut.fi/Software/smodels/priority/`

2.3 Rewriting Systems

In the context of logic programming, rewriting systems, consisting in a set of *program transformations*, have been defined to express precise logic programming semantics and to provide a procedural method for computing logic programming semantics [Brass et al., 2001; Dix et al., 2001; Brass and Dix, 1995, 1999]. They can also be considered an effective way to reduce the size (in terms of atoms and rules) of a logic program preserving its semantics.

As we will see in Chapter 5, program transformation will help to define a confluent rewriting system for Logic Programs with Possibilistic Ordered Disjunction (LPPODs) whose transformations can be used to reduce the size of an LPPOD and to propagate the necessity values between rules without affecting the LPPODs 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 $\rightarrow$. By $x \rightarrow^* y$ we say that x *reduces* to y. An *irreducible* element is said to be in *normal form*. A rewriting system is:

noetherian: if $\nexists x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_i \rightarrow x_{i+1} \rightarrow \ldots$, where $\forall i\ x_i \neq x_{i+1}$,
confluent: if $\forall u \in S$ such that $u \rightarrow^* x$ and $u \rightarrow^* y$ then $\exists z \in S$ such that $x \rightarrow^* z$ and $y \rightarrow^* z$,
locally confluent: if $\forall u \in S$ such that $u \rightarrow x$ and $u \rightarrow y$ then $\exists z \in S$ 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, several transformation rules have been defined for logic programs [Brass et al., 2001; Dix et al., 2001; Osorio et al., 2001b]. Here we only present some basic transformation rules (in particular the transformation we will extend in Chapter 5). Let $Prog_{\mathcal{L}}$ be the set of all normal logic programs with atoms from the signature $\mathcal{L}$. Given a normal program P, $HEAD(P) = \{head(r) \mid r \in P\}$ denotes the set of all head-atoms of P.

Definition 2.7. *A transformation rule is a binary relation on $Prog_{\mathcal{L}}$. The following transformation rules are called basic. Given a program $P \in 2^{Prog_{\mathcal{L}}}$:*

Elimination of Contradictions: P' results from P by elimination of contradictions ($P \rightarrow_{EC} P'$) if P contains a rule $r = a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ which has an atom b such that $b \in \mathcal{B}^+$ and $b \in \mathcal{B}^-$, and $P' = P \backslash \{r\}$

Positive Reduction: P' results from P by positive reduction RED^+ ($P \rightarrow_{RED^+} P'$), if there is a rule $r = a \leftarrow \mathcal{B}^+,\ not\ (\mathcal{B}^- \cup \{b\})$ in P and such that $b \notin HEAD(P)$, and $P' = (P \backslash \{r\}) \cup \{a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-\}$

Negative Reduction: P' results from P by negative reduction RED^- ($P \rightarrow_{RED^-} P'$), if P contains the rules $r = a \leftarrow \top$, and $r' = a \leftarrow \mathcal{B}^+,\ not\ (\mathcal{B}^- \cup \{a\})$, and $P' = (P \backslash \{r'\})$

Success: P' results from P by success ($P \rightarrow_S P'$), if P contains a fact $a \leftarrow \top$ and a rule $r = a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ such that $a \in \mathcal{B}^+$, and $P' = (P \backslash \{r\}) \cup \{a \leftarrow (\mathcal{B}^+ \backslash \{a\}),\ not\ \mathcal{B}^-\}$

Failure: P' results from P by failure ($P \rightarrow_F P'$), if P contains a rule $r = a \leftarrow \mathcal{B}^+$, not $\mathcal{B}^-$ such that $head(r) \in \mathcal{B}^+$ and $head(r) \notin HEAD(P)$, and $P' = (P \backslash \{r\})$.

Loop: P' results from P by loop ($P \rightarrow_{Loop} P'$), if there is a set of atoms A such that:

1. *$\forall r \in P$, if $head(r) \in A$, then $body(r) \cap A \neq \emptyset$*

2. $P' = \{r \in P \mid body(r) \cap A = \emptyset\}$
3. $P \neq P'$

Based on these transformation rules, we define the following rewriting system $\mathcal{CS} = \{EC, RED^+, RED^-, Success, Failure, Loop\}$.

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 2.6. Let P_1 be the following normal logic program:

$$P_1 = \begin{Bmatrix} b \leftarrow not\ a. \\ c \leftarrow not\ b. \\ c \leftarrow \quad a. \\ b \leftarrow \quad \top. \end{Bmatrix}$$

Since $a \notin HEAP(P_1)$, we can apply RED^+ to P_1. Thus we obtain $P_1 \rightarrow_{RED^+} P_2$:

$$P_1 = \begin{Bmatrix} b \leftarrow not\ a. \\ c \leftarrow not\ b. \\ c \leftarrow \quad a. \\ b \leftarrow \quad \top. \end{Bmatrix} \qquad P_2 = \begin{Bmatrix} c \leftarrow not\ b. \\ c \leftarrow \quad a. \\ b \leftarrow \quad \top. \end{Bmatrix}$$

Please observe that we can apply RED^- to the new program P_2, thus we get $P_2 \rightarrow_{RED^-} P_3$:

$$P_2 = \begin{Bmatrix} c \leftarrow not\ b. \\ c \leftarrow \quad a. \\ b \leftarrow \quad \top. \end{Bmatrix} \qquad P_3 = \begin{Bmatrix} c \leftarrow \ a. \\ b \leftarrow \top. \end{Bmatrix}$$

Finally, we can apply *Failure* to P_3. In this way we obtain $P_3 \rightarrow_{RED^-} P_4$

$$P_3 = \begin{Bmatrix} c \leftarrow \ a. \\ b \leftarrow \top. \end{Bmatrix} \qquad P_4 = \begin{Bmatrix} b \leftarrow \top. \end{Bmatrix}$$

P_4 is the normal form of P_1 w.r.t. $\mathcal{CS}$, because none of the transformation rules from $\mathcal{CS}$ can be applied. Please observe that the set of atom $\{b\}$ is a stable model both of P_4 and of P_1. The rewriting system $\mathcal{CS}$ is in fact *confluent* and *noetherian*.

2.4 Possibilistic Logic

Possibilistic logic emanates from possibility theory [Zadeh, 1978] and it was developed as a sound and complete logic system which extends classical logic for handling uncertainty in a qualitative way in a logical setting [Dubois et al., 1994]. Each classical logic formula is associated with a certainty level. Later on, possibilistic logic has shown that it can also be used for preference representation [Benferhat et al., 2001]. Under this latter view, each logic

formula represents a goal to be reached with its priority level rather than a statement that is believed to be true with some certainty level.

Standard possibilistic logic [Dubois et al., 1994] handles *necessity-valued formulas* of the form (φ, α) where φ is a classical logic formula and $\alpha \in (0, 1]$ is interpreted as a lower bound of a *necessity measure* N expressing that the formula φ is *certain* at least to the level α, *i.e.*, $N(\varphi) \geq \alpha$.

Necessity measures N are monotonic functions w.r.t. entailment and they are characterized by the decomposability property $N(\varphi \wedge \psi) = \min(N(\varphi), N(\psi))$. They are dual of *possibility measures* Π, namely $N(\varphi) = 1 - \Pi(\neg\varphi)$. Thanks to the decomposability property of necessity measures N w.r.t. conjunction, a possibilistic logic base can always be put in a clausal equivalent form.

Dual possibility and necessity measures Π and N are based on the same possibility distribution that essentially encodes a preorder that rank-orders interpretations. Generally speaking, a possibility distribution π on the universe of interpretations represents the compatibility of an interpretation ω with the available information (or beliefs) about the real world (encoded by a possibilistic logic base). By convention $\pi(\omega) = 0$ means that ω is impossible and $\pi(\omega) = 1$ means that nothing prevents ω for being true in the real world.

A possibility distribution π induces two dual measures grading respectively the possibility and certainty of a formula φ. The possibility measure Π, defined from a possibility distribution ω as $\Pi(\varphi) = max\{\pi(\omega) | \omega \models \varphi\}$, is the possibility degree which evaluates the extent to which φ is consistent with the available beliefs. The necessity measure N, defined as $N(\varphi) = 1 - \Pi(\neg\varphi)$, is the certainty degree which evaluates the extent to which φ is entailed by the available beliefs.

Given a propositional possibilistic logic base $\Sigma = \{(\varphi_i, \alpha_i) | 1 \leq i \leq n\}$, Σ is semantically associated with several possibility distributions which can be compatible with the set of constraints represented in Σ by $N(\varphi_i) \geq \alpha_i$. However, in practice, one is interested in the least specific distribution only. Formally, a possibility distribution π is said to be the least specific one between all compatible distributions if there is no possibility distribution π' with $\pi \neq \pi'$ compatible with Σ such that $\forall\omega$, $\pi'(\omega) \geq \pi(\omega)$. The least specific distribution always exists and is characterized by

$$\pi_\Sigma(\omega) = \min_{1 \leq i \leq n} \pi_{(\varphi_i, \alpha_i)}(\omega)$$

with $\pi_{(\varphi_i, \alpha_i)}(\omega) = 1$ if $\omega \models \varphi_i$, and $\pi_{(\varphi_i, \alpha_i)}(\omega) = 1 - \alpha_i$ if $\omega \not\models \varphi_i$.

Dubois *et al.* [Dubois et al., 1994] introduced a formal system for necessity-valued logic which is based in the following axiom 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) $((\neg\varphi \rightarrow \neg\psi) \rightarrow ((\neg\varphi \rightarrow \psi) \rightarrow \varphi)\ 1)$

where the inference rules for the axioms above are:

(GMP) $(\varphi\ \alpha), (\varphi \rightarrow \psi\ \beta) \vdash_{PL} (\psi\ \min\{\alpha, \beta\})$
(S) $(\varphi\ \alpha) \vdash_{PL} (\varphi\ \beta)$ if $\beta \leq \alpha$

Using this resolution rule repeatedly, a refutation-based proof procedure that is sound and complete w.r.t. the semantics exists for propositional possibilistic logic [Dubois et al., 1994]. The resolution rule allows to compute the maximal certainty level that can be attached to a formula according to the constraints expressed by the base Σ. This can be done by adding to K the clauses obtained by refuting the proposition to evaluate with a necessity level equal to 1. Then it can be shown that any lower bound obtained on $\bot$, by resolution, is a lower bound of the necessity of the proposition to evaluate. The treatment of the base in terms of formulas and necessity values (syntactical way) leads to the same results of the treatment done in terms of interpretations and possibility distribution (semantical way) [Dubois et al., 1994].

An important feature of possibilistic logic is its ability to deal with inconsistency. The level of inconsistency $Inc(\Sigma)$ of a possibilistic logic base Σ is defined as $Inc(\Sigma) = \max\{\alpha \mid K_\alpha \vdash \bot\}$, with $K_\alpha = \{\varphi \mid (\varphi, \beta) \in \Sigma \wedge \alpha \geq \beta\}$, with the convention $\max(\emptyset) = 0$. In case of partial inconsistency of Σ, *i.e.*, $Inc(\Sigma) > 0$, a refutation carried out in a situation where $Inc(\Sigma \cup \{(\neg\varphi, 1)\}) = \alpha > Inc(\Sigma)$ yields the non-trivial conclusion (φ, α), only using formulas whose certainty levels are strictly greater than the inconsistency level of the base. This is the syntactic *possibilistic entailment*, usually denoted by $\vdash_\pi$.

2.5 Possibilistic Answer Set Programming

Possibilistic Answer Set Programming (PASP) combines ASP and possibility logic by associating a necessity value with atoms and rules to provide a framework for reasoning about incomplete information and uncertainty [Nicolas et al., 2006]. A possibilistic normal (resp. definite) program is a finite set of pairs $\langle r, \alpha \rangle$ with r a normal (resp. definite) rule and α a certainty value associated with r. A possibilistic rule is indeed written as

$$\alpha : a \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_{m+n} \tag{2.9}$$

where $alpha$ belongs to an ordered linearly scale $\mathcal{S} = (0, 1]$ and a, b_i $(1 \leq i \leq m + n)$ are atoms. As in the case of normal logic program we represent by $head(r)$ the head of a rule r, $\mathcal{B}^+$ the positive and by $\mathcal{B}^-$ the negative part of the body of a rule respectively. $Nec(r)$ is the necessity value associate to a possibilistic rule r. In this way, in the PASP framework it is possible to represent logic programs associated with a certainty value.

Example 2.7. By using the following certainty scale: **1** is *absolutely certain*, **0.9** is *quasi certain*, **0.7** is *almost certain* and **0.3** is *little certain*, a possibilistic normal logic program P can be defined in the following way:

$$P = \left\{ \begin{array}{lll} r_1 = \mathbf{1} : & a \leftarrow & c, not\ b. \\ r_2 = \mathbf{1} : & b \leftarrow & d, not\ a. \\ r_3 = \mathbf{0.7} : & e \leftarrow & a, c. \\ r_4 = \mathbf{0.3} : & f \leftarrow & b, d. \\ r_5 = \mathbf{0.9} : & c \leftarrow & \top. \\ r_6 = \mathbf{0.7} : & d \leftarrow & \top. \end{array} \right\}$$

Nicolas *et al.* [Nicolas et al., 2006] captured the possibilistic semantics for ASP, first, by defining the semantics of possibilistic definite logic programs and, then, by generalizing this result by means of a program reduction. The semantics of possibilistic definite logic programs is defined by first computing the stable model of the corresponding definite logic program (by using a projection $*$ which returns necessity-value free atoms and program rules) and, then, by inferring the necessity values of the atoms belonging to the stable model.

It has also been proved how the semantical management of a program that is defined in term of a possibility distribution over all atom sets and a syntactical deduction process based on a fix-point operator defined on rules can lead to the same results. We reuse such results in Chapter 5.

An interesting feature of PASP is, similar to what happens in possibilistic logic, its ability to manage inconsistent possibilistic logic programs. Let us remember that, in such a case, a possibilistic normal logic program is inconsistent when it does not have any possibilistic answer set.

Definition 2.8. *Let P be a possibilistic normal logic program*

- *the strict α-cut of P is the subprogram $P_{>\alpha} = \{r \in P \mid Nec(r) > \alpha\}$*
- *the consistency cut degree of P is defined as*

$$ConsCutDeg(x) = \begin{cases} 0 : \textit{ if } P^* \textit{ is consistent} \\ \min_{\alpha \in \mathcal{S}}\{\alpha \mid P_{>\alpha} \textit{ is consistent}\} : \textit{ otherwise} \end{cases}$$

The concept of consistency cut is of interest in Chapter 10 where we use it to find a subset of consistent rules in a possibilistic logic program.

2.6 Classes of Logic Programs Considered in this Document

Figure 2.1 shows several classes of logic programs and the chapter(s) in which they are involved or defined. *Definite logic programs* are used in Chapter 4 as a specification to propose an approach to nonmonotonic reasoning based on implicit preferences rather than negation as failure. *Logic Programs with Ordered Disjunction* and *Possibilistic Normal Logic Programs* are considered in Chapter 5 to define the possibilistic framework of *Logic Programs with Possibilistic Ordered Disjunction*. This latter specification will serve different purposes. It is first introduced as an approach to deal with explicit preferences and uncertainty (Chapter 5). Then, in Chapter 7, it is used to manage user preferences in a context-aware system. Finally, in Chapter 10, it is used to compute an optimal decision in qualitative DMU. *Logic Programs with Ordered Disjunction* is also involved in several parts of the document. In Chapter 10, it is used as a specification to compute an optimal decision in qualitative decision making when knowledge is fully certain and preferences are all-or-nothing. In Chapter 8, it is extended into the framework of *Nested Logic Programs with Ordered Disjunction* for handling nested preferences. Finally, *Disjunctive Logic Programs*, *Generalized Disjunction Logic Programs*, *Nested Logic Programs* are particular cases of *Nested Logic Programs with Ordered Disjunction* and they are used in chapter 8.

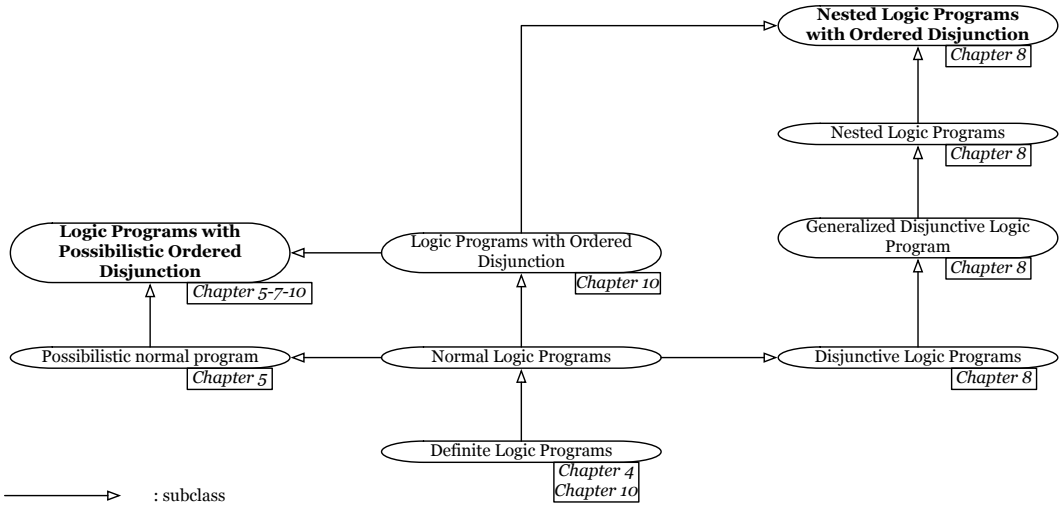


Fig. 2.1. Logic Programs Considered

Part I

Preferences in Nonmonotonic Reasoning and Uncertainty

3

Handling Incomplete and Uncertain Information

The aim of this chapter is twofold. On one hand, we give a picture of the different attempts to formalize nonmonotonic reasoning which have been proposed in the vast literature available. In presenting some noticeable approaches and their ideas behind the handling of default rules, *i.e.*, statements such as *normally p's are q's*, we will exhibit the following (classical) example:

Example 3.1. If we know that (i) *birds normally fly*, that (ii) *penguins normally do not fly*, and that (iii) *penguins are birds*, then we should be able to conclude, in a plausible way, that a bird can fly, but that a penguin can not.

We will see how the *birds-can-fly* example can be encoded in different approaches such as *default logic*, *circumscription*, *autoepistemic logic*, *possibilistic logic* and *logic programming*. In particular, we will discuss the role played by preferences in the selection of default rules and how preferences are understood in nonmonotonic reasoning.

On the other hand, we will describe some approaches able to represent and to reason about uncertain information. We will see that the joint handling of incomplete and uncertain information is addressed by only few approaches, despite the fact that reasoning under uncertainty is an extensively research area.

The chapter is organized as follows. After overviewing three seminal approaches for nonmonotonic reasoning (Section 3.1), in Section 3.2, we present the possibilistic logic approach for handling default rules. We dedicate Section 3.3 to present how default rules are handled in ASP. Section 3.4 overviews the main trends for handling nonmonotonicity and uncertainty. Finally, Section 3.5 concludes the chapter by discussing the overviewed approaches and by pointing out the directions followed in the next two chapters.

3.1 Seminal Approaches to Nonmonotonic Reasoning

Nonmonotonic reasoning is an attempt to formalize common-sense human reasoning. Since in classical logic there is no means to distinguish between classical models and it is not possible to retract or revise previous conclusions on the basis of the new information, a different type of logics, *nonmonotonic logics*, have been proposed to formalize nonmonotonic reasoning.

In this section, we overview three nonmonotonic logics, *default logic* [Reiter, 1980], *circumscription* [McCarthy, 1980], and the modal nonmonotonic *autoepistemic logic* [Moore, 1985] and we point out some extensions that were introduced to deal with preferences about default rules.

3.1.1 Default Logic

Default logic is a nonmonotonic logic introduced by Reiter [Reiter, 1980] in order to formalize default reasoning. A *default theory* $\Delta = \langle D, W \rangle$ consists of a set of first order logic or propositional formulas W and a set of defaults rules D of the form

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \tag{3.1}$$

in which α, $\beta_1, \ldots, \beta_n$, γ are first order logic or propositional formulas. In addition, α is called the *prerequisite*, $\beta_1, \ldots, \beta_n$ the *consistency conditions* or *justifications*, and γ the *conclusion* of the default. The intuitive reading of a default rule *normally p's are q's* in default logic is *an α is a γ if $\beta_1, \ldots, \beta_n$ are consistent* (there is no information that $\neg\beta_1, \ldots, \neg\beta_n$ hold).

Default rules can induce one or more *extensions* w.r.t. a set W to represent which default rules ought to be believed more appropriate. Generally, a default theory may have many extensions or none. The most direct use of default logic is in the encoding of default information. Let us consider the knowledge representation problem in Example 3.1.

Example 3.2. Let us consider the default theory $\Delta = \langle W, D \rangle$, where symbols b, p, f stand for *bird*, *penguin*, and *fly* respectively:[1]

$$W = \left\{ \begin{array}{c} p \rightarrow b \\ p \end{array} \right\} \qquad\qquad D = \left\{ \begin{array}{c} \frac{b:f}{f} \\ \frac{p:\neg f}{\neg f} \end{array} \right\}$$

We obtain two extensions: one in which $\{p, b, f\}$ are true and another in which $\{p, b, \neg f\}$ are true.

Intuitively, in the above example, we only want the last extension since the more specific default $\frac{p:\neg f}{\neg f}$ should take precedence over the less specific one $\frac{b:f}{f}$. In other words, it is desirable to be able to express preferences among default rules.

The usual solution, originally proposed in [Reiter and Criscuolo, 1981], is to establish a *precedence* among these two interacting defaults by adding the negation of the exception, p, to the justification of the less specific default rule. This amounts to replace $\frac{b:f}{f}$ by $\frac{b:f \wedge \neg p}{f}$ which yields to the desired results, *i.e.,* a single extension containing $\{p, b, \neg f\}$. This approach is based on the idea of resolving conflict by appealing to *specificity* following the *inheritance principle property* [Horty, 1994].

From a more general point of view, it is desirable to have a notion of preference w.r.t. default rules to be able to select the most appropriate rules to reason with. Indeed, several approaches have been proposed to extend default logic with an ordering on the set of default rules [Brewka, 1994; Delgrande and Schaub, 1997a, 2000; Rintanen, 1995, 1998]. Then,

[1] $\rightarrow$ denotes the classical implication.

an *ordered default theory* $\langle D, W, < \rangle$ is a finite set D of default rules, a finite set W of formulas, and a strict partial order $<$ on the default rules. Since there are a lot approaches, we refer to [Delgrande et al., 2004] for an overview and comparison. A noticeable approach, able to isolate interacting defaults with different specificity and to define exceptions to default, was proposed in [Delgrande and Schaub, 1997b].

3.1.2 Circumscription

Circumscription [McCarthy, 1980] is one of the best known formalisms of nonmonotonic reasoning. The idea behind circumscription is that of *logical minimization*. A formula α follows from a theory T by circumscription if α is true in all models of T that are minimal.

In applications to default reasoning, circumscription is used to minimize *abnormalities* of default rules. The intuitive reading of a default rule *normally p's are q's* in circumscription is *every p that is not abnormal is q*. For this purpose, the language is enriched by abnormality propositions designated ab_1, ab_2, ..., *etc*. These propositions address exceptional cases to a default rule at hand. In this way, a default statement like *normally p's are q's* is represented as a classical implication of the form

$$p \wedge \neg ab \rightarrow q \tag{3.2}$$

Then, following this general principle, a set of default rules is transformed into a set of implications. Given a generic world description (D, W), in which D and W represent the set of default and strict rule[2] respectively, the corresponding sets of classical implication rules are defined as $D^{ab} = \{p \wedge \neg ab \rightarrow q \mid p \rightsquigarrow q \in D\}$[3] and $W^* = \{p \rightarrow q \mid p \rightarrow q \in W\}$.

Example 3.3. Let us consider the world description $(D, W) = \langle \{b \rightsquigarrow f, p \rightsquigarrow \neg f\}, \{p \rightarrow b\} \rangle$. Then:

$$D^{ab} = \left\{ \begin{matrix} b \wedge \neg ab_1 \rightarrow f \\ p \wedge \neg ab_2 \rightarrow \neg f \end{matrix} \right\} \qquad\qquad W^* = \Big\{ p \rightarrow b \Big\}$$

The set of rules in $D^{ab} \cup W^*$ can be seen as a description of our birds scenario in Example 3.1.

In circumscription, we also have to express that things are considered as normal as possible provided that there is no evidence to the contrary. This assumption is formally accomplished by circumscribing a world description [McCarthy, 1980; Lifschitz, 1985]. The semantical underpinnings for circumscription are given by minimal models.

Example 3.4. Keeping with the birds example, let us consider the fact $\{p\}$. Then, the models of $\{p\}$, D^{ab}, and W^*, in which we just list the positive literals, are:

$$\{p, b, f, ab_1, ab_2\} \{p, b, ab_1, ab_2\} \{p, b, f, ab_2\} \{p, b, ab_1\}$$

[2] Strict rules are rules that do not have exceptions.
[3] $\rightsquigarrow$ denotes the no classical implication.

Circumscribing propositions ab_1 and ab_2 while varying b, f, p amounts to reasoning w.r.t. models that have the fewest abnormality propositions. There are two minimal models: $\{p, b, ab_1\}$ and $\{p, b, f, ab_2\}$.

As a consequence, it is not possible to derive ab_1, ab_2, or f, nor their negation. This shows that circumscription does not respect the principle of specificity [McCarthy, 1986].

For fixing this problem, McCarthy introduced a prioritized version of circumscription, *prioritized circumscription* by assuming that abnormality propositions are a priori assigned with different priorities. The idea is to partition the circumscribed propositions into priority layers. To this end, a set of propositions is partitioned into disjoint subsets, in which the propositions in a partition at the lower level should take priority over those in the above levels. Lifschitz shows in [Lifschitz, 1985] that a prioritized circumscription becomes a sequence of ordinary circumscriptions in which propositions in a higher layer are minimized while propositions in a lower layer are varied.

In the birds example, giving the abnormality proposition of the more specific default rule ab_2 priority over the one of the less specific rule ab_1, yields to conclude $\{p, b, \neg f\}$.

3.1.3 Autoepistemic Logic

Autoepistemic logic [Moore, 1985] belongs to the family of modal nonmonotonic logics. In modal logics, statements may be classified into three different categories, *i.e., provable*, *refutable*, and *conceivable*. In this respect, Moore adapts the approach of McDermott and Doyle [McDermott and Doyle, 1980] by referring to the dual notion of *conceivable* which is, according to him, the same as *believe*. Then, the intuitive reading of a default rule *normally p's are q's* in autoepistemic logic is *if a given p was not q, we would know it*.

Autoepistemic logic aims at formalizing an agent's reasoning about her own beliefs. To this end, the logical language is augmented by a modal operator $\Box$ in which a formula $\Box a$ is read as *a is believed*. A modal default is written as

$$p \wedge \neg\Box\neg q \rightarrow q \tag{3.3}$$

In a given theory T of such formulas, an autoepistemic extension or expansion E is defined as $E = Th(T \cup \{\neg\Box p \mid p \notin E\} \cup \{\Box p \mid p \in E\}$.

Example 3.5. The autoepistemic theory T for the default rules in Example 3.1 consists of the following three modal defaults:

$$T = \left\{ \begin{array}{l} b \wedge \neg\Box\neg f \rightarrow f \\ p \wedge \neg\Box f \rightarrow \neg f \\ p \wedge \neg\Box\neg b \rightarrow b \end{array} \right\}$$

Then, given p along with the three modal defaults, we obtain two autoepistemic extensions containing $\{\neg f\}$ and $\{f\}$.

For resolving conflicts among default modal rules, prioritized versions of autoepistemic logic have been defined in *hierarchic autoepistemic logic* [Konolige, 1989] and *prioritized autoepistemic logic* [Rintanen, 1994]. In the former approach, after defining a layered sets of

formulas, all stable expansions are generated layer by layer. Then, expansions are ordered according to the layer ordering. Instead, in the latter approach, a preference order assigned to the modal default induces an ordering on stable expansions.

3.2 Dealing with Incomplete Information in Possibility Theory

In [Benferhat et al., 1997], a complementary survey of the approaches to nonmonotonic reasoning outlined above and in [Sombé, 1990; Brewka et al., 1993] is presented. In particular, it is shown how *possibilistic logic* [Benferhat et al., 1992, 1998], *conditional objects* [Dubois and Prade, 1994], and *infinitesimal probabilities* [Pearl, 1988; Adams, 1975] can achieve a satisfactory treatment of default reasoning in the presence of incomplete information. Indeed, they can capture all the properties which a nonmonotonic consequence relation should satisfy according to Kraus, Lehmann and Magidor's System P [Kraus et al., 1990]. Since System P defines a set of postulates which provide a very cautious inference system which suffers from *irrelevance*[4], Lehmann [Lehmann and Magidor, 1992] and Pearl [Pearl, 1990] have proposed to add an inference rule, called *rational monotonicity*, and a particular entailment called *rational closure*. Then, one way of adding the rational monotony to System P is to use the System Z [Pearl, 1990]. Possibilistic logic, infinitesimal probabilities and conditional objects have shown to be able to satisfy this rational closure entailment.

3.2.1 Handling Default Rules in Possibilistic Logic

Possibilistic logic [Dubois et al., 1994] handles pairs of the form (φ, α) made of a classical logic formula φ and of a certainty weight α belonging to a totally ordered set $\mathcal{S}$ (see Chapter 2, Section 2.4). A pair (φ, α) expresses that φ is more certain than $\neg\varphi$, the latter being all the more impossible as α becomes higher. In the handling of default rules, the possibilistic logic inference mechanism turns out to express a *preferential entailment* [Benferhat et al., 1992, 1998]. Indeed, the semantics of possibilistic logic can be expressed in terms of a complete ordering over the interpretations.

Then, a conditional knowledge base or default base $\Delta = \{p_i \rightsquigarrow q_i \mid 1 \leq i \leq n\}$ made of default rules of the form *normally, if* p_i *then* q_i can be viewed as a set of constraints stating that $\Pi(p_i \wedge q_i) > \Pi(p_i \wedge \neg q_i)$, *i.e.*, $p_i \wedge q_i$ is strictly more possible than $p_i \wedge \neg q_i$. Strict rules are special cases of the above constraint in the sense that all the states where p_i and $\neg q_i$ are true result to be impossible. This is captured by assigning $\Pi(p_i \wedge \neg q_i) = 0$ to the above constraint. Default rules can then be turned into possibilistic logic formulas $(\neg p_i \vee q_i, \alpha)$ in which the weight α reflects a rule priority, computed from the least informed possibility measure compatible with the set of constraints.

In [Benferhat et al., 1992], it is shown that each default rule of a default base Δ, can be associated with a possibilistic formula $(\neg p_i \vee q_i, \alpha)$, in which α represents its specificity level $\alpha = \frac{Z(r)+1}{n+2}$, n being the index of the last stratum of the Z-ordering of Δ, and $Z(r)$

[4] If a formula δ is a plausible consequence of α, and if a formula β is a formula composed of propositional symbols which do not appear in the default base, then δ cannot be deduced from $\alpha \wedge \beta$. For instance, from the rule *normally birds fly*, it is not possible to deduce that *red birds fly too*, when no assertion in the knowledge base deals with *red*.

corresponding to the rank of the stratum to which the rule r belongs.[5] Under this rule ranking, possibilistic logic entailment is equivalent to rational closure entailment [Kraus et al., 1990; Gärdenfors and Makinson, 1994].

Example 3.6. Let us consider the default rules in Example 3.1. The Z-ordering of Δ is:

$$\Delta_0 = \left\{ b \rightsquigarrow f \right\} \qquad \Delta_1 = \left\{ \begin{matrix} p \rightarrow b \\ p \rightsquigarrow \neg f \end{matrix} \right\}$$

Then, the set of possibilistic formulas associated to Δ is

$$\Sigma_\pi = \left\{ \begin{matrix} (\neg b \vee f, \frac{1}{3}) \\ (\neg p \vee \neg f, \frac{2}{3}) \\ (\neg p \vee b, 1) \end{matrix} \right\}$$

in which the number is only a way to represent an order and it can be substituted by any other linearly ordered scale. Let $C = \{(p, 1)\}$ meaning that we are considering a penguin p. Then, applying the resolution rule of possibilistic logic, where $\vdash_\pi$ denotes the possibilistic entailment (see Chapter 2, Section 2.4), $Inc(\Sigma_\pi \cup C) = \frac{1}{3}$ since $\Sigma_\pi \cup C \vdash_\pi (f, \frac{1}{3})$ and $Inc(\Sigma_\pi \cup C) = \frac{2}{3}$ since $\Sigma_\pi \cup C \vdash_\pi (\neg f, \frac{2}{3})$. Hence, $\Sigma_\pi \cup C \vdash_\pi (\bot, \frac{1}{3})$. So, the final base only contains the formulas $\{(\neg p \vee \neg f, \frac{2}{3}), (\neg p \vee b, 1)\}$. So $\Sigma_\pi \cup C \vdash_\pi (\neg f, \frac{2}{3})$. One concludes that a penguin is unable to fly.

It can be observed how, in possibilistic logic, conflicts between default rules are solved by assigning a priority between default rules based on their specificity. However, this method suffers from the *blocking inheritance problem*.[6] For instance, if we had the rules *birds generally have legs* and *penguins generally do not fly*, then it would not be possible to conclude that *penguins have legs*. This problem has been addressed in [Dupin de Saint-Cyr and Prade, 2008] by a rewriting method which rewrites default rules into strict rules and adds special rules for handling incomplete information. This kind of rewriting is what we will mimic in the setting of extended definite logic program as we will see in Chapter 4.

3.2.2 Other Approaches

In the infinitesimal probabilities approach [Adams, 1975; Pearl, 1988], a default rule $p \rightsquigarrow q$ is modeled by the constraint $P(q \mid p) \geq 1 - \epsilon$, in which ϵ denotes an arbitrary small positive real number and P denotes a probability measure. Roughly speaking, given a default base Δ, the consequence relation is then defined as: q is a consequence of p w.r.t. Δ if the

[5] Pearl [Pearl, 1990] provides an algorithm which gives a stratification of a set of default rules in a way that reflects the specificity of the rules. Roughly speaking, the first stratum contains the most specific rules, *i.e.,* which do not admit exceptions (at least, expressed in the considered default base), the second stratum has exceptions only in the first stratum and so on. The algorithm allows to stratify Δ into $(\Delta_0, \Delta_1, \ldots, \Delta_n)$.

[6] Blocking inheritance is a weakness of System Z [Pearl, 1990]. Refinements of the Z-ordering have been proposed such as the maximum-entropy approach of Goldzsmidt in [Goldzsmidt et al., 1993] which also takes into account the number of rules tolerating a formula and not only their rank orders.

conditional probability $P(q \mid p)$ is very high whenever the conditional probability attached to each default rule in Δ is also very high.

Dubois and Prade discussed how Adams' infinitesimal probabilities can be expressed in terms of conditional objects in [Dubois and Prade, 1994]. A conditional object $q \mid p$ can be seen as a purely symbolic counterpart to the conditional probability $P(q \mid p)$. Thus, it shows that numerical probabilities do not play a crucial role in the modeling of preferential entailment, since no probability degrees, whether infinitesimal or not, are necessary when conditional objects are used.

More recently, in [Lukasiewicz, 2005],it is shown how probabilistic logic can also satisfy the rationality postulates of System P and the property of rational monotonicity.

3.3 Dealing with Incomplete Information in Logic Programming

Logic programming can serve both as a logical paradigm for declarative programming and as a framework for knowledge representation [Baral and Gelfond, 1994; Gelfond, 1994]. In *extended normal logic programs* under *answer set semantics* (see Chapter 2, Section 2.1), incomplete information is handled by *default negation* or *negation as failure*, *i.e., not*, which is allowed to appear in the body of the rules of a logic program. Then, a default rule, *i.e., normally p's are q's*, is usually modeled by a normal rule

$$q \leftarrow p, not\ ab \tag{3.4}$$

or by an extended normal rule

$$q \leftarrow p, not\ ab, not \neg q \tag{3.5}$$

when weak exceptions, *i.e., not*$\neg q$, need to be modeled. Informally, the piece of knowledge *not ab*, captures the knowledge about an abnormal condition *ab* w.r.t. the default rule. From a semantics point of view, modeling exceptions by means of negated-by-failure atoms amounts to check whether a strong exception, *i.e., ab*, or a weak exception, *i.e.,* $\neg q$ can be proved. If it is the case, more specific information is preferred to that which is more general, in accordance to the *inheritance reasoning principle* [Touretzky, 1986].

Therefore, in a logic program the concept of preference is already present at semantics level in an intrinsic way. The concept of preferred models is intrinsically defined in the answer set semantics definition of an answer set of a logic program. In fact, answer sets correspond to the preferred models of a logic program and tend to identify those models in which everything is *as normal as possible*.

However, this basic approach of assuming normality still faces a difficulty [Brewka et al., 2008]. There are cases in which extended rules can support conflicting conclusions, as shown in the following example.

Example 3.7. Let us consider the knowledge representation problem in Example 3.1. One possible encoding of this problem is by means of the following logic program P:

$$P = \begin{Bmatrix} r_1 : \ b \ \leftarrow \quad\quad p. \\ r_2 : \ f \ \leftarrow b, not\ (\neg f). \\ r_3 : \neg f \leftarrow \quad p, not\ f. \end{Bmatrix}$$

It can be checked that in context $\{p\}$ (penguin), P has two answer sets, $\{p, b, \neg f\}$ $\{p, b, f\}$ which are in conflict ($\neg f$ and f can be concluded).

In such situations, preferences can be used to solve conflicts. Two options are available so far [Brewka et al., 2008]: to use preferences that are *implicit* in the available information, or to use *explicit* preference information that needs to be manually specified.

3.3.1 Implicit Preferences in Logic Programs

Implicit preferences are based on the notion of *specificity* [Horty, 1994] of a rule as a way to assign a priority between rules that are in conflict. For instance, System Z [Pearl, 1990] provides a good example of a system that implicitly prefers more specific information. In the logic programming literature, there are some approaches that deal with specificity [You et al., 1999; Garcia et al., 2000, 2009].

In [You et al., 1999] an inheritance network is compiled to a general logic program and a query on the network is answered by a logic programming proof procedure w.r.t. the compiled logic program. This graphical model based on path computing does not allow to take into account rules with multiple atoms in the body.

The approach in [Garcia et al., 2000] translates a default theory, expressed through a set of conditional defaults, into an extended logic program whose semantics is defined in terms of the well-founded semantics [Van Gelder et al., 1991]. However, it suffers from cautiousness. In fact, it does not allow to infer floating conclusions [Horty, 2002] (*e.g.*, Nixon diamond example).[7] On the other hand, it faces *blocking inheritance* (*e.g.*, penguins do not have legs because of being abnormal birds).

Garcia *et al.* [Garcia et al., 2009] proposed to deal with the problem of representing default rules with exceptions by generating the suitable normal logic program from a compact representation of the information. Their approach is based on a previous work of Delgrande and Schaub [Delgrande and Schaub, 1997b] which proposed a way to compile specificity into nonmonotonic theories such as default logic, circumscription and autoepistemic logic. Since extended normal logic programs are closely related to default logic [Gelfond and Lifschitz, 1991], they have been able to adapt this method to extended programs both in theoretical and practical points of view. In particular, they capture exceptions to defaults by means of negated-by-failure atoms, as shown in the next example.

Example 3.8. Given the default specification in Example 3.1, the following extended normal logic program is compiled.

[7] The Nixon Diamond is a scenario in which default rules can lead to mutually inconsistent conclusions. Such scenario consists of the following set of defaults: *normally, Quakers are pacifist*; *normally, Republicans are not pacifist*; *Richard Nixon is both a quaker and a republican*. Hence, it is not possible to say whether Nixon is a pacifist or not [Reiter and Criscuolo, 1981]. The name diamond comes from the fact that such a scenario, when expressed in inheritance networks, has a diamond shape.

$$P = \left\{ \begin{array}{lll} f & \leftarrow b, not\ \neg f, not\ p. \\ b & \leftarrow \quad p. \\ \neg f & \leftarrow \quad p, not\ f. \end{array} \right\}$$

It can be checked that, given a penguin p, the only answer set is $\{p, b, \neg f\}$.

This approach is less cautious than the one in [Garcia et al., 2000] (it can handle the Nixon Diamond example) and it also faces the blocking inheritance problem. In Chapter 4, we propose a novel approach based on an alternative rewriting procedure, not based on negation as failure, which introduces exceptions and incomplete knowledge by means of strong negation. Then, it will be interesting to see how our approach relates to the approach in [Garcia et al., 2009].

When specificity is hard to be handled [Touretzky et al., 1987] or preferences need to be decided independently of specificity (such as in the law domain for instance), it is more convenient to express preferences on defaults in an explicit way.

3.3.2 Explicit Preferences in Logic Programs

Explicit preferences can be handled in different ways in ASP and numerous proposals have been made including [Gelfond and Son, 1998; Zhang and Foo, 1997; Brewka and Eiter, 1999; Sakama and Inoue, 2000; Delgrande et al., 2003; Brewka et al., 2004b; Van Nieuwenborgh and Vermeir, 2006]. A detailed comparison between these approaches can be found in [Schaub and Wang, 2001; Delgrande et al., 2004]. In [Delgrande et al., 2004], several categories have been identified to compare such approaches: whether they employ a metaformalism for characterizing *preferred answer sets* [Brewka and Eiter, 1999] versus they translate an ordered logic program into an ordinary logic program such that the preferred answer sets of the former correspond to the answer sets of latter [Gelfond and Son, 1998; Delgrande et al., 2003]; or whether they specify *preference among rules* [Zhang and Foo, 1997; Brewka and Eiter, 1999; Delgrande et al., 2003; Van Nieuwenborgh and Vermeir, 2006] versus *preference among atoms/literals* [Sakama and Inoue, 2000; Brewka et al., 2004b], are some of the categories considered.

In [Brewka et al., 2008], it is pointed out how approaches based on preferences among rules still face some shortcomings. In fact, preferences can depend on context, *i.e.,* what is preferred under certain circumstances may not be preferred in others, and explicit preference orderings among rules are not sufficiently flexible to take different contexts into account. Sometimes, it is more convenient or necessary to express preferences directly in terms of the literals contained in the answer sets. In particular, when we aim to represent conditional preferences differentiating between literals to be accepted based on other literals already accepted or rejected [Brewka et al., 2004b].

One way to deal with these limitations was proposed by Brewka *et al.* [Brewka et al., 2004b] in the logic programming framework of Logic Programs with Ordered Disjunction (LPODs). To express preferences among alternative literals, ordered disjunction $\times$ (first proposed in Qualitative Choice Logic [Brewka et al., 2004a]) is used in the heads of rules. Since several parts of this document make explicit reference to this approach, a technical

overview about LPODs was given in Chapter 2, Section 2.2.4, while its main characteristics from a nonmonotonic reasoning point of view are described in the next section.

3.3.3 LPODs: An Approach for Context-dependent Preferences Among Literals

Essentially, the LPODs specification is designed to do two things at once. First, it determines the space of answer sets. Secondly, LPODs uses preferences encoded by ordered disjunctions in the heads of rules and aggregates them into a global preference relation on answer sets. Preferences encoded by LPODs can be used to specify *explicit preferences about exceptions* (as we will consider in this section and in Chapter 5) or *user preferences* (as we will see later in Chapter 6).

From a nonmonotonic point of view, ordered disjunction can be used to select the best literals to reason with. In particular, ordered disjunction rules can express preferences such as *being c, a is preferred to b, i.e.,* they can be used to specify an explicit preference order among exceptions to default rules which depend on contextual knowledge. This idea is illustrated in the next examples.

Example 3.9. Let us consider the logic program in Example 3.7 in which we consider some extra knowledge stating that assuming that (i) *penguins do not fly* is more reasonable than assuming that (ii) *penguins do fly*.

$$P = \begin{Bmatrix} r_1: & b & \leftarrow & p. \\ r_2: & f & \leftarrow & b, not\ (\neg f). \\ r_3: & \neg f & \leftarrow & p, not\ f. \\ r_4: & \neg f \times f & \leftarrow & p. \\ r_5: & p & \leftarrow & \top. \end{Bmatrix}$$

The preference order among the exceptions to default rules r_2 and r_3 is captured by the ordered disjunction rule r_4. Therefore, according to the LPOD semantics (see Chapter 2, Section 2.2.4), P has two answer sets, $M_1 = \{p, b, \neg f, \}$, and $M_2 = \{p, b, f\}$. By taking the satisfaction degree of each answer sets w.r.t. rule r_4 into account, M_1 is preferred to M_2.

Thanks to negation as failure, LPODs can also capture preferences about exceptions w.r.t. default rules which can depend on some incomplete knowledge.

Example 3.10. Let us consider the following program in which, (i) if we consider *super-penguins*, then it is more reasonable to assume that *super-penguins fly*, since super-penguins are a special type of penguin. Otherwise, (ii) it is more reasonable to assume that *super-penguins do not fly*. This preference order among exceptions is captured by rules r_4 and r_5 in the following program:

$$P' = \begin{Bmatrix} r_1: & b & \leftarrow & p. \\ r_2: & f & \leftarrow & b, not\ (\neg f). \\ r_3: & \neg f & \leftarrow & p, not\ f. \\ r_4: & \neg f \times f & \leftarrow & p, not\ sp. \\ r_5: & f \times \neg f & \leftarrow & p, sp. \\ r_6: & sp & \leftarrow & \top. \end{Bmatrix}$$

According to the LPOD semantics, P' has two answer sets, $M'_1 = \{sp, b, \neg f, \}$, and $M'_2 = \{sp, b, f\}$. In this case, the most preferred answer set is M'_2.

Until now we have assumed that knowledge was completely certain. However, in many applications, knowledge can be pervaded with uncertainty. Dealing with incomplete information and uncertainty is part of most intelligent behavior, and, consequently, the production of intelligent behavior in machines often requires a proper representation and management of uncertainty as well. In the next section, we will overview different formalisms able to handle uncertainty.

3.4 Dealing with Incomplete Information and Uncertainty

Among logic-based approaches, the most noticeable works are presented in [Dupin de Saint-Cyr and Prade, 2008; Lukasiewicz, 2005] in the probabilistic and possibilistic logic settings. In [Lukasiewicz, 2005], a weak nonmonotonic probabilistic logic is proposed to handle statements such as *normally p's are q's with probability of at least* α in a uniform framework. Such framework can handle strict logical knowledge and purely probabilistic knowledge from probabilistic logic, as well as default logical knowledge from conditional knowledge bases. In the possibilistic setting [Dupin de Saint-Cyr and Prade, 2008], uncertain default rules such as *normally p's are q's with certainty at least* α are handled by a two stepped process. First, default rules are rewritten into possibilistic logic formulas and preferential entailment is used to draw plausible conclusions (see Section 3.2.1). Then, possibility logic resolution is used to handle the certainty associated to the information described in the possibilistic knowledge base (see Chapter 2, Section 2.4).

On the other hand, logic programming with uncertainty is an extensively research area and it has proceeded along various research lines of logic programming (for an interesting historical recollection in this topic, we refer to [Subrahmanian, 2007]).

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 different formalisms to handle uncertain information such as *annotated logics* [Kifer and Subrahmanian, 1992], *probabilistic logic* [Ng and Subrahmanian, 1992; Lukasiewicz, 1998; Kern-Isberner and Lukasiewicz, 2004; Saad and Pontelli, 2005; Baral et al., 2009], *fuzzy logic* [Alsinet and Godo, 2002; Nieuwenborgh et al., 2007; Bauters et al., 2010b], *bilattices* [Fitting, 1991], *evidence theory* [Baldwin, 1987; Wan, 2009], and *possibilistic logic* [Dubois et al., 1991; Nicolas et al., 2006; Nieves et al., 2011; Bauters et al., 2010a]. Basically, such approaches differ in the underlying notion of uncertainty and how uncertainty values, associated to clauses and facts, are managed. However, this long list of approaches can be radically shortened when one focuses only on works which can handle incomplete and uncertain information together. Such approaches are those which extend answer set semantics.

3.4.1 Fuzzy Answer Set Programming

Fuzzy answer set programming (FASP) is a generalization of ASP based on fuzzy logic [Nieuwenborgh et al., 2007]. Essentially, literals are associated with a truth degree from

the unit interval $[0,1]$ which refers to the intensity of a property as a matter of degree (*e.g.*, a person is sick if she suffers from *intensive* shivering and has a *high* temperature). These degrees are captured by means of functional relationships using connectives from particular fuzzy logics. As fuzzy logic gives the user great flexibility regarding the choice for the interpretation of the notions of negation, conjunction, disjunction and implication, the FASP framework is highly configurable.

As also stated in [Bauters et al., 2010b] by the same authors of FASP, FASP is basically a generalization of ASP to continuous domains and not a form of approximate reasoning. As such, FASP is not well-equipped to deal with vagueness and with uncertainty in general. A preliminary work on the combination of FASP with possibility theory is proposed in [Bauters et al., 2010b].

3.4.2 Probabilistic Answer Set Programming

The work in [Baral et al., 2009] presents a nonmonotonic probabilistic logic programming language (P-log) which is based on, and generalizes, logic programming under answer set semantics and Causal Bayesian networks. The P-log semantics is defined in terms of probabilistic models representing possible worlds of the domain being represented. Nonmonotonicity is achieved by means of an updating mechanism which changes the collection of possible worlds when new probabilistic information is added to the program.

Another example of probabilistic nonmonotonic formalism can be found in [Saad and Pontelli, 2005]. In this work, it is possible to represent rules in which every atom is annotated with a probability interval. Then, the associated semantics is defined in terms of a probabilistic well-founded semantics and a probabilistic stable model semantics.

On the other hand, the criticism to the probabilistic handling of uncertainty [McCarthy and Hayes, 1969; Dubois and Prade, 2004] has fostered other approaches based on possibilistic logic [Dubois et al., 1991] in which a certainty degree is attached to default rules statements. [Dubois et al., 1991] can be considered the first approach towards the uncertainty handling in logic programs from a possibility theory point of view. Since such approach was not dealing with nonmonotonicity, it has been superseded by more recent works in Possibilistic Answer Set Programming (PASP).

3.4.3 Possibilistic Answer Set Programming

Possibilistic Answer Set Programming (PASP) approaches [Nicolas et al., 2006; Nieves et al., 2011; Bauters et al., 2010a] extend answer set semantics with possibility theory in order to deal with a reasoning which is uncertain and nonmonotonic. PASP combines the nonmonotonic features of ASP with the uncertainty handling of possibilistic logic, according to which, necessity measures are associated to rules and atoms. The pioneer work of PASP of Nicolas *et al.* [Nicolas et al., 2006] captured a possibilistic semantics for ASP by first defining the semantics of possibilistic definite logic programs and then generalizing this result to possibilistic normal programs (see Chapter 2, Section 2.5). The work in [Nieves et al., 2011] basically extend the results of [Nicolas et al., 2006] to disjunctive logic programs.

More recently, [Bauters et al., 2010a] has proposed a revisited semantics for PASP. The core idea of this approach is to translate a normal program to a set of constraints on possibility distributions. Then, the least specific possibility distribution, which is compatible with these constraints, corresponds to the answer sets of the program. Such idea is extended to cover possibilistic (normal) logic programs. This revisited possibilistic semantics has also been applied to possibilistic disjunctive logic programs [Bauters et al., 2011].

3.5 Discussion

Nonmonotonic reasoning is an essential feature for systems that aim at formalizing human common-sense reasoning. In this chapter, we have seen several formalisms able to deal with incomplete knowledge. Such approaches differ on the way default rules and exceptions are represented and handled.

In the literature, two major trends can be identified which have been developed in an independent way: nonmonotonic logics approaches, and approaches which are based on preferential and rational closure entailments of System P [Kraus et al., 1990], and System Z [Pearl, 1990]. On the other hand, ASP is considered one of the most successful implementation of nonmonotonic logics. This is not surprising if we think that extended logic programs are a simple fragment of Reiter's default logic [Reiter, 1980].[8]

The notion of preference is strongly connected with nonmonotonic reasoning as preferences represent an effective way to choose between default rules (which can be in conflict). The way in which preferences are specified depends on the formalism chosen. Implicit preferences can emerge by considering specificity according to the information described by the defaults. For instance, in System Z and possibility theory, default rules are rank-ordered in such a way that the most specific rules, whose conclusions may contradict the conclusions of more general defaults, receive a higher level of priority. Instead, in logic programming, preferences are mainly understood in an explicit way and they are used to specify preference among rules or literals. While there are a lot of approaches for handling preferences in an explicit way in logic programming, only a few approaches have been proposed to handle preferences in an implicit way.

As such, in Chapter 4 we propose to use a novel approach, based on the notion of specificity, for handling incomplete information in logic programs. Our approach consists in an algorithm which makes it possible to handle incomplete knowledge without negation as failure. In fact, incomplete information is handled by making exceptions explicit in rules by means of strong negation and by adding completion rules.

On the other hand, reasoning with incomplete information and reasoning with uncertainty are important research trends that have been developed quite independently from each other. Indeed, not so many works in the literature deal with both problems at the same time. However, conclusions that we want to privilege in a given context may themselves be pervaded with uncertainty. In particular, in the case of LPODs, explicit context-dependent

[8] In particular, ASP is a special case of default logic by rewriting a rule of the form $a \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_{m+n}$ into the default $b_1 \wedge \ldots \wedge b_m : \neg b_{m+1}, \ldots, \neg b_n / a$. A direct mapping between extensions and answer sets have also been shown in [Gelfond and Lifschitz, 1991].

preferences can be specified to select default rules to be used in the reasoning. These rules may be pervaded with uncertainty. Indeed, when a rule of the form *normally p's are q's* and a rule of the form *normally if c then a is more reasonable than b* are stated, no estimate of the certainty of having q in context p and a preferred to b in context c is respectively provided.

This issue fosters the investigation of a logic programming framework able to reason about incomplete knowledge and explicit preferences about default rules at the same time. Such a framework is presented in Chapter 5.

4

An Approach to Nonmonotonic Reasoning using Implicit Preferences

According to the classical handling of nonmonotonic reasoning in ASP (see Chapter 3, Section 3.3), exceptions have to be made explicit by means of negation as failure. Indeed, negation as failure captures abnormal situations and blocks the applicability of default rules in the reasoning process. In this way, nonmonotonic conclusions can be derived from incomplete information.

Dealing with incomplete information by means of negation as failure is not the only solution. When several default rules contain different levels of *specificity*, the notion of specificity can be used to identify exceptions to default rules and to deal with incomplete information. Indeed, specificity can provide a basic mechanism to define a method for rewriting default rules into strict ones in which exceptions are made explicit.

Specificity is a well-known notion in nonmonotonic reasoning [Pearl, 1990; Goldzsmidt et al., 1993]. It can be understood as an *implicit preference order* over default rules. For instance, System Z [Pearl, 1990] provides a good example of a system that implicitly prefers more specific information. Approaches such as [Brewka, 1994; Delgrande and Schaub, 1997b] have shown that generating exceptions based on the notion of specificity is appealing in nonmonotonic logics such as default logic, autoepistemic logic and circumscription.

More recently, some approaches to exception rewriting based on specificity have been proposed in the context of ASP [Garcia et al., 2000, 2009]. For instance, Garcia *et al.* [Garcia et al., 2009] has recently adapted the work in [Delgrande and Schaub, 1997b] to propose a methodology for representing default rules with exceptions by automatically generating negated-by-failure exceptions from a compact representation of the information. Nonmonotonic reasoning can also be handled with preferences rather than negation as failure (as also shown in [Dimopoulos and Kakas, 1995]). Moreover, it can be sometimes interesting to model exceptions by means of strong negation. For instance, in our methodology for modeling qualitative decision making in ASP (see Chapter 10), knowledge about the world is encoded by means of extended definite logic programs, *i.e.,* negation as failure free. When such knowledge contains default rules, exceptions must be properly handled.

In this chapter, we propose a novel approach which offers an operational counterpart to negation as failure. Our approach amounts to an explicit rewriting of exceptions by means of strong negation $\neg$ rather than negation as failure, together with the addition of completion rules for coping with incomplete information. Therefore, it may be viewed as a dual

attitude w.r.t. classical methods that rely on negation as failure. In fact, in ASP, rules with negation as failure remain intrinsically default rules, while in our approach default rules become strict rules after a proper rewriting, as suggested in the following example.

Example 4.1. Let us consider the following set of default rules Δ representing the typical birds and penguins knowledge representation problem: (i) *birds normally fly*, (ii) *birds normally have legs*, (iii) *penguins normally are birds*, and (iv) *penguins normally do not fly*. Therefore, from the extended definite logic program:[1]

$$\Delta = \begin{Bmatrix} r_1 : f \leftarrow b. \\ r_2 : l \leftarrow b. \\ r_3 : b \leftarrow p. \\ r_4 : \neg f \leftarrow p. \end{Bmatrix}$$

we want to obtain a set of strict rules S and a set of completion rules CR

$$S = \begin{Bmatrix} r_1 : f \leftarrow b, \neg p. \\ r_2 : l \leftarrow b. \\ r_3 : b \leftarrow p. \\ r_4 : \neg f \leftarrow p. \end{Bmatrix} \qquad CR = \left\{ cr_1 : \neg p \leftarrow b. \right\}$$

The proposed method, described here, turns to be simple to process and to have several noticeable features: it defines a *rewriting of default rules* into strict ones and it is able to *restore the consistency* of inconsistent programs that implicitly involve specificity ordering between the rules. It allows to face the blocking inheritance problem and to infer floating information (as in the Nixon diamond example). It takes its inspiration from a proposal made in the setting of possibilistic logic [Dupin de Saint-Cyr and Prade, 2008].

The chapter is organized as follows. After clarifying some extra notation used in this chapter (Section 4.1), in Section 4.2, we describe the algorithm which allows to handle default knowledge by making exceptions explicit in rules while adding completion rules for coping with incomplete information. Section 4.3 relates our approach to the classical handling of default rules in ASP and to the method proposed in [Garcia et al., 2009]. Finally, in Section 4.4, we point out some concluding remarks about the proposed approach and the representational capabilities in terms of preference representation.

4.1 Preliminaries

In this chapter, we will assume that a program P consists of different sets of definite rules. In particular, we define:

Definition 4.1. *A definite logic program $P_{\langle \Delta, S, FC, CR \rangle}$ is a tuple $\langle \Delta, S, FC, CR \rangle$, in which*

- *Δ is a finite set of definite rules that we call default rules,*
- *S is a finite set of definite rules that we call strict rules,*

[1] Extended definite logic programs are logic programs which do not contain negated-by-failure atoms (see Chapter 2, Section 2.1).

- *FC is a finite set of facts that we call factual context,*
- *CR is a finite set of definite rules that we call completion rules.*

Intuitively, Δ is the set of rules which can admit exceptions, S is the set of rewritten rules from Δ in which exceptions have been made explicit, FC is the set of contextual knowledge, and CR is the set of additional knowledge needed to cope with incomplete information.

By convention, we omit the subindex in the writing of $P_{\langle \Delta,S,FC,CR\rangle}$ whenever the corresponding set of rules is empty, *e.g.*, $P_{\langle S,FC,CR\rangle}$ when $\Delta = \emptyset$, $P_{\langle S,FC\rangle}$ when $\Delta = \emptyset$ and $CR = \emptyset$. For representation issues throughout the chapter we denote rules belonging to Δ by $r_\Delta = l_\Delta \leftarrow \mathcal{B}_\Delta$ and rules belonging to S by $r_s = l_s \leftarrow \mathcal{B}_s$.

Example 4.2. Let us consider the set of default rules Δ in Example 4.1. It can be checked that the program $P_{\langle \Delta,FC\rangle}$, in which $FC = \{p\}$ is the factual context about a penguin p, is inconsistent, since $Cn(P) = \{p, b, l, f, \neg f\}$ is inconsistent.

Since we rewrite default rules into other rules in which atoms need to be negated (*e.g.*, a into $\neg a$) or to be made positive (*i.e.*, $\neg a$ into a), we define the following function:

Definition 4.2. *Let φ be a mapping function such that*

$$\varphi(x) = \begin{cases} \neg x : \text{ if } x \in \mathcal{L}_P \\ x : \text{ if } x \in \mathcal{L}'_P \setminus \mathcal{L}_P \end{cases}$$

in which $\mathcal{L}'_P$ is the language of the program P after the replacement of the extended atoms.

4.2 Handling Default Reasoning

In [Dupin de Saint-Cyr and Prade, 2008], Dupin de Saint-Cyr and Prade have made a proposal for handling uncertain default rules in the possibilistic logic setting. Indeed, possibility theory provides a framework both for modeling qualitative uncertainty and for modeling default rules of the form *if p then generally q* by means of constraints stating that having $p \wedge q$ true is strictly more possible than having $p \wedge \neg q$ true (see Chapter 3, Section 3.2.1). The exploitation of such constraints induces a priority ordering between defaults according to their specificity [Benferhat et al., 1998]. This ordering has been proved to be the same as the one given by System Z [Pearl, 1990]. Since uncertain default rules are thus associated with both an uncertainty level and a priority level, there was a need for rewriting defaults as ordinary possibilistic logic formulas associated only with an uncertainty level. It is this kind of rewriting idea that we are now introducing in the ASP setting.

The algorithm we will propose allows to mirror nonmonotonic reasoning in logic programming without negation as failure by making exceptions explicit in rules while adding completion rules for coping with incomplete information. The method consists of four general steps (Algorithm 1) we briefly summarize before discussing them in details in the next subsections.

First, program rules which have not the same specificity have to be identified; this is done by using the notion of tolerance of a rule of System Z [Pearl, 1990] adapted to ASP

Algorithm 1 General Algorithm

Input: $\begin{cases} \Delta : \text{a set of default rules} \\ FC : \text{factual context} \end{cases}$

Output: $\{ M : \text{the minimal model of } P_{\langle S,FC,CR \rangle}$

1: $S = \emptyset$; // the set of strict rules
2: $CR = \emptyset$; // the set of completion rules
3: $\langle \Delta_0, \ldots, \Delta_n \rangle \leftarrow Z - ordering(\Delta)$
4: $\langle S, CR \rangle \leftarrow writeExceptions(\langle \Delta_0, \ldots, \Delta_n \rangle)$
5: $P_{\langle S,FC,CR \rangle} \leftarrow testConsistency(S, FC, CR)$
6: $M \leftarrow Cn(P_{\langle S,FC,CR \rangle})$
7: **return** $\langle M \rangle$

programs (line 3). Secondly, default rules which are exceptional w.r.t. other more specific rules are rewritten in order to make their condition parts more complete. At the same time, completion rules aiming at completing the encountered exceptional situations are added; this is done by the rewriting algorithm (line 4). Then, when factual context is added to the rewritten program, a consistency check is made between the context, the completion rules, and the relevant rewritten rules (line 5), *i.e.,* rules in which exceptions are made explicit by the rewriting algorithm. Finally, factual context and consistent completion rules are added to the rewritten program and the minimal model is computed (line 6).

4.2.1 Ordering Rules by Specificity

Pearl [Pearl, 1990] provides an algorithm which gives a stratification of a set of default rules in a way that reflects the *specificity* of the rules. Roughly speaking, the first stratum contains the most specific rules, *i.e.,* rules which do not admit exceptions (at least, expressed in the considered default base); the second stratum has exceptions only in the first stratum, and so on. Therefore, in System Z, a set of rules Δ is stratified into subsets $\Delta_0, \ldots, \Delta_n$ in which the resulting partitioning is called Z-ordering. Although there can be several orderings compatible with a set of default rules, the *minimal specificity ordering* is unique [Pearl, 1990]. In the following, we refer to this unique ordering.

As we are dealing with set of atoms, we have to adapt the notion of tolerance of a rule to ASP. For this reason, we have reused the definition of *rule tolerance* introduced in [Garcia et al., 2009].[2]

Definition 4.3. *For a rule $r_\Delta = l_\Delta \leftarrow \mathcal{B}_\Delta$, a set of atoms A verifies r_Δ if $\mathcal{B}_\Delta \subseteq A \wedge l_\Delta \in A$.*

Definition 4.4. *A rule r_Δ is said to be* tolerated *by a set of default rules Δ if and only if there is a set of atoms A, closed under Δ and consistent, which verifies r_Δ.*

The tolerance of the rule characterizes the fact that its application does not generate any contradiction. From this notion of tolerance, it is possible to obtain a stratification of the program which allows to stratify Δ into $(\Delta_0, \Delta_1, \ldots, \Delta_n)$ such that: Δ_0 contains the set of rules of Δ tolerated by Δ; Δ_1 contains the set of rules of $\Delta \backslash \Delta_0$ tolerated by $\Delta \backslash \Delta_0$; Δ_2 contains the set of rules of $\Delta \setminus (\Delta_0 \cup \Delta_1)$ tolerated by $\Delta \backslash (\Delta_0 \cup \Delta_1)$, *etc.*

[2] The notions of consistent and closed are defined in Chapter 2, Section 2.1.

Example 4.3. Let us consider the set of rules in Example 4.1 in which symbols b, f, l, p denote *birds*, *fly*, *legs*, and *penguins* respectively:

$$\Delta = \begin{Bmatrix} r_1 : \ f \ \leftarrow b. \\ r_2 : \ l \ \leftarrow b. \\ r_3 : \ b \ \leftarrow p. \\ r_4 : \neg f \leftarrow p. \end{Bmatrix}$$

$A = \{b, f, l\}$ verifies r_1, it is closed under Δ, and it is consistent. Thus, r_1 is tolerated in Δ and $r_1 \in \Delta_0$. Similarly, r_2 belongs to Δ_0. The only set which verifies r_3 and is closed w.r.t. r_1, r_2, and r_4 is $A' = \{b, p, l, f, \neg f\}$, but it is inconsistent. Thus, $r_3 \in \Delta_1$. In the same way, we obtain $r_4 \in \Delta_1$. Finally, the Z-ordering associated to Δ is:

$$\Delta_0 = \begin{Bmatrix} r_1 : f \leftarrow b. \\ r_2 : l \leftarrow b. \end{Bmatrix} \qquad \Delta_1 = \begin{Bmatrix} r_3 : \ b \ \leftarrow p. \\ r_4 : \neg f \leftarrow p. \end{Bmatrix}$$

4.2.2 Rules Rewriting

The general idea of the rewriting is to generate automatically from Δ a set of rules in which the condition parts explicitly state that we are not in an exceptional context to which other default rules refer. In practice, this amounts to transform the set of default rules Δ into a set of strict rules S with explicit exceptions and a set of completion rules CR.

In order to make exceptions explicit, the proposed rewriting considers rules specificity. At the beginning, rules in the last stratum of the Z-ordering are accepted as strict rules, since they are the most specific ones. Then, lower strata are processed in order to identify the set of exceptions of less specific default rules (if any) w.r.t. the strict rules. Default rules are rewritten into strict ones in which exceptions are made explicit. In case of exceptions, completion rules are added in order to complete contextual and incomplete information. The procedure keeps on fixing the next strata based on the computed strict rules until all strata have been considered. In order to define the algorithm in a more precise way, we need several definitions.

Definition 4.5 (Exceptional Set). *Let $S = \{r_{s_i} = l_{s_i} \leftarrow \mathcal{B}_{s_i} \mid 1 \leq i \leq n\}$ be a set of strict rules. For any given default rule $r_\Delta = l_\Delta \leftarrow \mathcal{B}_\Delta$, we define the exceptional set in r_{s_i} to the rule r_Δ as*[3]

$$E_i(r_\Delta, r_{s_i}) = \mathcal{B}_{s_i} \ s.t. \ l_{s_i} \leftarrow \mathcal{B}_{s_i} \in S \wedge \mathbf{cons}(\mathcal{B}_\Delta \cup \mathcal{B}_{s_i}) \ \wedge \ \mathbf{incons}(\{l_\Delta\} \cup \{l_{s_i}\})$$

Then, the set of exceptional sets in S to a rule r_Δ is defined as

$$\mathcal{E}(r_\Delta, S) = \{E_i(r_\Delta, r_{s_i}) \mid r_{s_i} \in S\}$$

The above definition collects all the exceptional sets in all strict rules to a given default rule r_Δ. This distinction is used for handling the case when the body of a strict rule is not a singleton.

[3] By $\mathbf{cons}(A)$ (resp., $\mathbf{incons}(A)$), we denote two Boolean functions which return true (resp., false) if the set of atoms A is consistent (resp., inconsistent).

Example 4.4. Let us consider the set of default rules Δ':

$$\Delta' = \begin{Bmatrix} r_1 : f \leftarrow b. \\ r_2 : b \leftarrow p. \\ r_3 : \neg f \leftarrow a, c. \\ r_4 : \neg f \leftarrow p. \end{Bmatrix}$$

Then, the Z-ordering associated to Δ' is

$$\Delta'_0 = \begin{Bmatrix} r_1 : f \leftarrow b. \\ r_2 : b \leftarrow a. \end{Bmatrix} \qquad \Delta'_1 = \begin{Bmatrix} r_3 : b \leftarrow a, c. \\ r_4 : \neg f \leftarrow p. \end{Bmatrix}$$

Please observe that the set Δ'_1 contains rules which do not have any exceptions, *i.e.,* it is a set of strict rules S. Then $\mathcal{E}(r_1, S) = \{\{a, c\}, \{p\}\}$.

Once all the exceptions of a default rule w.r.t. a set of strict rules have been generated, the default rule can be rewritten in the following way.

Definition 4.6 (Default Rule Rewriting). *Let $r_\Delta = l_\Delta \leftarrow \mathcal{B}_\Delta$ be a default rule, $S = \{r_{s_i} = l_{s_i} \leftarrow \mathcal{B}_{s_i} \mid 1 \leq i \leq n\}$ be a set of strict rules, and $\mathcal{E}(r_\Delta, S)$ be the set of exceptional sets in S to the rule r_Δ. Then, for each $E_i(r_\Delta, r_{s_i})$, r_Δ is rewritten into a new strict rule r_s by setting:*

$$r_s = l_\Delta \leftarrow \mathcal{B}_\Delta \cup \{\varphi(x) | x \in E_i(r_\Delta, r_{s_i})\}$$

The idea is to generate automatically from Δ a set of rules in which the condition parts explicitly state that we are not in an exceptional context to which other default rules refer. The rewritten rule is added to the set S.

Please observe that when the exception set is not a singleton, a default rule is rewritten into more than one strict rule.

Example 4.5. For instance, following with Example 4.4, the set of default rules Δ' is rewritten into the following set of strict rules S':

$$S' = \begin{Bmatrix} r_1 : f \leftarrow b, \neg a. \\ r_2 : f \leftarrow b, \neg c. \\ r_3 : f \leftarrow b, \neg p. \\ r_4 : b \leftarrow p. \\ r_5 : \neg f \leftarrow a, c. \\ r_6 : \neg f \leftarrow p. \end{Bmatrix}$$

If we has defined only one exception set for all the exceptions of a default rule w.r.t. a strict one, we would have obtained a different (less intuitive) rewriting (*e.g.,* rule r_1 would have been rewritten into one rule only $f \leftarrow b, \neg a, \neg c, \neg p$) which would have prevented the derivation of some intuitive conclusions (*e.g.,* in context $\{a, \neg c\}$ it would have not been possible to conclude f).

Algorithm 2 writeExceptions($\langle \Delta_0, \ldots, \Delta_n \rangle$): $\langle S, CR \rangle$

Input: $\{ \langle \Delta_0, \ldots, \Delta_n \rangle$: the stratification given by the Z-ordering

Output: $\begin{cases} S : \text{the set of strict rules obtained by rewriting default rules} \\ CR : \text{the set of completion rules} \end{cases}$

$k \leftarrow n-1; CR \leftarrow \emptyset; S = \{r_{s_i} = l_{s_i} \leftarrow \mathcal{B}_{s_i} \mid r_{\Delta_{ni}} \in \Delta_n\}$ //initialization
while $k \geqslant 0$ **do**
  $S_k \leftarrow \emptyset$
  for all $r_{\Delta_k} = l_{\Delta_k} \leftarrow \mathcal{B}_{\Delta_k}$ s.t. $r_{\Delta_k} \in \Delta_k$ **do**
    $\mathcal{E}(r_\Delta, S) \leftarrow \emptyset$ //Set of exceptional sets of a default rule r_Δ w.r.t. S
    for all $r_{s_i} \in S$ s.t. $\mathbf{cons}(\mathcal{B}_{\Delta_k} \cup \mathcal{B}_s) \wedge \mathbf{incons}(\{l_{\Delta_k}\} \cup \{l_s\})$ **do**
      $CR \leftarrow CR \cup \{\varphi(x) \leftarrow \mathcal{B}_{\Delta_k} | x \in \mathcal{B}_{s_i}\}$
      $E_i(r_{\Delta_k}, r_{s_i}) \leftarrow \mathcal{B}_{s_i}$ //Exceptional set of a default rule r_{Δ_k} w.r.t. r_{s_i}
      $\mathcal{E}(r_\Delta, S) \leftarrow \mathcal{E}(r_\Delta, S) \cup E_i(r_{\Delta_k}, r_{s_i})$
    end for
    if $\mathcal{E}(r_\Delta, S) \neq \emptyset$ **then**
      for all $E_i(r_{\Delta_k}, r_{s_i}) \in \mathcal{E}(r_\Delta, S)$ **do**
        $S_k \leftarrow S_k \cup \{l_{\Delta_k} \leftarrow \mathcal{B}_{\Delta_k} \cup \{\varphi(x) \mid x \in E_i(r_\Delta, r_{s_i})\}\}$
      end for
    else
      $S_k \leftarrow S_k \cup \{l_{\Delta_k} \leftarrow \mathcal{B}_{\Delta_k}\}$
    end if
  end for
  $S \leftarrow S \cup S_k$
  $k \leftarrow k-1$
end while
return $\langle S, CR \rangle$

Special (strict) rules called *completion rules* stating that we are not in an exceptional situation are added to a new set denoted by CR. The use of these completion rules is motivated by the need of reasoning in presence of incomplete information. In fact, completion rules allow to apply strict rules which now have a more precise condition part.

Definition 4.7 (Completion Rule). *Let $r_\Delta = l_\Delta \leftarrow \mathcal{B}_\Delta$ be a default rule, and $r_s = l_s \leftarrow \mathcal{B}_s$ be a strict rule such that $E(r_\Delta, r_s) = \mathcal{B}_s$. Then, a completion rule related with r_Δ is defined as $\varphi(x) \leftarrow \mathcal{B}_\Delta$ in which $x \in E(r_\Delta, r_s)$.*

Based on the above definitions we can proceed with the description of Algorithm 2. Note that the rules of the last stratum Δ_n do not admit exceptions w.r.t. Δ since they are the most specific ones and they are directly copied into S. Thus, the algorithm begins with the rules of the stratum $n-1$. The stratum $n-1$ contains rules that admit exceptions only because of rules in the last stratum. More generally, a stratum k contains rules that admit exceptions only because of rules in strata with rank greater or equal to $k+1$. More precisely, for each rule in a given stratum, all its exceptions (coming from strata with a greater rank) are computed in order to rewrite this rule by explicitly stating that the exceptional situations are excluded in its condition part. Moreover, completion rules are added for each exceptional case found. A completion rule is used only if it is consistent with the current context and the relevant rewritten rules.

Algorithm 3 testConsistency(S,FC,CR): $P_{\langle S,FC,CR\rangle}$

Input: $\begin{cases} S : \text{the set of strict rules} \\ FC : \text{factual context} \\ CR : \text{the set of completion rules} \end{cases}$

Output: $\left\{ P_{\langle S,FC,CR\rangle} : \text{the definite program} \right.$

$App_{CR} = Apl(CR, FC, S)$
$Rel_{CR} = Rel(App_{CR}, S)$
if incons$(Cn(FC \cup App_{CR} \cup Rel_{CR}))$ **then**
  $App_{CR} \leftarrow \emptyset$
end if
$P_{\langle S,FC,CR\rangle} \leftarrow S \cup FC \cup App_{CR}$
return $P_{\langle S,FC,CR\rangle}$

Example 4.6. Let us apply the algorithm to rewrite the rules in Example 4.3 by describing explicitly their exceptions starting from the last stratum of the Z-ordering. It can be checked that the algorithm rewrites the set of default rules Δ into:

$$S = \begin{Bmatrix} r_1 : & f & \leftarrow & b, \neg p. \\ r_2 : & l & \leftarrow & b. \\ r_3 : & b & \leftarrow & p. \\ r_4 : & \neg f & \leftarrow & p. \end{Bmatrix} \qquad CR = \left\{ cr_1 : \neg p \leftarrow b. \right\}$$

As we will see later, the completion rule will only be used if it is consistent with the current context and the set of rewritten rules.

4.2.3 Consistency Test

As said before, completion rules are useful to state in which cases the current context is not exceptional and they are used to apply the rewritten rules which now have a more precise condition part. However, completion rules can only be used if they are consistent with the context described by FC and the set of rewritten rules S. Hence, a consistency test is required (Algorithm 3).

To retrieve all the applicable completion rules w.r.t. the factual context and the set of strict rules, we provide the following definitions.

Definition 4.8. *Let S be a set of strict rules obtained by Δ. Let CR be the set of completion rules and FC the factual context to be added to S. Then, we define:*

- $Apl(CR, FC, S) = \{cr \in CR \mid body(cr) \subseteq Cn(S \cup FC)\}$
- $Rel(Apl(CR, FC, S), S) = \{r_s \in S \mid r_s \text{ is associated with } Apl(CR, FC, S)\}$[4]

The consistency test amounts to check whether the set of atoms produced by the set of rules obtained by merging the factual context FC, the set of the completion rules which are applicable w.r.t. the current context (App_{CR}), and the set of strict rules associated to

[4] Informally, *associated* is a mapping between a rewritten rule r_s and its completion rules produced in the rewriting.

the completion rules (Rel_{CR}), is consistent. Depending on this consistency check, a new program $P_{\langle S,FC,CR\rangle}$ is returned which contains the applicable completion rules (if any). It is worthy to point out that, once the default rules have been rewritten, this consistency test is performed only when the context FC is changed.

Example 4.7. Let us consider the set of strict rules S, the set of completion rules CR in Example 4.6, and the contexts $FC_1 = \{b\}$, $FC_2 = \{p\}$, and $FC_3 = \{b, \neg f\}$. The completion rule $\{cr_1 = \neg p \leftarrow b\}$ is consistent w.r.t. the context $FC_1 = \{b\}$ and the strict rule r_1. Thus, the program $P_{\langle S,FC_1,CR\rangle}$ is built. In $FC_2 = \{p\}$, CR is not consistent w.r.t. the factual knowledge. Therefore, it cannot be taken into account. Finally, $FC_3 = \{b, \neg f\}$ illustrates the case in which CR is consistent w.r.t. the context, but it is not consistent with the rewritten rule r_1. Also in this case, the completion rule cannot be considered.

4.2.4 Minimal Model Computation

Since the program obtained by the consistency test is a definite logic program, we can apply the fix-point operator in order to compute the minimal model $Cn(P)$ of a definite logic program P.[5]

Example 4.8. Let us consider the set of strict rules S in Example 4.6 and the consistency tests in Example 4.7 w.r.t. the factual contexts $FC_1 = \{b\}$, $FC_2 = \{p\}$, and $FC_3 = \{b, \neg f\}$. Different programs are obtained after the consistency tests. Let $P_{\langle S,FC_1,CR\rangle} = P_1$, $P_{\langle S,FC_2\rangle} = P_2$, $P_{\langle S,FC_3\rangle} = P_3$, then:

$$P_1 = \left\{ \begin{array}{lll} r_1: & f & \leftarrow b, \neg p. \\ r_2: & l & \leftarrow b. \\ r_3: & b & \leftarrow p. \\ r_4: & \neg f & \leftarrow p. \\ cr_1: & \neg p & \leftarrow b. \\ fc_1: & b & \leftarrow \top. \end{array} \right\} \quad P_2 = \left\{ \begin{array}{lll} r_1: & f & \leftarrow b, \neg p. \\ r_2: & l & \leftarrow b. \\ r_3: & b & \leftarrow p. \\ r_4: & \neg f & \leftarrow p. \\ fc_2: & p & \leftarrow \top. \end{array} \right\} \quad P_3 = \left\{ \begin{array}{lll} r_1: & f & \leftarrow b, \neg p. \\ r_2: & l & \leftarrow b. \\ r_3: & b & \leftarrow p. \\ r_4: & \neg f & \leftarrow p. \\ fc_{3_1}: & b & \leftarrow \top. \\ fc_{3_2}: & \neg f & \leftarrow \top. \end{array} \right\}$$

Different minimal models are retrieved for each of the above programs.

$$\begin{array}{ll} T_{P_1}(\emptyset) & = \{b\} \\ T_{P_1}(\{b\}) & = \{b, \neg p\} \\ T_{P_1}(\{b, \neg p\}) & = \{b, \neg p, f, l\} \\ \underline{T_{P_1}(\{b, \neg p, f, l\})} & \underline{= \{b, \neg p, f, l\}} \\ Cn(P_1) & = \{b, \neg p, f, l\} = M_1 \end{array}$$

$$\begin{array}{ll} T_{P_2}(\emptyset) & = \{p\} \\ T_{P_2}(\{p\}) & = \{p, b, \neg f\} \\ T_{P_2}(\{p, b, \neg f\}) & = \{p, b, \neg f, l\} \\ \underline{T_{P_2}(\{p, b, \neg f, l\})} & \underline{= \{p, b, \neg f, l\}} \\ Cn(P_2) & = \{p, b, \neg f, l\} = M_2 \end{array}$$

[5] $Cn(P)$ is defined as $Cn(P) = \bigcup_{k \geq 0} T_P^k(\emptyset)$ (see Chapter 2, Section 2.1.3).

$$\begin{array}{ll} T_{P_3}(\emptyset) & = \{b, \neg f\} \\ T_{P_3}(\{b, \neg f\}) & = \{b, \neg f, l\} \\ T_{P_3}(\{b, \neg f, l\}) & = \{b, \neg f, l\} \\ \hline Cn(P_3) & = \{b, \neg f, l\} = M_3 \end{array}$$

$M_1 = \{b, \neg p, f, l\}$ is the minimal model of $P_{\langle S, FC_1, CR\rangle}$. $M_2 = \{b, p, l, \neg f\}$ is the minimal model of $P_{\langle S, FC_2\rangle}$. Finally, $M_3 = \{b, \neg f, l\}$ is the minimal model of $P_{\langle S, FC_3\rangle}$.

All these results agree with the intuition behind the bird and penguin knowledge representation problem. In particular, the proposed method refines the Z-ordering, since it can deal with the blocking inheritance problem. Our method also handles floating information, as shown in the following example.

Example 4.9. Let Δ'' be a set of default rules representing the Nixon Diamond example: *Quakers normally are pacifists*, *Quakers normally are Americans*, *Americans normally like baseball*, *Quakers generally do not like base-ball* and *Republicans are generally not pacifists*. Then:

$$\Delta'' = \left\{ \begin{array}{l} r_1 : \ p \ \leftarrow q. \\ r_2 : \ a \ \leftarrow q. \\ r_3 : \ b \ \leftarrow a. \\ r_4 : \neg b \leftarrow q. \\ r_5 : \neg p \leftarrow r. \end{array} \right\}$$

The rewriting produces:

$$S'' = \left\{ \begin{array}{l} r_1 : \ p \ \leftarrow \ q. \\ r_2 : \ a \ \leftarrow \ q. \\ r_3 : \ b \ \leftarrow a, \neg q. \\ r_4 : \neg b \leftarrow \ q. \\ r_5 : \neg p \leftarrow r, \neg q. \end{array} \right\} \qquad CR = \left\{ \begin{array}{l} cr_1 : \neg q \leftarrow a. \\ cr_2 : \neg q \leftarrow r. \end{array} \right\}$$

In context $FC = \{q, r\}$, CR is inconsistent w.r.t. FC. Therefore, $M = \{p, a, \neg b\}$ is the minimal model of $P_{\langle S, FC\rangle}$. An intuitive interpretation of the fact that *pacifist* is obtained is that the *Quaker* is more specific than *Republican* in Δ, since *Republican* is compatible with all the rules which is not the case for *Quaker*.

We can observe that there are some interesting properties of our algorithm.

Proposition 4.1. *Algorithm 1 terminates.*

Moreover, our approach offers a straightforward methodology to restore the consistency of inconsistent definite programs which implicitly involve some specificity order between the rules.

Proposition 4.2. *The program $P_{\langle S, FC, CR\rangle}$ returned by Algorithm 1 is consistent.*

It is natural now to wonder how our method behaves w.r.t. the classical handling of nonmonotonic reasoning in ASP.

4.3 Related Work

In the literature, several proposals have been made for handling exceptions and specificity in logic programming [Kowalski and Sadri, 1991; Dimopoulos and Kakas, 1995; Garcia et al., 2000; García and Simari, 2004]. More recently, Garcia *et al.* [Garcia et al., 2009] has proposed a rewriting procedure for making exceptions explicit.

Although the automatic generation of exceptions in [Garcia et al., 2009] may look similar to ours at first glance, it is intrinsically different in spirit since exceptions are captured by means of negation as failure. Therefore, it is interesting to compare our approach to nonmonotonicity, based on rules rewriting and completion rules, to the standard nonmonotonic reasoning characterized in terms of *not*. For doing this, let us consider the following example taken from [Garcia et al., 2009].

Example 4.10. From the set of default rules Δ':

$$\Delta' = \begin{Bmatrix} r_1: & f & \leftarrow b. \\ r_2: & l & \leftarrow b. \\ r_3: & b & \leftarrow p. \\ r_4: & \neg f & \leftarrow p. \end{Bmatrix}$$

using the algorithm in [Garcia et al., 2009], the following extended normal logic program is obtained:

$$P'_s = \begin{Bmatrix} r_1: & f & \leftarrow & b, not\ \neg f, not\ p. \\ r_2: & l & \leftarrow & b. \\ r_3: & b & \leftarrow & p. \\ r_4: & \neg f & \leftarrow & p, not\ f. \end{Bmatrix}$$

Let us consider the contexts $FC_1 = \{b\}$, $FC_2 = \{p\}$, and $FC_3 = \{p, f\}$. For illustrative purposes, by answer set semantics definition and by considering the set of atoms $M_1^{'} = \{l, b, f\}$, we can see that

$M_1^{'}$	$(P'_s \cup \{b\})^{M_1^{'}}$		$Cn((P'_s \cup \{b\})^{M_1^{'}})$
$\{l, b, f\}$	f	$\leftarrow b$	$\{l, b, f\}$
	l	$\leftarrow b$	
	b	$\leftarrow p$	
	b	$\leftarrow \top$	

$M_1^{'}$ is answer set of $P'_s \cup \{b\}$. In a similar way, it can be checked that $M'_2 = \{p, b, l, \neg f\}$ and $M'_3 = \{p, f, b, l\}$ are answer sets of $(P'_s \cup \{p\})$ and $(P'_s \cup \{p, f\})$ respectively.

Comparing our generated program (Example 4.8) with $(P'_s \cup \{b\})$ above, it can be observed that a model in our approach generally contains more atoms than the corresponding model of $P'_s \cup \{b\}$. Indeed, $Cn(P_{\langle S, FC_1, CR\rangle}) = \{l, b, f, \neg p\}$ is also telling that a bird that flies is not a penguin. These extra atoms can be justified by the corresponding dual attitude w.r.t. negation as failure played by the completion rules. In fact, the intuitive meaning of a rule with default negation like $f \leftarrow b, not\ p$ is that *if it can be proved that we are in the bird context b and nothing proves that we are in a penguin context p, then we can conclude f*. In order to use the

default rule about birds, we should not derive p. Alternatively, in our approach, the role of the completion rule ($\neg p \leftarrow b$) is to complete the factual knowledge and to trigger the applicability of the rule $f \leftarrow b, \neg p$ whose condition part now is more specific. In this way, the completion rule is playing the dual part of strong exception rules used in the classical handling of exceptional situations in ASP. On a closer inspection, when the completion rule is inconsistent w.r.t. the rewritten rule and it is not added to the program (see Example 4.7), it also serves as weak exception rule and it prevents the rewritten rule to be used.

Since our approach only rewrites exceptions which are implicit inherited by the specificity order among the set of default rules, any (weak) exception related to the rules in the last stratum of the Z-ordering is ignored (*e.g.,* the exception captured in $\neg f \leftarrow p, not\ f$ in P'_s). In this case, the two approaches can lead to different solutions (*e.g.,* in context $\{p, f\}$). To recover the same results we add a pair of *additional rules* to define an *abstract specificity level* in order to introduce an exceptional situation w.r.t. the class of the last stratum.

Definition 4.9 (Abstract Specificity Level). *Let $\{c_1, \ldots, c_k\}$ be a set of atoms representing classes with a hierarchy relation ($c_1 \supset c_2 \supset \ldots \supset c_k$), encoded by rules $c_i \leftarrow c_{i+1}$ (with $1 \leq i \leq k-1$). Let $\{p_1, \ldots, p_{k'}\}$ be a set of atoms representing class properties, encoded by rules $p_j \leftarrow c_i$ (with $1 \leq j \leq k'$ and $1 \leq i \leq k$). Then, an* abstract specificity level *c_{k+1} with property $p_{k'+1}$ is captured by two additional rules $c_k \leftarrow c_{k+1}$ and $p_{k'+1} \leftarrow c_{k+1}$ s.t. $p_{k'+1} = \varphi(p_{k'})$.*

Thanks to the additional rules, our approach agrees with the results obtained in [Garcia et al., 2009]. Moreover, it leads to larger models due to effect of completion rules which can contribute to add more factual knowledge. We illustrate this idea by means of the following example.

Example 4.11. Let us consider Δ in Example 4.1 with an abstract specificity level introduced by by adding the rules *super-penguins are penguins* (r_5) and *super-penguins normally fly* (r_6). Then $\Delta \cup \{r_5, r_6\}$ is:

$$\Delta' = \begin{Bmatrix} r_1: & f & \leftarrow & b. \\ r_2: & l & \leftarrow & b. \\ r_3: & b & \leftarrow & p. \\ r_4: & \neg f & \leftarrow & p. \\ r_5: & p & \leftarrow & sp. \\ r_6: & f & \leftarrow & sp. \end{Bmatrix}$$

The Z-ordering associated to Δ' induced by the specificity of the rules is:

$$\Delta'_0 = \begin{Bmatrix} r_1 : f \leftarrow b. \\ r_2 : l \leftarrow b. \end{Bmatrix} \qquad \Delta'_1 = \begin{Bmatrix} r_3 : b \leftarrow p. \\ r_4 : \neg f \leftarrow p. \end{Bmatrix} \qquad \Delta'_2 = \begin{Bmatrix} r_3 : b \leftarrow sp. \\ r_4 : f \leftarrow sp. \end{Bmatrix}$$

Intuitively, r_6 belongs to a higher stratum w.r.t. r_4, since it encodes more specific information. Applying Algorithm 1, Δ' is rewritten into:

$$S' = \begin{Bmatrix} r_1 : & f & \leftarrow & b, \neg p. \\ r_2 : & f & \leftarrow & b, \neg sp. \\ r_3 : & l & \leftarrow & b. \\ r_4 : & b & \leftarrow & p. \\ r_5 : & \neg f & \leftarrow & p, \neg sp. \\ r_6 : & p & \leftarrow & sp. \\ r_7 : & f & \leftarrow & sp. \end{Bmatrix} \qquad CR = \begin{Bmatrix} cr_1 : \neg p \leftarrow b. \\ cr_2 : \neg sp \leftarrow b. \\ cr_3 : \neg sp \leftarrow p. \end{Bmatrix}$$

In context $FC = \{p, f\}$, only completion rule cr_3 can be considered. Then, it can be checked that the set $\{p, f, b, sp, l\}$ is a minimal model of the program $P_{\langle S,FC,CR\setminus\{cr_1,cr_3\}\rangle}$.

Please observe that since the initial set of default rules contains an implicit inheritance relation, the generated normal program in [Garcia et al., 2009] is a stratified logic program. The class of stratified logic program has an important property: they have a unique answer set [Baral, 2003]. Based on this observation, we can establish the following relation between our approach and the approach in [Garcia et al., 2009].

Proposition 4.3. *Let Δ be a set of extended definite rules with an implicit inheritance relation, FC be a factual context, and AR be a pair of additional rules according to Definition 4.9. Let P' be the extended normal logic program obtained by the algorithm in [Garcia et al., 2009]. Let $\Delta' = \Delta \cup AR$, and $P_{\langle S,FC,CR\rangle}$ be the definite program obtained from Δ' by Algorithm 1. If M is answer set of $P' \cup FC$ and $M' = Cn(P_{\langle S,FC,CR\rangle})$ then $M \subseteq M'$.*

Therefore, our approach can offer an operational counterpart to negation as failure in knowledge representation problems which consist of default rules with an implicit specificity order between them.

4.4 Discussion and Concluding Remarks

In this chapter, we have proposed an algorithm for handling exceptions in default rules in extended definite logic programs. This method applies to a set of default rules that have an inheritance hierarchy relation. It amounts to automatically rewrite default rules into strict rules in which exceptions are made explicit and completion rules are added for coping with incomplete information.

The rewriting of default rules proposed is based on the Z-ordering, a well-known stratification of System Z [Pearl, 1990] (introduced also in System P [Kraus et al., 1990] to define the rational closure of a set of conditionals). In System Z, defaults are partitioned and ordered according to their specificity level (based on the concept of rules tolerance). By adapting the Z-ordering into the logic programming setting (following the work in [Garcia et al., 2009]), we have been able to provide a rewriting method for the generation of exceptions which amounts to generate exceptions in less specific rules which are in conflict with rules with higher specificity.

However, the main weakness of System Z is known to be its inability to deal with property inheritance from classes to exceptional sub-classes, also known as the *blocking inheritance problem*, in a proper way. This is a drawback that cannot be remedied by a method

based solely on the Z-ordering of defaults. Indeed, refinements of the Z-ordering have been proposed such as the maximum-entropy approach of Goldzsmidt in [Goldzsmidt et al., 1993] (which also takes into account the number of rules tolerating a formula and not only their rank orders). Other approaches equip nonmonotonic specifications with (explicit) preference ordering to default rules such as in [Brewka, 1994; Brewka and Eiter, 1999; Lifschitz, 1985; Rintanen, 1994]. In such approaches, the notion of preference ordering is merged with the notion of inheritance properties. Then, depending on how the notion of preference is interpreted and how preferences over rules are assigned, different conclusions can be drawn.

We have also sanctioned property inheritance, but in an indirect way. While the blocking inheritance problem is due to a ranking over the interpretations obtained by the Z-ordering, here we have used specificity only to identify exceptional rules and to rewrite them. Then, we have applied the standard inference of logic programming based on the fix-point computation. The notion of preference exists in our approach as well, but it is implicit, since it is related to specificity. As such, implicit preferences are interpreted in only one way and this can guarantee that, in a given context, the same conclusions are always drawn.

We have established that, under certain conditions, we can recover the results obtained by the method proposed in [Garcia et al., 2009] which is based on the rewriting of exceptions by means of negation as failure (Proposition 4.3). This result is significant, since it suggests how negation as failure can be captured by an operational approach in the class of stratified logic programs. As such, the proposed approach can provide a dual view to nonmonotonic reasoning in logic programs.

We have claimed that the method can be used to restore the consistency of logic programs encoding knowledge representation problems with inheritance hierarchies (Proposition 4.2). So far, we have identified two scenarios which can benefit from the the proposed method. In decision making problems (see Chapter 10), the knowledge base may consist of a set of default rules. Then, the rewriting algorithm can be used to rewrite default rules into strict ones before proceeding to the computation of an optimal decision. In knowledge scenarios, when pieces of knowledge (encoded by logic programs) coming from different sources are integrated, the resulting program can be inconsistent and the rewriting can be applied to solve inconsistencies. We illustrate this idea in the next example.

Example 4.12. Let us consider two pieces of knowledge (P_1,P_2) encoding some default knowledge about going to the *beach* in a day of *may*. On one hand, both knowledge bases contain some common default knowledge such as *spring is the end of winter* and *may is spring*. On the other hand, different information is stored about the weather condition and whether going to the beach is a valid alternative or not. In particular, in P_1, it is known that *in spring it normally rains* and *normally if it rains I do not go to the beach*, while in P_2, it is known that *in may it normally does not rain* and *normally in may I go to the beach*. Moreover, we observe that we are in a day of may. We can model this scenario by means of the following two extended definite logic programs:

$$P_1 = \begin{Bmatrix} r_2 : \neg beach \leftarrow rain. \\ r_4 : rain \leftarrow spring. \\ r_5 : spring \leftarrow may. \\ r_6 : \neg winter \leftarrow spring. \\ r_7 : may \leftarrow \top. \end{Bmatrix} \qquad P_2 = \begin{Bmatrix} r_1 : beach \leftarrow may. \\ r_3 : \neg rain \leftarrow may. \\ r_5 : spring \leftarrow may. \\ r_6 : \neg winter \leftarrow spring. \\ r_7 : may \leftarrow \top. \end{Bmatrix}$$

It is easy to see that, since P_1 and P_2 are already consistent, the rewriting of default rules does not produce any exceptions and different minimal models are retrieved as expected (*e.g.*, $\{may, spring, \neg winter, rain, \neg beach\}$ and $\{may, spring, \neg winter, \neg rain, beach\}$ respectively). However, when P_1 and P_2 are integrated some inconsistencies arise.

$$P_1 \cup P_2 = \begin{Bmatrix} r_1 : beach \leftarrow may. \\ r_2 : \neg beach \leftarrow rain. \\ r_3 : \neg rain \leftarrow may. \\ r_4 : rain \leftarrow spring. \\ r_5 : spring \leftarrow may. \\ r_6 : \neg winter \leftarrow spring. \\ r_7 : may \leftarrow \top. \end{Bmatrix}$$

The program above contains several default statements which must be properly handled before being able to use them. Indeed, default rules r_1,r_2, and r_3, r_4 are in conflict, since in context *may* they lead to conflicting conclusions (*e.g.*, $beach/\neg beach$ and $rain/\neg rain$). It is importance to observe that some rules are more specific than others. For instance, r_1 is more specific than r_2 and r_3 than r_4 (they belong to different specificity levels due to rule r_5). In this kind of representation, we can apply the rewriting algorithm proposed in this chapter.

By Z-ordering, we obtain the following ranking of defaults: $\Delta_0 = \{r_2, r_4, r_6\}$, $\Delta_1 = \{r_1, r_3, r_5\}$. By a proper rewriting, we obtain a new program, in which exceptions are made explicit, and a set of completion rules:

$$P' = \begin{Bmatrix} r_1 : beach \leftarrow may. \\ r_2 : \neg beach \leftarrow rain, \neg may. \\ r_3 : \neg rain \leftarrow may. \\ r_4 : rain \leftarrow spring, \neg may. \\ r_5 : spring \leftarrow may. \\ r_6 : \neg winter \leftarrow spring. \end{Bmatrix} \qquad CR = \begin{Bmatrix} cr_1 : \neg may \leftarrow rain. \\ cr_2 : \neg may \leftarrow spring. \end{Bmatrix}$$

From the preference representation point of view, the approach described in this chapter allows to handle some basic default preference (such as the ones above). However, the representation of simple default statements is of quite limited applicability for representing user knowledge in an intelligent system that supports user decisions. In particular, having only one option in each context does not give so much flexibility in the preference specification.

On the other hand, LPODs [Brewka et al., 2004b] allows to specify more than one options in each context and it supports preference statements built by means of the ordered disjunction operator $\times$ (see Chapter 2, Section 2.2.4). In LPODs, preferences statements can de considered default statements as well. For instance, default rules such as *generally if it does not rain I prefer going to the beach over going to the cinema* (encoded as $beach \times cinema \leftarrow not\ rain$), and *generally if it rains I prefer going to the cinema over going to the beach* (encoded as $cinema \times beach \leftarrow rain$) can be handled in LPODs by means of negation as failure. In this way, context-dependent preference expressions with incomplete information can be properly handled. For instance, the scenario in Example 4.12 can be enriched by the LPODs specification in the following way:

$$P'' = \left\{ \begin{array}{lcl} r_1 : beach \times cinema & \leftarrow & may, not\ rain. \\ r_2 : cinema \times beach & \leftarrow & rain, not\ may. \\ r_3 : \neg rain & \leftarrow & may. \\ r_4 : rain & \leftarrow & spring, not\ may. \\ r_5 : spring & \leftarrow & may. \\ r_6 : \neg winter & \leftarrow & spring. \\ r_7 : may & \leftarrow & \top. \end{array} \right\}$$

r_1 and r_2 are default rules which express defeasible preferences which depend on incomplete knowledge captured by negated-by-failure atoms. Although LPODs offers a richer syntax for expressing context-dependent preferences, it does not allow to associate weights to preference rules (as we will discuss in Chapter 5) and to capture more complex preferences expressions (as we will discuss in Chapter 6-7-8).

5

LPPODs: a Framework for Dealing with Explicit Preferences and Uncertainty

Handling incomplete information amounts to use implicit or explicit preferences for the selection of default rules. As far as explicit preferences are concerned, LPODs is a logic programming specification that allows to specify context-dependent explicit preference statements of the form *normally, in context c it is more reasonable to assume a rather than b* (encoded as $a \times b \leftarrow c$). From a nonmonotonic reasoning point of view, these preference statements can be used to specify a preference order among exceptions w.r.t. the default rules in a logic program, and, consequently, to select the preferred default rules to be used in the reasoning (see Chapter 3, Section 3.3.3).

However, preferences about exceptions to default rules that we want to privilege may be pervaded with uncertainty. When the preference statement above *normally, in context c it is more reasonable to assume a rather than b* is stated, no estimate of the certainty of having *a* preferred to *b* in context *c* is provided.

The selection of the default rules to be used in the reasoning can be a matter of uncertainty as well. Indeed, I might be interested in selecting the most certain pieces of information for building my reasoning. For instance, one may consider that a preference rule stating that *it is more reasonable to assume an exception* ab_1 *rather than an exception* ab_2 is less certain than another preference rule stating that *it is more reasonable to assume* ab_2 *rather than* ab_1. In such a case, a mechanism which allows one to select the most certain preference rules can be useful. There may also exist more complex cases in which preference rules are equally certain but there exist other uncertain rules (or facts) in the program on which such preference rules depend. Then, an effective mechanism for propagating uncertainty values between program rules is needed in order to select the most certain pieces of knowledge. To illustrate these ideas, let us consider the following example.

Example 5.1. Continuing with the classical bird-fly example, let us consider a more complicated version of Example 3.10. Let us suppose to have the following default rules (i) *antarctic birds are birds*, (ii) *birds normally fly* and (iii) *antarctic birds normally do not fly*. Furthermore, we have two preference rules expressing our preferences among exceptions to the previous rules. In particular, let us suppose that one preference rule states that (iv) *in case of a penguin, it is more reasonable to assume that birds cannot fly rather than antarctic birds can fly*, and another rule states that (v) *in case of a super-penguin, it is more reasonable to assume that antarctic birds can fly rather than birds cannot fly*. Finally, we have two further rules expressing that (vi) *antarctic birds are penguins* and (vii) *antarctic birds are super-penguins*. The

latter rules are uncertain, since we can imagine that in Antarctica there are many penguins that do not fly together with only few super-penguins that can fly. Let us suppose, for now in an informal way, to associate to each rule a label indicating its certainty. The resulting program looks like (we also observe an antarctic bird):

$$P = \left\{ \begin{array}{llcl} r_1 = \mathbf{1}: & b & \leftarrow & ant. \\ r_2 = \mathbf{1}: & f & \leftarrow & b, not\ ab_1. \\ r_3 = \mathbf{1}: & \neg f & \leftarrow & ant, not\ ab_2. \\ r_4 = \mathbf{1}: & ab_1 \times ab_2 & \leftarrow & p. \\ r_5 = \mathbf{1}: & ab_2 \times ab_1 & \leftarrow & sp. \\ r_6 = \mathbf{1}: & \bot & \leftarrow & ab_1, ab_2. \\ r_7 = \mathbf{0.6}: & p & \leftarrow & ant. \\ r_8 = \mathbf{0.4}: & sp & \leftarrow & ant. \\ r_9 = \mathbf{1}: & ant & \leftarrow & \top. \end{array} \right\}$$

It can be observed that the above program without any certainty label is a classical LPOD. In such a case, no preference order can be achieved between the two alternative answer sets $\{ant, p, sp, b, ab_1, \neg f\}$ and $\{ant, p, sp, b, ab_2, f\}$. In other words, the conflict between the defaults r_2 and r_3 cannot be solved. However, intuitively, since the rule r_7 is more certain then the rule r_8, the selection on which exception to privilege, in order to sanction the conflict between the rules r_2 and r_3, should also take into account the certainty about the information encoded in the program. This can be done in principle by considering that the preference rule r_4 is more important than r_5, since it depends on more certain information. Then, the following question arises: how is it possible to consider the certainty of other rules on which a preference rule may depend? A possible way is by means of a program transformation able to propagate certainty values between program rules.

The above example suggests that we need to extend LPODs with a proper management of uncertainty in order to select the most preferred and certain beliefs to be used in the reasoning. To this end, we extend LPODs by a possibilistic semantics [Nicolas et al., 2006] which associates necessity values to program rules for measuring to *what extent a statement is certain* (according to possibilistic logic [Dubois et al., 1994]). Moreover, we need to study a suitable mechanism for transforming program rules in order to propagate uncertainty values between the rules themselves. This can be achieved by reusing the theory of rewriting system [Dershowitz and Plaisted, 2001] and by defining a set of transformation rules which preserve the possibilistic semantics.

In this chapter, we present a logic programming framework which can represent explicit preferences and necessity values together. The framework properly generalizes LPODs to support the expression of explicit context-dependent preference statements to select the default rules to be used in the reasoning in the presence of uncertain information. We will assume that the necessity values associated to program rules are discovered by a knowledge discovery process. We will show how LPODs rule satisfaction degrees and necessity values can be considered to order possibilistic answer sets. To achieve this, we merge the formalism of *Logic Programs with Ordered Disjunction* (LPODs) [Brewka et al., 2004b] and

possibilistic normal logic programs [Nicolas et al., 2006] into the single framework of *Logic Programs with Possibilistic Ordered Disjunction* (LPPODs).[1]

The LPPODs framework embeds in a unified way several aspects of common-sense reasoning such as nonmonotonicity, preferences, and uncertainty. Each part is underpinned by a well established formalism. On one hand, from LPODs, it inherits the distinctive feature of expressing qualitative context-dependent preferences among different alternatives (modeled as the atoms of a logic program). On the other hand, possibilistic normal logic programs allow to capture qualitative certainty labels about the rules themselves (modeled as necessity values according to possibilistic logic). In this way, the LPPODs framework supports a reasoning which is nonmonotonic, preference- and uncertainty-aware.

We define a syntax which extends LPODs to be able to associate necessity values to program rules. In this way, each rule in an LPPOD is provided with a value which can be used as complementary scale to the rule satisfaction degree of LPODs to compare LPPODs solutions (in particular it can be seen as a priority). We define a possibilistic semantics for capturing LPPODs which is a generalization of the LPODs semantics according to two different characterizations. Similar to what happens in possibilistic logic and in possibilistic normal logic programs, an LPPOD can be seen as a compact representation of the possibility distribution defined on the interpretations representing the information. Indeed, two equivalent characterizations of the LPPODs semantics can be given.

Preferences and necessity values can be used to specify an order among possibilistic answer sets of an LPPOD. For this purpose, we define a set of transformation rules for providing en effective mechanism to propagate necessity values between the rules of an LPPOD. Based on these results, we define a possibilistic preference relation which compares the possibilistic answer sets of an LPPOD using two scales: *rule satisfaction degrees* and *rule necessity values*. This approach can provide a flexible solution for comparing possibilistic answer sets, since the order between them can also depend on the certainty associated to program rules.

The chapter is organized as follows. After defining the LPPODs syntax which extends LPODs to be able to consider necessity values associated to program rules (Section 5.1), in Section 5.2, we define a possibilistic semantics for capturing LPPODs according to two different characterizations: a syntactical characterization based on a fix-point operator (Section 5.2.1), and a semantical characterization defined in terms of the least possibility distribution (Section 5.2.2). In Section 5.3, we present a set of program transformations for propagating necessity values between program rules. Based on these transformations, in Section 5.4, we show how preferences and necessity values can be used to specify an order among possibilistic answer sets of an LPPOD. In Section 5.5, we describe a simple algorithm to compute the LPPODs semantics. Section 5.6 is devoted to compare our framework with other works dealing with uncertainty and preferences. Finally, Section 5.7 concludes

[1] In this chapter, we present the LPPODs framework in terms of selection of the most certain beliefs to be used in the reasoning taking into account certainty values. However, the framework can be also understood in terms of weighted user preferences in which necessity values associated to preference rules are interpreted as priority labels. This interpretation is first discussed in Section 5.7 and then, in Chapter 7, we present a real use-case scenario in which the LPPODs framework is employed to model user profiles.

the chapter by discussing the representational capabilities of the LPPODs framework in terms of user preference modeling.

5.1 LPPODs Syntax

The syntax of an LPPOD is based on the syntax of ordered disjunction rules (Chapter 2, Section 2.2.4) and of possibilistic logic (Chapter 2, Section 2.4). A *possibilistic atom* is a pair $p = (a, \alpha) \in \mathcal{A} \times \mathcal{S}$ in which $\mathcal{A}$ is a set of atoms and $\mathcal{S} \subseteq (0, 1]$ is a finite totally ordered set of necessity values. The projection $*$ for any possibilistic atom p is defined as $p^* = a$. Given a set of possibilistic atoms M, the projection of $*$ over M is defined as $M^* = \{p^* \mid p \in M\}$. A possibilistic ordered disjunction rule r is of the form

$$\alpha : c_1 \times \ldots \times c_k \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- \tag{5.1}$$

in which $\alpha \in \mathcal{S}$ and $c_1 \times \ldots \times c_k \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ is an ordered disjunction rule.

The projection $*$ for a possibilistic ordered disjunction rule r is $r^* = c_1 \times \ldots \times c_k \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$. $Nec(r) = \alpha$ is a necessity degree representing the certainty level of the information described by r. A possibilistic constraint c is of the form $\mathbf{1} : \bot \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$. Observe that the necessity value of a possibilistic constraint is $\mathbf{1}$, since, like constraints in standard ASP, the purpose of the possibilistic constraint is to eliminate possibilistic answer sets. As such, we assume that there is no uncertainty about the information captured by a possibilistic constraint. As in possibilistic ordered disjunction rules, the projection $*$ for a possibilistic constraint c is $c^* = \bot \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$. To manage possibilistic constraints, we will replace each possibilistic rule of the form $\mathbf{1} :\leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ by a new rule of the from $\mathbf{1} : f \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-,\ not\ f$ such that f is a new atom symbol which does not appear in the signature of the possibilistic program.

A *Logic Program with Possibilistic Ordered Disjunction* (LPPOD) is a finite set of possibilistic ordered disjunction rules and/or possibilistic constraints. If an LPPOD does not contain possibilistic ordered disjunction rules, then it is known as a *possibilistic normal logic program*. If a possibilistic normal logic program does not contain negation as failure, then it is a *possibilistic definite logic program*. The projection of $*$ over P is defined as $P^* = \{r^* \mid r \in P\}$. Please observe that P^* is an LPOD.

Example 5.2. We are now able to represent our introductory program described before. By using the following certainty scale: **1** is *absolutely certain*, **0.9** is *quasi certain*, **0.6** is *almost certain*, and **0.4** is *little certain*, we can associate a certainty degree to each rule:

$$P = \left\{ \begin{array}{llcl} r_1 = \mathbf{1}: & b & \leftarrow & ant. \\ r_2 = \mathbf{0.9}: & f & \leftarrow & b, not\ ab_1. \\ r_3 = \mathbf{0.6}: & \neg f & \leftarrow & ant, not\ ab_2. \\ r_4 = \mathbf{1}: & ab_1 \times ab_2 & \leftarrow & p. \\ r_5 = \mathbf{1}: & ab_2 \times ab_1 & \leftarrow & sp. \\ r_6 = \mathbf{1}: & \bot & \leftarrow & ab_1, ab_2. \\ r_7 = \mathbf{0.6}: & p & \leftarrow & ant. \\ r_8 = \mathbf{0.4}: & sp & \leftarrow & ant. \\ r_9 = \mathbf{1}: & ant & \leftarrow & \top. \end{array} \right\}$$

The first rule states that we are *absolutely certain* that an antarctic bird is a bird (r_1). Then, we are *quasi certain* that birds normally fly (r_2), while we are *almost certain* that antarctic birds normally do not fly (r_3). We also consider that rule r_4, *i.e.,* being a penguin it is more reasonable to assume that antarctic birds cannot fly rather than they can fly, and rule r_5, *i.e.,* being a super-penguin it is more reasonable to assume that antarctic birds can fly rather than they cannot fly, are *absolutely certain*. However, r_7 and r_8 express that we are *almost certain* that an antarctic bird is a penguin and that we are *little certain* that an antarctic bird is a super-penguin. The last rule (r_9) indicates that the we are *absolutely certain* to observe an antarctic bird.

In the following, we suppose that necessity values associated to program rules are determined by a knowledge discovery process. These values induce a certainty scale which has to be taken into account when inferring the possibilistic answer sets of an LPPOD. In the sequel, we show how it is possible to characterize the LPPODs semantics in two different ways which lead to the same possibilistic answer sets.

5.2 LPPODs Semantics

Similar to what happens in possibilistic logic and in possibilistic normal logic programs, an LPPOD can be seen as a compact representation of the possibility distribution defined on the interpretations representing the information. Indeed, two equivalent characterizations of the LPPODs semantics can be given, a *syntactical characterization* based on a fix-point operator (Section 5.2.1), and a *semantical characterization* defined in terms of the least possibility distribution (Section 5.2.2).

5.2.1 Possibilistic Answer Sets of LPPODs by Fix-Point

The first characterization of the LPPODs semantics is in terms of the *possibilistic least model* presented in [Nicolas et al., 2006] for possibilistic normal logic programs. Before presenting this characterization, we introduce some relevant concepts.

Given a finite set of atoms $\mathcal{A}$ and $\mathcal{S} \subseteq (0, 1]$, the finite set of all the possibilistic atom sets induced by $\mathcal{A}$ and $\mathcal{S}$ is denoted by $\mathcal{PS} = 2^{\mathcal{A} \times \mathcal{S}}$. Each $A \in \mathcal{PS}$ is a set of possibilistic atoms such that $\forall a \in A, |\{(a, \alpha\})| \leq 1$, *i.e.,* every atom a occurs at most one time in A.

The LPPODs semantics is based on a fix-point operator ΠC_n which was introduced in [Nicolas et al., 2006] by considering possibilistic definite logic programs. To be able to

reuse such result, we first define a syntactic reduction for LPPODs which represents the possibilistic extension of the LPODs reduction (see Chapter 2, Section 2.2.4).

Definition 5.1 (Possibilistic Reduction $r_{\times}^{M}$). *Let $r = \alpha : c_1 \times \ldots \times c_k \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ be a possibilistic ordered disjunction rule and M be a set of atoms. The $\times$-possibilistic reduction $r_{\times}^{M}$ is defined as follows:*

$$r_{\times}^{M} = \{\alpha : c_i \leftarrow \mathcal{B}^+ | c_i \in M \wedge M \cap (\{c_1, \ldots, c_{i-1}\} \cup \mathcal{B}^-) = \emptyset\} \tag{5.2}$$

Definition 5.2 (Possibilistic Reduction $P_{\times}^{M}$). *Let P be an LPPOD and M be a set of atoms. The $\times$-possibilistic reduction $P_{\times}^{M}$ is defined as follows:*

$$P_{\times}^{M} = \bigcup_{r \in P} r_{\times}^{M} \tag{5.3}$$

Example 5.3. Let P be the LPPOD in Example 5.2 and let us consider the following sets of atoms M_1^*=$\{ant, p, sp, ab_1, b, \neg f\}$ and $M_2^* = \{ant, p, sp, ab_2, b, f\}$. We can see that:[2]

$$P_{\times}^{M_1^*} = \begin{Bmatrix} r_1 = \mathbf{1}: & b & \leftarrow ant. \\ r_2 = \mathbf{0.9}: & f & \leftarrow b. \\ r_4 = \mathbf{1}: & ab_1 & \leftarrow p. \\ r_5 = \mathbf{1}: & ab_1 & \leftarrow sp. \\ r_7 = \mathbf{0.6}: & p & \leftarrow ant. \\ r_8 = \mathbf{0.4}: & sp & \leftarrow ant. \\ r_9 = \mathbf{1}: & ant & \leftarrow \top. \end{Bmatrix} \qquad P_{\times}^{M_2^*} = \begin{Bmatrix} r_1 = \mathbf{1}: & b & \leftarrow ant. \\ r_3 = \mathbf{0.9}: & \neg f & \leftarrow ant. \\ r_4 = \mathbf{1}: & ab_2 & \leftarrow p. \\ r_5 = \mathbf{1}: & ab_2 & \leftarrow sp. \\ r_7 = \mathbf{0.6}: & p & \leftarrow ant. \\ r_8 = \mathbf{0.4}: & sp & \leftarrow ant. \\ r_9 = \mathbf{1}: & ant & \leftarrow \top. \end{Bmatrix}$$

Please observe that the programs $P_{\times}^{M_1^*}$ and $P_{\times}^{M_2^*}$ are possibilistic definite logic programs.

In general, once an LPPOD P has been reduced by a set of atoms M^*, it is possible to test whether M is a possibilistic answer set of the program P by considering the possibilistic least model ΠCn of P. ΠCn is defined in terms of the β-applicability of a rule w.r.t. a set of possibilistic atoms and in terms of the possibilistic consequence operator ΠT.

Definition 5.3. [Nicolas et al., 2006] *Let P be a possibilistic definite logic program such that r is of the form $\alpha : a \leftarrow b_1, \ldots, b_m$ and M be a set of possibilistic atoms,*

- *r is β-applicable in M with $\beta = min\{\alpha, \alpha_1, \ldots, \alpha_n\}$ if $\{(b_1, \alpha_1), \ldots, (b_m, \alpha_n)\} \subseteq M$.*
- *r is 0-applicable otherwise.*

And then, for all atom $\varphi \in \mathcal{L}_{P^*}$ we define:

$$App(P, M, \varphi) = \{r \in head(P, \varphi) | r \text{ is } \beta\text{-applicable in } M \text{ and } \beta > 0\}$$

inn which $\varphi \in \mathcal{L}_{P^*}$, $head(P, \varphi) = \{r \in P | head(r^*) = \varphi\}$, and $head(P^*) = \{head(r^*) | r^* \in P^*\}$.

Given a set of possibilistic atoms M, the applicability degree β of a rule β-applicable in M captures the certainty of the conclusion that the rule can produce w.r.t. M. If the body is

[2] r_6 is deleted by our possibilistic reduction by the way in which we manage possibilistic constraints in our LPPODs (see Section 5.1).

empty, then the rule is applicable with its own certainty degree. If the body is not satisfied by M, then the rule is not applicable at all. Otherwise, the applicability level of the rule depends on the certainty level of the propositions.

Based on the β-applicability of a rule w.r.t. a set of possibilistic atoms, the consequence operator ΠT_P for logic programs has been extended in the following way:

Definition 5.4. [Nicolas et al., 2006] *Let P be a possibilistic definite logic program and M be a set of possibilistic atoms. The immediate possibilistic consequence operator ΠT_P maps a set of possibilistic atoms to another one by this way:*
$\Pi T_P(M) = \{(\varphi, \delta) | \varphi \in head(P^*), App(P, M, \varphi) \neq \emptyset,$
$\delta = max_{r \in App(P,M,\varphi)}\{\beta | r \text{ is } \beta\text{-applicable in } M\}\}$
Then, the iterated operator ΠT_P^k is defined by

$$\Pi T_P^0 = \emptyset \text{ and } \Pi T_P^{n+1} = \Pi T_P(\Pi T_P^n), \forall n \geq 0$$

Example 5.4. Let P_1 be the possibilistic definite logic program in Example 5.3 obtained by the possibilistic reduction $P_\times^{M_1^*}$. We can see that if we consider $M = \emptyset$ and the atom ant, then r_9 is 1-applicable in M. In fact, we can see that $App(P, \{\emptyset\}, ant) = \{r_5\}$. Also, we can see that if $M = \{(ant, 1)\}$ and the atoms p and sp are considered, then r_7 and r_8 are 0.6-applicable and 0.4-applicable in M respectively.

The consequence operator is monotonic and it always reaches a fix-point. The possibilistic model of a possibilistic definite logic program is characterized in the following way.

Proposition 5.1. [Nicolas et al., 2006] *Let P be a possibilistic definite logic program, then ΠT_P has a least fix-point $\bigsqcup_{n \geq 0} \Pi T_P^n$ that is called the set of possibilistic consequences of P and it is denoted by $\Pi Cn(P)$.*

By considering the fix-point operator $\Pi Cn(P)$ and the possibilistic reduction $P_\times^M$, the LPPODs semantics can be related to the LPODs semantics and the possibilistic answer set semantics. This directly leads to the following definition.

Definition 5.5 (Possibilistic Answer Set for LPPODs). *Let P be an LPPOD, 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 $M = \Pi Cn(P_\times^{M^*})$. We denote by $SEM_{LPPOD}(P)$ the set of all possibilistic answer sets of P and by SEM_{LPPOD} the LPPOD semantics.*

Example 5.5. Let P be the LPPOD introduced in Example 5.2 and let us consider the sets of atoms M_1^*=$\{ant, p, sp, ab_1, b, \neg f\}$ and $M_2^* = \{ant, p, sp, ab_2, b, f\}$. In order to infer the possibilistic answer sets of P, we have to infer the answer set of the LPOD P^*. It can be proved that both M_1^* and M_2^* are valid answer sets of P^*. As seen in Example 5.3, $P_\times^{M_1^*}$ and $P_\times^{M_2^*}$ are possibilistic definite logic programs. Therefore, the possibilistic answer sets of P are:

$$\Pi Cn(P_\times^{M_1^*}) = \{(ant, 1), (p, 0.6), (sp, 0.4), (b, 1), (\neg f, 0.9), (ab_1, 0.6)\}$$

since

$$
\begin{aligned}
&\Pi T^0_{P_\times^{M_1^*}} = \emptyset \\
&\Pi T^1_{P_\times^{M_1^*}} = \Pi T_{P_\times^{M_1^*}}(\emptyset) = \{(ant,1)\} \\
&\Pi T^2_{P_\times^{M_1^*}} = \Pi T_{P_\times^{M_1^*}}(\{(ant,1)\}) = \{(ant,1),(p,0.6),(sp,0.4),(b,1)\} \\
&\Pi T^3_{P_\times^{M_1^*}} = \Pi T_{P_\times^{M_1^*}}(\{(ant,1),(p,0.6),(sp,0.4),(b,1)\}) = \\
&\qquad \{(ant,1),(p,0.6),(sp,0.4),(b,1),(\neg f,0.9),(ab_1,0.6)\} \\
&\Pi T^4_{P_\times^{M_1^*}} = \Pi T_{P_\times^{M_1^*}}(\{(ant,1),(p,0.6),(sp,0.4),(b,1),(\neg f,0.9),(ab_1,0.6)\}) = \\
&\qquad \{(ant,1),(p,0.6),(sp,0.4),(b,1),(\neg f,0.9),(ab_1,0.6)\} \\
&\Pi T^{k+1}_{P_\times^{M_1^*}} = \Pi T^k_{P_\times^{M_1^*}}, \forall k > 3
\end{aligned}
$$

and

$$\Pi Cn(P_\times^{M_2^*}) = \{(ant,1),(p,0.6),(sp,0.4),(b,1),(f,0.6),(ab_2,0.6)\}$$

since

$$
\begin{aligned}
&\Pi T^0_{P_\times^{M_2^*}} = \emptyset \\
&\Pi T^1_{P_\times^{M_2^*}} = \Pi T_{P_\times^{M_2^*}}(\emptyset) = \{(ant,1)\} \\
&\Pi T^2_{P_\times^{M_2^*}} = \Pi T_{P_\times^{M_2^*}}(\{(ant,1)\}) = \{(ant,1),(p,0.6),(sp,0.4),(b,1)\} \\
&\Pi T^3_{P_\times^{M_2^*}} = \Pi T_{P_\times^{M_2^*}}(\{(ant,1),(p,0.6),(sp,0.4),(b,1)\}) = \\
&\qquad \{(ant,1),(p,0.6),(sp,0.4),(b,1),(f,0.6),(ab_2,0.6)\} \\
&\Pi T^4_{P_\times^{M_2^*}} = \Pi T_{P_\times^{M_2^*}}(\{(ant,1),(p,0.6),(sp,0.4),(b,1),(f,0.6),(ab_2,0.6)\}) = \\
&\qquad \{(ant,1),(p,0.6),(sp,0.4),(b,1),(f,0.6),(ab_2,0.6)\} \\
&\Pi T^{k+1}_{P_\times^{M_2^*}} = \Pi T^k_{P_\times^{M_2^*}}, \forall k > 3
\end{aligned}
$$

From Definition 5.5, we can observe that there is an important condition w.r.t. the definition of a *possibilistic answer set* of an LPPOD: a possibilistic set M cannot be a possibilistic answer set of an LPPOD P, if M^* is not an answer set of an LPOD P^*.

When all the possibilistic rules of an LPPOD P have the necessity value of $\mathbf{1}$, each answer set of P^* can be directly generalized to be a possibilistic answer set of P.

Proposition 5.2 (Generalization). *Let P be an LPPOD, and M^* be an answer set of P^*. If $\forall r \in P$, $Nec(r) = \mathbf{1}$, then $M = \{(\varphi, \mathbf{1}) \mid \varphi \in M^*\}$ is a possibilistic answer set of P.*

An alternative characterization of the LPPODs semantics can be given by the least possibilistic distribution associated to a possibilistic definite logic program as it will be discussed in the next section.

5.2.2 Possibilistic Answer Sets of LPPODs by Possibility Distribution

Definition 5.5 proposes a syntactical way to compute the possibilistic answer sets of an LPPOD by means of the possibilistic $\times$-reduction (Definition 5.2) and the fix-point operator

$\Pi C n$. Similar to what has been done in [Nicolas et al., 2006] for possibilistic normal programs, we characterize the LPPODs semantics in terms of the least specific distribution for possibilistic definite logic programs.

Definition 5.6. [Nicolas et al., 2006] *Let P be a possibilistic definite logic program, M be a set of atoms, and $\pi_P : 2^{\mathcal{L}_P} \longmapsto [0,1]$ be a possibility distribution. π_P is compatible with P and it is the least specific distribution for P if:*[3]

$$\forall M \in 2^{\mathcal{L}_P} \begin{cases} \pi_P(M) = 0, & M \nsubseteq head(App(P^*, M)) \\ \pi_P(M) = 0, & App(P^*, M) \text{ is not grounded} \\ \pi_P(M) = 1, & M \text{ is an answer of } P^* \\ \pi_P(M) = 1 - \max_{r \in P}\{Nec(r) \mid M \not\models r^*\}, & \text{otherwise} \end{cases}$$

As it happens in the possibilistic logic settings, the possibility distribution π_P induces two dual measures which can be used to infer, in the ASP setting, the necessity degrees of each atom of a possibilistic definite logic program.

Definition 5.7. [Nicolas et al., 2006] *Let P be a possibilistic definite logic program and π_P its associated possibility distribution. The possibility and necessity measures are defined as:*

- $\Pi_P(\varphi) = \max_{M \in 2^{\mathcal{L}_P}}\{\pi_P(M) \mid \varphi \in M\}$
- $N_P(\varphi) = 1 - \max_{M \in 2^{\mathcal{L}_P}}\{\pi_P(M) \mid \varphi \notin M\}$

Given the above definitions and the $\times$-possibilistic reduction $P_\times^M$, it is possible to give a characterization of a possibilistic answer set of an LPPOD in terms of the possibility distribution for possibilistic definite logic programs.

Proposition 5.3. *Let P be an LPPOD and M be a set of possibilistic atoms such that M^* is an answer set of P^*, then*

$$\Pi M(P_\times^{M^*}) = \{(\varphi, N_{P_\times^{M^*}}(\varphi)) | \varphi \in \mathcal{L}_{P_\times^{M^*}}, N_{P_\times^{M^*}}(\varphi) > 0\}$$

is a possibilistic answer set of P.

Example 5.6. Let P be the LPPOD in Example 5.2, M_1^*=$\{ant, p, sp, ab_1, b, \neg f\}$, $M_2^* = \{ant, p, sp, ab_2, b, f\}$ be the answer sets of P^*, and $P_\times^{M_1^*}$, $P_\times^{M_2^*}$ be the reductions of P w.r.t. M_1^* and M_2^* respectively (Example 5.3).

Let us consider the possibility distribution π for a possibilistic definite logic program in Definition 5.6 and let $\mathcal{L}_{P_\times^{M_1^*}} = \{ant, p, sp, ab_1, b, \neg f\}$. The least specific possibility distribution on $2^{\mathcal{L}_{P_\times^{M_1^*}}}$ induced by $P_\times^{M_1^*}$ is:

- $\pi_{P_\times^{M_1^*}}(\{ant, b\}) = 1 - max\{0.9, 0.6, 0.4\} = 0.1$
- $\pi_{P_\times^{M_1^*}}(\{ant, b, \neg f\}) = 1 - max\{0.6, 0.4\} = 0.4$

[3] A definite logic program P^* is said to be grounded if it can be ordered as a sequence $\langle r_1, \ldots, r_n \rangle$ such that $\forall i, 1 \leq i \leq n, r_i \in App(P, head(\{r_1, \ldots, r_{i-1}\}))$.

- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, p\}) = 1 - max\{0.4, 0.9\} = 0.1$
- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, sp\}) = 1 - max\{0.6, 0.9\} = 0.1$
- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, \neg f, p\}) = 1 - max\{0.6\} = 0.4$
- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, \neg f, p\}) = 1 - max\{0.4\} = 0.6$
- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, \neg f, p, b\}) = 1$ (the answer set)
- for all the other sets $S \in 2^{\mathcal{L}_{P_\times^{M_1^*}}}$, $\pi_{P_\times^{M_1^*}}(S) = 0$

As all the atoms in the signature of $P_\times^{M_1^*}$ belong to the answer set of P^*, their level of consistency with respect to $P_\times^{M_1^*}$ is the most possible value. As expected. the possibility measures associated to each of the atoms are:

- $\Pi_{P_\times^{M_1^*}}(ant) = \Pi_{P_\times^{M_1^*}}(p) = \Pi_{P_\times^{M_1^*}}(sp) = \Pi_{P_\times^{M_1^*}}(ab_1) = \Pi_{P_\times^{M_1^*}}(b) = \Pi_{P_\times^{M_1^*}}(\neg f) = 1$

Necessity measures instead evaluate the certainty level at which each atom is inferred from $P_\times^{M_1^*}$. As expected, necessity values inferred by the possibility distribution $\pi_{P_\times^{M_1^*}}$ are consistent with the necessity values associated to each possibilistic atom in the possibilistic answer set of M_1 of P which was inferred by the fix-point in Example 5.5.

- $N_{P_\times^{M_1^*}}(ant) = 1$
- $N_{P_\times^{M_1^*}}(p) = 0.6$
- $N_{P_\times^{M_1^*}}(sp) = 0.4$
- $N_{P_\times^{M_1^*}}(ab_1) = 0.6$
- $N_{P_\times^{M_1^*}}(b) = 1$
- $N_{P_\times^{M_1^*}}(\neg f) = 0.9$

The inference of the necessity values related to $M_2^* = \{ant, p, sp, ab_2, b, f\}$ in the possibilistic definite logic program $P_\times^{M_2^*}$ can be done in a similar way.

Therefore, the LPPODs semantics can be defined by two different but equivalent characterizations: the syntactical characterization based on a fix-point operator and the semantical characterization based on the possibility distribution for possibilistic definite logic programs. For the sake of computation, in Section 5.5, we will see how the syntactical characterization provides a straightforward methodology for the LPPODs semantics computation.

In the next section, we present a set of transformation rules which allow one to propagate necessity values between rules while preserving the LPPODs semantics. These results are especially important for the definition of a comparison criterion. In fact, based on these transformations, in Section 5.4, we will see how to define a comparison criterion for selecting possibilistic answer sets which can consider both *necessity values* and *satisfaction degrees* of possibilistic ordered disjunction rules.

5.3 Transformation Rules for LPPODs

Generally speaking, a *transformation rule* is a syntactic rule which specifies the conditions under which a logic program P can be transformed to another program P'. In the logic programming semantics literature, transformation rules have been studied for normal logic programs and disjunctive logic programs in order to reduce the complexity of computing the answer sets of a logic program [Brass et al., 2001; Dix et al., 2001; Osorio et al., 2001b; Brass and Dix, 1995, 1999]. A common requirement is that the transformations can be used to reduce the size of a logic program provided that they do not affect its semantics. In this context, several notions of equivalence between logic programs have been defined [Lifschitz et al., 2001]. We generalize some basic transformations of normal programs (*EC*, RED^+, RED^-, *Success*, *Failure*, *Loop*) (see Chapter 2, Section 2.3) to LPPODs and we show how these transformations can reduce the structure of an LPPOD to a normal form without affecting the LPPODs semantics. To this end, we first prove that the transformations preserve *weak equivalence* and we reuse the theory of rewriting system [Dershowitz and Plaisted, 2001] to guarantee that the normal form is always unique. We denote a possibilistic ordered disjunction rule r (of the form expressed by 5.1) by $\alpha : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$. We write $HEAD(P)$ for the set of all atoms occurring in rule heads of an LPPOD P. We generalize Lifschitz's notion of *weak equivalence* [Lifschitz et al., 2001] and we say that two LPPODs P and P' are *equivalent* w.r.t. the LPPODs semantics (denoted $SEM_{LPPOD}(P) \equiv SEM_{LPPOD}(P')$) if they possess the same possibilistic answer sets.

In the following, we refer to a rewriting rule as a program transformation as defined in [Osorio et al., 2001a]. A program transformation $\rightarrow$ is a binary relation on $Prog_{\mathcal{A}-\mathcal{S}}$ where $Prog_{\mathcal{A}-\mathcal{S}}$ is the set of all LPPODs with atoms from the signature $\mathcal{A}$ and necessity values from $\mathcal{S} \subseteq (0,1]$. $\rightarrow$ maps an LPPOD P to another LPPOD P'. We use $P \rightarrow_T P'$ for denoting that we get P' from P by applying a transformation rule T to P.

We illustrate the transformations by means of the following running example:

Example 5.7.

$$P_0 = \left\{ \begin{array}{lll} r_1 = \mathbf{1}: & a \times b \leftarrow & not\ c, not\ d. \\ r_2 = \mathbf{1}: & c \times d \leftarrow & e, not\ e. \\ r_3 = \mathbf{1}: & b \times a \leftarrow & c. \\ r_4 = \mathbf{1}: & a \times b \leftarrow & d. \\ r_5 = \mathbf{1}: & \bot \leftarrow & a, b. \\ r_6 = \mathbf{0.6}: & c \leftarrow & not\ e. \\ r_7 = \mathbf{0.4}: & d \leftarrow & not\ e. \end{array} \right\}$$

Definition 5.8 (Possibilistic Elimination of Contradictions). *Given two LPPODs P and P', P' results from P by possibilistic elimination of contradictions ($P \rightarrow_{PeC} P'$) if P contains a rule $r = \alpha : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ which has an atom b such that $b \in \mathcal{B}^+$ and $b \in \mathcal{B}^-$, and $P' = P \backslash \{r\}$.*

By applying this first transformation to the program in Example 5.7, we get rid of rule r_2 and we obtain P_1:

Example 5.8 ($P_0 \rightarrow_{PeC} P_1$).

$$P_1 = \left\{ \begin{array}{lllll} r_1 = & \mathbf{1}: & a \times b & \leftarrow & not\ c, not\ d. \\ r_3 = & \mathbf{1}: & b \times a & \leftarrow & c. \\ r_4 = & \mathbf{1}: & a \times b & \leftarrow & d. \\ r_5 = & \mathbf{1}: & \bot & \leftarrow & a, b. \\ r_6 = & \mathbf{0.6}: & c & \leftarrow & not\ e. \\ r_7 = & \mathbf{0.4}: & d & \leftarrow & not\ e. \end{array} \right\}$$

We would like also to delete *not e* from all rule bodies whenever e does not appear in the head of an LPPOD. This can be guaranteed by the *Possibilistic Positive Reduction*. Contrariwise, if an LPPOD contains $\alpha : a \leftarrow \top$, then the atoms in the head must be true, so rules containing in their bodies *not a* are surely false and should be deleted. This can be guaranteed by the *Possibilistic Negative Reduction*.

Definition 5.9 (Possibilistic Positive Reduction). *Given two LPPODs P and P', P' results from P by possibilistic positive reduction $PRED^+$ ($P \rightarrow_{PRED^+} P'$), if there is a rule $r = \alpha : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ (\mathcal{B}^- \cup \{b\})$ in P and such that $b \notin HEAD(P)$, and $P' = (P \backslash \{r\}) \cup \{\alpha : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-\}$.*

Definition 5.10 (Possibilistic Negative Reduction). *Given two LPPODs P and P', P' results from P by possibilistic negative reduction $PRED^-$ ($P \rightarrow_{PRED^-} P'$), if P contains the rules $r = \alpha : a \leftarrow \top$, and $r' = \beta : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ (\mathcal{B}^- \cup \{a\})$, and $P' = (P \backslash \{r'\})$.*

An application of these reductions reduces the size of an LPPOD. In our example, we can apply $\rightarrow_{PRED^+}$ (two times) to obtain P_2 and then $\rightarrow_{PRED^-}$ to obtain P_3.

Example 5.9 ($P_1 \rightarrow_{PRED^+} P_2 \rightarrow_{PRED^+} P_3$, $P_3 \rightarrow_{PRED^-} P_4$).

$$P_3 = \left\{ \begin{array}{lllll} r_1 = & \mathbf{1}: & a \times b & \leftarrow & not\ c, not\ d. \\ r_3 = & \mathbf{1}: & b \times a & \leftarrow & c. \\ r_4 = & \mathbf{1}: & a \times b & \leftarrow & d. \\ r_5 = & \mathbf{1}: & \bot & \leftarrow & a, b. \\ r_6 = & \mathbf{0.6}: & c & \leftarrow & \top. \\ r_7 = & \mathbf{0.4}: & d & \leftarrow & \top. \end{array} \right\} \qquad P_4 = \left\{ \begin{array}{lllll} r_3 = & \mathbf{1}: & b \times a & \leftarrow & c. \\ r_4 = & \mathbf{1}: & a \times b & \leftarrow & d. \\ r_5 = & \mathbf{1}: & \bot & \leftarrow & a, b. \\ r_6 = & \mathbf{0.6}: & c & \leftarrow & \top. \\ r_7 = & \mathbf{0.4}: & d & \leftarrow & \top. \end{array} \right\}$$

The following two transformations are usually used to replace the Generalize Principle of Partial evaluation ($GPPE$) [Brass and Dix, 1995, 1999].

Definition 5.11 (Possibilistic Success). *Given two LPPODs P and P', P' results from P by possibilistic success ($P \rightarrow_{PS} P'$), if P contains a fact $\alpha : a \leftarrow \top$ and a rule $r = \beta : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ such that $a \in \mathcal{B}^+$, and $P' = (P \backslash \{r\}) \cup \{min\{\alpha, \beta\} : \mathcal{C}^\times \leftarrow (\mathcal{B}^+ \backslash \{a\}),\ not\ \mathcal{B}^-\}$.*

Definition 5.12 (Possibilistic Failure). *Given two LPPODs P and P', P' results from P by possibilistic failure ($P \rightarrow_{PF} P'$), if P contains a rule $r = \alpha : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ such that $a \in \mathcal{B}^+$ and $a \notin HEAD(P)$, and $P' = (P \backslash \{r\})$.*

The *Possibilistic Success* is particularly important for LPPODs because it allows to propagate necessity values between LPPOD rules. Please observe how such transformation reflects the possibilistic modus ponens (GMP) (see Chapter 2, Section 2.4). By applying $\rightarrow_{PF}$ to P_4 (two times), we obtain:

Example 5.10 ($P_4 \rightarrow_{PS} P_5 \rightarrow_{PS} P_6$).

$$P_6 = \left\{ \begin{array}{lll} r_3 = \mathbf{0.6} : b \times a & \leftarrow & \top. \\ r_4 = \mathbf{0.4} : a \times b & \leftarrow & \top. \\ r_5 = \mathbf{1} : \quad \bot & \leftarrow & a, b. \\ r_6 = \mathbf{0.6} : \quad c & \leftarrow & \top. \\ r_7 = \mathbf{0.4} : \quad d & \leftarrow & \top. \end{array} \right\}$$

The last transformation that we add is the possibilistic loop. For its definition it is useful to map an LPPOD to a possibilistic normal program. Given a possibilistic ordered disjunction rule $r = \alpha : \mathcal{C}^{\times} \leftarrow \mathcal{B}^{+},\ not\ \mathcal{B}^{-}$, we write $ordis - nor(r)$ to denote the set of possibilistic normal rules $\{\alpha : c \leftarrow \mathcal{B}^{+},\ not\ (\mathcal{B}^{-} \cup (\mathcal{C}^{\times} \backslash \{c\}))|c \in \mathcal{C}^{\times}\}$ and we extend this definition to LPPODs as follows. Let P be an LPPOD, then $ordis - nor(P)$ denotes the possibilistic normal program $\bigcup_{r \in P} ordis - nor(r)$. Given a possibilistic normal program $ordis - nor(P)$, we write $def(ordis - nor(P))$ to denote the possibilistic definite logic program that is obtained from $ordis - nor(P)$ by removing every negated-by-failure atom in $ordis - nor(P)$. Given a definite logic program $def(ordis - nor(P))^*$, $Cn(def(ordis - nor(P))^*)$ denotes the unique minimal model of $def(ordis - nor(P))^*$ (that always exists for definite logic program, see [Lloyd, 1987]).

Definition 5.13 (Possibilistic Loop). *Given two LPPODs P and P', P' results from P by possibilistic loop ($P \rightarrow_{PLoop} P'$), if $P' = \{\alpha : \mathcal{C}^{\times} \leftarrow \mathcal{B}^{+},\ not\ \mathcal{B}^{-} | \mathcal{B}^{+} \cap unf(P) = \emptyset\}$, where $unf(P) = \mathcal{L}_{P^*} \backslash Cn(def(ordis - nor(P)^*)$ and $P \neq P'$.*

Based on these transformations, we define an abstract rewriting system that contains the possibilistic transformation rules introduced in the previous definitions.

Definition 5.14 (Rewriting System for LPPODs). *Let P be an LPPOD and $\mathcal{CS}_{LPPOD}$ be the rewriting system based on the possibilistic transformation rules $\{\rightarrow_{PeC}, \rightarrow_{PRED^+}, \rightarrow_{PRED^-}, \rightarrow_{PS}, \rightarrow_{PF}, \rightarrow_{PLoop}\}$. We denote the normal form of P w.r.t. $\mathcal{CS}_{LPPOD}$ by $norm_{\mathcal{CS}_{LPPOD}}(P)$.*[4]

As stated before, an essential requirement of program transformations is that they preserve the semantics of the programs to which they are applied. The following Lemma is an important result, since it allows to reduce an LPPOD without affecting its semantics.

Lemma 5.1 ($\mathcal{CS}_{LPPOD}$ preserves SEM_{LPPOD}). *Let P and P' be two LPPODs related by any transformation in $\mathcal{CS}_{LPPOD}$. Then $SEM_{LPPOD}(P) \equiv_p SEM_{LPPOD}(P')$.*

Returning to the description of the running example, it can be observed that the program P_6 has the property that it cannot be further reduced because none of our transformations is applicable. Therefore, $P_6 = norm_{\mathcal{CS}_{LPPOD}}(P_0)$. However, the normal form of P could have been obtained by applying a different set of transformations. In particular, the sequence $\rightarrow_{PRED^+}$ (three times), $\rightarrow_{PRED^-}$, $\rightarrow_{PF}$, and $\rightarrow_{PS}$ (two times) reduces the program P_0 to a program equivalent to P_6. The following theorem is a strong result as it shows that a different application of our transformations (in a different ordering) leads to the same reduced program.

[4] Soon, we will show that this normal form is unique.

Theorem 5.1 (confluence and termination). *Let P be an LPPOD. Then $\mathcal{CS}_{LPPOD}$ is confluent, noetherian and $norm_{\mathcal{CS}_{LPPOD}}(P)$ is unique.*

The results we have obtained are important, since they have some theoretical and practical implications. In the case of the LPPODs semantics implementation, the confluence guarantees that the order in which the transformations are applied does not matter (*i.e.*, it allows to apply the less costly transformation) and that an LPPOD can always be reduced to an unique normal form. Consequently, the semantics of LPPODs can be computed on the normal form of an LPPOD. Moreover, the *Possibilistic Success* is particularly important for LPPODs because it also allows to propagate necessity values between LPPODs rules. This will be of interest in the next section.

5.4 Possibilistic Answer Sets Selection

The $\times$ connective can be used to express a preference order among atoms. Then, an order among the answer sets of a logic program can be achieved, since each answer set is associated with a rule satisfaction degree. Such degree can be used to specify comparison criteria for the answer sets selection (Chapter 2, Section 2.2.4). Similar to what happens in LPODs, we can define the rule satisfaction degree of a possibilistic answer set of an LPPOD w.r.t. a possibilistic ordered disjunction rules as:

Definition 5.15. *Let P an LPPOD and let M be a possibilistic answer set of P. Then, M satisfies a possibilistic ordered disjunction rule $r = \alpha : c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1} \ldots,\ not\ b_{m+n}$, denoted by $deg_{M^*}(r^*)$:*

- *to degree 1 if $b_j \notin M^*$ for some j $(1 \leq j \leq m)$, or $b_i \in M^*$ for some i $(m+1 \leq i \leq m+n)$,*
- *to degree j $(1 \leq j \leq k)$ if all $b_l \in M^*$ $(1 \leq l \leq m)$, $b_i \notin M^*$ $(m+1 \leq i \leq m+n)$, and $j = min\{r \mid c_r \in M^*, 1 \leq r \leq k\}$.*

The satisfaction degree of a possibilistic answer set M w.r.t. a rule can provide a ranking of the possibilistic answer sets of an LPPOD. An order among the possibilistic answer sets can be obtained by means of the following comparison criterion.

Definition 5.16. *Let P be an LPPOD, and let M_1 and M_2 be two possibilistic answer sets of P. Then, M_1^* is preferred to M_2^* (denoted by $M_1^* \succ M_2^*$) iff $\exists\ r^* \in P^*$ such that $deg_{M_1^*}(r^*) < deg_{M_2^*}(r^*)$, and $\nexists r'^* \in P^*$ such that $deg_{M_2^*}(r') < deg_{M_1^*}(r'^*)$.*

The condition in the above definition follows the principle of Pareto optimality according to which an object is preferred if it is better or equal to another in all attributes and strictly better in at least one attribute.

As the reader has probably observed, the above definition only considers the no possibilistic part of an LPPOD. To illustrate the above definition, let us consider the following example.

Example 5.11. Let us return to the LPPOD in Example 5.2.

$$P = \left\{ \begin{array}{llll} r_1 = \mathbf{1}: & b & \leftarrow & ant. \\ r_2 = \mathbf{0.9}: & f & \leftarrow & b, not\ ab_1. \\ r_3 = \mathbf{0.6}: & \neg f & \leftarrow & ant, not\ ab_2. \\ r_4 = \mathbf{1}: & ab_1 \times ab_2 & \leftarrow & p. \\ r_5 = \mathbf{1}: & ab_2 \times ab_1 & \leftarrow & sp. \\ r_6 = \mathbf{1}: & \bot & \leftarrow & ab_1, ab_2. \\ r_7 = \mathbf{0.6}: & p & \leftarrow & ant. \\ r_8 = \mathbf{0.4}: & sp & \leftarrow & ant. \\ r_9 = \mathbf{1}: & ant & \leftarrow & \top. \end{array} \right\}$$

As seen before (Examples 5.5-5.6), P has two possibilistic answer sets: $M_1 = \{(ant, 1), (p, 0.6), (sp, 0.4), (b, 1), (\neg f, 0.9), (ab_1, 0.6)\}$ and $M_2 = \{(ant, 1), (p, 0.6), (sp, 0.4), (b, 1), (f, 0.6), (ab_2, 0.6)\}$. Let us consider the rule satisfaction degrees of M_1 and M_2:

Table 5.1. Example of Rule Satisfaction Degrees

Rule	$deg_{M_1^*}$	$deg_{M_2^*}$
$r_4^* = ab_1 \times ab_2 \leftarrow p$	1	2
$r_5^* = ab_2 \times ab_1 \leftarrow sp$	2	1

By Pareto-based comparison, it is easy to see that it is not possible to compare the possibilistic answer sets.

We can observe that rule satisfaction degrees are not enough to compare the possibilistic answer sets of a program such as P. However, the rules in our LPPODs are associated with certainty values, *i.e.,* an ordered disjunction rule can be more certain than another one. This observation motivates the need to explore for a more precise comparison criterion able to take rules necessity values into account when needed.

By exploiting the set of transformation rules we have defined in the previous section (which are a mechanism able to propagate necessity values between LPPODs rules), we can define a preference relation that takes the necessity values of LPPOD rules into account in order to compare possibilistic answer sets. The following *possibilistic preference relation* is able to consider rules satisfaction degrees and rules necessity values at the same time to specify an order between the possibilistic answer sets of an LPPOD.

Definition 5.17 (Possibilistic Preferred Relation). *Let P be an LPPOD, M_1 and M_2 be possibilistic answer sets of P, $norm_{\mathcal{CS}_{LPPOD}}(P)$ be the normal form of P w.r.t. the rewriting system $\mathcal{CS}_{LPPOD}$. M_1 is possibilistic preferred to M_2 ($M_1 \succ_{pp} M_2$) iff $\exists\ r \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_1^*}(r^*) < deg_{M_2^*}(r^*)$, and $\nexists r' \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_2^*}(r'^*) < deg_{M_1^*}(r'^*)$ and $Nec(r) < Nec(r')$.*[5]

This definition tells us that, once we have rewritten an LPPOD P by applying the transformations seen before, we are able to compare the possibilistic answer sets using directly the normal form of the program (as the LPPODs semantics is not affected by the transfor-

[5] $Nec(r)$ denotes the necessity value of a rule r as explained in Section 5.1.

mations and the normal form is unique). To illustrate the above definition, let us consider the following example.

Example 5.12. Let P be the LPPOD in Example 5.2, and $\mathcal{CS}_{LPPOD}$ be the abstract rewriting system we defined for LPPODs. By applying *Possibilistic Success* (three times) we can first rewrite r_1, r_7, and r_8 in P to obtain P_3:

$$P = \left\{ \begin{array}{llll} r_1 = \mathbf{1}: & b & \leftarrow & ant. \\ r_2 = \mathbf{0.9}: & f & \leftarrow & b, not\ ab_1. \\ r_3 = \mathbf{0.6}: & \neg f & \leftarrow & ant, not\ ab_2. \\ r_4 = \mathbf{1}: & ab_1 \times ab_2 & \leftarrow & p. \\ r_5 = \mathbf{1}: & ab_2 \times ab_1 & \leftarrow & sp. \\ r_6 = \mathbf{1}: & \bot & \leftarrow & ab_1, ab_2. \\ r_7 = \mathbf{0.6}: & p & \leftarrow & ant. \\ r_8 = \mathbf{0.4}: & sp & \leftarrow & ant. \\ r_9 = \mathbf{1}: & ant & \leftarrow & \top. \end{array} \right\} \quad P_3 = \left\{ \begin{array}{llll} r_1 = \mathbf{1}: & b & \leftarrow & \top. \\ r_2 = \mathbf{0.9}: & f & \leftarrow & b, not\ ab_1. \\ r_3 = \mathbf{0.6}: & \neg f & \leftarrow & ant, not\ ab_2. \\ r_4 = \mathbf{1}: & ab_1 \times ab_2 & \leftarrow & p. \\ r_5 = \mathbf{1}: & ab_2 \times ab_1 & \leftarrow & sp. \\ r_6 = \mathbf{1}: & \bot & \leftarrow & ab_1, ab_2. \\ r_7 = \mathbf{0.6}: & p & \leftarrow & \top. \\ r_8 = \mathbf{0.4}: & sp & \leftarrow & \top. \\ r_9 = \mathbf{1}: & ant & \leftarrow & \top. \end{array} \right\}$$

By applying the same transformation other two times, we can rewrite rule r_4 and r_5 and obtain P_5 which represents the normal form of P

$$P_5 = \left\{ \begin{array}{llll} r_1 = \mathbf{1}: & b & \leftarrow & \top. \\ r_2 = \mathbf{0.9}: & f & \leftarrow & b, not\ ab_1. \\ r_3 = \mathbf{0.6}: & \neg f & \leftarrow & ant, not\ ab_2. \\ r_4 = \mathbf{0.6}: & ab_1 \times ab_2 & \leftarrow & \top. \\ r_5 = \mathbf{0.4}: & ab_2 \times ab_1 & \leftarrow & \top. \\ r_6 = \mathbf{1}: & \bot & \leftarrow & ab_1, ab_2. \\ r_7 = \mathbf{0.6}: & p & \leftarrow & \top. \\ r_8 = \mathbf{0.4}: & sp & \leftarrow & \top. \\ r_9 = \mathbf{1}: & ant & \leftarrow & \top. \end{array} \right\}$$

It can be checked that $P_5 = norm_{\mathcal{CS}_{LPPOD}}(P)$ and that it has the same possibilistic answer sets of P, as expected. Once we have obtained the normal form of P, we can apply the possibilistic preference relation to compare M_1 and M_2. By considering rule satisfaction degrees $deg_{M_1^*}(r_4^*) = 1$, $deg_{M_1^*}(r_5^*) = 2$ and $deg_{M_2^*}(r_4^*) = 2$, $deg_{M_2^*}(r_5^*) = 1$, the necessity values of rules r_4 and r_5, $Nec(r_4) = 0.6$ and $Nec(r_5) = 0.4$, it is not difficult to see that $M_1 \succ_{pp} M_2$, since $Nec(r_4) > Nec(r_5)$ ($M_2 \not\succ_{pp} M_1$ follows by Definition 5.17 as well).

From a knowledge representation point of view, we can interpret this result in the following way. In the original LPPOD P, we were assuming that rule r_4, *i.e.,* being a penguin it is more reasonable to assume that birds cannot fly rather than antarctic birds can fly, and rule r_5, *i.e.,* being a super-penguin it is more reasonable to assume that antarctic birds can fly rather than birds cannot fly, were *absolutely certain*. However, these rules are supported by two pieces of knowledge, rules r_7 and r_8, which are *almost certain* and *little certain* respectively. As expected, these certainty values must be considered at the moment of deciding which one of the exceptions ab_1 and ab_2 w.r.t. the default rules r_2 and r_3 is the most plau-

sible. This has been achieved by means of a mechanism able to propagate necessity values between the rules of LPPOD P.

The reader may observe that, when necessity values are equal, it is not possible to achieve a total order between possibilistic answer sets of an LPPOD. However, this is in someway what it can be expected when both scales prevent such decision.

Nevertheless, we can observe that there is an important property w.r.t. the possibilistic preference relation for possibilistic answer sets: a possibilistic answer set M is comparable if M^* is comparable in the LPOD P^*. In fact, our extension generalizes the preference relation between answer sets of LPODs w.r.t. the preference relation for LPODs.

Proposition 5.4. *Let M_1 and M_2 be possibilistic answer sets of an LPPOD P, $\succ_p$ be the preference relation based on rules satisfaction degrees only, and $\succ_{pp}$ be the possibilistic preference relation. If $M_1^* \succ_p M_2^*$ then $M_1 \succ_{pp} M_2$.*

It is worthy to add that the approach we have defined in this section for comparing possibilistic answer sets based on the rewriting system CS_{LPPOD} can be replaced by other strategies. An alternative strategy can be obtained by associating a unique value, which merges necessity and satisfaction degree together, to each possibilistic answer set and to use such value to rank-order the possibilistic answer sets.[6] Other possible strategies can be proposed by defining an alternative program reduction which focuses on the propagation of necessity values only; or by transforming an LPPOD to an equivalent possibilistic normal logic program where special atoms are introduced for denoting the satisfaction degree of possibilistic ordered disjunction rules and by defining a comparison criteria based on the necessity values associated to these atoms.

5.5 Computing the LPPODs Semantics

In Section 5.2, we have seen how the LPPODs semantics can be characterized in two different ways: syntactically, in terms of a syntactic reduction and a fix-point operator, and semantically, in terms of the least specific possibility distribution compatible with an LPPOD.

For the sake of computation, the former characterization is of special interest as it defines a straightforward methodology for the LPPODs semantics implementation. In fact, according to Definition 5.5, the LPPODs semantics is defined in terms of the LPODs semantics and the possibilistic answer set semantics. As both semantics are computable, the decision problem of existence of a possibilistic answer set of an LPPOD is computable, as stated by the following theorem.

Theorem 5.2. *Let P be an LPPOD, such that $P \subseteq Prog_{\mathcal{A}-\mathcal{S}}$, where $Prog_{\mathcal{A}-\mathcal{S}}$ is the set of all LPPODs which can be generated by a finite set of atoms $\mathcal{A}$ and a finite set of necessity values $\mathcal{S}$. Then, the LPPODs semantics is computed by the function $SEM_{LPPOD} : P \rightarrow \mathcal{PS}$.*

[6] A possible way to associate a unique value to each possibilistic answer set is by means of the following function: $\sum_{1 \leq i \leq n} \frac{Nec(r_i)}{deg_{M^*}(r_i) n_r}$, in which n_r is the number of rules in an LPPOD P.

Algorithm 4 $SEM_{LPPOD}(P) : \mathcal{PM}$

Input: An LPPOD P
Output: Ordered Set of Possibilistic Answer Sets

```
PM ← ∅
P_LPOD ← P*
M ← SEM_LPOD(P_LPOD)
while M ≠ ∅ do
    M ← pop(M)
    PM ← ΠCn(P_×^M)
    push(PM, PM)
end while
return PM
```

In order to prove Theorem 5.2, since the LPPODs semantics maps any LPPOD to its possibilistic answer sets, it is sufficient to find an algorithm that can compute the possibilistic answer sets given an LPPOD P. The algorithm exists and it is described below.

Algorithm 4 accepts an LPPOD P, does some processing and returns an ordered set of possibilistic answer sets of P. To be able to infer the possibilistic answer sets of P, the algorithm follows the methodology provided by the LPPODs semantics syntactical characterization in a straightforward way. As first step, the projection $*$ is applied to P in order to obtain its classical part (P_{LPOD}). Since P_{LPOD} is an LPOD, the LPOD semantics (SEM_{LPOD}) can be used to infer its answer sets ($\mathcal{M}$). If no answer set of P_{LPOD} is found, then the algorithm terminates. Otherwise, each answer set of P_{LPOD} is used to reduce P ($P_{\times}^{M}$) into a possibilistic definite logic program to which the fix-point operator ΠCn can be applied to infer the necessity values of the atoms corresponding to the possibilistic answer sets of P. Since the consequence operator is monotonic, $\mathcal{PS}$ is finite, and each $ps \subset \mathcal{PS}$ is also finite, the operator always reaches a fix-point in a finite number of steps. Therefore, the algorithm always terminates. In this way, the algorithm provides an effective way to compute the LPPODs semantics.

Furthermore, the relation of the LPPODs semantics with the LPODs and possibilistic answer set semantics suggests an important results. In fact, it can be observed how the problem of deciding whether an LPPOD has a possibilistic answer set stays in the same complexity of finding an answer set of an LPOD, *i.e.*, Σ_2^P. In fact, finding a possibilistic answer set of an LPPOD can be done in polynomial time from the moment an answer set of its classical part is provided. This comes in a straightforward way from the fact that (i) the $\times$-possibilistic reduction converts an LPPOD to a possibilistic definite logic program and, (ii) the computation of the possibilistic least model of a possibilistic definite logic program P can be done in polynomial time w.r.t. $k \times |P|$ in which k is the number of different levels of certainty occurring in P. This is an important result, since it shows how LPPODs can yield a more expressive framework without increasing the complexity of its classical part.

5.6 Related Work

Since the LPPODs framework combines preferences, nonmonotonicity and uncertainty in a logic programming framework together, there are several works which relate to ours from

different perspectives. Most of the existing approaches either deal with uncertainty and nonmonotonic reasoning [Dubois et al., 1991; Saad and Pontelli, 2005; Nicolas et al., 2006; Nieves et al., 2011; Baral et al., 2009] or deal with preferences and nonmonotonic reasoning [Gelfond and Son, 1998; Zhang and Foo, 1997; Brewka and Eiter, 1999; Sakama and Inoue, 2000; Delgrande et al., 2003; Brewka et al., 2004b; Van Nieuwenborgh and Vermeir, 2006].

5.6.1 Uncertainty in Logic Programming

Concerning research on logic programming with uncertainty, existing approaches in the literature interpret the uncertainty in a qualitative or in a quantitative way.

As far as qualitative treating of uncertainty is concerned, the most popular approaches in the literature are based on possibility theory. The first approach which combines possibilistic logic and logic programming is an idea presented in [Dubois et al., 1991]. In this work, logic programming rules are associated with necessity measures and certainty about the atoms in valid interpretations of a logic program are inferred using possibilistic resolution [Dubois et al., 1994]. This approach is able to deal with inconsistent programs and with nonmonotonicity by means of the α-cut [Dubois et al., 1991, 1994], but it is not able to capture incomplete knowledge as it does not consider negation as failure as it is usually done in ASP.

In the context of ASP, one of the pioneer works about qualitative uncertainty handling is the work of Nicolas *et al.* (see Chapter 2, Section 2.5). It combines possibility theory with normal logic programs. Our approach clearly relates to this work, since the LPPODs semantics is based on the possibilistic semantics. Moreover, when all possibilistic rules in an LPPOD P are of the form $\alpha : c_1 \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$, the LPPODs semantics coincides with the possibilistic semantics of normal programs in which each possibilistic answer set has the maximum allowed satisfaction degree, *i.e.,* 1.

The approach of Nicolas *et al.* is extended by possibilistic disjunctive programs [Nieves et al., 2011]. The main motivation of this approach is to define a description language and a reasoning process in which the user can consider relative likelihoods for modeling different levels of uncertainty. Our approach differs in this respect, since we expect necessity values to be recovered by a knowledge discovery process and we use these values as priority levels between LPPOD rules for possibilistic answer sets selection. Moreover, uncertainty labels in [Nieves et al., 2011] belong to a partially ordered set, whereas in our approach necessity values belong to a totally ordered scale. Despite these slight differences, it is not difficult to see how the two semantics relate to each other. Since $\times$ in LPPODs is a special disjunction, and since the possibilistic disjunctive semantics [Nieves et al., 2011] is based on minimal models,[7] it is possible to represent an LPPOD by means of a possibilistic disjunctive program. The main difference is in the selection of the possibilistic answer sets of the program. Nevertheless, it seems possible to recover the satisfaction degree of LPPODs in possibilistic disjunctive programs as well. Intuitively, as in LPPODs a satisfaction degree is associated to a possibilistic answer set w.r.t. a possibilistic ordered disjunction rule, this

[7] According to the minimal model, given the possibilistic disjunctive rule: $\alpha : a \vee b$, $\{(a, \alpha)\}$ and $\{(b, \alpha)\}$ are valid possibilistic answer sets, but $\{(a, \alpha), (b, \alpha)\}$ is not.

degree can be recovered in a possibilistic disjunctive rule by defining a satisfaction relation for a possibilistic disjunction rule that takes the position of the satisfied atoms into account.

Another possibilistic extension of the semantics of LPODs, based on pstable semantics [Osorio et al., 2006], is the approach proposed in [Confalonieri et al., 2009a]. The main motivation behind this work is the handling of possibilistic rules of the form $\alpha : a \leftarrow not\ a$, which under answer set semantics are considered inconsistent. The core idea of the approach is first to characterize the LPODs semantics in terms of the pstable semantics and then to generalized such result to possibilistic LPODs [Confalonieri et al., 2009a]. Although the research line of such a framework is in paraconsistent logic, possibilistic pstable semantics for LPODs relates to LPPODs semantics, since each possibilistic answer set is a possibilistic pstable model in which the main difference consists in the necessity values of the inferred atoms (see Proposition 2 in [Confalonieri et al., 2009a]).

A noticeable approach to uncertainty handling in ASP based on possibilistic theory is reported in [Bauters et al., 2010a]. The authors proposed a revisited semantics for PASP. The core idea of this approach is to translate a normal program to a set of constraints on possibility distributions and then to extend this idea to cover possibilistic ASP. Our approach differs from [Bauters et al., 2010a] in the way in which the possibilistic answer sets are computed. Indeed, we first compute the answer set of the no possibilistic program and, then, we use the possibilistic semantics for inferring the necessity values (according to the approach in [Nicolas et al., 2006]). This revisited possibilistic semantics has been extended to possibilistic disjunctive logic programs [Bauters et al., 2011].

In the case the uncertainty is treated quantitatively, the most used formalisms is probability theory [Saad and Pontelli, 2005; Baral et al., 2009]. These approaches basically differ from ours in the underlying notion of uncertainty and how uncertainty values associated to clauses and facts are managed.

5.6.2 Preferences in Logic Programming

Concerning works about preference handling, several approaches have been proposed as extension to classical ASP. As already pointed out in Chapter 3, these approaches differs in the way in which preferences are expressed. Since our approach is based on LPODs [Brewka et al., 2004b], we invite the reader to have a look at [Delgrande et al., 2004] for a comprehensive discussion about LPODs and other approaches. A more recent approach, which is not discussed in [Delgrande et al., 2004], is the work in [Van Nieuwenborgh and Vermeir, 2006]. In [Van Nieuwenborgh and Vermeir, 2006], preferences are used in the context of inconsistent programs for defining a partial order among program rules with the purpose to allow less important rules to be violated in order to satisfy rules with higher importance. Therefore, while in LPPODs preferences are used to select program solutions, in [Van Nieuwenborgh and Vermeir, 2006], they are used to select program rules.

5.7 Discussion and Concluding Remarks

In this chapter, we have proposed a possibilistic logic programming framework which is able to capture explicit preferences about rules having exceptions and uncertainty values in terms of necessity measures according to possibilistic logic [Dubois et al., 1994].

As discussed in Section 5.6, the use of possibilistic logic in ASP is not new and several proposals under different specifications have been studied [Nicolas et al., 2006; Nieves et al., 2011; Bauters et al., 2010a]. Nevertheless, LPPODs is the first logic programming specification able to consider explicit preferences and uncertainty together based on possibilistic answer set semantics.[8]

For achieving this, we have embedded several aspects of common-sense reasoning, *i.e.*, nonmonotonicity, preferences, and uncertainty, in a single framework. Each part is underpinned by two well established formalisms: LPODs and possibilistic normal logic programs. In joining these two works together, we have obtained a framework which is able to consider two scales to specify an order between the solutions of an LPPOD: the *rule satisfaction degree* of a possibilistic answer set w.r.t. a possibilistic ordered disjunction rule and the *necessity values* of the possibilistic rules themselves. In working towards this direction, we have progressively built all the theoretical knowledge needed to define a possibilistic preference relation which can consider the satisfaction degrees and the necessity values of program rules for comparing LPPODs solutions.

First, we have extended the LPODs syntax with possibilistic logic and we have defined the LPPODs semantics in terms of the LPODs and possibilistic answer set semantics (Definition 5.5). Then, we have introduced a set of transformation rules (Definition 5.14) which represent an efficient way to reduce the size of an LPPOD and to propagate the necessity values between LPPOD rules (preserving the semantics, Lemma 5.1 and Theorem 5.1). In this way, necessity values can be taken into account in the possibilistic answer sets selection (Definition 5.16). Moreover, since the LPPODs semantics is a generalization of both the LPODs semantics and the possibilistic answer set semantics (Definition 5.5, Proposition 5.2), the rewriting system we have defined for LPPODs can be applied, in principle, both to LPODs and to possibilistic normal programs.

We have described a simple algorithm for computing the LPPODs semantics (Algorithm 4) and we have discussed how the complexity of the LPPODs semantics belongs to the same complexity class of its classical part, *i.e.*, LPODs. This is an important result since:

- LPPODs can yield a more expressive framework without affecting the complexity, and
- the LPPODs framework can be used in a practical way.[9]

The LPPODs framework is not only a matter of selecting default rules explicitly when information is uncertain. The LPPODs capability to draw an order among possibilistic answer sets is a nice feature to have in systems that have to assist users in taking decisions.

[8] As discussed in Section 5.6, in [Confalonieri et al., 2009a], we proposed a possibilistic extension of LPODs based on possibilistic pstable semantics [Osorio et al., 2006; Osorio and Nieves, 2007].

[9] Indeed, we have implemented the LPPODs semantics in an ASP-based solver, that we called *posPsmodels*. We have integrated the solver in a context-aware system in which weighted preferences are considered to handle user profiles in order to enhance the system personalization capabilities (see Chapter 7).

The ordering can be an effective way to decide which options to choose considering uncertain information or to decide which preferences to use for selecting content from some available information. In such scenarios, the LPPODs specification can be used to represent context-dependent preferences with uncertain information or with importance weights. The modeling of degrees associated to LPPODs rules in terms of necessity values allows one to interpret such values either as *certainty* or as *priority* labels about the rules of a logic program. This is due to the fact that, in possibilistic logic, necessity values can be seen as a measure to qualitatively model the certainty level of a classical logic formula [Dubois et al., 1994] or as a measure to model the priority of a classical formula intended as a goal to be reached [Kaci et al., 2006]. In this way, it is possible to associate to each preference statement a scale indicating to *what extent a statement is certain* or to *what extent a statement is important*.

In order to illustrate the former interpretation, let us consider a revised version of the Example 4.12 presented in Chapter 4 in which preferences about going to the beach or to the cinema now depend on some uncertain weather conditions. In particular:

Example 5.13. A tourist visiting Barcelona has to decide whether to go to the beach or to the cinema. She has the following preferences: (i) *she normally prefers going to the beach over going to the cinema if she does not have any information about the weather condition (rain and sultriness)*. Otherwise, (ii) *she prefers to go to the cinema over going to the beach if it rains* and (iii) *she prefers to go to the beach over going to the cinema if it is sultry*. It is also known that (iv) *normally, if it is cloudy, then it will rain* and that (v) *normally, if it is humid, then it will make sultriness*. These last rules are uncertain: we are *quasi certain* (λ) about the former rule and *almost certain* (γ) about the latter one respectively (with $0 < \gamma < \lambda < 1$). Finally, we observe that it is a cloudy and humid day. A possible encoding of this scenario is:

$$P = \left\{ \begin{array}{llll} r_1 = \mathbf{1} : & beach \times cinema & \leftarrow & not\ rain, not\ sultriness. \\ r_2 = \mathbf{1} : & cinema \times beach & \leftarrow & rain. \\ r_3 = \mathbf{1} : & beach \times cinema & \leftarrow & sultriness. \\ r_4 = \mathbf{1} : & \bot & \leftarrow & cinema, beach. \\ r_5 = \lambda : & rain & \leftarrow & cloudy. \\ r_6 = \gamma : & sultriness & \leftarrow & humid. \\ r_7 = \mathbf{1} : & cloudy & \leftarrow & \top. \\ r_8 = \mathbf{1} : & humid & \leftarrow & \top. \end{array} \right\}$$

It can be observed that the above program without any certainty value is a classical LPOD. In such a case, no preference order can be achieved between the two alternatives *beach* and *cinema* (similarly to what happens in the antarctic birds example presented in this chapter). However, by considering certainty degrees and transformation rules, we can propagate certainty values, λ and γ, to ordered disjunction rules r_2 and r_3. Therefore, the preference about a *cinema* can be chosen, since it is *more certain* that it will rain rather than it will make *sultriness*.

The presence of *preferences* and *uncertainty* may let the reader think that we are dealing with decision making under uncertainty. Although it is tempting, this is not the case. Indeed, qualitative decision under uncertainty, as understood in approaches such as

[Boutilier, 1994; Tan and Pearl, 1994a,b; Dubois and Prade, 1995; Bonet and Geffner, 1996; Brafman and Tennenholtz, 1996], advocates for the use of two separate knowledge bases. One which represents uncertain knowledge of the world and another one which contains the preferences (associated with priority labels). Then, an optimal decision is computed according to pessimistic and optimistic criteria. This formulation, referred to as *qualitative decision making under uncertainty*, will be our research topic in Chapter 10.[10]

Instead, when knowledge and preferences are tied together and necessity values are interpreted as weights, the representational capabilities of the LPPODs framework can be used to model user preferences. Such preferences can depend both on contextual incomplete information and they can be associated with a weight representing their importance.

Indeed, in context-aware systems, users can have many preferences and an approach that considers all the specified preferences as equally important at the moment of selecting the content to suggest based on user preferences can be too rigid. Therefore, necessity values interpreted as weights can be used to express the importance of preference rules, and, consequently, of the preferences the rules contain. Such labels can be considered a kind of meta-preference which can serve to further distinguish between preferences which belong to the same LPPOD solution. We illustrate this idea by means of the following example.

Example 5.14. Let us consider some preferences that a user has about cinema suggestions: (i) *she prefers to watch an horror over an action movie if time is between 20 and 22*, otherwise (ii) *she prefers action over horror movies*. Then, (iii) *she prefers to pay by Master Card over using her Visa* and (iv) *she prefers a cinema accessible by metro line L1 rather than line L3* with the meta-information that preferences about the metro line are more important than the ones about the paying method. A possible way to encode her preferences is by means of the following LPPOD:

$$P' = \begin{Bmatrix} r_1 = \mathbf{1}: & horror \times action & \leftarrow & cinema, 20-22h. \\ r_2 = \mathbf{1}: & action \times horror & \leftarrow & cinema, not\ 20-22h. \\ r_3 = \mathbf{0.7}: & masterCard \times visa & \leftarrow & cinema. \\ r_4 = \mathbf{0.9}: & accessL1 \times accessL3 & \leftarrow & cinema. \\ r_5 = \mathbf{1}: & cinema & \leftarrow & \top. \end{Bmatrix}$$

The above program leads to several possibilistic answer sets which represent all preference combinations (8), in which $\{(cinema, 1), (accessL1, 0.9), (masterCard, 0.7), (action, 1)\}$ is the most preferred set and $\{(cinema, 1), (horror, 1), (visa, 0.7), (accessL3, 0.9)\}$ is the less preferred one (according to rule satisfaction degrees). When the system proceeds with the content selection, for each preference set it has to consider all the preferences which are listed. For instance, for the most preferred set $\{(cinema, 1),\ (accessL1, 0.9),\ (masterCard, 0.7),\ (action, 1)\}$ the most satisfactory result would be a cinema which has *action* movies, is reachable by metro *line L1*, and accepts *Master Card*. Such cinema may not exist. Preference weights can be used for considering preferences with higher importance only. Hence, a bit less satisfactory result would be a cinema which has *action* movies and is reachable by metro *line L1* (*i.e.,* the *Master Card* preference is dropped).

[10] We will show how LPPODs can be used as a computational machinery to compute an optimal decision in agreement with the qualitative decision theory formulated in the possibilistic logic setting [Dubois et al., 2001; Dubois and Prade, 1995].

To summarize, the LPPODs framework provides a flexible solution for: (i) the selection of default rules by means of explicit preferences and uncertain information; (ii) providing an effective reasoning tool for defining a user-oriented preference handling method (Chapter 7), and (iii) for supporting the computation of an optimal decision in qualitative decision making under uncertainty according to optimistic and pessimistic criteria formulated in the possibilistic logic setting (Chapter 10).

Part II

User Preference Modeling

6

User Preference Handling

One of the major issues in the process of preference handling is the modeling of user preferences. A preference model should be compact and rich enough to capture preferences and preference ordering relations. In the AI literature, several approaches have been proposed for modeling preferences. Approaches which extend classical logic express preferences by means of logical formulas and define preferred models among valid interpretations of the preference model. Graphically-based approaches offer a more intuitive preference representation. Logic programming approaches are able to express preference statements which can depend both an contextual information and incomplete knowledge. On the other hand, context is an important aspect in any personalization system which aims at staying closer to user desires and needs and several notions of context-awareness have been defined.

In this chapter, we will survey some of the most representative approaches of preference handling in AI. We also present the main characteristics of context-aware systems and we overview several definitions of context-awareness.

The chapter is organized as follows. After describing two non-classical logic based approaches (Sections 6.1-6.2), in Section 6.3, we present CP-nets. Section 6.4 overviews logic programming-based specifications which are able to express preferences and incomplete information. In Section 6.5, we describe the basic elements of context-aware systems, *i.e.*, user profiles and context. Finally, Section 6.6 concludes the chapter by pointing out the directions followed in the next two chapters.

6.1 User Preferences in Qualitative Choice Logic

Qualitative Choice Logic (QCL) [Brewka et al., 2004a] is a nonmonotonic propositional logic which extends classical logic in order represent qualitative choices. The non-standard part of QCL is a new logical connective $\overrightarrow{\times}$ capable to capture *ordered disjunctions*. Intuitively, if A and B are formulas, then $A \overrightarrow{\times} B$ says: if possible A, but if A is impossible then (at least) B.

QCL can be used to represent alternative, ranked options for problem solutions, as shown in the following example.

Example 6.1. [Brewka et al., 2004a] Let us imagine we want to represent the preferences of a user concerning a hotel booking. Let us assume there are four hotels available (h_1, h_2, h_3,

h_4) out of which she has to pick one. Each hotel has some characteristics, *e.g.,* whether it can be reached by walk, whether it has an hotel transportation service, or whether it can be reached by public transport. The user has the following preferences: she prefers to walk, over using the hotel transportation service over going by public transport. This scenario can be encoded by means of the following QCL knowledge base:

$$KB_{QCL} = \left\{ \begin{array}{c} h_1 \rightarrow walking \\ h_2 \rightarrow \neg walking \wedge hoteltransport \\ h_3 \rightarrow \neg walking \wedge \neg hoteltransport \wedge publictransport \\ h_4 \rightarrow \neg walking \wedge \neg hoteltransport \wedge \neg publictransport \\ h_1 \vee h_2 \vee h_3 \vee h_4 \\ walking \overrightarrow{\times} hoteltransport \overrightarrow{\times} publictransport \end{array} \right\}$$

Given the above formulas, the conclusion h_1 is obtained, since this is the only hotel satisfying the user's most intended option. However, if we suddenly discover that h_1 is unavailable (*i.e.,* we add $\neg h_1$ to KB_{QCL}), the conclusion h_2 is obtained, since the user's most favored property, *i.e., walking*, cannot be satisfied.

The above example shows how the inference relation of QCL is nonmonotonic (it actually satisfies all the postulates of System P [Kraus et al., 1990] as well as rational monotony [Pearl, 1990]). In fact, the semantics of this logic is defined in terms of *preferred models*. The definition of preferred models proceeds in two steps. First, each formula of the logic leads to a ranking of models, based on how well the models satisfy the formula. This is achieved by a degree of satisfaction of a formula in a particular (classical) model. For instance, if A and B are atoms and an interpretation I contains A, then I satisfies the formula $A \overrightarrow{\times} B$ as good as possible (1). If I contains no A but B, then it also satisfies $A \overrightarrow{\times} B$, but only in a worse way (2). The satisfaction degree is general enough to cover any qualitative choice formulas composed by any valid propositional formula (see [Brewka et al., 2004a] for details). Based on the satisfaction degree of a formula in a model, a satisfaction relation can be defined [Brewka et al., 2004a]. Secondly, a preference entailment is defined on the basis of the satisfaction relation of the single formulas. Different preference entailments characterize the QCL inference relation, *e.g.,* lexicographic-based, inclusion-based, ranking function-based ordering (depending on the cautiousness).

As stated in [Benferhat and Sedki, 2008], one of the limitations of QCL is that it does not correctly deal with negated and conditional preferences. In fact, using the QCL semantics, there is no difference between the material implication $klm \overrightarrow{\times} airfrance \rightarrow hotelpackage$ and the propositional formula $klm \vee airfrance \rightarrow hotelpackage$. Moreover, the negation in QCL misses some desirable properties from propositional calculus. In particular, when a negation is used in a QCL formula with ordered disjunction, that negated QCL formula is logically equivalent to a propositional formula obtained by replacing the ordered disjunction by the propositional disjunction [Benferhat and Sedki, 2008]. For instance, $(airfrance \overrightarrow{\times} klm) \rightarrow hotelpackage$ is equivalent to $(klm \overrightarrow{\times} airfrance) \rightarrow hotelpackage$ since $\neg(airfrance \overrightarrow{\times} klm) \vee hotelpackage$ and $\neg(klm \overrightarrow{\times} airfrance) \vee hotelpackage$ are translated into $(\neg airfrance \wedge \neg klm) \vee hotelpackage$. For this reason, in [Benferhat and Sedki, 2008], two other logics are proposed: $PQCL$ (prioritized qualitative choice logic) and $QCL+$ (pos-

itive qualitative choice logic). They are both based on the same QCL language, but they use different inference rules (see [Benferhat and Sedki, 2008] for details).

6.2 User Preferences in Possibilistic Logic

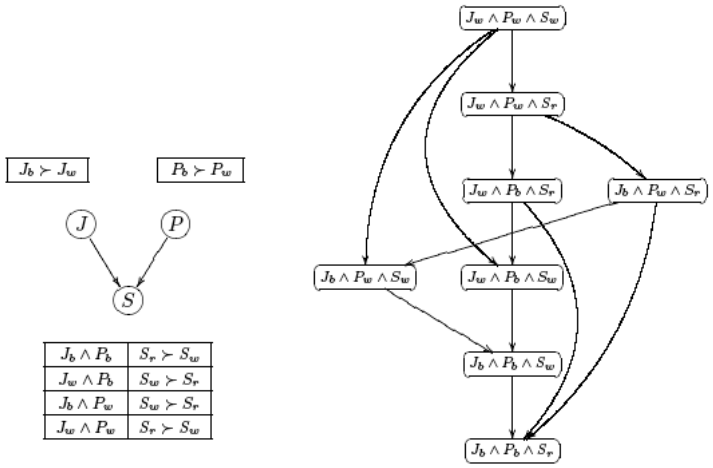
Possibilistic logic was initially introduced as a framework for representing uncertainty [Dubois et al., 1994]. Later on, it has also shown to be able to represent incomplete information [Benferhat et al., 1992, 1998]. More recently, it has appeared that it also allows to represent preferences [Benferhat et al., 2001], bipolarity [Benferhat et al., 2006], preference queries [Hadjali et al., 2008] and preference queries to possibilistic databases [Bosc et al., 2010].

In possibilistic logic there are two ways of interpreting possibilistic distributions associated to a set of possibilistic logic formulas. These two interpretations allow to characterize preferences according to two different readings: a *negative* and a *positive*.

The negative reading reflects what is not (fully) impossible and, consequently, remains potentially possible. It induces (prioritized) constraints delimiting the potentially satisfactory choices and it amounts to translate preferences into a set of more or less imperative goals [Benferhat et al., 2001]. Negative preferences can be encoded by necessity-based possibilistic logic formulas. Indeed, a possibilistic formula (φ, α) encodes $N(\varphi) \geq \alpha$, which is equivalent to $\Pi(\neg\varphi) \leq 1 - \alpha$, and thus reflects the impossibility of $\neg\varphi$, which is all the stronger as α is high. Therefore, a preference statement such as *I prefer tea to coffee* can be represented by the necessity-based possibilistic base $\Sigma = \{(tea \vee coffee, 1), (tea, 1 - \alpha)\}$ with $\alpha < 1$. Namely, Σ states that *tea* is somewhat imperative and that $tea \vee coffee$ is still more imperative (in fact is compulsory, assuming that the choice is between *tea* and $coffee$). Please observe that the preferences are expressed negatively: *nothing is possible outside* $coffee$ *or tea* and *nothing is strongly possible outside tea*. This reading correspond to the standard possibilistic logic at representation level.

The positive reading expresses what is really desirable and it is encoded by a new type of formula based on a set function called *guaranteed possibility measure* (Δ) [Dubois and Prade, 1992]. Positive preferences can be encoded by guaranteed-based possibilistic logic formulas. Therefore, the preference statement above is represented by the guaranteed-based (Δ) base $\Sigma' = \{[tea, 1], [\neg tea \wedge coffee, \alpha]\}$. Namely, Σ' states that having *tea* is fully satisfactory, and that having $coffee$ instead is less satisfactory. This reading corresponds to the guaranteed possibilistic logic at representation level. Since the guaranteed based representation is not based on classical possibilistic logic formulas, it must be translated into a necessity-based representation before it can be properly handled in possibilistic logic [Benferhat et al., 2002]. The guaranteed preference representation has shown to be equivalent to QCL.

When positive and negative preferences are used together, the representation capabilities of possibilistic logic can be enlarged in the bipolar possibilistic setting [Benferhat et al., 2006]. In this setting, what is rejected or considered unacceptable and what is desired are matters of degrees handled by two distinct possibilistic logic distributions.



(a) The CP-net (b) The induced preference graph

Fig. 6.1. Evening Dressing Example [Boutilier et al., 2004]

Possibilistic logic can be also used to encode conditional preference statements such as in *context c, a is preferred to b*. This amounts to encode a conditional preference by means of a pair of possibilistic formulas $\{(\neg c \vee a \vee b, 1), (\neg c \vee a, 1-\alpha)\}$ [Dubois and Prade, 2004]. Such modeling has been used in [Kaci and Prade, 2007, 2008] for representing preferences in the approximation of CP-nets and in [Hadjali et al., 2008] for representing database preferences queries using symbolic weights. Recently, the approach to database preferences queries has been extended to the case in which flexible queries and uncertain databases are considered [Bosc et al., 2010].

6.3 User Preferences in Conditional Preference Networks

Conditional Preference Networks (CP-nets) [Boutilier et al., 2004] are a graphical representation format for modeling conditional preferences. CP-nets are based on the expression of conditional *ceteris paribus* preference statements, *i.e.*, everything else being equal. With CP-nets, the user describes how his/her preferences over the values of one variable depend on the value of other variables. Given a set of attributes $V = \{X_1, \ldots, X_n\}$, for each attribute X_i, a set of parent attributes $Pa(X_i)$ is identified. For each value of $Pa(X_i)$ the user specifies a preference order over possible values of X_i, all other things being equal.

Formally, a CP-net N over the set of attributes $V = \{X_1, \ldots, X_n\}$ is a directed graph in which nodes represent attributes and edges express preference links between them. Each node is associated with a conditional preference table $CPT(X_i)$ that associates a strict partial order ($\succ_{CP}$) with each possible instantiation of the parents of X_i. A complete preference ordering satisfies a CP-net N if and only if it satisfies each conditional preference expressed in N. In this case, the preference ordering is said to be consistent with N. To illustrate these concepts, let us consider the following example.

Example 6.2. [Boutilier et al., 2004] A CP-net example is shown in Figure 6.1(a). As it can be seen, black to white is unconditionally preferred as a color for both the jacket (J) and

the pants (P), while preference between the red and white shirts (S) is conditioned on the combination of jacket and pants. Otherwise, if the jacket and the pants are of different colors, then a red shirt is preferred.

Another graphical representation of a CP-net is the *induced preference graph* (Figure 6.1(b)). An induced preference graph is a directed graph, in which the set of all nodes is the set of all outcomes. Then, an arc from outcome o_1 to outcome o_2 indicates that o_2 is preferred over o_1 (which can directly be determined from the preference table). The top elements of the preference graph are the worst outcomes and the bottom elements are the best outcomes.

Continuing with the above example, there exist 8 tuples corresponding to the specified preferences: *bbr*, *bbw*, *bwr*, *bww*, *wbr*, *wbw*, *wwr*, *www*. The preferences associated to the induced CP-net, applying the *ceteris paribus principle*, are: $bPS \succ_{CP} wPS, \forall P \in \{b, \neg b\}$, $\forall S \in \{r, \neg r\}$; $JbS \succ_{CP} J\neg bS, \forall J \in \{b, \neg b\}, \forall S \in \{r, \neg r\}$; $bbr \succ_{CP} bbw$, $wwr \succ_{CP} www$, $bww \succ_{CP} bwr$, $wbw \succ_{CP} wbr$.

Then, the following partial ordering holds under the *ceteris paribus* assumption: $bbr \succ_{CP} bbw \succ_{CP} bww \succ_{CP} bwr \succ_{CP} wwr \succ_{CP} www$; $bbr \succ_{CP} bbw \succ_{CP} wbw \succ_{CP} wbr \succ_{CP} wwr \succ_{CP} www$.

CP-nets have also been proposed for database preference queries in which preference are represented by a binary relation over a relation schema [Brafman and Domshlak, 2004]. As pointed out in [Brafman and Domshlak, 2004], database preference queries can be interpreted according to two semantics: the totalitarian semantics and the ceteris paribus semantics.[1] Since preference queries generally consist of several preference statements, the different preference orders have to be composed into a single preference order. Several proposal have been made in the database community [Chomicki, 2003] in which a totalitarian interpretation with intersection composition appears to be the implicit choice. Since this interpretation is semantically inappropriate and yields unintuitive or empty results even for very simple queries, in [Brafman and Domshlak, 2004], an approach based on *ceteris paribus semantics* and union-based composition was proposed (see [Brafman and Domshlak, 2004] for details).

Finally, several works improve CP-nets in different directions. TCP-nets [Brafman et al., 2006], for instance, enrich CP-nets by allowing the expression of relative importance statements between variables. CP-theories [Wilson, 2004] are still more general; they allow conditional preference statements on the values of a variable, together with a set of variables that are allowed to vary when interpreting the preference statement. Conditional Importance Networks (CI-nets) [Bouveret et al., 2009] express preferences between sets of goods.

6.4 User Preferences in Logic Programming

Although QCL, Possibilistic Logic, and CP-nets own some nice features w.r.t. preference handling, they seem to miss an important aspect which is commonly encountered in knowl-

[1] The totalitarian semantics does not assume everything else being equal, *i.e.,* it does not fix the values of the attributes not involved in a preference relation (as it happens under ceteris paribus semantics).

edge modeling: the representation of incomplete information. Incomplete information can be captured in logic programming approaches by means of negation as failure. In this section, we overview some representative approaches able to represent preferences which both depend on contextual and incomplete information.

6.4.1 Logic Programs with Ordered Disjunction

LPODs is the logic programming incarnation of QCL. Indeed, the $\times$ connective can be used to capture context-dependent user preferences. For a full overview about LPODs syntax and semantics, we refer to Chapter 2, Section 2.2.4.

LPODs is a useful specification to encode user preferences as shown in the following example.

Example 6.3. [Brewka, 2002] Let us consider the user preferences about a menu. The menu can consists of a $soup$ or a $salad$ as $starter$, $fish$, $beef$ or $lasagne$ as main $course$, $icecoffee$ or $tiramisu$ for $dessert$, and red, $white$ wine or $water$ as $beverage$. Then, the user has the following preferences: (i) as a $starter$ she prefers $soup$ over $salad$; (ii) as $main$ course she prefers $fish$ over $beef$ over $lasagne$; (iii) for $beef$ she prefers red to $white$ wine over $water$, otherwise, she prefers $white$ over red wine; (iv) if she has $tiramisu$, then she wants an extra $coffee$. Finally, (v) for $coffee$ she prefers $espresso$ over $cappuccino$.

User preferences in the menu example can be modeled in LPODs by:

$$P = \left\{ \begin{array}{rcl} soup \times salad & \leftarrow & starter. \\ fish \times beef \times lasagne & \leftarrow & main. \\ red \times white \times water & \leftarrow & beverage, beef. \\ white \times red \times water & \leftarrow & beverage, not\ beef. \\ espresso \times cappuccino & \leftarrow & coffee. \\ \neg vegetarian & \leftarrow & beef. \\ \neg vegetarian & \leftarrow & fish. \\ \neg vegetarian & \leftarrow & soup. \\ alcohol & \leftarrow & white. \\ alcohol & \leftarrow & red. \\ coffee & \leftarrow & tiramisu. \\ icecoffee & \leftarrow & dessert, not\ tiramisu. \\ tiramisu & \leftarrow & dessert, not\ icecoffee. \end{array} \right\}$$

Given a description of the case at hand, *e.g.*, whether the user is vegetarian or not, drinks alcohol or not, likes fish *etc*, the preferred answer sets will determine a menu which satisfies user preferences as much as possible. Let us consider the following contextual knowledge C:

$$C = \left\{ \begin{array}{rcl} \neg vegetarian & \leftarrow & \top. \\ starter & \leftarrow & \top. \\ main & \leftarrow & \top. \\ dessert & \leftarrow & \top. \\ beverage & \leftarrow & \top. \end{array} \right\}$$

It can be checked that the program $P \cup C$ has 54 answer sets. Among them $\{beverage, dessert, main, starter, \neg vegetarian, fish, soup, icecoffee, alcohol, white\}$ and $\{beverage, dessert, main, starter, \neg vegetarian, fish, soup, tiramisu, alcohol, white, coffee, espresso\}$ are the most preferred options for the user menu.

The LPODs syntax only allows to specify elementary preference expressions. For this reason an extension able to consider disjunction in the head of rules has been proposed in [Kärger et al., 2008].

6.4.2 Logic Programs with Ordered and Unordered Disjunction

Logic Programs with Ordered and Unordered Disjunction (DLPODs) is an LPODs extension proposed by Kärger *et al.* [Kärger et al., 2008] which is able to deal with ordinary disjunction $\vee$ in the head of rules. DLPODs is able to specify preference equalities statements such as

$$pub \times (cinema \vee tv) \leftarrow not\ sunny$$

The intuition behind this expression is that *pub* is the most preferred option and, in case *pub* is not possible, both *cinema* and *tv* are equally preferred.

Generally speaking, a DLPOD is a set of rules of the form $h \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_{m+n}$ in which h is an ordered disjunctive term defined as any arbitrary combination of literals and $\vee$, $\times$ connectives, and b_j's ($1 \leq j \leq m+n$) are literals. Then, the DLPOD semantics is defined in terms of disjunctive split programs on the ordered disjunctive normal form (ODNF) of an DLPOD, *i.e.,* when all h's are under the form

$$\vee_{i=1}^{n} \times_{j=1}^{m_i} = (C_{1,1} \times \ldots \times C_{1,m_1}) \vee \ldots \vee (C_{n,1} \times \ldots \times C_{n,m_n}). \tag{6.1}$$

$(C_{i,1} \times \ldots \times C_{i,k_i})$ is the i-th Ordered Disjunct of the ODNF. The ODNF can be obtained by exhaustive application of the following rewriting rules:

(i): $a \times (b \vee c) = (a \times b) \vee (a \times c)$
(ii): $(a \vee b) \times c = (a \times c) \vee (b \times c)$
(iii): $a \times (b \times c) = a \times b \times c$
(iv): $(a \times b) \times c = a \times b \times c$

Example 6.4. For instance, $pub \times (cinema \vee tv) \leftarrow not\ sunny$ yields to the following ODNF

$$(pub \times cinema) \vee (pub \times tv) \leftarrow not\ sunny$$

A split program of a DLPOD P is defined as the result of replacing each rule r of P by one of its option. Similarly to what happens in LPODs, a set of literals M is an answer set of a DLPOD P if M is an answer set of some split program P' of P. Optimal answer sets of DLPODs are characterized by a *satisfaction degree vector* defined as the measure of how much an answer set satisfies a rule. Intuitively, to each Ordered Disjunct a penalty is associated representing how much the answer set satisfies the disjunct. For each rule, these penalties build up a vector $D = (d_1, \ldots, d_n)$ of degree values, one dimension for

each disjunct, where each d_i is either a natural number or a constant ϵ. The ϵ denotes that a particular disjunct does not tell anything about how much an answer set is preferred. Similarly to LPODs, the best satisfaction degree is assigned (*i.e.*, $D = (1, \ldots, 1)$) in case the body of a rule is not satisfied.

Example 6.5. The ODNF rule $(pub \times cinema) \vee (pub \times tv) \leftarrow not\ sunny$ has four disjunctive split programs. Valid answer sets are $\{pub\}$, $\{tv\}$, $\{cinema\}$. The satisfaction vectors associated to each of the answer sets are $(1, 1)$, $(\epsilon, 2)$, and $(2, \epsilon)$ respectively.

The preference relation between answer sets is defined according to a Pareto-based ordering. First, for each rule r in the program it is checked whether an answer set M_1 Pareto-dominates an answer set M_2 (comparing the satisfaction vectors $d_{M_1}(r)$ and $d_{M_2}(r)$). Then, the comparison is generalized to the whole program.

Example 6.6. Given the ODNF rule $(pub \times cinema) \vee (pub \times tv) \leftarrow not\ sunny$ and the three answer sets $M_1 = \{pub\}$, $M_2 = \{tv\}$, $M_3 = \{cinema\}$, M_1 Pareto dominates M_2 and M_3, while M_2 and M_3 are incomparable.

In [Kärger et al., 2008], an implementation design is also presented which follows the ideas behind the LPODs implementation.

6.4.3 Answer Set Optimization Programs

Answer Set Optimization (ASO) [Brewka et al., 2003] is a decoupled approach to preference representation. Rather than specifying a preference relation among literals in a single logic program like in LPODs, ASO uses two different programs. Formally, ASO is a pair $\langle P_{gen}, P_{pref} \rangle$, where P_{gen} is a logic program, called the *generator program*, and P_{pref} is a collection of preference rules, called the *preference program*. Each preference rule r in the preference program is of the form

$$C_1 > \ldots > C_k \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_{m+n} \tag{6.2}$$

in which the C_i are boolean combinations built from a set of literals $\mathcal{L}$ by means of disjunction, conjunction, and negation as failure, with the restriction that strong negation is allowed to appear only in front of atoms and negation as failure only in front of literals.

Given a preference rule r and an outcome M, the *satisfaction degree* $v_M(r)$ of r in M is defined to be i if the body of r is satisfied by M, some ranking condition C_i is satisfied and i is the minimum index among the satisfied C_i. Otherwise, the degree is I (irrelevant). The satisfaction degrees are ordered in a pre-order $\geq$ where values I and 1 are equally good but better than all other degrees i for which $i \geq i + 1$ hold. For a set of preference rules $\{r_1, \ldots, r_n\}$, an answer set M induces a satisfaction vector $(v_M(r_1), \ldots, v_M(r_n))$ in which $v_M(r_i)$ is the satisfaction degree of r_i in M. Answer sets are ordered as follows: $M_1 \geq M_2$ if $v_{M_1}(r_i) \geq v_{M_2}(r_i)$, for each i, $1 \leq i \leq n$, and $M_1 > M_2$ if $M_1 \geq M_2$, and for some i, $1 \leq i \leq n$, $v_{M_1}(r_i) > v_{M_2}(r_i)$. An answer set M is optimal if there is no answer set M' such that $M' > M$.

As pointed out in [Brewka et al., 2003], an advantage of ASO programs w.r.t. LPODs is that the answer set generation and the answer set comparison are decoupled. This decoupling splits knowledge generation from preference specification and facilitates preference elicitation.

Example 6.7. Let us represent the preference program in Example 6.3 by means of an ASO. The ASO will consist of two programs.[2]

$$P_{gen} = \left\{ \begin{array}{rcl} 1\{soup, salad\}1 & \leftarrow & starter. \\ 1\{fish, beef, lasagne\}1 & \leftarrow & main. \\ 1\{red, white, water\}1 & \leftarrow & beverage. \\ 1\{espresso, cappuccino\}1 & \leftarrow & coffee. \\ 1\{icecoffee, tiramisu\}1 & \leftarrow & dessert. \\ \neg vegetarian & \leftarrow & fish. \\ \neg vegetarian & \leftarrow & soup. \\ alcohol & \leftarrow & white. \\ alcohol & \leftarrow & red. \\ coffee & \leftarrow & tiramisu. \\ \neg vegeterian & \leftarrow & \top. \\ starter & \leftarrow & \top. \\ main & \leftarrow & \top. \\ dessert & \leftarrow & \top. \\ beverage & \leftarrow & \top. \end{array} \right\}$$

$$P_{pref} = \left\{ \begin{array}{rcl} soup > salad & \leftarrow & starter. \\ fish > beef > lasagne & \leftarrow & main. \\ red > white > water & \leftarrow & beverage, beef. \\ white > red > water & \leftarrow & beverage, not\ beef. \\ espresso > cappuccino & \leftarrow & coffee. \end{array} \right\}$$

It can be checked that the P_{gen} generates 54 answer sets and P_{pref} induces an order over them which is consistent with the ordering obtained in LPODs.

ASO programs have also been combined with CP-nets [Brewka et al., 2005], in which techniques from CP-nets are used to compute the most preferred answer sets.

6.5 Context-aware Systems

Context-aware systems [Baldauf et al., 2007] are ubiquitous information systems which are able to adapt their operations to the current context without explicit user intervention. In this way, they aim at increasing usability and effectiveness by taking context into account.

[2] $1\{soup, salad\}1$ is a cardinality constraint. A cardinality constraint is an atom of the form $n\{l_1, \ldots, l_k\}m$ with the semantics that at least n and at most m of the literals l_i's are true. Cardinality constraint can be used to generate answer sets. Although rules built from cardinality constraints can, in principle, be replaced by sets of rules with negation as failure, they make problem specifications much more concise and readable. For the precise definitions we refer to [Simons et al., 2002].

In information systems, personalization focuses on collecting, modeling, and adapting *user profiles* [Adomavicius and Tuzhilin, 2005]. In context-aware systems, the personalization must also take *contextual information* into account. In fact, an effective personalization can be achieved only if the system is aware of user preferences and context. If this is not the case, the system will not be able to distinguish users as individuals and, usually, it will provide the same results to users with different characteristics and in different contexts. In general, personalized systems represent features of the user as an individual. On the other hand, mobile and ubiquitous systems, in which the context is essential, need to be *context-aware*.

6.5.1 User Profile

The *user profile* (also called user model) is the distinctive feature of personalized information systems as it is essential for the system to behave differently for different users. The typical user features modeled, which mainly depends on the application domain, are [Codina, 2009; Palau, 2010]:

- *user knowledge*: it represents the expertise level of the user in a specific subject or domain. This user feature tends to be the used one in educational and hypermedia systems;
- *user interests*:[3] they constitute the most important part of the user profile in information retrieval and filtering systems that deal with large volumes of information;
- *user goals or needs*: they represent the immediate purpose for a user task. Depending on the type of the system, they can be an immediate information need (in information access systems) or a learning goal (in educational systems);
- *user background*: it represents user experience. Background information is usually used for content adaptation, although there are examples of its use within search and navigation support;
- *user individual trait*: it defines the user as an individual. Examples are personality traits (*e.g.,* introvert/extravert, *etc*), cognitive styles, cognitive factors (*e.g.,* working memory capacity, *etc*) and learning styles. Similar to user background, individual traits are stable features of a user that either cannot be changed at all or can be changed only over a long period of time.

These features may be collected from user interactions in an *explicit* or *implicit way*.

On one hand, *explicit user information collection* relies on inputs from the users. Two typical methods to capture explicit feedback are: via Web forms, in which the users can provide personal and demographic information such as birthday, current job, preferences or personal data; and via ratings, which allow users to express their opinions by selecting a value from a range. Processing explicit feedback costs time and relies on user willing to provide personal information. Therefore, the tendency is to simplify them by inheriting user profiles from social-networks or by using simple like/dislike buttons. In addition, stereotypes based on generalizations about communities of users are used to model new users.

[3] Due to the characteristics of the type of context-aware system we will target, user interests are the only feature modeled, as we will see in Chapter 7.

On the other hand, *implicit user information collection* is based on usage data of the users. From this data, the system tries to predict user interests by taking implicit indicators associated to specific patterns of user behavior into account. The typical heuristic indicators used by implicit user modeling methods are the time spent viewing a specific item, the frequency of item selection, and if the item is consumed or acquired. The advantage of implicit feedback is that it does not require any additional intervention by the user during the user modeling process. However, one drawback of implicit feedback techniques is that they can typically only capture positive feedback. When a user clicks on an item, it seems reasonable to assume that this indicates some degree of preference in the item; however, it is not as clear, when a user fails to examine some data item, that this is an indication of disinterest.

The adaptation of such profiles is an essential requirement for personalized systems that need to adjust to changes quickly in order to keep user preferences up-to-date. Profile updating can be done automatically or manually. Although automatic methods are less intrusive for the end user, in most personalized systems, adaptation is achieved through user feedback. The main disadvantage of this adaptation method is that old preferences are not forgotten. This can cause not only an exponential growth of the user profile, but also a decrease in precision by considering old preferences.

6.5.2 Context

The context can be defined as the information that characterizes the environment in which an application or service runs. While user profiles contain information about user characteristics, which generally are assumed to vary slowly over time, the context of a user, in contrast, assesses the status of the person at a particular situation (*i.e.,* a profile can be applied in different contexts). There are many classifications of context (as many as definitions). One possible categorization, depending on who applies, is [Greenfield, 2006; Park et al., 2009]:

- *individual context*: contextual information that relates to or describes the environment of a particular person or entity;
- *group context*: information extracted from a group of people (a family, passengers on a train, a group of friends, people attending a conference, *etc*). It is not the mere aggregation of individual contexts, as it is necessary to consider what context properties spread to the entire group, and whether the existence of the group causes the appearance of new properties which do not apply individually;
- *collective context*: the context differs from group to large-scale aggregation. Application contexts are large populations, and in them the important thing is to determine what features (with the margin of error considered) are generalizable to the populations analyzed.

Another possible classification is by the source of context:

- *spatial context*: it consists of the physical location of the user;
- *temporal context*: it consists of the date and current time the user is interacting with the system;

- *environmental context*: it includes weather conditions, *e.g.,* temperature, if it is sunny, if it rains, *etc*;
- *personal context*: it refers to the activity the user is performing while he/she is interacting with the system, *e.g.,* working, reading, *etc*.

In the next chapter, we will see how some of these sources of context, in particular, *current time*, *date*, *weather* and *location*, can be used in combination with user preferences to build user profiles in terms of logic programming rules.

6.6 Discussion

In this chapter, we have overviewed different approaches that support the specification of a compact preference model and the achievement of an order among the outcomes of the model. We have seen that the way in which preferences are expressed and the preference order is achieved depend on the formalism chosen.

Although QCL, possibilistic logic and CP-nets own some nice features w.r.t. preference handling, they seem to miss an important aspect which is commonly encountered when modeling knowledge: the representation of incomplete information. In this respect, logic programming approaches based on answer set semantics such as LPODs [Brewka, 2002; Brewka et al., 2004b], DLPODs [Kärger et al., 2008], and ASOs [Brewka et al., 2003] allow to express context-dependent preferences and incomplete knowledge by means of negation as failure.

On the other hand, context-aware systems start to play an important role nowadays. They offer users an easy access to big amount of information. However, these systems usually do not distinguish users as individuals and similar activities are suggested to users with different characteristics. As such, it results impossible for the system to satisfy users who are typically looking for the most appropriate suggestions according to their *preferences* in a specific *context*.

One concrete way to achieve an effective personalization is by means of *user profiles*. A user profile consists of a set of user preferences and contextual information. Indeed, in a context-aware system, user preferences depend on contextual information, *i.e.,* the position where the user is located, the time the user is making the petition to the system, weather conditions, *etc*. Therefore, contextual information is important because a different set of preferences may apply in different contexts. More generally, in the modeling of user preferences in such systems, several issues related to knowledge representation arise:

- preferences can depend both on contextual and incomplete information,
- preferences in a given context can be more important than others, and
- the specification of the preference model should be compact enough and fast to process.

As discussed at the end of Chapter 5, LPPODs is a suitable logic programming specification for representing context-dependent preferences, incomplete knowledge and to associate preferences with importance weights. In Chapter 7, we describe a use case related to a context-aware system in which we employ LPPODs for modeling user profiles.

Another important aspect in the modeling of user preferences is to have flexibility in capturing preference statements. For instance, in LPPODs, only singleton preference literals can be specified in each preference rule. However, there are cases in which more complex expressions need to be specified. Preferences statements, expressing equalities (*e.g., I prefer cinema to pub or tv*) such as in DLPODs, or more generic boolean combinations, such as in ASO programs, may allow to capture richer preferences rules, but they still impose some syntactical restrictions on the way in which preference rules can be formulated. As such, in Chapter 8, we propose an extension of LPODs which supports the writing of nested preferences expressions built by means of connectives $\{\vee, \wedge, \neg, not, \times\}$.

7

Handling User Preferences in a Context-Aware System

The appearance of ubiquitous systems has made it possible to convey information from different sources and to make it accessible to everyone. One field in the wide range of ubiquitous systems are the so-called context-aware systems (see Section 6.5). A new trend on context-aware systems are Interactive Community Displays (ICDs).

ICDs are multimedia information points that offer interactive services on the public thoroughfare in order to provide information to people living in or visiting a city [Ceccaroni et al., 2009]. Some examples of ICDs are the *i-kiosks*[1] and *i+*[2] (respectively located in Aberdeen and Bristol). More recently, many other initiatives have been deployed in commercial malls and other public spaces such as *Punts BCN*[3], a Barcelona city council initiative to offer information public services.

However, these services are often isolated or designed with predefined static sources and they do not actually exploit the benefits offered by the *Internet of Things* [Tselentis et al., 2009]. Furthermore, they suffer from some limitations [Gómez-Sebastià et al., 2009; Palau et al., 2010].

On one hand, these services are supported by a manually-controlled infrastructure in order to manage and convey information provided by (internal and external) Web services that may fail or change at any time. To address this limitation, the ALIVE[4] methodology and framework have been recently considered. In this way, it has been possible to design and implement a robust infrastructure which is able to compose and coordinate ICDs' Web services dynamically (since we do not explicitly deal with these issues here we refer the interested reader to [Gómez-Sebastià et al., 2009; Palau et al., 2010]).

On the other hand, ICDs usually do not distinguish users as individuals and similar information is suggested to users with different characteristics. As such, these systems are

[1] `http://www.aberdeencity.gov.uk/regeneration/ikiosk/reg_ikiosks.asp`

[2] `http://www.bristollegiblecity.info/projects/21/21.html`

[3] `http://w3.bcn.es/fitxers/premsa/dppuntsbarcelona.126.pdf`

[4] The ALIVE framework [Lam et al., 2009] combines Model Driven Development (MDD) with coordination and organizational mechanisms [Vázquez-Salceda et al., 2010], providing support for highly dynamic and open systems of services. The ALIVE framework extends current trends in engineering by defining three levels in the design and management of distributed systems: a Service Level, a Coordination Level and an Organization Level. An essential distinction between the ALIVE approach and existing ones is that it provides an organizational context (such as, for instance, objectives, structures and regulations) that can be used to select, compose and invoke services dynamically. Thus, (agentified) services are organization-aware services able to coordinate with others according to a given organization and coordination model.

unable to filter available information in a personalized way and to satisfy users who are typically looking for the most appropriate suggestions according to their *preferences* in a specific *context*. Moreover, the system has to be able to filter and adapt big amount of information. These issues suggest that an enhancement of actual ICDs with a mechanism for handling user preferences can be valuable.

A possible way to create such mechanism is by means of the logic programming framework of LPPODs.[5] The use of LPPODs for modeling user preferences has already been motivated (see Section 1.4.2 and Section 5.7). First, it can represent preferences in a compact way and it can capture contextual and incomplete information. Secondly, it can measure preference importance by means of weights associated to preference rules. Thirdly, it can achieve an order between the outcomes of the preference model. Finally, it offers a practical way to compute the user preference model (indeed, we provide an implementation of the LPPODs framework).

In this chapter, we describe how LPPODs can be used to enhance the personalization capabilities of an ICD. To this end, we employ LPPODs to model user profiles and to generate, in a given context, the most preferred outcomes to be taken into account in the content filtering. The results of this chapter have been settled in the context of a use case targeted in the ALIVE EU project.[6].

The rest of the chapter is organized as follows. After providing a brief description of the considered use case and its requirements (Section 7.1), in Section 7.2, we design an abstract user profile adaptation process and, based on that, we motivate the need for context-dependent preferences and preference weights. Then, in Sections 7.3-7.4, we propose a knowledge level approach for representing context-dependent preferences by means of an ontology. Section 7.5 presents some user profiles we have designed and integrated in the system. Section 7.6 describes the LPPODs solver implementation and discusses its use. Section 7.7 provides some hints about how personalized suggestions can be retrieved using the solver outputs. Section 7.8 discusses some relations with other works in the literature. Finally, Section 7.9 concludes the chapter.

7.1 Use Case and Requirements

Throughout this chapter we will use a typical ICD usage scenario. The scenario starts when a user interacts with the systems interface (the ICD). At this point, the user logs in the system and his/her profile is loaded. If the user profile does not exist, *i.e.,* the user has just registered for the first time, a predefined profile filled with default preferences is loaded. The system is now ready to presents an initial interface composed by several activities, such as cinemas, restaurants and night venues. When the user requests for suggestions related with one of these activities, the system reasons about the most appropriate user preferences for that specific context. Contextual information consists of — but, it is not limited to — *current time*, *date*, *weather* and *location*. Context-dependent preferences related to different activities

[5] An LPPOD consists of context-dependent preference rules of the form: $\alpha : c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_{m+n}$ (see Chapter 5).

[6] `http://www.ist-alive.eu/`

can have different importance. For instance, some preferences can be requirements, while other preferences can be less mandatory and they can have different priorities. To this end, preferences are associated with a weight to specify the preference importance. User preferences are used to filter available content, *i.e.*, information about available venues is gathered and displayed on a map together with basic information, such as a brief description, address and pictures, *etc*. Once suggestions have been provided, the user can express some feedback over them. This can be done in an explicit way (explicit feedback), by evaluating the quality of the suggestions received, or in an implicit way (implicit feedback), by showing more interests in some type of venues (*e.g.*, selecting several cultural suggestions). This feedback is processed and the preferences are updated according to each context and domain. For instance, a negative feedback in a recommendation about a restaurant venue for lunch, will trigger a lowering in the weight of the corresponding preference in the profile. When the weight is very low or equal to 0, such preferences become obsolete.

From the aforementioned use-case, a list of requirements the system should be able to fulfill has been compiled. In particular:

R_1: *addressing the cold-start problem*: when the user interacts with the system for the first time, the system should be able to create a predefined profile;
R_2: *weighted context-dependent preferences*: the system should be able to represent weighted context-dependent preferences in a compact way and obtain an ordering among them;
R_3: *dynamic recommendations*: the list of suggestions based on contextual preferences can change over the time due to the change of interests of the user;
R_4: *user feedback*: user feedback should be taken into account in order to keep the user profile up-to-date;
R_5: *domain independency*: the system should be flexible enough to allow to extend preference types and content information in an easy way;
R_6: *preference representation independency*: the preference handling mechanism should be flexible enough to allow its replacement in an easy way if needed.

7.2 User Profile Adaptation Process and Context-Dependent Preferences

To address requirements R_1, R_2, R_3 and R_4, a multi-stage user profile adaptation process has been designed (Figure 7.1).

The user profile adaptation process follows four sequential phases each time a user interacts with the system. The *Profile Manager* deals with the representation of user preferences. At the beginning, an initial user profile is created from a user group. The profile is stored according to a *profile ontology* which contains a list of weighted contextual preference relations. New user profiles are created by inheriting preferences from predefined user groups. As next, the abstract user profile representation is converted in a processable format for the *Preference Reasoner*. This module collects the current context and reasons about context-aware preference rules. The reasoning generates an ordered list of context-aware preference items. Preference items are processed by the *Content Manager* which is in charge of filtering the appropriate content to be provided. Finally, once the suggestions are pro-

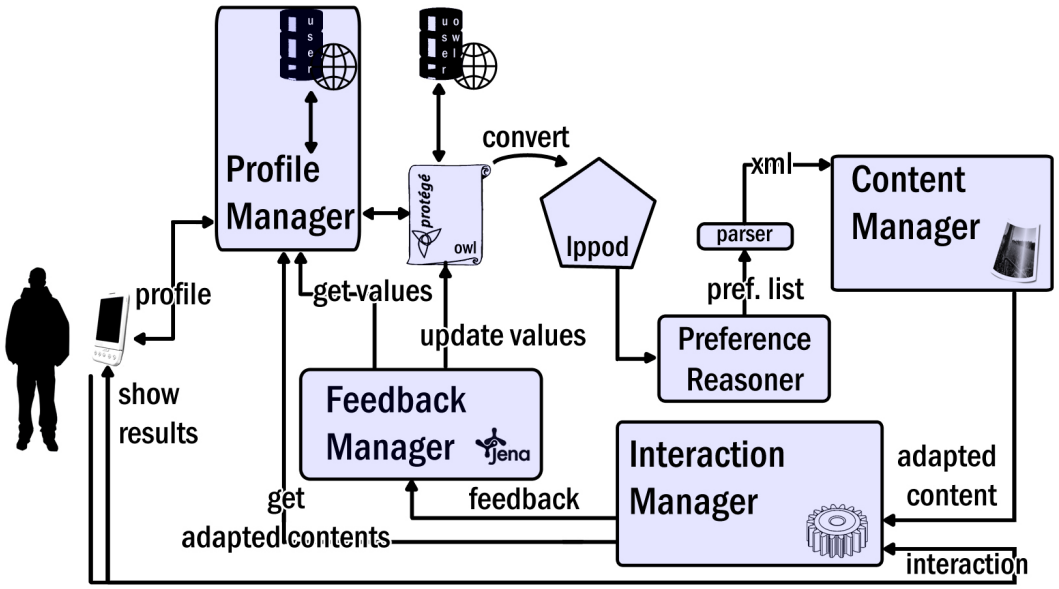


Fig. 7.1. Overview of User Profile Adaptation Process

vided, the user can express some feedback that is processed by the *Feedback Manager* to update the user profile.

The adaptation is concerned with several issues typically encountered in the representation of user profiles, such as *cold-start problem*, *weighted context-dependent preference* representation, *content filtering* and *user feedback*. Despite all these issues, here we will only focus on modeling and reasoning about *weighted context-dependent preferences*, *i.e.*, we will specially show how to deal with R_2.

In order to represent context-dependent preferences and weights, we will adopt the logic programming framework of LPPODs. In fact, as discussed in Chapter 5, by interpreting LPPODs framework in terms of user profiles and user preferences, it can be noticed how such framework can support:

- the specification of context-dependent preferences by means of ordered disjunction rules;
- the specification of a weight to each context-dependent preference rule;
- the reasoning about user preferences to obtain an order among them.

Therefore, LPPODs can be used as a specification for user context-dependent preferences associated with a weight.

To satisfy requirement R_5, domain knowledge in the application has been modeled by means of a *preference ontology* which describes the application domain (Section 7.3).[7,8]

[7] An ontology, in computer science, is a *formal, explicit specification of a shared conceptualization* [Gruber, 1993]. An ontology provides a shared vocabulary, which can be used to model a domain *i.e.*, the type of objects and/or concepts that exist, and their properties and relations.

[8] In fact, the use of ontologies has become a common practice in knowledge-based information systems in order to foster system reusability [Liu et al., 2007; Wang and Kong, 2007].

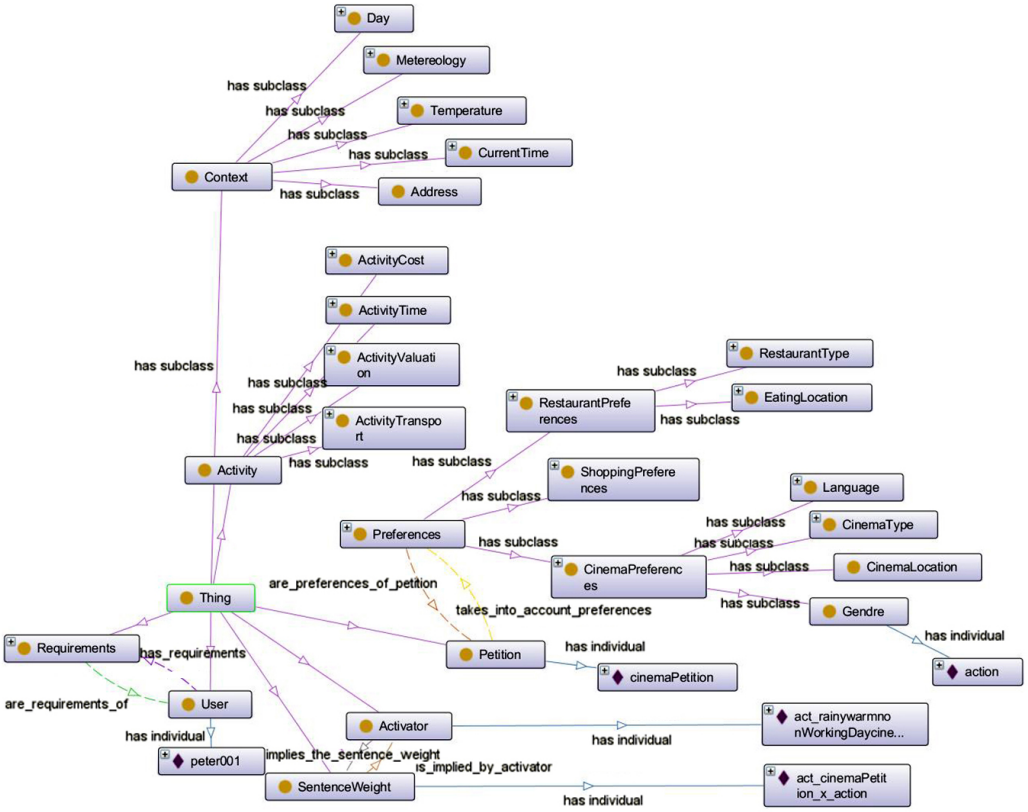


Fig. 7.2. Preference Ontology

On the other hand, R_6 imposes that the preference handling method chosen can be replaced by another mechanism (if necessary) in an easy way. This flexibility has been achieved reusing some concepts of Model-Driven Design (MDD).[9]

We have specified a *user preference* and *profile ontology* (Section 7.3) upon which a particular instance of a user profile can be created. Then, by the specification of an ontology for LPPODs and a transformation function it is possible to automatically translate the user profile abstract representation into the symbolic level representation of LPPODs (as we will see in Section 7.4).

[9] Model-Driven Design (MDD) refers to the systematic use of models as primary artifacts throughout the Software Engineering (SE) development process. The defining characteristic of MDD is the use of models to represent the important artifacts in a system [Nicolas et al., 2003]. Each of the models in a MDD system is constructed from a language specified in a meta-model, which captures the concepts and relationships of the language in a structured and regular form. In relation to these meta-models, the models can then be stored, manipulated and transformed to other models, and to implementation artifacts.

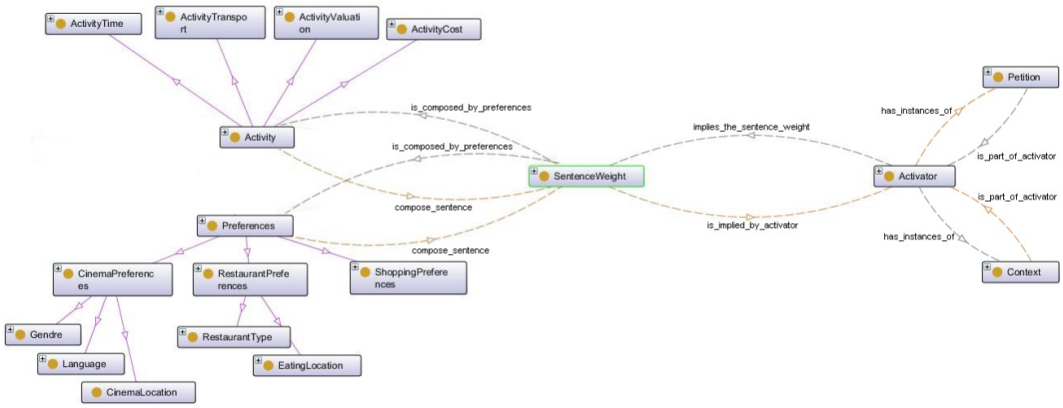


Fig. 7.3. Ontology for Rules containing User Preferences related to Context

7.3 User Preferences and Profile Ontology

The *preference ontology* developed with the Protégé Ontology Editor[10] is shown in Figure 7.2. The graph represents the taxonomy of the classes involved in the use case and their relations along with some individuals and various relations among users, their petitions and preferences.

On the other side, user profiles are modeled according to a *profile ontology* which represents the relation between contexts and preferences by means of preference rules (see Figure 7.3). In order to keep preferences independent from context, each preference rule is modeled involving several ontological concepts that contain a set of preferences and an activator.

The set of preferences of a rule is composed by several user preferences for a specific domain, and the activator of a rule is built by different contexts and a domain petition. The *SentenceWeight* links *Preferences*, *Activity* (*i.e.*, preferences without domain) and *Activator*, and they make it possible to represent preferences in different contexts and domains, along with the management of priority among preferences.

This representation abstractly represents context-aware preference rules which, at runtime, are converted into the format of the Preference Reasoner module.

7.4 Mapping User Profiles to LPPODs

As we do not want to stick to a particular formalism, language or technology for representing context-dependent preferences, we have defined a model describing how user profiles should be specified in a general way (Section 7.3). The advantage of having a meta-model

[10] `http://protege.stanford.edu/`

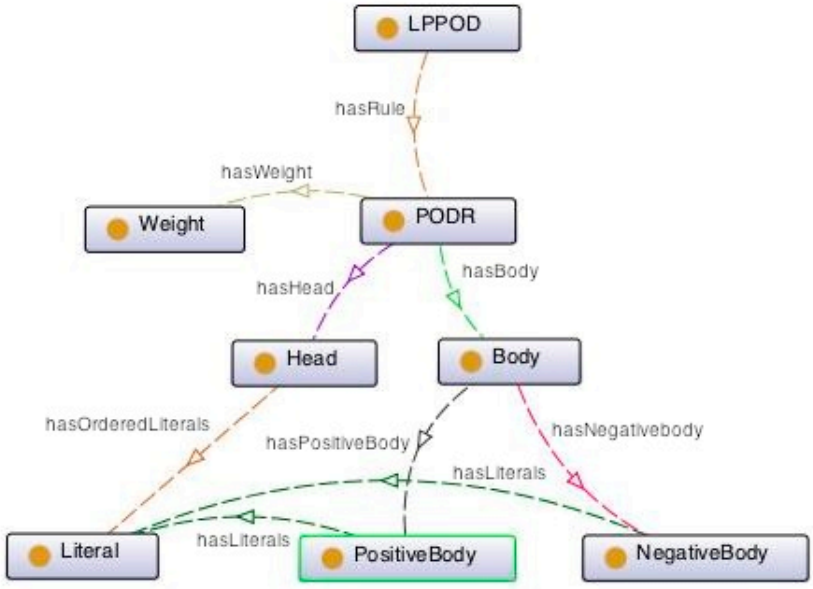


Fig. 7.4. Ontology for LPPODs

is to have a specification general enough which can be translated to other languages or formalisms through model transformations [Mellor et al., 2004]. Generally speaking, a model transformation takes as input a model, built according to a given meta-model and it produces as output another model conforming to another meta-model. The process of translating a model to another is specified by a mapping. The main advantage of this approach is that, from the same meta-model specification, several mappings to different models can be done through different mapping functions [Metzger, 2005].

We have defined a transformation t of a user profile UP specified according to the user profile ontology PO to an LPPOD P specified according to an LPPOD ontology $LPPOD$ as $t : UP_{PO} \rightarrow P_{LPPOD}$.

To be able to define the mapping, we first need an ontology which describes the formalism of LPPODs. A first proposal of an ontology for LPPODs is shown in Figure 7.4. An *LPPOD* consists of a finite set of *PODR*, *i.e.,* Possibilistic Ordered Disjunction Rules, associated with a *Weight*. Each rule has a *Head* and a *Body*, in which the head has an ordered set of literals and the body consists of a negative and a positive part.

According to MDD the mapping t is specified by a script which translates the corresponding elements of UP into elements of P (Figure 7.5). In this way, an LPPOD-based encoding is produced.

7.5 User Preference Representation

At symbolic level, user preferences are represented according to the syntax of LPPODs. Then, a user profile consists of a list of weighted context-dependent preferences and it is encoded by a set of possibilistic ordered disjunction rules (see Chapter 5 Section 5.1):

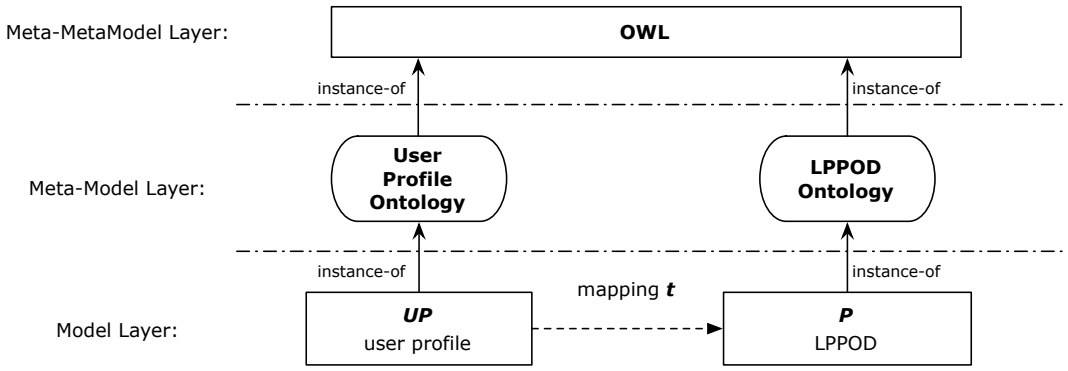


Fig. 7.5. MDD applied to User Preference Modeling

$$w_i : p_1 \times \ldots \times p_k \leftarrow b_1, \ldots, b_m, \; not \; b_{m+1}, \ldots, \; not \; b_{m+n}$$

in which p_i's ($1 \leq i \leq k$) are preference literals and each of the b_j ($1 \leq j \leq m+n$) is a context literal. w is an integer between $[0, 100]$ representing the importance of a rule r. For convention, we associate the left part of a rule r, $p_1 \times \ldots \times p_k$, to an ordered disjunction of preferences related with the activities discussed in the aforementioned scenario, with the reading if possible p_1, if p_1 is not possible, then p_2 and so on. We map the right part of a rule r, $b_1, \ldots, b_m, \; not \; b_{m+1}, \ldots, \; not \; b_{m+n}$, with the context information needed to activate the rule. In such a way, we are able to represent that the user has some preferences in a particular context. Please observe how we are able to capture incomplete knowledge by means of negation as failure.

Example 7.1. Let us consider some preferences a user can have about restaurants. The user prefers (i) *to eat in a Mediterranean restaurant rather than in a Vegetarian unless time is between 13 and 15*, otherwise she prefers (ii) *to eat in a Vegetarian restaurant rather than in a Mediterranean*. Moreover, she requires that (iii) *she prefers to pay by Master Card over using her Visa*. This simple user profile UP can be encoded as:

$$UP = \left\{ \begin{array}{c} r_1 = \mathbf{90} : type\_Mediterranean \times type\_Vegetarian \leftarrow pet\_Restaurant, not \; 13-15h. \\ r_2 = \mathbf{90} : type\_Vegetarian \times type\_Mediterranean \leftarrow pet\_Restaurant, 13-15h. \\ r_3 = \mathbf{100} : req\_MasterCard \times req\_Visa \leftarrow pet\_Restaurant. \end{array} \right\}$$

Taking the preference ontology presented in Section 7.3, user preferences are represented with the following information:

- *Location*: it represents spatial preferences. This class contains two subclasses, the district and the street;
- *Access*: it represents the user preferred (public) transports;
- *Cost*: it represents the maximum cost the user wants to assume;
- *Rate*: it represents the minimum valuation the user wants to accept;
- *Type*: it represents the preferred types within the given domain, *e.g.*, restaurants, cinema, and night venues;

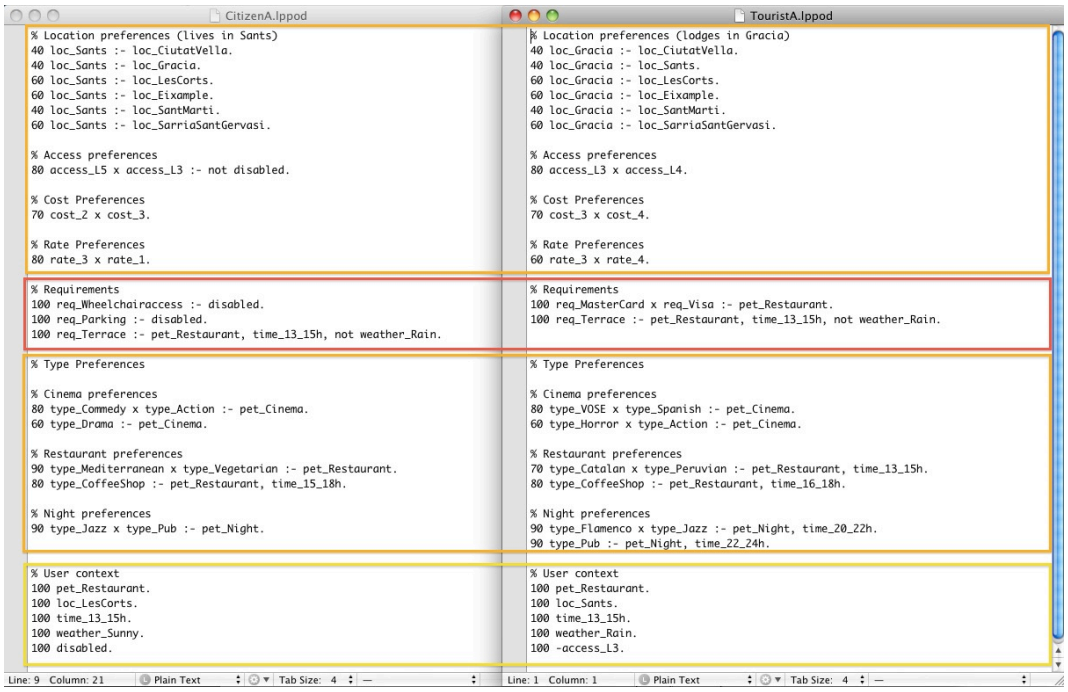


Fig. 7.6. Examples of User Profiles (*Citizen* and *Tourist*) encoded by LPPODs in which Red boxes denote *Requirements*, Orange boxes denote *Preferences* and Yellow boxes denote *Context*.

- *Requirements*: they represent the mandatory preferences of the user.

Generally speaking, LPPODs user profiles (see Figure 7.6) consider that the user may be lodging or living in a district, while the user interacts with the system from another district. Thus, proportional weights have been applied to the corresponding rules, obtaining as location preferences both the district where they are located and the district where they are staying (*e.g.*, home or hotel district). For instance, a *Citizen* living in *Sants*, and interacting with an ICD located in the *Les Corts* district, will have the following two location preferences: *Les Corts* and *Sants*.

Some *Access*, *Cost* and *Rate* preference rules may have empty bodies. In such a case, they are always considered. The ordered disjunction connector $\times$ is used to specify a preference order between several solutions. For instance, preferred access (public transport) for a *Tourist* are *L3* and *L4* metro lines. Instead, since *Citizen* is a disabled person, the rule about transport preferences will not be considered.

Requirements rules are considered as preferences with maximum weights, *i.e.*, 100, and generally higher than *Type*-based preference rules.

Type preference rules express general preference related to three types of activities the system is able to consider: *restaurants*, *cinema*, and *night venues*. Each preference type rule is associated with a weight corresponding to the priority of the preference rule. The weight allows us to give priority to the preference type themselves in order to look for suggestions or to filter preferences that do not satisfy a threshold. Moreover, these weights can be updated by an implicit or explicit user feedback. For instance, the weight of a rule can be increased or decreased when the user chooses or evaluates suggestions that are related

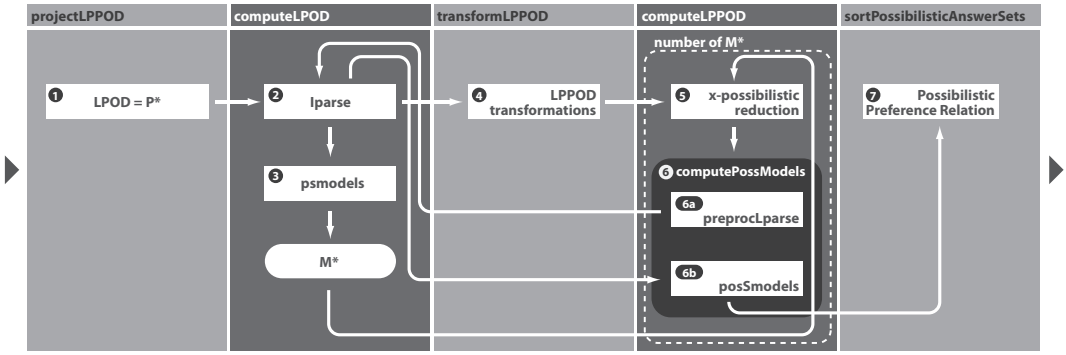


Fig. 7.7. Overview of the *posPsmodels* system

with such preference rule. In case the weight of a rule is decreased to the minimum value, *i.e.*, 0, the rule will be not considered any more.

As stated in Section 6.5.2, the user context follows a dynamic behavior. Therefore, LPPODs user profiles consider current user context (*i.e.*, user and environmental contexts are updated at each petition). The contextual activation of specific preference rules is achieved by means of *contextual information* in our LPPODs. This means that the context, along with the user petition, is mapped into facts that activate other rules and, consequently, other preferences.

7.6 User Preference Reasoning

Once user preferences have beens modeled, an order between them can be established. This is achieved using the theory of LPPODs. In fact, in LPPODs, ordered user preferences can be obtained by computing the solutions of an LPPOD, *i.e.*, its possibilistic answer sets. The LPPODs semantics specifies a formal way to obtain possibilistic answer sets and the possibilistic preference relation provides a comparison criterion to rank-order LPPODs solutions (see Chapter 5, Section 5.2).

At a practical level, the LPPODs semantics is implemented by the *posPsmodels* solver [Confalonieri et al., 2010c]. *PosPsmodels* is an ASP-based solver able to process LPPODs that can be used to compute possibilistic preferred answer sets under LPPODs semantics.

7.6.1 LPPODs Solver

The LPPOD solver implementation is depicted in Figure 7.7. The solver basically implements Algorithm 4 in a C++ program called *posPsmodels*.[11]

The system accepts an LPPOD and it returns an ordered set of possibilistic answer sets. In our system, we have used *lparse*[12], *psmodels*[13] and *posSmodels*[14] in an interleaved fashion

[11] A beta version of the *posPsmodels* system can be downloaded from `http://github.com/rconfalonieri/posPsmodels/tarball/master`.
[12] `http://www.tcs.hut.fi/Software/smodels/src/lparse-1.1.2.tar.gz`
[13] `http://www.tcs.hut.fi/Software/smodels/priority/`
[14] `http://www.info.univ-angers.fr/pub/pn/Softwares/posSmodels.html`

and we have implemented the modules for the LPPOD projection, the LPPODs transformations, the LPPODs possibilistic $\times$-reduction, and the possibilistic preference relation.[15]

The procedure to compute the possibilistic answer sets of an LPPOD can be described as follows. First, we apply the projection $*$ to the input file containing the LPPOD P (Step 1). Then, we retrieve the answer sets of its classical part P^*, *i.e.,* an LPOD (Step 2 and 3). At the same time, we apply the transformation rules for LPPODs (Step 4). As next, we build the representation needed by *posSmodels* which only accepts possibilistic normal logic programs. This can be done via the implementation of the possibilistic $\times$-reduction (Step 5). The reduction used the answer sets of the LPOD P^* in order to reduce the LPPOD P. Once program P has been reduced, its possibilistic answer sets can be generated (Step 6). Step 5 and Step 6 are repeated for each answer set retrieved in Step 3. Finally, the possibilistic answer sets are ordered (Step 7).

7.6.2 User Preference Computation and Ordering

The execution of the *posPsmodels* system over user profiles modeled as LPPODs produces an ordered list of user preferences.

Below, we present some of most preferred sets generated for the *Citizen* profile (Figure 7.6). Concerning this profile, 8 solutions are computed, among which a preference order can be obtained.

```
******** Possibilistic Model 1**********
(weather_Sunny,100) (time_13_15h,100) (loc_LesCorts,100)
(pet_Restaurant,100)  (type_Mediterranean,90) (cost_2,70)
(req_Terrace,100) (req_Parking,100) (req_Wheelchairaccess,100)
 (rate_3,80) (loc_Sants,60) (disabled,100)

******** Possibilistic Model 2**********
(weather_Sunny,100) (time_13_15h,100) (loc_LesCorts,100)
(pet_Restaurant,100) (type_Vegetarian,90) (cost_2,70)
(req_Terrace,100) (req_Parking,100) (req_Wheelchairaccess,100)
(rate_3,80) (loc_Sants,60) (disabled,100)

******** Possibilistic Model 3**********
(weather_Sunny,100) (time_13_15h,100) (loc_LesCorts,100)
(pet_Restaurant,100) (type_Mediterranean,90) (cost_2,70)
(req_Terrace,100) (req_Parking,100) (req_Wheelchairaccess,100)
(rate_1,80) (loc_Sants,60) (disabled,100)

******** Possibilistic Model 4**********
(weather_Sunny,100) (time_13_15h,100) (loc_LesCorts,100)
```

[15] Briefly, the projection eliminates weights from the rules of an LPPOD; transformations reduce the size of an LPPOD program preserving the LPPODs semantics; the possibilistic $\times$-reduction reduces an LPPOD according to a set of atoms M and it is part of the LPPODs semantics definition; finally, the possibilistic preference relation compares and orders the possibilistic answer sets of an LPPOD. All these concepts have been formally defined in Chapter 5.

```
(pet_Restaurant,100) (type_Mediterranean,90) (cost_3,70)
(req_Terrace,100) (req_Parking,100) (req_Wheelchairaccess,100)
(rate_3,80) (loc_Sants,60) (disabled,100)

...

******** Possibilistic Model 8**********
(weather_Sunny,100) (time_13_15h,100) (loc_LesCorts,100)
(pet_Restaurant,100) (type_Vegetarian,90) (cost_3,70)
(req_Terrace,100) (req_Parking,100) (req_Wheelchairaccess,100)
(rate_1,80) (loc_Sants,60) (disabled,100)
```

Concerning the solutions produced for the *Citizen*, we can observe the following. The solver computes *terrace* and *wheel chair access* in all possibilistic answer sets. This can be motivated by the fact that there are three requirement rules in the *Citizen* profile. Two rules are computed when the user is a *disabled* person (the ones related to the *wheel chair access* and the *parking*). These are general rules, since they are also valid in other contexts different from a restaurant petition (*e.g.,* in cinema or night petitions). The third rule is computed when there is a restaurant petition, time is between *13 and 15h*, and it is *not raining*. In this example, the contextual information tell us (among other information) that the *weather is sunny*, time is between *13h and 15h*, and that the user is *disabled*. Therefore, *terrace* and *wheel chair access* are added to all the possibilistic models. Among the context-dependent preferences related to restaurants, the user mostly prefers *Mediterranean* food, to eat in restaurants with a category rate at least of *rate 3* and cost category at most of *cost 2*. All the other solutions are rank-ordered according to the order in which preferences have been specified.

Similarly, concerning the *Tourist* profile (Figure 7.6), we illustrate below some of the most preferred solutions among the several ones which can be obtained (16). It can be observed that, in this case, the requirement about having a *terrace* cannot be satisfied, since in this particular context it is *raining*. Moreover, we can see that the solver cannot consider access to metro line *L3* as the user context excludes such a choice (probably because the line is not working or it is far away from the user location). Therefore, only access to metro line *L4* is considered.

```
******** Possibilistic Model 1**********
(-access_L3,100) (weather_Rain,100) (time_13_15h,100)
(loc_Sants,100)  (pet_Restaurant,100) (access_L4,80)
(type_Catalan,70) (req_MasterCard,100) (rate_3,60)
(cost_3,70) (loc_Gracia,40)

******** Possibilistic Model 2**********
(-access_L3,100) (weather_Rain,100) (time_13_15h,100)
(loc_Sants,100) (pet_Restaurant,100) (access_L4,80)
(type_Catalan,70) (req_Visa,100) (rate_3,60)
(cost_3,70) (loc_Gracia,40)
```

```
******** Possibilistic Model 3**********
(-access_L3,100) (weather_Rain,100) (time_13_15h,100)
(loc_Sants,100) (pet_Restaurant,100) (access_L4,80)
(type_Peruvian,70) (req_MasterCard,100) (rate_3,60)
(cost_3,70) (loc_Gracia,40)

******** Possibilistic Model 4**********
(-access_L3,100) (weather_Rain,100) (time_13_15h,100)
(loc_Sants,100) (pet_Restaurant,100) (access_L4,80)
(type_Catalan,70) (req_MasterCard,100) (rate_3,60)
(cost_4,70) (loc_Gracia,40)

...

******** Possibilistic Model 16**********
(-access_L3,100) (weather_Rain,100) (time_13_15h,100)
(loc_Sants,100) (pet_Restaurant,100) (access_L4,80)
(type_Peruvian,70) (req_Visa,100) (rate_4,60)
(cost_4,70) (loc_Gracia,40)
```

The preferred answer sets, along with their associated weights (enclosed in parentheses), are considered to look for personalized suggestions. To this end, the *Content Manager* will first use the most preferred possibilistic answer sets to build preference queries. If not enough suggestions are found, less preferred answer sets are considered.

7.7 Retrieving Personalized Suggestions

The retrieval of personalized suggestions for the user consists in the following five-steps process:

- *parsing preference items*: context-dependent user preferences returned by the *posPsmodels* system are parsed in order to map each element to the corresponding domain element (defined according to the *preference ontology*). To this end, prefixes at the beginning of atom names are used. For instance, *pet_Restaurant* is mapped into a *Petition*, while *req_Terrace* is mapped into a *Requirement*;
- *preference query generation*: each possibilistic answer set is mapped into one or more preference queries. The query generation takes preference weights into account and it is explained below;
- *mapping queries into XPath expressions*: since information sources retrieve XML-based data, preference queries are mapped into XPath expressions;
- *retrieving suggestions*: XPath queries are executed against the XML-based information;
- *visualization*: the results retrieved in the previous step are composed in an ordered suggestion set and converted into a visualization format (*e.g.,* HTML).

Preference queries are generated as follows:

- *Location preferences*: some results can provide more than one alternative for a *Location* preference. Therefore, a disjunction between *Location* elements is considered;
- *Rate preferences*: these preferences are considered as the minimum *Rate* to be accepted. Therefore, higher or equal rates (*e.g.*, $\geq$) are considered;
- *Cost preferences*: these preference are considered as the maximum *Cost* to be accepted. Therefore, a lower or equal costs (*e.g.*, $\leq$) are considered;
- *Type preferences*: all preference *Types* are considered; however, low weighted preferences are filtered for the benefit of the others;
- *Requirements*: all *Requirements* have to be considered. To this end, a conjunction of requirements is considered.

We explain the preference query generation by means of the following example.

Example 7.2. Based on the above interpretations, given the following possibilistic answer set:

```
(weather_Sunny,100) (time_13_15h,100) (loc_LesCorts,100)
(pet_Restaurant,100) (type_Mediterranean,90) (cost_2,70)
(req_Terrace,100) (req_Parking,100) (req_Wheelchairaccess,100)
(rate_3,80) (loc_Sants,60) (disabled,100)
```

the following pseudo-code for a preference query is generated:

```
(Petition = 'Restaurant') and (Rate >= 3) and (Cost <= 2) and
(Type = 'Mediterranean') and (Requirement = 'Parking' and
Requirement = 'Terrace' and Requirement = 'Wheelchairaccess') and
(Location = 'LesCorts' or Location = 'Sants')
```

Please observe that, generally, a given query can be too demanding and its execution returns an empty set of results. On the contrary, it can also happen that a query is too relaxed and, in such a case, it returns too many items. In such cases, preference weights can be used to relax or to over-constrain some query parameters.

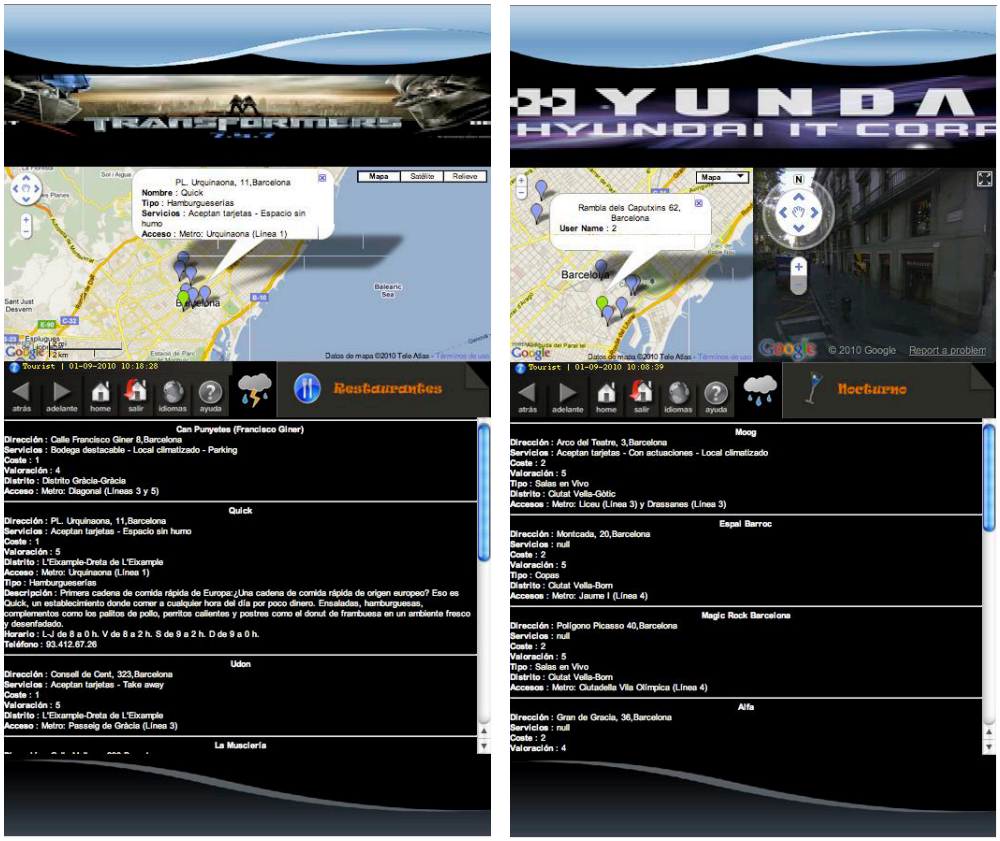
For instance, from the preference query above, the following queries are obtained by relaxing query parameters with a weight lower than 100:

```
(Petition = 'Restaurant') and (Rate >= 3) and (Type = 'Mediterranean') and
(Requirement = 'Parking' and Requirement = 'Terrace' and
Requirement = 'Wheelchairaccess') and
(Location = 'LesCorts' or Location = 'Sants')

(Petition = 'Restaurant') and (Type = 'Mediterranean') and
(Requirement = 'Parking' and Requirement = 'Terrace' and
Requirement = 'Wheelchairaccess') and
(Location = 'LesCorts' or Location = 'Sants')
```

On the contrary, from the preference query above, the following query is obtained:

```
(Petition = 'Restaurant') and (Rate >= 3) and (Cost <= 2) and
(Type = 'Mediterranean') and (Requirement = 'Parking' and
Requirement = 'Terrace' and Requirement = 'Wheelchairaccess') and
(Location = 'LesCorts')
```



(a) Tourist's Results (b) Citizen's Results

Fig. 7.8. Final GUIs showing several suggestions

Preference queries are mapped into XPath expression which are queried against the data offered by information sources (we omit these details). Therefore, results are obtained using the most preferred solutions. Less preferred ones are used only if a sufficient number (application-dependent) of suggestions cannot be achieved.

Once sufficient data have been retrieved, the results are visualized on (interactive) city maps (see Figure 7.8). From this interface, the user can express his/her feedback about the results obtained in order to help the system to keep his/her profile up-to-date as much as possible.

7.8 Related Work

The reason for introducing LPPODs into the ICD was to enhance the system with a preference handling method in order to represent and to manage user profiles. The representation of user profiles has been done by considering contextual and incomplete knowledge, as well as weights between the preferences themselves. The process of retrieving personalized suggestions has been decoupled in a dedicate module (*i.e.*, the Content Manager) which is responsible to interpret preferences and associated weights. This has been a design choice, since the information is collected from several Web services (coordinated by

the ALIVE-based infrastructure) which are queried according to user preferences specified in the parameters of the service call.

Instead, in some of the approaches that deal with user preferences, preference representation and preference queries are not decoupled. This is the case for the different lines of research in the literature which deal with preferences queries in databases and in possibilistic databases.

7.8.1 Database Preference Queries

Since classical queries can sometimes either return an empty set of answers (because the condition(s) imposed by the query are too restrictive) or they can return too many results (because there are too many items which can be matched), the interest in expressing preferences inside database queries has been growing. In fact, plenty of proposals to flexile querying can be found in the literature such as approaches based on databases [Lacroix and Lavency, 1987; Kießling, 2002; Chomicki, 2003; Godfrey and Ning, 2004], on fuzzy set [Dubois and Prade, 1997], on CP-nets [Brafman and Domshlak, 2004], and, more recently, on possibilistic logic [Hadjali et al., 2008].

The main reasons behind introducing preferences in the querying process are different. Main streamlines of research have focused on offering more expressive query languages and on providing preference relations for rank-ordering the retrieved items. These different points have been considered in the approaches mentioned above, although from different perspectives. In particular, works in the database community study and refine algorithms aiming at the efficient computation of non Pareto-dominated answers (*i.e.,* the skyline point of view [Börzsönyi et al., 2001]). Fuzzy set-based approaches use fuzzy set membership functions for describing the user profiles on each attribute domain involved in the query. This is especially convenient when dealing with numerical domains (a good comparison between database vs. fuzzy set-based methods can be found in [Bosc and Pivert, 1992]). Moreover, fuzzy-set based approaches have contributed with some noticeable ordering refinements for min- and Pareto-based comparison criteria such as *discrimin* and *leximin* [Dubois et al., 1997]. In [Brafman and Domshlak, 2004], it is shown how CP-nets can be used for the specification of preference queries and of a composition operator for composing different preference orders into a single preference one. A more effective approach to the handling of database preference queries, which takes its roots in the fuzzy set-based approaches, is represented by possibilistic logic. Indeed, possibilistic logic improves the CP-net based method from a preference-based ranking point of view [Hadjali et al., 2008]. A good discussion and comparison between CP-nets and possibilistic logic can be found in [Kaci and Prade, 2007, 2008].

Preference queries have also been considered in the setting of possibilistic databases [Hadjali et al., 2008; Bosc et al., 2009]. In this setting, possibilistic logic is used to handle priority and certainty degrees in order to manage uncertain information and to rank-order retrieved items. More recently, it has also been shown how possibilistic logic can be used to encode preference queries built by means of weighted disjunction and conjunction [Bosc et al., 2010]. Based on that, we have been able to draw a closer relation between possibilistic logic and LPPODs. As such, a more detailed discussion is presented in the next section.

7.8.2 Preference Queries in Possibilistic Databases

In the context of possibilistic databases, the use of possibilistic logic for representing uncertain data can provide a strong representation system for the whole relational algebra [Bosc et al., 2009]. In particular, a possibilistic database can be directly encoded in possibilistic logic by mapping keys into variables, attributes into predicates, and database tuples into instantiated formulas [Bosc et al., 2009]. Then, answering a query such as $\exists x$ $C_1(x) \wedge \ldots \wedge C_n(x)$ (resp. $\exists x\ C_1(x) \vee \ldots \vee C_n(x)$) amounts to add the possibilistic logic formula $\{(\neg C_1(x) \vee \ldots \neg C_n(x) \vee answer(x), 1)\}$ (resp. $\{(\neg C_1(x) \vee answer(x), 1),\ldots,(\neg C_n(x) \vee answer(x), 1)\}$) to the possibilistic base and to evaluate the query by the repeated application of possibilistic resolution $(\neg P(x) \vee Q(x,y), \alpha), (P(a) \vee R(z), \beta) \vdash_{PL} (Q(a,y) \vee R(z), min(\alpha, \beta))$ [Dubois et al., 1994].

However, it can sometimes happen that these queries either return an empty set of answers or they return too many results. Such cases have motivated the study of a way to incorporate qualitative preferences inside queries in the context of possibilistic logic. When modeling preferences in possibilistic logic, the necessity measure α associated to a classical formula p in (p, α) is understood as the priority of p rather than its certainty level.[16] The incorporation of preferences inside a query can be achieved assigning a (total) order over the necessity values associated to the query. In [Bosc et al., 2010], the authors show how it is possible to represent *conjunctive* and *disjunctive* preference queries asking for items satisfying conditions $C_1, C_2, \ldots, C_{n-1}, C_n$ with the information that C_1 *is required and if possible* C_2 *also,. . . , and if possible* C_n *too* and C_n *is required, or better* C_{n-1}*,. . .,or still better* C_1 respectively.

For instance, given three conditions $C_1(x)$, $C_2(x)$, $C_3(x)$ and the necessity measures $1, \alpha, \beta$ (seen as priority levels associated to the possibilistic formula representing the queries) with $1 > \alpha > \beta$, the possibilistic logic encoding of a conjunctive (resp. disjunctive) query is {($\neg C_1(x) \vee \neg C_2(x) \vee \neg C_3(x) \vee answer(x)$,1), ($\neg C_1(x) \vee \neg C_2(x) \vee answer(x)$,$\alpha$), ($\neg C_1(x) \vee answer(x)$,$\beta$)} (resp. {($\neg C_1(x) \vee answer(x)$,1), ($\neg C_2(x) \vee answer(x)$,$\alpha$), ($\neg C_3(x) \vee answer(x)$,$\beta$)}).

Once a preference query is evaluated, the retrieved items are associated with two levels: to *what extent the answer satisfies the query* (*i.e.,* $1, \alpha, \beta$), and to *what extent the data used in the query evaluation are certain* (*i.e.,* the necessity measures in the database). These scales can be used to achieve a totally ordered set of query results [Bosc et al., 2010].

In the following, we show how it is possible to encode uncertain data and preference queries in the LPPODs framework. To this end, we first show how a preference query is represented and processed in possibilistic logic and, then, how it can be represented and processed in the LPPODs framework. We only show the case of a conjunctive query, since an equivalence between conjunctive and disjunctive queries exists [Bosc et al., 2010]. For the comparison, we will borrow an example presented in [Bosc et al., 2010].

Let us consider a database example DB with three relations R, S, and T containing uncertain pieces of data (Table 7.1). The context considered is such that uncertainty may pervade attribute values. We adopt the model proposed in [Bosc et al., 2009]. According to

[16] This point of view has been discussed in Chapter 6, Section 6.2.

Table 7.1. Relations R (top-left), S (top-right), and T (bottom) (example taken from [Bosc et al., 2010])

id	Name	City
1	John	(Brest,**a**)
2	Mary	(Lannion,**b**)
3	Peter	(Quimper,**c**)

City	Flea Market
Brest	(yes,**d**)
Lannion	(no,**e**)
Quimper	(no,**f**)
Rennes	(yes,**g**)

City	Museum
Rennes	(modern,**h**)
Quimper	(contemporary,**i**)
Brest	(modern,**k**)

it, each uncertain attribute value is associated with a certainty degree which is modeled as a lower bound of a necessity measure.

Given the possibilistic database DB, let us consider the following preference query asking:

Example 7.3. [Bosc et al., 2010] *Find people living in a city with a flea market ($fleaMarket(x)$) and preferably a museum of modern art ($modern(x)$) and, if possible, a museum of contemporary art ($contemp(x)$).*
The query is clearly conjunctive and its possibilistic logic representation is:

$$Q_\wedge = \left\{ \begin{array}{c} (\neg fleaMarket(x) \vee \neg modern(x) \vee \neg contemp(x) \vee answer(x), 1) \\ (\neg fleaMarket(x) \vee \neg modern(x) \vee answer(x), \alpha) \\ (\neg fleaMarket(x) \vee answer(x), \beta) \end{array} \right\}$$

where $1 > \alpha > \beta$.
By applying possibilistic logic resolution, it can be proved that valid answers to the query are: $\{answer("John"), \alpha, min(a,d,k)\}$ and $\{answer("John"), \beta, min(a,d)\}$.

In the encoding of the database DB in LPPODs, we adopt the same convention used in [Bosc et al., 2009]. According to it, keys become variables, attributes become predicates, and database tuples are encoded by instantiated predicates. As attribute certainty is modeled as a necessity measure, we can directly map it to the necessity values of program rules in our LPPODs framework. Therefore, the LPPOD representing the DB in Table 7.1, denoted by P_{DB}, is:

$$P_{DB} = \left\{ \begin{array}{cc} r_1 = \mathbf{a} : city("John","Brest"). & r_6 = \mathbf{f} : \neg fleaMarket("Quimper"). \\ r_2 = \mathbf{b} : city("Mary","Lannion"). & r_7 = \mathbf{g} : fleaMarket("Rennes"). \\ r_3 = \mathbf{c} : city("Peter","Quimper"). & r_8 = \mathbf{h} : modern("Rennes"). \\ r_4 = \mathbf{d} : fleaMarket("Brest"). & r_9 = \mathbf{i} : modern("Brest"). \\ r_5 = \mathbf{e} : \neg fleaMarket("Lannion"). & r_{10} = \mathbf{k} : contemp("Quimper"). \end{array} \right\}$$

Let us consider now the query in Example 7.3. Its encoding in the LPPODs framework, denoted by $P_{Q_\wedge}$, is:

$$r_{11} = 1 : q_1(X) \leftarrow city(X,Y), fleaMarket(Y), modern(Y), contemp(Y), not\ q_2(X), not\ q_3(X).$$
$$r_{12} = 1 : q_2(X) \leftarrow city(X,Y), fleaMarket(Y), modern(Y), not\ q_1(X), not\ q_3(X).$$
$$r_{13} = 1 : q_3(X) \leftarrow city(X,Y), fleaMarket(Y), not\ q_1(X), not\ q_2(X).$$
$$r_{14} = 1 : q_1(X) \times q_2(X) \times q_3(X).$$

The main difference w.r.t. the possibilistic logic encoding is in the way in which we represent the query priority. In fact, we encode priority about a query by a possibilistic ordered disjunction rule (r_{14}) rather than by means of necessity values as done in possibilistic logic. Then, we associate queries and preferences about the queries with a necessity of **1** in order to keep the necessity values of the queries results as the certainty values computed by the fix-point operator (which reflects the possibilistic modus ponens, see Chapter 5, Section 5.2.1).

Concerning the query execution, it can be checked that the program $P_{DB} \cup P_{Q_\wedge}$ does not have any result matching q_1 (as in Example 7.3), while we obtain $M_1 = \{q2(''John''),min(a, d, k)\}$ with $deg_{M_1}(r_{14}) = 2$ matching q_2 and $M_2 = \{q3(''John''), min(a, d)\}$ with $deg_{M_2}(r_{14}) = 3$ matching q_3. Please observed that the retrieved items are associated with two levels: (i) the satisfaction degree of the possibilistic answer set w.r.t. the ordered disjunction rule, (ii) and the necessity values as certainty measures w.r.t. the data used in the query evaluation. This is clearly in accordance with the result obtained in the possibilistic logic setting. Moreover, the total order of the preference queries in the possibilistic logic setting is reflected in LPPODs as well. In fact, according to the possibilistic preference relation, M_1 is preferred to M_2 since $deg_{M_1}(r_{14}) < deg_{M_2}(r_{14})$ (see Chapter 5, Section 5.4).

To conclude, the representation of preference queries in the LPPODs setting seems to be feasible and there is a close relation between the two approaches. However, at representational level the possibilistic setting seems to be more expressive. In fact, in [Prade, 2009; Bosc et al., 2010] it is claimed how possibilistic logic can also accommodate some cases of disjunctive information (for instance the case in which the third tuple of relation R is $\langle Peter, (Quimper \vee Rennes, c)\rangle$). Instead, the LPPODs syntax does not allow disjunction in the head of program rules. A possible way to overcome this limitation is by extending the LPPODs syntax and semantics with possibilistic disjunction [Nieves et al., 2011].

7.9 Discussion and Concluding Remarks

In this chapter, we have presented the enhancement of the personalization capabilities of a specific instance of context-aware systems, an ICD, by means of user profiles. At knowledge level, we have represented user profiles according to a user profile and preference ontology. User profiles have then been translated, at symbolic level, into the logic programming specification of LPPODs. Indeed, LPPODs has proved to be a suitable specification in order to express user preferences which can depend both on contextual and incomplete information. Furthermore, preferences can be associated with weights expressing the preference importance. Such weights can be considered to relax or the over-constrain preference conditions at the moment of retrieving personalized suggestions and (although not explicitly covered here) to keep the user profile up-to-date.

The LPPODs features have been implemented in an ASP-based solver called *posPsmodels*. The solver has been encapsulated in a dedicated service which integrates with the system infrastructure implemented with the ALIVE tools[17]. The integration of the LPPODs solver into the system, due to its complexity, has only been done at a basic level. In the

[17] `http://sourceforge.net/projects/ict-alive/files/`

current version, the solver is wrapped by a Java-based interface which exposes some basic methods for querying the preference profile against the most preferred options of the user depending on the context. More functionalities should be added in order to support the actualization of user profiles. Indeed, methods for changing preference rules and weights, by using user feedback, can provide the basis to adapt user profiles according to the most recent user tastes and dislikes.

Apart of the aforementioned improvements, another important aspect that needs to be further considered is the preference representation. In fact, the syntax supported by LPPODs is quite limited. Since LPPODs generalizes LPODs (see Chapter 5), LPPODs intrinsically inherits, from a syntactical point of view, the expressiveness of LPODs. This basically means that the LPPODs syntax only allows to express alternatives between preference literals rather than more general expressions. However, more complex preference statements may need to be encoded in the ICD.

For instance, let us consider the user preferences about restaurants introduced in Example 7.1. In that example, the user preferred *to eat in a Mediterranean restaurant rather than in a Vegetarian* and she preferred *to pay by Master Card over using her Visa*. In such a case, it can be valuable to be able to capture complex preference expressions in order to express equalities between options and/or combinations over alternatives. Indeed, the user may prefer to have *Mediterranean* over *Vegetarian* food over not having any of them, *i.e.,* expressions such as $\mathit{Mediterranean} \times \mathit{Vegetarian} \times (\mathit{not}\ \ \mathit{Mediterranean} \wedge \mathit{not}\ \ \mathit{Vegetarian})$; or she may prefer to pay either by Master Card or by Visa over paying cash, *i.e.,* expressions such as $(\mathit{MasterCard} \vee \mathit{Visa}) \times \mathit{cash}$.

In order to extend LPPODs, we first need to extend LPODs to support a richer formalism to represent and to reason about these preference statements. As such, in Chapter 8, we propose a first step towards an extension of LPPODs. We present a more general syntax which allows the writing of nested (or non-flat) preferences expressions built by means of connectives $\{\vee, \wedge, \neg, \mathit{not}, \times\}$. To represent these formulas and to capture their semantics, we define an extension of LPODs called *Nested Logic Programs with Ordered Disjunction* (LPODs$^+$).

8

LPODs$^+$: A Framework for Handling Nested Preferences

Handling user preferences in context-aware systems, which provide users with personalized information, is an essential feature. As such, finding a suitable preference handling method plays an important role in the development of these systems. LPPODs has shown to be a suitable specification to represent and to reason about user preferences.[1] Indeed, LPPODs can support: (i) the representation of preferences which can depend on contextual and incomplete information and which can have different importance; (ii) the specification of a preference relation for achieving an order among preferred alternatives. Nevertheless, the LPPODs syntax is too restrictive when more complex preference expressions need to be encoded. This limitation is due to the fact that LPPODs relies on the LPODs framework. Therefore, planning an extension of LPPODs means to extend LPODs first.

As far as LPODs is concerned, its syntax allows to express alternatives between preference literals. An LPOD can encode simple statements such as: *At night I prefer going to a pub over going to a cinema over watching tv* (encoded as $pub \times cinema \times tv \leftarrow night$), but the LPODs syntax is limited when other types of preference statements need to be formulated. For example, at night when going to a *pub* is not possible, one might prefer to watch a movie at the *cinema* or on *tv* with equal preference. This kind of preference equality has been addressed in the DLPODs framework (see Chapter 6, Section 6.4.2). The DLPODs syntax allows one to express preference equalities in ordered disjunction rules by means of combinations of literals connected by $\vee$. For instance, the preference equality statement about *cinema* and *tv* at night can be encoded as $pub \times (cinema \vee tv) \leftarrow night$.

However, there may exist more complex preference statements that not even DLPODs is able to express. In the nighttime preferences above, we can instead be concerned about having both options at the same time, expressing that in case *pub* is not possible we *do* mind watching a movie in the *cinema* and not on the *tv*, *i.e.*, expressions such as $pub \times (cinema \wedge \neg tv) \leftarrow night$, or that we even want to have preference equalities and combinations at the same time, *i.e.*, expressions such as $(pub \vee bar) \times (cinema \wedge \neg tv) \leftarrow night$, or even more complex preference expressions such as $(pub \wedge (expensive \times cheap)) \times bar \times (not\ \ pub \wedge not\ \ bar) \leftarrow night \vee (not\ \ busy \wedge afternoon)$. These examples suggest that studying less restricted syntaxes could be valuable.

[1] In Chapter 7, we have shown how LPPODs can be employed to handle user preferences in a context-aware system such as ICD.

In this chapter, we present a more general syntax which make it possible to specify nested (or non-flat) preference expressions built by means of connectives $\{\vee, \wedge, \neg, not, \times\}$. To represent these formulas and to capture their semantics, we define an extension of LPODs called *Nested Logic Programs with Ordered Disjunction* (LPODs$^+$).

The syntax of LPODs$^+$ is based on a language which supports nested formulas that can provide a richer syntax at the moment of writing conditional preferences. The language we consider combines the propositional language of QCL (see Chapter 6, Section 6.1) with negation as failure.

To define the LPODs$^+$ semantics, we reuse and extend some of the results defined in [Lifschitz et al., 1999] in which a semantics for nested logic programs is presented. As such, we capture the semantics of a nested ordered disjunction program (OD$^+$-program) in a simple way and we show that, when an OD$^+$-program syntactically corresponds to an LPOD, the LPOD$^+$ semantics and the LPODs semantics coincide.

With the presence of nested expressions in the head of preference rules, determining the satisfaction degree of an answer set of an OD$^+$-program can be sometimes more involved in our approach. For this reason, we propose a recursive function to compute the optionality of complex preference formulas in order to determine the rule satisfaction degree. As the optionality function is a generalization of the rule satisfaction degree in LPODs, we can directly use LPODs preference relations for comparing the answer sets of an OD$^+$-program. Consequently, our approach generalizes the LPODs framework in a proper way.

Since the LPOD$^+$ syntax is too complex to be handled by existent answer set solvers, we propose a translation procedure to map any OD$^+$-program to an equivalent disjunctive logic program. This result is significant, since the complexity added at the syntactic level does not interfere with the computation of the LPOD$^+$ semantics whose complexity coincides with the complexity class Σ_2^p of disjunctive logic programs (propositional case [Eiter and Gottlob, 1995]).

The rest of the chapter is structured as follows. In Section 8.1 and Section 8.2, we respectively present the LPODs$^+$ syntax and the LPODs$^+$ semantics. In Section 8.3, we introduce a recursive optionality function for computing the satisfaction degree of an answer set of an OD$^+$-program. In Section 8.4, we design a translation procedure which maps any OD$^+$-program into a disjunctive logic program. We also provide some guidelines about the implementation of a tester program for computing the optimal answer set of an OD$^+$-program. Section 8.5 compares our work with other approaches in the literature. Finally, Section 8.6 concludes the chapter.

8.1 LPODs$^+$ Syntax

In the following, literals[2], $\bot$ and $\top$ are considered *elementary formulas*, while $\{\vee, \wedge, not, \times\}$ *formulas* (denoted $A, B, C, \ldots$) are constructed from elementary formulas using the connectives $\{\vee, \wedge, not, \times\}$ arbitrarily nested (strong negation $\neg$ is allowed to appear only in front

[2] A more general background on ASP can be found in Chapter 2, Section 2.1. Please remember that a literal is an atom or an extended atom.

Table 8.1. Example of rules captured by the LPODs$^+$ syntax

Syntax	Rule Type
$(a \wedge (b \times c)) \times (d \vee \neg e) \leftarrow g \vee (not\ \neg i \wedge f)$.	*Nested Ordered Disjunction rule*
$a \wedge not\ b \leftarrow p \wedge not\ (\neg q \vee r)$.	*Nested rule* [Lifschitz et al., 1999]
$a \times (b \vee c) \leftarrow d \wedge\ not\ e$.	*Ordered Disjunctive rule* [Kärger et al., 2008]
$a \times b \leftarrow c \wedge not\ d$.	*Ordered Disjunction rule* [Brewka et al., 2004b]
$a \vee b \leftarrow c \wedge not\ \neg e$.	*Disjunctive rule* [Gelfond and Lifschitz, 1991]

of atoms). We assume that the connectives $\wedge$, $\vee$, and *not* have stronger bindings than $\times$. We also assume that $\times$ is associative.

Then, given a finite set of literals $\mathcal{L}$, a *nested ordered disjunction rule* (rule for short) is an expression of the form

$$H \leftarrow B \tag{8.1}$$

where H is either an elementary formula or a $\{\vee, \wedge, not, \times\}$ formula (known as the *head*) and B is either an elementary formula or a $\{\vee, \wedge, not\}$ formula (known as the *body*) built using the literals in $\mathcal{L}$. Some particular cases are *facts*, of the form $H \leftarrow \top$ (written as H), and *constraints*, $\bot \leftarrow B$ (written as $\leftarrow B$). If no occurrences of *not* appear in a rule, then the rule is a *definite nested ordered disjunction rule* (similarly a definite nested ordered disjunction formula). If no occurrences of $\times$ appear in a rule, then the rule is a *nested rule* (similarly a nested formula). If no occurrences of $\times$ and *not* appear in a rule, then the rule is known as a *definite nested rule* (similarly a definite nested formula). Different formula combinations lead to different rules as shown in Table 8.1.

A *Nested Logic Program with Ordered Disjunction*, or an OD$^+$-program, is a finite set of nested ordered disjunction rules and/or constraints and/or facts. If the program does not contain *not*, then the program is called a *definite* OD$^+$-program.

The use of $\{\vee, \wedge, not, \times\}$ formulas in the head of nested ordered disjunction rules, rather than elementary formulas or disjunctive formulas only, as in LPODs and DLPODs, provides a rich syntax for specifying conditional qualitative preferences. Indeed, using $\vee$ and $\wedge$, we can express that some options are equally preferred, or that some combinations are preferred over other combinations. The following program exemplifies some preference statements that can be modeled by means of an OD$^+$-program.

Example 8.1. Let P be an OD$^+$-program representing some user preferences of the form:

$$P = \begin{Bmatrix} r_1 : italian \times peruvian \times (not\ italian \wedge not\ peruvian) \leftarrow \top. \\ r_2 : (pub \vee bar) \times (cinema \wedge \neg tv) \leftarrow night. \\ r_3 : night \leftarrow \top. \end{Bmatrix}$$

Briefly, the intuitive reading of each of the rules of program P is the following: r_1 expresses that we generally prefer to have *italian* over *peruvian* food and we prefer to have one of the options over not having any of them; r_2 tells that at *night* we prefer to go to a *pub* or a *bar*, and, if not possible, we want to watch a movie at the *cinema* but not on the *tv*; r_3 just states that it is night.

8.2 LPODs$^+$ Semantics

To define the semantics of LPODs$^+$, we consider and extend some of the results of nested logic programs [Lifschitz et al., 1999]. The first definitions are simply reported as they represent some basic results. Instead, Definition 8.4 and 8.5 extend the original ones for nested programs to account for the case of $\times$ which characterizes our approach.

Definition 8.1. *[Lifschitz et al., 1999] Let M be a set of literals. M satisfies a definite nested formula A (denoted by $M \models A$), recursively as follows:*

- *for elementary A, $M \models A$ if $A \in M \vee A = \top$*
- *$M \models A \wedge B$ if $M \models A \wedge M \models B$*
- *$M \models A \vee B$ if $M \models A \vee M \models B$*

Definition 8.2. *[Lifschitz et al., 1999] Let P be a definite nested logic program. A set of literals M is closed under P if, $\forall r \in P$ such that $r = H \leftarrow B$, $M \models H$ whenever $M \models B$.*

Definition 8.3. *[Lifschitz et al., 1999] Let M be a set of literals and P a definite nested logic program. M is called an answer set for P if M is minimal among the sets of literals closed under P.*

Definition 8.4. *The reduction of a nested ordered disjunction formula with respect to a set of literals M is recursively defined as follows:*

- *for elementary A, $A^M = A$*
- *$(A \wedge B)^M = A^M \wedge B^M$*
- *$(A \vee B)^M = A^M \vee B^M$*
- $(not\ A)^M = \begin{cases} \bot, & \text{if } M \models A^M \\ \top, & \text{otherwise} \end{cases}$
- $(A \times B)^M = \begin{cases} A^M, & \text{if } M \models A^M \\ B^M, & \text{if } M \not\models A^M \wedge M \models B^M \\ \bot, & \text{otherwise} \end{cases}$

Definition 8.5. *The reduction $P^M_{\times^+}$ of an OD$^+$-program P with respect to a set of literals M is recursively defined as follows:*

- *$(H \leftarrow B)^M = H^M \leftarrow B^M$*
- *$P^M_{\times^+} = \{(H \leftarrow B)^M \mid H \leftarrow B \in P\}$*

Please observe that $P^M_{\times^+}$ is a definite nested logic program. Hence, the following definition follows from the answer set definition for definite nested logic programs in a straightforward way.

Definition 8.6. *Let P be an OD$^+$-program and M a set of literals. M is an answer set of P if it is an answer set of $P^M_{\times^+}$. We denote the LPODs$^+$ semantics of program P by means of $SEM_{LPOD^+}(P)$.*

Example 8.2. Let us consider the OD$^+$-program P in Example 8.1[3] and the set of literals $M = \{b, d, g\}$. Then $P^{\{b,d,g\}}_{\times^+}$ can be obtained in three recursive steps by applying Definition 8.4

[3] Due to representation issues we have changed the signature of the program from $\{italian, peruvian, pub, bar, cinema, tv, night\}$ to $\{a, b, c, d, e, f, g\}$.

and 8.5:

(1)	(2)	(3)
$(a \times b \times (not\ a \wedge not\ b))^{\{b,d,g\}}$ $((c \vee d) \times (e \wedge \neg f) \leftarrow g)^{\{b,d,g\}}$ $(g)^{\{b,d,g\}}$	$b^{\{b,d,g\}}$ $(c \vee d)^{\{b,d,g\}} \leftarrow g^{\{b,d,g\}}$ g	b $c \vee d \leftarrow g$ g

Clearly, M is an answer set of $P^{\{b,d,g\}}_{\times^+}$ and so M is an answer set of P. Similarly, it can be checked that the sets of literals $\{a,c,g\}$, $\{a,d,g\}$, $\{a,e,\neg f,g\}$, $\{b,c,g\}$, $\{b,e,\neg f,g\}$, $\{c,g\}$, $\{d,g\}$ and $\{e,\neg f,g\}$ are valid answer sets of P as well.

It can be observed that an immediate property of the above construction is that, when the formulas in LPODs$^+$ are formulas without the $\times$ connective, the LPODs$^+$ semantics and the semantics for nested logic programs (denoted by $SEM_{NLP}(P)$) [Lifschitz et al., 1999] coincide.

Proposition 8.1. *Let P be an OD$^+$-program such that $\forall r \in P$, $r = H \leftarrow B$, H and B be formulas $\times$ free, and M be a set of literals. $M \in SEM_{LPOD^+}(P)$ if and only if $M \in SEM_{NLP}(P)$.*

Considering the relationship between LPODs$^+$ and LPODs, we can observe the following. Besides the fact that an OD$^+$-program has a richer syntax, it also has a richer semantics. Indeed, when all the rules of an OD$^+$-program P are ordered disjunction rules, the LPODs$^+$ semantics and the LPODs semantics of P (denoted by $SEM_{LPOD}(P)$) coincide.

Proposition 8.2. *Let P be an OD$^+$-program such that $\forall r \in P$, $r = A_1 \times \ldots \times A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_{m+n}$ in which A_i's ($1 \leq i \leq k$) and B_j's ($1 \leq j \leq m+n$) are literals and M be a set of literals. $M \in SEM_{LPOD^+}(P)$ if and only if $M \in SEM_{LPOD}(P)$.*

As a consequence, our approach can offer an alternative characterization of the LPODs semantics and it generalizes the LPODs semantics in a proper way. Nevertheless, determining the satisfaction degree of an answer set of an OD$^+$-program can sometimes be more involved in our approach as we discuss in the next section.

8.3 LPODs$^+$ Answer Sets Selection

LPODs$^+$ is a specification to express context-dependent preferences, in which a preference order among its answer sets is induced by means of the $\times$ connective. Therefore, the key question now is how such ordering can be achieved. In the simplest case, when each rule of an OD$^+$-program P corresponds to an ordered disjunction rule, a satisfaction degree k ($1 \leq i \leq k$) can be associated with an answer set M w.r.t. each rule of P. In this basic case, the satisfaction degree corresponds to the position of the best satisfied literal [Brewka et al., 2004b].

However, when we consider a nested ordered disjunction rule $r = H \leftarrow B$, in which H and B can be formulas built from $\{\vee, \wedge, not, \times\}$ and $\{\vee, \wedge, not\}$ respectively, the assignment of the satisfaction degree is not so trivial. In fact, this degree depends on how many options H admits. For this reason, we first define the *optionality* of an answer set M w.r.t. a nested ordered disjunction formula H as follows:

Definition 8.7. *Let H be a $\{\vee, \wedge, not, \times\}$ formula, and M be an answer set of an OD$^+$-program P. Then, the optionality of M w.r.t. H, denoted by $opt_M(H)$, is recursively defined as:*

$$opt_M(H) = \begin{cases} 1, \text{ if } H = A \text{ and } A \text{ is an elementary formula} \\ k \begin{cases} \text{if } H = (not\ A) \text{ then } k = opt_M(A) \\ \text{if } H = (A \wedge B) \text{ then } k = max\{opt_M(A), opt_M(B)\} \\ \text{if } H = (A \vee B) \text{ then } k = max\{opt_M(A), opt_M(B)\} \\ \text{if } H = (A \times B) \text{ then } \begin{cases} k = opt_M(A), \text{ if } M \models A^M \\ k = opt_M(A) + opt_M(B), \text{ otherwise} \end{cases} \end{cases} \end{cases}$$

Based on this function, the satisfaction degree of an answer set M w.r.t. a nested ordered disjunction rule $r = H \leftarrow B$ can be defined as:

Definition 8.8. *Let M be an answer set of an an OD$^+$-program P. Then, the satisfaction degree M w.r.t. a rule $r = H \leftarrow B$, denoted by $deg^+_M(r)$, is:*

$$deg^+_M(r) = \begin{cases} 1, & \text{if } M \not\models B^M \\ opt_M(H), & \text{otherwise} \end{cases}$$

The degrees can be viewed as penalties; in fact, a higher degree expresses a lesser satisfaction. Therefore, if the body of a rule is not satisfied, then there is no reason to be dissatisfied and the best possible degree 1 is obtained [Brewka et al., 2004a,b].

We can observe that there is a direct property w.r.t. the optionality function we have just defined for an answer set of an OD$^+$-program. The optionality function clearly generalizes the satisfaction degree of an answer set of an LPOD. More generally, the following property holds.

Proposition 8.3. *Let P be an OD$^+$-program such that $\forall r \in P$, $r = A_1 \times \ldots \times A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_{m+n}$ in which A_i's ($1 \leq i \leq k$) and B_j's ($1 \leq j \leq m+n$) are literals and M be an answer set of P. Then, $deg^+_M(r) = deg_M(r)$.*[4]

Therefore, our approach also generalizes in a consistent way the preference relation between answer sets of LPODs.

Definition 8.9. *Let P be an OD$^+$-program and let M_1, M_2 be two answer sets of P. Then M_1 is preferred to M_2 (denoted by $M_1 \succ M_2$) iff $\exists\ r \in P$ such that $deg^+_{M_1}(r) < deg^+_{M_2}(r)$, and $\nexists r' \in P$ such that $deg^+_{M_2}(r') < deg^+_{M_1}(r')$.*

The condition in the above definition follows the principle of Pareto optimality according to which an object is preferred if it is better or equal to another in all attributes (in our case in all nested ordered disjunction rules) and strictly better in at least one attribute.

Finally, we provide the definition of a preferred answer set of an OD$^+$-program P:

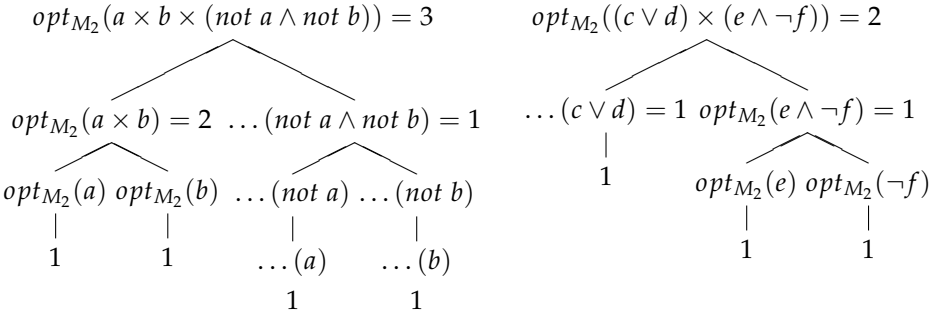
Definition 8.10. *Let P be an OD$^+$-program and M be an answer set of P. M is called a preferred answer set of P if there is no answer set M' of P for which $M' \succ M$ holds.*

[4] For the definition of $deg_M(r)$, we refer to Chapter 2, Section 2.2.4.

Example 8.3. Let P be the OD$^+$-program in Example 8.1 and $M_1 = \{a, c, g\}$, $M_2 = \{e, \neg f, g\}$ be two of the answer sets of P in Example 8.2. We want to show that $M_1 \succ M_2$. To this end, we first have to compute the satisfaction degrees of M_1 and M_2 w.r.t. all the rules in P. Applying Definitions 8.7 and 8.8, we can see how the degrees of M_1 w.r.t. r_1 and r_2 correspond to $deg^+_{M_1}(r_1) = 1$ and $deg^+_{M_1}(r_2) = 1$ respectively, since:

$$\begin{array}{c} opt_{M_1}(a \times b \times (not\ a \wedge not\ b)) = 1 \\ | \\ opt_{M_1}(a \times b) = 1 \\ | \\ opt_{M_1}(a) \\ | \\ 1 \end{array} \qquad \begin{array}{c} opt_{M_1}((c \vee d) \times (e \wedge \neg f)) = 1 \\ | \\ opt_{M_1}(c \vee b) = 1 \\ | \\ 1 \end{array}$$

The case of M_2 looks more interesting as it shows how determining the degree of satisfaction can sometimes be more involved. As M_2 satisfies only the third condition in the head of rule r_1, its optionality w.r.t. this rule has to take into account that the first two conditions are not satisfied, *i.e.*, $deg^+_{M_2}(r_1) = 2 + opt_{M_2}(not\ a \wedge not\ b)$. As shown below, the degrees of M_2 w.r.t. r_1 and r_2 correspond to $deg^+_{M_2}(r_1) = 3$ and $deg^+_{M_2}(r_2) = 2$ respectively, since:



The simplest case of r_3 is not represented as r_3 is clearly satisfied by degree 1 by both answer sets. Once the degrees are computed, the preference relation $\succ$ can be used to compare M_1 and M_2. It is easy to see how M_1 is preferred to M_2, since M_1 satisfies the rules of the program in a better way ($M_2 \not\succ M_1$ follows from Definition 5.16 as well).

8.4 Implementation Design

The LPODs$^+$ syntax is too complex to be handled by existent answer set solvers such as DLV, SMODELS, or GRINGO. In this section, we present a translation procedure which can be used to map any OD$^+$-program to an ordinary disjunctive logic program in order to compute its answer sets. We also sketch a method for computing the optimal answer set of an OD$^+$-program.

8.4.1 Mapping LPODs$^+$ to Disjunctive Logic Programs

Lifschitz *et al.* in [Lifschitz et al., 1999] already provides a procedure to translate nested logic programs into disjunctive ones by defining several equivalent transformations. We extend

this procedure by proposing a translation which maps any nested ordered disjunction rule into a nested rule.

In order to keep the presentation clear, we first fix the notation used throughout this section. We mainly deal with four classes of logic programs: LPODs$^+$, Nested Logic Programs ($NLPs$), Generalized Disjunctive Logic Programs ($GDLPs$) and Disjunctive Logic Programs ($DLPs$).[5]

Transformations sometimes introduce new literals in the signature of a program. For this reason, we explicitly use $LPODs^+_{\mathcal{L}}$ to denote the class of all nested ordered disjunction logic programs over alphabet $\mathcal{L}$; $NLPs_{\mathcal{L}}$ to denote the subclass of nested logic program over $\mathcal{L}$; furthermore, $GDLPs_{\mathcal{L}}$ stands for the subclass of $NLPs_{\mathcal{L}}$ containing all generalized disjunctive logic programs over $\mathcal{L}$; and $DLPs_{\mathcal{L}}$ is the class of all disjunctive logic programs over $\mathcal{L}$. In the following, we refer to a $GDLP_{\mathcal{L}}$ and to a $DLP_{\mathcal{L}}$ by $P^{\langle not,\vee\rangle}$ and $P^{\langle\vee\rangle}$ respectively.

The translation procedure can be described as follows. Given an OD$^+$-program $P \in LPODs^+_{\mathcal{L}}$, we perform the following steps:

1. transform each nested ordered disjunction rule into a nested rule: the resulting program is a nested logic program;
2. transform each rule in the nested logic program into the corresponding negative normal form, *i.e.,* rules made up of (possibly negated-by-failure) literals, conjunctions and disjunctions:
3. translate the program into a program containing only rules with conjunctions of (possibly negated-by-failure) literals in their bodies and disjunctions of (possibly negated-by-failure) literals in their heads;
4. eliminate double negations in bodies and heads: the resulting program is a generalized disjunctive logic program;
5. transform the generalized disjunctive logic program into an ordinary disjunctive logic program, *i.e.,* make all heads negation free.

Step 1 is realized using the transformation we propose in Proposition 8.4. Steps 2-4 are achieved by means of exhaustive application of program properties and distributive laws according to Lifschitz *et al.* [Lifschitz et al., 1999] (Proposition 8.5). Step 5 exploits a procedure due to Janhunen [Janhunen, 2001] (Definition 8.12).

Based on [Lifschitz et al., 1999] we follow the notion of formula equivalence according to which:

Definition 8.11. *[Lifschitz et al., 1999] A formula F is equivalent to a formula G ($F \equiv G$) if, for any (consistent) sets X and Y of literals, $X \models F^Y$ if and only if $X \models G^Y$.*

Moreover, two programs are said to be equivalent if they possess the same answer sets. Based on the above definition, we define the following equivalent transformation which maps a nested ordered disjunction rule into a nested rule.

Proposition 8.4. *For any formula A, B:*

$$A \times B \equiv A \vee (not\ A \wedge B)$$

[5] For the syntax of a DLP and a GDLP, please refer to Chapter 2, Sections 2.2.1 and 2.2.2.

Once we have obtained a nested logic program, we can reuse the transformations presented in [Lifschitz et al., 1999] to transform it into a generalized disjunctive logic program.[6]

Proposition 8.5. *[Lifschitz et al., 1999] For any formula A, B, C,*

1. $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$ *and* $(A \vee B) \vee C \equiv A \vee (B \vee C)$
2. $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
3. $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
4. $not\ (A \wedge B) \equiv not\ A \vee not\ B$
5. $not\ (A \vee B) \equiv not\ A \wedge not\ B$
6. $not\ not\ not\ A \equiv not\ A$
7. $A \leftarrow B \wedge not\ not\ C \equiv A \vee not\ B \leftarrow C$
8. $A \vee not\ not\ B \leftarrow C \equiv A \leftarrow not\ B \wedge C$
9. $A \wedge B \leftarrow C \equiv \begin{cases} A \leftarrow C \\ B \leftarrow C \end{cases}$
10. $A \leftarrow B \vee C \equiv \begin{cases} A \leftarrow B \\ B \leftarrow C \end{cases}$

By means of transformations in Proposition 8.4 and 8.5, it is possible to map a nested ordered disjunction logic program into an equivalent generalized disjunctive logic program.

Lemma 8.1. *Any OD^+-program $P \in LPODs^+_{\mathcal{L}}$ is equivalent to a generalized disjunctive program $P^{\langle not,\vee\rangle} \in GDLPs_{\mathcal{L}}$.*

Example 8.4. Let P be the OD^+-program in Example 8.1.[7] The corresponding generalized disjunctive logic program $P^{\langle not,\vee\rangle} \in GDLPs_{\mathcal{L}}$ is (r_3 is omitted).

$$a \times b \times (not\ a \wedge not\ b) \equiv \begin{cases} a \vee not\ a \\ a \vee b \vee not\ b \\ a \vee b \vee not\ a \end{cases} \qquad (c \vee d) \times (e \wedge \neg f) \leftarrow g \equiv \begin{cases} c \vee d \vee not\ c \leftarrow g \\ c \vee d \vee not\ d \leftarrow g \\ c \vee d \vee e \leftarrow g \\ c \vee d \vee \neg f \leftarrow g \end{cases}$$

Finally, we eliminate any remaining negation possibly occurring in the heads of rules. To this end a procedure due to Janhunen can be employed [Janhunen, 2001].

Definition 8.12. *[Janhunen, 2001] A generalized disjunctive logic program $P^{\langle not,\vee\rangle} \in GDLPs_{\mathcal{L}}$ can be mapped into a disjunctive logic program $P^{\langle\vee\rangle} \in DLPs_{\mathcal{L}\cup\mathcal{L}'}$ by setting:*

$$P^{\langle\vee\rangle} = P'^{\langle\vee\rangle} \cup \{\bot \leftarrow (p \wedge p'),\ p' \leftarrow not\ p \mid not\ p \text{ occurs in the head of some rules in } P^{\langle not,\vee\rangle}\}$$

in which $P'^{\langle\vee\rangle}$ results from $P^{\langle not,\vee\rangle}$ by replacing each occurrence of a literal $not\ p$ in the head of a rule in $P^{\langle not,\vee\rangle}$ by p'. $\mathcal{L}'$ is the set of the new literals introduced in $P'^{\langle\vee\rangle}$.

[6] We omit some transformations, for a complete list we refer to Proposition 7 and Proposition 8 in [Lifschitz et al., 1999].

[7] As done in Example 8.2 we have replaced the signature of the program from $\{italian, peruvian, pub, bar, cinema, tv, night\}$ to $\{a, b, c, d, e, f, g\}$.

Janhunen showed that replacements of the above kind lead to a transformation which is faithful, *i.e.*, it preserves the semantics of the transformed program. Therefore:

Lemma 8.2. *Any generalized disjunctive logic program $P^{\langle not,\vee\rangle} \in GDLPs_{\mathcal{L}}$ is equivalent to a disjunctive logic program $P^{\langle\vee\rangle} \in DLPs_{\mathcal{L}\cup\mathcal{L}'}$.*

Example 8.5. Let $P^{\langle not,\vee\rangle}$ be the *GDLP* in Example 8.4. The corresponding disjunctive logic program $P^{\langle\vee\rangle} \in DLPs_{\mathcal{L}\cup\mathcal{L}'}$ with $\mathcal{L}' = \{a', b', c', d'\}$ is:

$$P^{\langle\vee\rangle} = \left\{ \begin{array}{cccc} a \vee a' \leftarrow \top. & c \vee d \vee d' \leftarrow g. & a' \leftarrow not\ a. & \bot \leftarrow a \wedge a'. \\ a \vee b \vee b' \leftarrow \top. & c \vee d \vee e \leftarrow g. & b' \leftarrow not\ b. & \bot \leftarrow b \wedge b'. \\ a \vee b \vee a' \leftarrow \top. & c \vee d \vee \neg f \leftarrow g. & c' \leftarrow not\ c. & \bot \leftarrow c \wedge c'. \\ c \vee d \vee c' \leftarrow g. & g \leftarrow \top. & d' \leftarrow not\ d. & \bot \leftarrow d \wedge d'. \end{array} \right\}$$

The described procedure makes it possible to compute the answer sets of an OD$^+$-program by translating it into an equivalent disjunctive logic program as captured by the following result.

Theorem 8.1. *Let $P \in LPODs^+_{\mathcal{L}}$ and $P^{\langle\vee\rangle} \in DLPs_{\mathcal{L}\cup\mathcal{L}'}$ the disjunctive logic program obtained by transforming P. If M' is an answer set of $P^{\langle\vee\rangle}$ then $M = M' \cap \mathcal{L}$ is answer set of P.*

Example 8.6. Let P be the OD$^+$-program in Example 8.2 and $P^{\langle\vee\rangle}$ be the *DLP* in Example 8.5. It can be checked that $\forall M' \in SEM(P^{\langle\vee\rangle})$, $M = M' \cap \mathcal{L}$ is a answer set of P.

As far as complexity is concerned, it is known from [Eiter and Gottlob, 1995] that deciding whether a disjunctive logic program has at least one answer set is Σ_2^P-complete (propositional case). Unfortunately, the complexity of the transformations involved in Steps 2-4 can be exponential in the worst case due to the fact that these transformations rely on distributive laws [Pearce et al., 2002]. Later on, Pearce *et al.* [Pearce et al., 2002] proposed a technique based on the introduction of new atoms, called *labels*, abbreviating sub-formula occurrences, which avoids such an exponential blow-up and whose complexity is polynomial. Since our main construction (Step 1) occurs before any other step involved in the mapping to disjunctive logic programs, distributive laws in Steps 2-4 can be replaced by Pearce's translation procedure. Under this hypothesis, deciding whether an OD$^+$-program has at least one answer set is Σ_2^P-complete, since our construction is linear and any nested logic program can be faithfully reduced to a disjunctive logic program in polynomial time. Therefore, the implementation of a solver for LPODs$^+$ can be sketched as follows: after extending the compiler for nested logic programs (NLP[8]) to treat the $\times$ connector, the answer sets of LPODs$^+$ can be computed by a disjunctive logic programming system such as DLV.

8.4.2 Optimal Answer Set Computation

In order to compute the optimal answer set of an OD$^+$-program, a technique similar to the one adopted for ASO programs [Brewka et al., 2003] can be employed.

[8] `http://www.cs.uni-potsdam.de/~torsten/nlp/`

An ASO program is a pair $\langle P_{gen}, P_{pref} \rangle$, where P_{gen} is a logic program (of any type) called the generating program and P_{pref} is a collection of preference rules. The computation strategy followed in [Brewka et al., 2003] is based on a tester program that takes as input an answer set generated by P_{gen} and generates a strictly better one if such an answer set exists. If the tester fails to generate a strictly better one, then an optimal answer set is found. Otherwise, if a strictly better answer set is discovered, the tester is run with the new answer set as input. This continues until an optimal answer set is reached.

The translation procedure described in the previous section (or the one which can be built by generalizing the procedure in [Pearce et al., 2002]) already provides us with a method for computing the answer sets of an OD$^+$-program. Therefore, the disjunctive program $P^{\langle \vee \rangle}$ obtained by an OD$^+$-program P can be employed as the generating program. On the other hand, it is possible to create a tester $P_T^{\langle \vee \rangle}$ for P by adding specific testing rules to $P^{\langle \vee \rangle}$ in order to:

- compute the satisfaction degrees of an answer set w.r.t. the nested ordered disjunction rules;
- perform a comparison of two answer sets based on their satisfaction degrees.

In the following, we assume that for any given answer set M there exists an associated satisfaction vector $V_M = \{v_M(r_1), \ldots, v_M(r_n)\}$ in which each $v_M(r_i)$ is the rule satisfaction degree of M w.r.t. the rules in P. To compute the satisfaction degree of an answer set w.r.t. a rule r, each nested ordered disjunction rule can be translated into a set of normal rules (possibly nested) that explicitly states the satisfaction degrees w.r.t. the available options. We illustrate this idea by means of the following example:

Example 8.7. Let us consider $r_1 = a \times b \times (not\ a \wedge not\ b)$ and $r_2 = (c \vee d) \times (e \wedge \neg f) \leftarrow g$ in Example 8.1. These rules can be translated into:

$$
\begin{aligned}
v(r_1, 1) &\leftarrow a \\
v(r_1, 2) &\leftarrow not\ v_1(r_1, 1) \wedge b \\
v(r_1, 3) &\leftarrow not\ v_1(r_1, 2) \wedge (not\ a \wedge not\ b) \\
v(r_2, 1) &\leftarrow not\ g \\
v(r_2, 1) &\leftarrow g \wedge (c \vee d) \\
v(r_2, 2) &\leftarrow not\ v_1(r_2, 1) \wedge g \wedge (e \wedge \neg f)
\end{aligned}
$$

Each rule captures the satisfaction degree that corresponds to the formulas a given answer set satisfies.[9] For instance, for the answer sets $M_1 = \{a, c, g\}$ and $M_2 = \{e, \neg f, g\}$ (taken from Example 8.3) the above rules generate the sets $\{v(r_1, 1), v(r_2, 1)\}$ and $\{v(r_1, 3), v(r_2, 2)\}$ respectively.

Once we have obtained the satisfaction degrees, the comparison between two satisfaction vectors V_{M_0}, V_{M_1} can be defined as $geq(V_{M_1}, V_{M_0})$ if $v_{M_1}(r_i) \geq v_{M_0}(r_i)$ (for all $r_i \in P$ with $1 \leq i \leq n$). This comparison can be realized by the following rules:

$$
\begin{aligned}
&\leftarrow not\ better \\
better &\leftarrow v_0(R, V_0) \wedge v_1(R, V_1) \wedge geq(V_1, V_0) \wedge not\ geq(V_0, V_1) \\
&\leftarrow v_0(R, V_0) \wedge v_1(R, V_1) \wedge not\ geq(V_1, V_0)
\end{aligned}
$$

[9] Instead, the fourth rule captures the idea that if the body of the rule is not satisfied, then the degree of satisfaction is 1.

Finally, the optimal answer set computation can be sketched as follows. The tester takes as input an answer set M_0 and its satisfaction vector modeled as facts $v_0(r_i, deg^+_{M_0}(r_i))$ (for each r_i in the OD$^+$-program P). If the tester program $P_T^{\langle\vee\rangle}$ has an answer set M_1, then M_1 restricted to the original language of $P^{\langle\vee\rangle}$ is an answer set for P that is strictly preferred to M_0. If $P_T^{\langle\vee\rangle}$ does not have any answer sets, then no better answer set exists. It can be observed that the tester program can be a nested logic program. In such cases, we can reuse the translation procedure in Section 7.4, omitting Step 1, to convert the tester into a disjunctive logic program.

Considering the complexity of finding an optimal answer set, we can observe the following. First, determining whether an optimal answer set exists is not more difficult than determining whether any answer sets exist. Similar to [Kärger et al., 2008], complexity results of LPODs can be lifted by the Σ_2^P-completeness of disjunctive logic programs. Consequently, the existence of a preferred answer set of an OD$^+$-program has complexity Σ_2^P. Then, a similar lifting also works for checking whether an answer set M is optimal [Kärger et al., 2008]. Therefore, deciding whether an answer set M is an optimal answer set of an OD$^+$-program P is Π_2^P-complete.

8.5 Related Work

Preference handling in logic programming has been a hot investigation topic in the last two decades. Indeed, preferences are not only an effective way to determine a choice among different problem solutions, but they have also shown to have a strong connection with nonmonotonic reasoning as we stressed in Chapter 3. As such, many ASP extensions have been proposed [Delgrande et al., 2004; Osorio et al., 2004; Osorio and Zepeda, 2006; Osorio et al., 2011; Zepeda, 2005]. Besides providing an effective way to express an order among rules or among beliefs to be used in the reasoning process, preferences are also very important when user preferences have to be encoded in a compact way. As our work is situated in the latter world, we compare our approach with other works focused on the representation of user preferences.

Besides the LPODs approach, which is properly captured by LPODs$^+$, there are two main logic programming approaches which relate to ours: DLPODs [Kärger et al., 2008] and ASO programs [Brewka et al., 2003].

DLPODs has been proposed as an extension of LPODs to deal with ordinary disjunction $\vee$ in the head of rules. DLPODs makes it possible to specify preference equality statements such as $pub \times (cinema \vee tv) \leftarrow not\ sunny$.[10]

Since our semantics applies to any $\{\vee, \wedge, not, \times\}$ formula, it should also provide a semantics for DLPODs. Therefore, the interesting question is whether our semantics and the DLPODs semantics coincide. Unfortunately, this is not the case as shown by the following example [Cabalar, 2011].

Example 8.8. Let us consider a DLPOD P consisting of rules $r_1 = pub \times (cinema \vee tv) \leftarrow not\ sunny$ and $r_2 = tv$. Then the ODNF of r_1 is $(pub \times cinema) \vee (pub \times tv) \leftarrow not\ sunny$

[10] An overview about DLPODs can be found in Chapter 6, Section 6.4.2.

obtained by application of rewriting rule (i).[11] Then, P has the following four split programs:

$$P_1 = \begin{Bmatrix} pub \vee pub \leftarrow not\ sunny \\ tv \quad \leftarrow \quad \top \end{Bmatrix} \qquad P_3 = \begin{Bmatrix} cinema \vee pub \leftarrow not\ sunny \wedge not\ pub \\ tv \quad \leftarrow \quad \top \end{Bmatrix}$$

$$P_2 = \begin{Bmatrix} pub \vee tv \leftarrow not\ sunny \wedge not\ pub \\ tv \quad \leftarrow \quad \top \end{Bmatrix} \quad P_4 = \begin{Bmatrix} cinema \vee tv \leftarrow not\ sunny \wedge not\ pub \\ tv \quad \leftarrow \quad \top \end{Bmatrix}$$

The DLPOD P has three valid answer sets, since $M_1 = \{pub, tv\}$, $M_2 = \{tv\}$, $M_3 = \{cinema, tv\}$ are the answer sets of its split programs. Instead, under LPODs$^+$ semantics the only admissible answer sets are M_1 and M_2.

Although the DLPODs semantics can generally yield more answer sets than the LPODs$^+$ semantics, some solutions under DLPODs semantics look rather counterintuitive. For instance, answer set M_3 in the above example satisfies the second choice with both literals even if the regular disjunction $\vee$ becomes satisfied when any of its disjuncts are true. Another counterintuitive example of the DLPODs semantics can be given by a program of the form: $\{r_1 = pub \vee (cinema \times tv) \leftarrow not\ sunny, r_2 = tv\}$. In such a case, $\{pub, tv\}$ is an admissible answer set under DLPODs semantics, even if once tv is true and we do not choose $cinema$, it is clear that $(cinema \times tv)$ holds and there is no clear reason to make pub true. Also in this case, the counterintuitive solution $\{pub, tv\}$ is not admissible under LPODs$^+$ semantics. The reason for this discrepancy is that rewriting rule (ii)[12], which is employed in DLPODs, is not allowed in our setting because it leads to pairs of formulas which are not equivalent. In fact, how the $\times$ operator irregularly behaves w.r.t. some distributive properties of conjunction and disjunction can be checked. In particular, $\times$ distributivity w.r.t. conjunction is not satisfied for the following pair of formulas $(A \times B) \wedge C$, $(A \times B) \wedge (A \times C)$, while distributivity w.r.t. disjunction only holds for the following formulas $A \times (B \vee C)$, $(A \times B) \vee (A \times C)$ (as formally studied in [Cabalar, 2011]).

ASO has been proposed by Brewka *et al.* in [Brewka et al., 2003] as a decoupled approach for representing and handling preferences in logic programming.[13] Our approach differs from ASO programs, since in the latter, answer set generation and comparison are separated. Then, the preference handling is independent of the type of program used for the answer set generation. In fact, in ASO programs preference rules have no role in generating answer sets; while, in our approach, preference rules affect which sets are and are not answer sets.

Among quantitative approaches to preference specification, the work of Costantini *et al.* [Costantini and Formisano, 2009], in which an extension of ASP is designed to model quantitative resources and to enable the specification of non-linear preferences (both in the heads and in the bodies), is worth mentioning. This approach has also been applied to pure

[11] $a \times (b \vee c) = (a \times b) \vee (a \times c)$.
[12] $(a \vee b) \times c = (a \times c) \vee (b \times c)$.
[13] An overview about ASO program can be found in Chapter 6, Section 6.4.3.

ASP in order to capture weight constraints with preferences [Costantini and Formisano, 2011].

Finally, other noticeable approaches, treating the problem of conditional (qualitative) preference specification, are the works in the possibility theory [Benferhat et al., 2004, 2001; Bosc et al., 2010] and in the CP-nets [Boutilier et al., 2004; Wilson, 2004] settings. Furthermore, for a more general overview about preferences in AI, we refer to [Domshlak et al., 2011; Kaci, 2011].

8.6 Discussion and Concluding Remarks

In this chapter, we have defined the syntax and the semantics of Nested Logic Programs with Ordered Disjunction (LPOD$^+$). LPODs$^+$ is a logic programming framework that allows the specification of nested conditional preference expressions. The LPODs$^+$ syntax is based on formulas built from connectives $\{\vee, \wedge, not\}$ plus an ordered disjunction connective $\times$ which was first introduced in QCL and then used in LPODs. As a result, rules in LPODs$^+$ are more general than rules which can be specified in related approaches of qualitative context-dependent preferences between literals such as LPODs and DLPODs.

We have shown how the LPODs$^+$ semantics can be defined in a concise way (Definitions 8.4, 8.5 and 8.6) and how it can offer an alternative characterization of the LPODs semantics (which is based on split programs). Indeed, when an OD$^+$-program syntactically corresponds to an LPOD, the LPODs and LPODs$^+$ semantics coincide (Proposition 8.2). Moreover, when the formulas of an OD$^+$-program are $\times$ free, the LPODs$^+$ semantics coincide with the semantics of nested logic programs (Proposition 8.1). As LPODs$^+$ supports the specification of complex formulas in the head of a rule, we have characterized the satisfaction degree of an answer set w.r.t. a nested ordered disjunction rule (Definition 2.6) by means of a recursive optionality function (Definition 8.7). The optionality function generalizes the satisfaction degree defined for LPODs (Proposition 8.3). The answer sets of an OD$^+$-program are compared by means of a comparison criterion that generalizes the Pareto-based LPODs criterion. As a consequence, our approach generalizes the LPODs framework in a proper way.

Considering the computation of the LPODs$^+$ semantics, we have designed a translation procedure for inferring the answer set of an OD$^+$-program by inferring the answer sets of its equivalent disjunctive logic program (Theorem 8.1). This is an important result, since the complexity added at the syntactic level does not interfere with the computation of the LPODs$^+$ semantics whose complexity class stays in the complexity class of disjunctive logic programs, *i.e.*, Σ_2^P. An implementation design for a tester program for the computation of the optimal answer set of an OD$^+$-program has also been described.

To summarize, by defining the LPODs$^+$ framework, we have put the basis for extending the LPPODs framework, and, in principle, the user preference handling we adopted in Chapter 10. Indeed, an implementation of the LPODs$^+$ framework can support the specification of more complex preference expressions. This is a desired feature to add in a context-aware system such as ICD which can require to represent more complex user preferences.

Part III

Preferences in Decision under Uncertainty

9

Qualitative Decision Making Under Uncertainty

The formulation of classical decision theory relies on a probabilistic framework. The ranking of decisions is done according to the maximization of an expected utility function in terms of probabilities, modeling uncertainty about states, and utilities, modeling preferences over consequences of these decisions. Although this classical approach is satisfactory in economics frameworks, new decision making scenarios which require a more qualitative handling of information, have shifted the paradigm of decision theory from a quantitative to a qualitative representation of uncertainty and preferences. As such, qualitative decision making frameworks have appeared in the AI literature.

The chapter is organized as follows. After introducing some basic knowledge about classical decision theory and introducing some of the motivations behind qualitative decision making (Section 9.1), in Section 9.2, we present the possibilistic semantics for making an optimal decision in terms of a pessimistic and an optimistic criteria. In Section 9.3, the classical and possibilistic logic views of a decision problem and a decision making problem under uncertainty are described. Section 9.4 overviews some logic programming approaches for modeling qualitative decision. Finally, in Section 9.5, we discuss the need of a different logic programming methodology, closer to possibilistic logic, for modeling decision problems under uncertainty and for computing optimal decisions in the sense of the possibilistic criteria.

9.1 Classical Decision Theory

Decision making, in its basic instance, can be formulated as the problem of selecting an optimal decision among the possible ones, taking into account the consequences decisions can produce and the preferences expressed about the consequences themselves.

More formally, in a decision problem, given a set S of *states* of the world and a set X of *consequences*, a *decision* d is represented by a function d from S to X. Preferences among consequences are modeled by an utility function u from X to a numerical scale U, where U represents a preference scale.

In this decision process, uncertainty can usually appear in different stages: either the decision functions are precisely defined, but the real situation is not fully known; or the real situation is known, but the consequences of decisions are not. Therefore, decision making

frameworks able to deal with uncertainty have been proposed [von Neumann and Morgenstern, 1944; Savage, 1972]. Classical approaches to decision making under uncertainty (DMU) assume that uncertainty is handled by means of a probabilistic approach. For instance, if a probability distribution p on the set of situations is assumed to be known, and a numerical utility u is assigned to consequences, then decisions can be ranked according to an *expected utility function* defined as (finite case):

$$u(d) = \sum_{s \in S} p(s)u(d(s)) \tag{9.1}$$

According to 9.1, a decision d is strictly preferred to a decision d' if and only if $u(d) > u(d')$. The ranking of decisions according to the expected utility of the consequences of these decisions was a proposal made by Neumann and Morgenstern [von Neumann and Morgenstern, 1944] in the 1950's. Later on, Savage [Savage, 1972] proposed an axiomatic framework of a theory of subjective expected utility, in which only a decision rule had a quantitative nature, while preferences and uncertainties were derived from a total ordering of decisions.

However, these classical approaches have several limitations [Doyle and Thomason, 1999]. First, an expected utility approach needs numerical probabilities for each state and numerical utilities for all possible consequences. Unfortunately, such functions and distributions are sometimes not always directly accessible or they are difficult to obtain. Then, since preferences and knowledge are often human-oriented, a qualitative representation of information is more adequate than a quantitative one as assumed in classical decision theory. On the other hand, the emergence of new information technologies has generated many new decision problems involving intelligent systems and it was not clear that a classical decision theory approach, which was developed in the framework of economics, was fully adapted to these new problems. The above limitations and new decision making scenarios fostered the appearance of several frameworks of *qualitative decision making*.

9.2 Qualitative Decision Making

Qualitative decision making under uncertainty (DMU) relies only on a qualitative representation of uncertainty and preferences. As such, a decision problem is formulated in a qualitative way, in which knowledge and preferences have the form of human-like expressions. Decision rules for choosing among decisions no longer relies on probability theory nor numerical utility functions [Doyle and Thomason, 1999]. However, the term qualitative decision making can refer to more than one kind of representation. Several qualitative decision making frameworks have been proposed in the AI literature including [Boutilier, 1994; Tan and Pearl, 1994a,b; Dubois and Prade, 1995; Bonet and Geffner, 1996; Brafman and Tennenholtz, 1996]. Among them, some works consider all-or-nothing notions of utility and plausibility [Bonet and Geffner, 1996], while other works still use sets of integers to describe rough probabilities or utilities [Tan and Pearl, 1994a,b]. In the following, we describe the possibilistic approach for qualitative decision in which a commensurateness

hypothesis between uncertainty and utility scales is assumed. We also point out others approaches which relax this assumption.[1]

9.2.1 Possibilistic Semantics of Qualitative Decision

A purely possibilistic approach to qualitative DMU has been proposed in [Dubois et al., 2001; Dubois and Prade, 1995]. According to this approach, preferences and uncertainty are graded on ordinal linearly ordered scales which are commensurate. In particular, two sets of postulates for qualitative decision have been proposed that turn to be respectively equivalent to the maximization of a pessimistic criterion and of an optimistic one [Dubois et al., 2001; Dubois and Prade, 1995]. These two criteria are respectively estimating the necessity and the possibility that a sufficiently satisfactory state is reached.

When uncertainty and preferences are matters of degrees, possibility distributions, which map a set of interpretations to a scale, are a convenient way for expressing complete pre-orderings. In particular, uncertain knowledge and preference can be represented in terms of two different possibility distributions, π and μ, which respectively map the set of states S and the set of consequences X to a scale $\mathcal{S}$ [Dubois et al., 2001; Dubois and Prade, 1995]. The scale $\mathcal{S}$ is made of $n+1$ levels $\alpha_1 = \mathbf{1} > \alpha_2 > \ldots > \alpha_n > \alpha_{n+1} = \mathbf{0}$. Certainty and priority degrees are commensurate by an order reversing map n of scale $\mathcal{S}$ such that $n(\alpha_i) = \alpha_{n+2-i}$ for $i = 1, \ldots, n+1$.

The intuitive reading behind the two possibility distributions is that: the higher $\pi(s)$, the more plausible d as being the real state of the world; the greater $\mu(x)$, the more acceptable x as a consequence. Therefore, the utility of a decision d whose consequence in state s is x (for all states s) can be evaluated by combining the possibilities $\pi(s)$ and utilities $\mu(x)$ in a suitable way. In this setting, a pessimistic and an optimistic criteria which evaluate the goodness of a decision d have been proposed in the literature:

$$u_*(d) = \min_{s \in S} \max(n(\pi(s)), \mu(f(s))) \tag{9.2}$$

$$u^*(d) = \max_{s \in S} \min(\pi(s), \mu(f(s))) \tag{9.3}$$

where n is the order reversing mapping defined above.

Maximizing $u_*(d)$ means to look for a decision d whose highly plausible consequences are among the most preferred ones. This pessimistic utility is small as soon as it exists a possible consequence of d which is highly plausible and bad w.r.t. the preferences. The pessimistic criterion corresponds to a risk-averse attitude. On the other hand, the other utility function $u^*(d)$ corresponds to an optimistic, risk-prone, attitude, since its value is high as soon as there exists a possible consequence of d which is both highly plausible and highly preferred.

[1] For a general discussion about qualitative decision theory from different perspectives, we refer to [Doyle and Thomason, 1999].

9.2.2 Other Approaches

Boutilier proposed a nonmonotonic logic based on possible worlds semantics which allows to represent and reason with qualitative probabilities and preferences [Boutilier, 1994]. According to this logic, conditional defeasible preferences of the form $I(\alpha \mid \beta)$, *i.e., if* α, *then ideally* β, are given a semantics in terms of preference ordering on possible worlds. Since the scales of normality and preference are incomparable, preferential inference of nonmonotonic reasoning can be exploited to make decisions on the basis of the most plausible states.

The case in which uncertainty and utility scales are incomparable was also explored in the possibilistic logic setting [Dubois et al., 2002]. Under this assumption, a decision procedure based on the *likely dominance rule* was drawn. Similar to nonmonotonic reasoning, this rules assumes that one of the most normal situations compatible with the available information prevails. As such, optimal decisions were made on the basis of the most plausible states of the world. Although the commensurability hypothesis between uncertainty and utility scales can be found too demanding, without this hypothesis, the obtained decision rule is either not very decisive or it may lead to very risky decisions [Dubois et al., 2002]. Consequently, commensurateness is a requirement for effective decision rules.

Another noticeable approach is Brafman and Tenenholtz's axiomatization of pessimistic attitude to decision making [Brafman and Tennenholtz, 1996].

9.3 Modeling Qualitative Decision in Possibilistic Logic

In possibilistic logic [Dubois et al., 1994], a possibility distribution encodes the semantics of a possibilistic logic base. A set of classical formulas can be gathered into several layers according to their levels of certainty or priority. Then, the semantics of a possibilistic logic base can be expressed by means of a function encoding an ordering on the set of possible interpretations associated with the language. This ordering can either express certainty or priority.

In this section, we will describe how qualitative DMU has been formulated in the possibilistic logic setting, in which knowledge and preferences are expressed in terms of stratified knowledge bases. In particular, in [Dubois et al., 1999], it is shown how computing a pessimistic and an optimistic utility (9.2 and 9.3) amounts to solve two maximization problems.

First, the classical case, which shows how a decision problem can be stated in a way similar to abductive diagnosis is presented.

9.3.1 Qualitative Decision in Propositional Bases

The logical view of a decision problem can be stated in the following way. Let K be the knowledge base describing what is known about the world, D be the set of decision literals, and P be another base describing goals delimiting preferred states of the world.

Then, a decision, defined as a conjunction d of decision literals such that:[2]

[2] $\vdash_{CL}$ denotes the logical consequence of classical logic.

$$K \wedge d \vdash_{CL} P \tag{9.4}$$

(with $K \wedge d \neq \bot$) is for sure a good decision (if it exists), since it makes certain that all the goals are satisfied. Looking only for such a decision corresponds to a cautious, pessimistic, attitude.

A much more optimistic attitude would correspond to consider also potential decisions d such that:

$$K \wedge d \wedge P \neq \bot \tag{9.5}$$

which expresses that the possibility of satisfying all the goals remains open.

These two points of view can be extended to the case where K and P are possibilistic logic bases [Dubois et al., 1994], *i.e.*, when uncertainty and preferences are matters of degrees.

9.3.2 Qualitative Decision in Stratified Propositional Bases

When uncertainty and preferences are matters of degrees, K is a set of more or less certain pieces of knowledge and P is a set of goals with associated levels of priority. The certainty and priority levels are supposed to belong to the same linearly ordered scale $\mathcal{S}$ and are commensurate by means of the order reversing map n.

Then, given K_α as the set of formulas in K having certainty at least equal to α *without* their certainty levels, and $P_{\underline{\beta}}$ as the set of goals having a priority strictly greater than β *without* their priority levels, the exact counterpart of the pessimistic and optimistic criteria (expression 9.2 and 9.3), when the knowledge and the preferences are expressed under the form of two possibilistic knowledge bases, have been shown in [Dubois et al., 1999] to correspond to the following definitions:

Definition 9.1. *The pessimistic utility $u_*(d)$ of a decision d is the maximal value of $\alpha \in [0,1]$ such that*

$$K_\alpha \wedge d \vdash_{CL} P_{\underline{n(\alpha)}} \tag{9.6}$$

Definition 9.2. *The optimistic utility $u^*(d)$ of a decision d is the maximal value of $n(\alpha) \in [0,1]$ such that*

$$K_{\underline{\alpha}} \wedge d \wedge P_{\underline{\alpha}} \neq \bot \tag{9.7}$$

with the convention $\max \emptyset = \mathbf{0}$. The intuition below $u_*(d)$ is that we are interested in finding a decision d (if it exists) such that $K_\alpha \wedge d \vdash_{CL} P_{\underline{\beta}}$ with α high and β low, *i.e.*, such that the decision d together with the most certain part of K entails the satisfaction of the goals, even those with low priority. Taking $\beta = n(\alpha)$ requires that the certainty and priority scales be commensurate. The optimistic utility can be understood in a similar way.

The computation of pessimistic and optimistic decisions has been explored in [Dubois et al., 1999] in the context of Assumption Truth Maintenance Systems (ATMSs).

9.4 Modeling Qualitative Decision in Logic Programming

There is not so much work in the logic programming literature addressing qualitative decision problems. However, existing approaches [Brewka, 2005; Grabos, 2004] are based on LPODs (Chapter 2, Section 2.2.4).

9.4.1 Answer Sets and Qualitative Decision Making

Brewka in [Brewka, 2005] proposed a general methodology for qualitative decision making based on LPODs [Brewka et al., 2004b]. The basic idea is to encode the elements of a decision making problem, *i.e.,* generic knowledge about the states of the world, possible decisions, their consequences and preferences, by means of an LPOD. According to this approach, generic knowledge, decisions, and consequences are modeled by extended normal rules, while preferences are represented by ordered disjunction rules. In this way, solutions to the decision problem correspond to those answer sets which are consistent with the encoded knowledge.

The use of ordered disjunction induces a partial order on the answer sets. Based on this order, an order on the available decisions can be specified according to some comparison criteria, *e.g.,* cardinality, inclusion, or Pareto (Chapter 2, Section 2.2.4). However, in a decision making setting, it is not always sufficient to consider the most preferred answer sets only. Indeed, this represents an extremely optimistic attitude. Moreover, uncertainty is not explicitly represented and all states which belong to the answer sets are considered plausible. This means that no distinction between the degrees of plausibility of the states can be made, and no further distinctions between the generated answer sets are possible.

For this reason, other selection strategies based on the order among the answer sets have to be defined a posteriori in order to choose an optimal decision. For instance, an optimistic selection strategy might reason from maximally preferred answer sets, choosing a decision d over another decision d' if the most preferred answer sets containing d are better than the most preferred answer sets containing d'. Instead, a pessimistic selection strategy might choose a decision whose worst outcome is most tolerable, *i.e.,* reasoning from less preferred answer sets and selecting a decision d over d' if the least preferred answer sets containing d are better than the least preferred answer sets containing d'.

9.4.2 Qualitative Model of Decision Making

In [Grabos, 2004], a generalization and extension of the work of Brewka [Brewka, 2005] is presented. Similar to [Brewka, 2005], a decision making problem is modeled by means of an LPOD, in which extended normal rules encode general knowledge about the world, decisions and their consequences. The main difference relies on the use of ordered disjunction rules. According to this approach, ordered disjunction rules represent two dimensions: the plausibility about the states of the world (called belief rules) and the preferences over the consequences of decisions.

The solutions of the decision problem have still the forms of answer sets of the LPOD program, but answer sets can also be ranked according to the plausibility of states rather

than only according to the preferences over the consequences. This allows to define different selection strategies which can correspond to an optimistic and a pessimistic selection criterion.

If a commensurability assumption between the plausibility and preference order is not taken, then different attitudes can be used to select an optimal answer set and, consequently, an optimal decision. When plausibility of states is assumed to be more important than preferences, answer sets are compared on the basis of decisions with the most preferred consequences in the most likely states. On the contrary, answer sets are compared on the basis of decisions with the highest degree of plausibility with the most preferred consequence.

Instead, if a commensurability assumption between the plausibility and preference order is assumed, then selection strategies which correspond to a pessimistic and an optimistic criterion can be provided. A pessimistic optimal decision will be a decision made among those answer sets whose decisions lead to the best among the worst consequences of all decisions. On the contrary, an optimistic optimal decision will be a decision made among those answer sets whose decisions lead to the highest plausible states among all the most preferred consequences.

Although this approach looks closer to qualitative DMU than [Brewka, 2005], it lacks a formal justification of the optimistic and pessimistic attitude. Indeed, answer sets selection is based on empirical criteria.

9.5 Discussion

Existing ASP-based methodologies for handling qualitative DMU [Brewka, 2005; Grabos, 2004] amount to compile a decision problem as a logic program able to generate the space of possible decision solutions and to specify an order between them by means of the ordered disjunction connective $\times$. Although such approaches are enough to cover decisions in completely certain environment, they become less effective when the knowledge is pervaded with uncertainty.

The DMU problem with qualitative preferences and uncertainty has been studied in the setting of possibility theory assuming a commensurateness hypothesis between the level of certainty and the preference priority. As in classical utility theory, pessimistic and optimistic criteria have been proposed and justified on the basis of postulates [Dubois et al., 2001]. This approach has been adapted in the setting of possibilistic propositional logic in which the available knowledge is described by formulas which are more or less certainty true and the goals are described in a separate prioritized propositional base.

For this reason, it is interesting to look for a different logic programming methodology, closer to possibilistic logic, for modeling decision problems and for computing optimal decisions in the sense of the possibilistic criteria. To this end, a logic programming specification able to deal with necessity values (according to possibilistic logic) should be considered. On the other hand, the similarity between decision and abduction problems, as already stressed in [Sabbadin, 1998], is striking and logic programming can be also used to model abduction problems [Eiter et al., 1999; Brewka, 2006]. In particular, in [Brewka,

2006] it is shown how LPODs can provide an elegant modeling of abduction problems. Therefore, it is natural to try to model decision making problems in terms of abduction and by means of LPODs. When decisions are matters of uncertainty and preferences degrees, like it happens in possibilistic logic, the possibilistic extension of LPODs, Logic Programs with Possibilistic Ordered Disjunction (presented in Chapter 5) can be used to handle both a knowledge base pervaded with uncertainty and a prioritized preference base. Then, a logic programming counterpart to the computation of the possibilistic decision criteria can be obtained by applying both a classic and a possibilistic ASP-based methodology. This methodology is proposed in Chapter 10.

10

An ASP-based Methodology for Computing Decision Under Uncertainty

The capability of computing decision under uncertainty represents a desirable and essential feature in intelligent systems that assist users in taking decisions. Unfortunately, the computation of an optimal decision is not an easy task. Indeed, it amounts to find a good tradeoff between the uncertainty about the states of the world and the preferences about the consequences of a decision.

The formalization of qualitative DMU has been a hot investigation topic for a long time. Several logic-based frameworks have been proposed in the literature such as the works in [Boutilier, 1994; Dubois and Prade, 1995; Brafman and Tennenholtz, 1996]. In particular, possibilistic logic offered a qualitative framework for modeling decision under uncertainty [Dubois and Prade, 1995]. In this setting, pessimistic and optimistic decision criteria have been formally justified.

On the other hand, not too much work can be found in the logic programming literature. The most noticeable approaches are the LPODs-based proposals in [Brewka, 2005; Grabos, 2004]. Although such approaches can capture preferences, they lack a satisfactory handling of uncertainty.[1] To this end, Possibilistic Answer Set Programming (PASP) can offer a qualitative framework for handling uncertain information. As a consequence, it is reasonable to think that the LPPODs framework, the possibilistic extension of LPODs, can be employed for the computation of optimal decisions under uncertainty.[2] Unfortunately, LPPODs cannot be used in a direct way. First, LPPODs handles uncertainty and preferences together, whereas in decision making uncertain knowledge and prioritized preferences are handled in two different knowledge bases. Secondly, uncertainty and preference degrees must be commensurate in a proper way in order to provide effective decision rules. These issues suggest to look for an alternative LPPODs-based construction.

In this chapter, we present an ASP-based methodology for modeling qualitative DMU and for computing an optimal decision in the sense of the possibilistic criteria. In offering the logic programming counterpart to the computation of the decision criteria, we formulate qualitative DMU in terms of abduction. In this way, we can handle both a knowledge base pervaded with uncertainty and a prioritized preference base. Concerning the computation of the decision criteria, we will see that both a classic and a possibilistic ASP-based

[1] For this discussion, please refer to Chapter 9.

[2] The LPPODs framework has been presented in this document in the context of nonmonotonic reasoning under uncertainty (see Chapter 5).

methodology can be defined. Throughout the chapter, we will use the following running example to exemplify our approach:

Example 10.1. An agent is supposed to know that (i) *if I have an umbrella then I will be not wet*; *if it rains and I do not have an umbrella, then I will be wet*; and *typically if it is cloudy it will rain* (this rule is uncertain) ii) it is known that the sky is cloudy, and iii) being not wet is more important than not carrying an umbrella. The problem then is to decide whether or not to take an umbrella.

The chapter is organized as follows. In Section 10.1, we address the decision problem in ASP by proposing a modeling based on abduction. To this end, we present a methodology to compute optimal minimal inclusion-set decisions when knowledge is fully certain and preferences are all-or-nothing. In Section 10.2, we cope with the general case, *i.e.,* when certainty and preferences are matters of degrees. We propose a classic and a possibilistic ASP-based methodology based on the LPPODs framework for computing optimal decisions and utility values. In Section 10.3, we compare the proposed approach to previous works. Finally, in Section 10.4, we present some concluding remarks and discussions.

10.1 Modeling Decision Making in ASP

Similarly to what happens in the logical view of qualitative decision making (see Chapter 9), we assume that knowledge about the world and preferences are separated. Considering this separation, the similarity between decision making and abduction is striking [Sabbadin, 1998]. Then, it is natural to want to model qualitative decision making in terms of abduction.

In logic programming, several abductive frameworks [Kakas et al., 1992; Eiter et al., 1997; Denecker and Kakas, 2002] and several ways for computing abduction problems have been proposed [Eiter et al., 1999; Brewka, 2006]. A framework of abductive logic programming was established and made popular by the survey paper [Kakas et al., 1992]. More recently, some other abductive frameworks have been proposed [Eiter et al., 1999; Brewka, 2006]. For instance, Eiter *et al.* [Eiter et al., 1999] has modeled abduction in the context of disjunctive logic programs (and the DLV system), while Brewka [Brewka, 2006] has shown an encoding by means of LPODs.

10.1.1 Abduction and LPODs

According to [Eiter et al., 1999; Brewka, 2006], an abductive framework is a triple $\langle P, H, O\rangle$ in which P is a logic program, H is a set of facts, referred to as *hypotheses*, and O is a set of literals, referred to as *observations*. Generally speaking, abduction is the process of generating explanations for a set of observations. Then, a set $\Delta \subseteq H$ is a minimal explanation of O w.r.t. P if there exists an answer set of $P \cup \Delta$ which contains O (credulous reasoning).[3] An explanation Δ_i is minimal, if for every other explanation Δ_j ($i \neq j$) of O, $\Delta_j \not\subset \Delta_i$ holds.

[3] In the literature, some definitions of abduction are based on cautious reasoning, *i.e., O* must be contained in all answer sets of $P \cup \Delta$.

Therefore, in logic programming, an abduction problem is cast in the problem of finding a minimal set $\Delta \subseteq H$ such that $P \cup \Delta \models_c O$.[4] Brewka [Brewka, 2006] has shown that, given an abduction problem $\langle P, H, O \rangle$, an explanation for O can be computed via the following construction:

$$P_{abd}(P,H,O) = P \cup \{\leftarrow not\ o \mid o \in O\} \cup \{\neg ass(h) \times ass(h) \mid h \in H\} \cup \{h \leftarrow ass(h) \mid h \in H\}$$

where $ass(h)$ reads h is assumed. Using the above construction, it can be proved that the set of explanation literals belonging to a consistent maximally preferred answer set of an LPOD corresponds to a (minimal) explanation for the abduction problem. This result was formally captured by the following proposition.

Proposition 10.1. [Brewka, 2006] *Let H and O be sets of literals and let P be a logic program. Δ is an explanation for O given P and H iff there exists a consistent maximally preferred answer set M of $P_{abd}(P,H,O)$ such that $\Delta = \{h \in H \mid ass(h) \in M\}$.*

In principle, several minimal explanation sets may exist. It can be observed that the minimality condition is guaranteed by the way in which $\times$ rules in $P_{abd}(P,H,O)$ are constructed.

10.1.2 Fully Certain Knowledge and All-or-nothing Preferences

In the following, we assume that the knowledge about the world is encoded by an extended positive logic programs and preferences by singleton literals.[5] Then, we define:

Definition 10.1. *A decision making problem DM is represented as a tuple $\langle K, D, Pref \rangle$ where*

- *K is an extended positive logic program representing the knowledge about the world,*
- *$D = \{d_1, \ldots, d_m\}$ is the set of decision literals, and*
- *$Pref = \{p_1, \ldots, p_n\}$ is the set of goal or preference literals.*

Example 10.2. Let us consider the decision problem in Example 10.1 without any uncertainty and keeping all the goals as equally important. By denoting *taking an umbrella* by u, *not taking an umbrella* by $\neg u$, *being wet* by w, *not being wet* by $\neg w$, *it rains* by r, and *it is cloudy* by c, the decision problem can be modeled as follows:

$$K = \begin{Bmatrix} r_1 : \neg w \leftarrow & u. \\ r_2 : & w \leftarrow r, \neg u. \\ r_3 : & r \leftarrow & c. \\ r_4 : & c \leftarrow & \top. \end{Bmatrix} \qquad D = \Big\{u, \neg u\Big\} \qquad Pref = \Big\{\neg w, \neg u\Big\}$$

As in the case of the logical view of a decision problem (see Chapter 9, Section 9.3.1), we can define optimal decisions according to a pessimistic and an optimistic criterion.

[4] $\models_c$ has been formally introduced in Chapter 2, Section 2.1.3.

[5] A possible extension will be to consider preferences as propositional formulas and to extend $\models_c$ to any propositional formula like it happens in ASO programs [Brewka et al., 2003].

When K expresses completely certain knowledge and $Pref$ are all-or-nothing, optimal decisions, according to a pessimistic point of view, are decisions that, in conjunction with the knowledge, lead to the satisfaction of all the preferences. Based on the above definitions, the pessimistic criterion can be formulated as follows.

Definition 10.2. *Given a $DM = \langle K, D, Pref \rangle$, a pessimistic optimal decision is a minimal set of decision literals $\Delta \subseteq D$ such that $K \cup \Delta \models_c Pref$.*

In general, more than one minimal set of decisions may exist for a set of preference literals $Pref$. Therefore, we introduce the definition of *label* of $Pref$.

Definition 10.3. *Given a $DM = \langle K, D, Pref \rangle$, the label of $Pref$, denoted by $label_K(Pref)$, is the set of all $\Delta_i \subseteq D$ minimal for set-inclusion such that $K \cup \Delta_i \models_c Pref$.*

As far as an optimistic point of view is concerned, an optimistic optimal decision is a set of decision literals $\Delta \subseteq D$ such that Δ is consistent with the knowledge and the preferences. The optimistic criterion can be defined as:

Definition 10.4. *Given a $DM = \langle K, D, Pref \rangle$, an optimistic optimal decision is a minimal set of decision literals $\Delta \subseteq D$ such that $K \cup \Delta \cup Pref$ is consistent.*

Please observe how the above definition is less restrictive than the pessimistic one. It only requires that $K \cup \Delta \cup Pref$ is consistent.

10.1.3 Computation of Decisions

Based on the above modeling and on the abduction modeling in Section 10.1.1, the computation of an optimal decision can be done translating a decision making problem DM into an LPOD. As we will see, pessimistic and optimistic decision criteria require a different construction.

Computation of Pessimistic Optimal Decisions

Let us start with the computation of the pessimistic criterion. To this end, we define the following translation where the main construction is borrowed from [Brewka, 2006].

Definition 10.5. *Given a $DM = \langle K, D, Pref \rangle$, a decision Δ for DM can be computed via an LPOD $P_{dm}(\langle K, D, Pref \rangle) = K \cup \{\leftarrow not\ p \mid p \in Pref\} \cup \{\neg ass(d) \times ass(d) \mid d \in D\} \cup \{d \leftarrow ass(d) \mid d \in D\}$, where $ass(d)$ reads d is assumed.*

The computation of an optimal pessimistic decision is shown in Algorithm 5. The algorithm accepts a decision making problem DM and returns the label of $Pref$ together with its pessimistic utility value. The behavior of the algorithm can be explained as follows.

The abstract DM is converted into an LPOD P_{dm} according to the method `decisionMakingToLPOD(DM)` which implements Definition 10.5. In the resulting LPOD P_{dm}, the use of ordered disjunction rules generates all the possible combinations of decisions, while the use of constraints eliminates those answer sets in which preferences are not satisfied. As such, once the answer sets of P_{dm} are computed ($SEM_{LPOD}(P_{dm})$), the optimal set of

Algorithm 5 computePessimisticDecisions(DM) : $\langle label_K(Pref), u_* \rangle$

Input: $\begin{cases} A\ DM = \langle K, D, Pref \rangle \end{cases}$

Output: $\begin{cases} label_K(Pref) : \text{optimal decisions} \\ u_* : \text{pessimistic utility} \end{cases}$

```
label_K(Pref) ← {}; u_* ← 0;
P_dm ← decisionMakingToLPOD(DM)
if (SEM_LPOD(P_dm) ≠ {}) then
  u_* ← 1
  for all M_i ∈ maxPreferredAS(SEM_LPOD(P_dm)) do
    Δ_i ← getDecisionLiterals(D, M_i)
    label_K(Pref) ← label_K(Pref) ∪ Δ_i
  end for
end if
return ⟨label_K(Pref), u_*⟩
```

decisions (`getDecisionLiterals(D,M)`) which are minimal ($label_K(Pref)$) belongs to the most preferred answer(s) set only (`maxPreferredAS(`$SEM_{LPOD}(P_{dm})$`)`). If P_{dm} does not have any answer sets, then the set of decision is empty and its utility is equal to 0.

Example 10.3. Let us consider the DM in Example 10.2 and its computation with Algorithm 5. The function `decisionMakingToLPOD(DM)` generates the following LPOD

$$P_{dm} = \left\{ \begin{array}{llll} r_1 : & \neg w \leftarrow u. & r_6 : & \neg ass(\neg u) \times ass(\neg u) \leftarrow \top. \\ r_2 : & w \leftarrow r, \neg u. & r_7 : & u \leftarrow ass(u). \\ r_3 : & r \leftarrow c. & r_8 : & \neg u \leftarrow ass(\neg u). \\ r_4 : & c \leftarrow \top. & r_9 : & \bot \leftarrow not\ \neg w. \\ r_5 : & \neg ass(u) \times ass(u) \leftarrow \top & r_{10} : & \bot \leftarrow not\ \neg u. \end{array} \right\}$$

It can be checked that, according to the LPODs semantics (see Chapter 2, Section 2.2.4), the program P_{dm} does not have any answer sets. In fact, all the preferences cannot be satisfied. Therefore, the set of best decisions is empty and the pessimistic utility is equal to **0**.

More generally, the following result holds.

Proposition 10.2. *Let $DM = \langle K, D, Pref \rangle$ be a decision making problem and let P_{dm} be the LPOD generated by Algorithm 5. Then, $\Delta \in label_K(Pref)$ is an optimal pessimistic decision if and only if there exists a consistent maximally preferred answer set M of P_{dm} such that $\Delta = \{d \in D \mid ass(d) \in M\}$.*

Computation of Optimistic Optimal Decisions

As far as the optimistic criterion is concerned, Algorithm 5 can be modified to provide the computation of the optimistic utility in a straightforward way. It amounts to replace the function `decisionMakingToLPOD(DM)` with the function `decisionMakingToLPOD'(DM)` which modifies Definition 10.5 in the following way:

Algorithm 6 computeOptimisticDecision(DM) : $\langle \mathcal{D}, u^* \rangle$

Input: $\begin{cases} A\ DM = \langle K, D, Pref \rangle \end{cases}$

Output: $\begin{cases} \mathcal{D} : \text{optimal decisions} \\ u^* : \text{optimistic utility} \end{cases}$

$\mathcal{D} \leftarrow \{\}$; $u^* \leftarrow \mathbf{0}$;
$P_{dm'} \leftarrow decisionMakingToLPOD'(DM)$
if $(SEM_{LPOD}(P_{dm'}) \neq \{\})$ **then**
  $u_* \leftarrow \mathbf{1}$
  for all $M_i \in maxPreferredAS(SEM_{LPOD}(P_{dm}))$ **do**
    $\Delta_i \leftarrow getDecisionLiterals(D, M_i)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \Delta_i$
  end for
end if
return $\langle \mathcal{D}, u^* \rangle$

Definition 10.6. *Given a* $DM = \langle K, D, Pref \rangle$*, a decision* Δ *for* DM *can be computed via an LPOD* $P_{dm'}(\langle K, D, Pref \rangle) = K \cup \{p \leftarrow \top \mid p \in Pref\} \cup \{\neg ass(d) \times ass(d) \mid d \in D\} \cup \{d \leftarrow ass(d) \mid d \in D\}$*, where* $ass(d)$ *reads* d *is assumed.*

Based on the above definition, we can introduce Algorithm 6 for computing an optimal decision according to an optimistic criterion.

The main difference with previous Definition 10.5 is in the treatment of the preferences. Indeed, in the optimistic case, preferences are added as facts rather than constraints rules. This expresses that the possibility of satisfying all the goals remains open. Consequently, the existence of an optimal decision depends whether the generated LPOD is consistent or not.

Example 10.4. Let us consider the DM in Example 10.2 and its computation with Algorithm 6. The generated LPOD for computing an optimistic optimal decision corresponds to:

$$P_{dm'} = \left\{ \begin{array}{llll} r_1 : & \neg w & \leftarrow u. & r_6 : \neg ass(\neg u) \times ass(\neg u) \leftarrow \top. \\ r_2 : & w & \leftarrow r, \neg u. & r_7 : u \leftarrow ass(u). \\ r_3 : & r & \leftarrow c. & r_8 : \neg u \leftarrow ass(\neg u). \\ r_4 : & c & \leftarrow \top. & r_9 : \neg w \leftarrow \top. \\ r_5 : & \neg ass(u) \times ass(u) & \leftarrow \top & r_{10} : \neg u \leftarrow \top. \end{array} \right\}$$

In this case $SEM_{LPOD}(P_{dm'}) = \{\}$. Then, the set of optimal decision is empty and the optimistic utility is $\mathbf{0}$.

Remarks

Concerning the complexity of the computation of the pessimistic and optimistic optimal decision, we can observe the following. Since the employed ASP formalism of LPODs offers a complexity of Σ_2^P [Brewka et al., 2004b], we can bound the complexity of Algorithms 5 and 6 to the same complexity class.

Similar to what happens in possibilistic logic, these criteria can be extended to the case where K is a possibilistic logic program and $Pref$ is a set of possibilistic literals encoding prioritized goals, *i.e.*, when uncertainty and preferences are matters of degrees.

10.2 Modeling Decision Making Under Uncertainty in ASP

In stratified propositional bases, *i.e.*, when K is a set of more or less certain pieces of knowledge and P is a set of goals with associated levels of priority, the certainty and priority levels are supposed to belong to the same linearly ordered scale $\mathcal{S}$ and to be commensurate according to an order reversing map n.

Therefore, in order to be able to model qualitative DMU in a logic programming setting, we have to consider a specification which allows one to associate certainty and priority degrees to logic program rules and literals respectively. Since we follow the same principle behind the modeling adopted in Section 10.1, we will consider LPPODs which offers an extended version of LPODs with possibilistic logic (see Chapter 5).

In the following, we show how to use LPPODs to model qualitative DMU and to compute optimal decisions based on two alternative but equivalent computation methodologies.

10.2.1 Uncertain Knowledge and Prioritized Preferences

When knowledge about the world is uncertain and preferences are prioritized, definitions in Section 10.1.2 can be extended in the following way.

Definition 10.7. *A decision making problem under uncertainty DMU is represented as a tuple* $\langle K, D, Pref \rangle$ *where,*

- K *is a possibilistic definite logic program representing the knowledge about the world,*
- D *is a set of decision literals, and*
- $Pref = \{((p_1, \beta_1) \ldots, (p_n, \beta_n)\}$ *is a set of possibilistic literals, where* $\beta_i \in \mathcal{S}$ *is the priority of preference* p_i*.*

Several notations have to be introduced to deal with certainty and priority levels.

Let K_α denote the α-cut of K as $K_\alpha = \{r^* \in K \mid Nec(r) \geq \alpha\}$[6], and let $Pref_\beta$ be the β-cut of $Pref$ as $Pref_\beta = \{(p_i, \beta_i)^* \in Pref \mid \beta_i \geq \beta\}$. We also use the notations $K_{\underline{\alpha}}$ and $Pref_{\underline{\beta}}$ (with $\alpha < \mathbf{1}$ and $\beta < \mathbf{1}$, $\mathbf{1}$ being the top element of the scale) for denoting the set of rules and the set of preferences with certainty and priority strictly greater than α and β respectively (without certainty and priority degrees). In particular $K_{\underline{\mathbf{0}}} = K^*$ and $Pref_{\underline{\mathbf{0}}} = Pref^*$ ($\mathbf{0}$ being the bottom element of the scale) where K^* and $Pref^*$ denote the set of rules K and the set of preferences $Pref$ without their certainty and priority levels respectively.

Example 10.5. Let us consider the decision problem in Example 10.1 with certainty levels and prioritized goals. Then:

[6] As seen in Chapter 5, $Nec(r)$ denotes the necessity value α of a rule r and $*$ is the projection of a rule r and a program P defined as $r^* = l_0 \leftarrow l_1, \ldots, l_m$ and $P^* = \{r^* \mid r \in P\}$ respectively.

$$K = \begin{Bmatrix} r_1 = \mathbf{1} : \neg w \leftarrow u. \\ r_2 = \mathbf{1} : w \leftarrow r, \neg u. \\ r_3 = \lambda : r \leftarrow c. \\ r_4 = \mathbf{1} : c \leftarrow \top. \end{Bmatrix} \qquad D = \{u, \neg u\} \qquad Pref = \begin{Bmatrix} (\neg w, \mathbf{1}) \\ (\neg u, \delta) \end{Bmatrix}$$

where $\mathbf{0} < \lambda < \mathbf{1}$ and $\mathbf{0} < \delta < \mathbf{1}$. In the example, certainty and priority degrees can be read as follows: we are λ certain that when the sky is cloudy, it rains (rule r_3 in K). Then, we do not like to carry an umbrella, $(\neg u, \delta)$, but we consider more important to be dry, $(\neg w, \mathbf{1})$.

In the context of possibilistic logic, it has been proved that the pessimistic utility $u_*(d)$ of a decision d is the maximal value of $\alpha \in [0,1]$ such that $K_\alpha \wedge d \vdash_{CL} P_{\underline{n(\alpha)}}$ (Definition 9.1). Its counterpart in our approach can be formulated as:

Definition 10.8. *Given a $DMU = \langle K, D, Pref \rangle$, a pessimistic optimal decision is a set of decision literals $\Delta \subseteq D$ that maximizes α such that $K_\alpha \cup \Delta \models_c Pref_{\underline{n(\alpha)}}$.*

The above definition expresses the fact that a pessimistic optimal decision is computed with the most certain part of K and that selected preferences, even those with low priority, are satisfied.

Then, the label of a set of preferences can be defined as:

Definition 10.9. *Given a $DMU = \langle K, D, Pref \rangle$, the label of $Pref_{\underline{n(\alpha)}}$, denoted by $label_{K_\alpha}(Pref_{\underline{n(\alpha)}})$, is the set of all $\Delta_i \subseteq D$ minimal for set-inclusion such that $K_\alpha \cup \Delta_i \models_c Pref_{\underline{n(\alpha)}}$.*

Concerning an optimistic attitude, the computation of an optimistic decision in possibilistic logic amounts to find the maximum value of $n(\alpha) \in [0,1]$ such that $K_{\underline{\alpha}} \wedge d \wedge P_{\underline{\alpha}} \neq \bot$ (Definition 9.2). Its counterpart in our approach can be formulated as:

Definition 10.10. *Given a $DMU = \langle K, D, Pref \rangle$, an optimistic optimal decision is a set of decision literals $\Delta \subseteq D$ that maximizes $n(\alpha)$ such that $K_{\underline{\alpha}} \cup \Delta \cup Pref_{\underline{\alpha}}$ is consistent.*

The above definition expresses the fact that an optimistic optimal decision is a decision that belongs to a consistent answer set in which the preferred literals are among the most plausible ones.

Regarding the computation of a pessimistic optimal decision, there basically exist two possible ways to proceed. The first one is based on successive computations of labels of $Pref$. It amounts to maximize α in applying α- and β-cuts (with $\beta = n(\alpha)$) to the knowledge and preference base respectively (this means that the computation is reduced to the classical case). The second one, as we will see, amounts to compute optimal values of α according to an approach, closer to possibilistic logic inference, based on the LPPODs semantics. The computation of an optimistic decision can be realized in a similar way.

10.2.2 Classical ASP-based Computation

Computation of Pessimistic Optimal Decisions

For the sake of computation of the pessimistic optimal decision we analyze different cases.

Algorithm 7 computePessimisticDecisionsUU(DMU) : $\langle \mathcal{D}, u_* \rangle$

Input: $\begin{cases} A\ DMU = \langle K, D, Pref \rangle \end{cases}$

Output: $\begin{cases} \mathcal{D} : \text{the set of best pessimistic decisions} \\ u_* : \text{the utility of the best pessimistic decisions} \end{cases}$

$\alpha, u_* \leftarrow \mathbf{0}$; $\mathcal{D} \leftarrow \{\}$; $finish \leftarrow false$;
while $(not\ finish)$ **do**
  $\{Inc(\alpha)$ is a function changing the value of α into the immediately above value$\}$
  $\alpha \leftarrow Inc(\alpha)$
  $label_{K_\alpha}(Pref_{\underline{n(\alpha)}}) \leftarrow computePessimisticDecisions(\langle K_\alpha, D, Pref_{\underline{n(\alpha)}} \rangle)$
  if $(label_{K_\alpha}(Pref_{\underline{n(\alpha)}}) = \{\})$ **then**
    $finish \leftarrow true$
  else
    $u_* \leftarrow \alpha$
    $\mathcal{D} \leftarrow label_{K_\alpha}(Pref_{\underline{n(\alpha)}})$
  end if
end while
return $\langle \mathcal{D}, u_* \rangle$

In the case K is a set of rules associated with certainty degrees and $Pref$ are all-or-nothing preferences, an optimal set of decision literals Δ is the set of that allows to satisfy any preference literal, considering only the totally sure rules in $K_{\mathbf{1}}$. However, such set may not exist. Thus, the certainty level of K has to be considered. In terms of labels, the search of Δ can be formulated as the optimization problem of maximizing α such that $label_{K_\alpha}(Pref)$ is not empty. In this case, the pessimistic optimal utility corresponds to the level of α encountered.

In the case K is a set of fully certain rules and $Pref$ is a set of prioritized preferences, an optimal set of decision literals Δ is the set that allows to satisfy any preference literal, whatever its degree of priority is. Again, such decision may not exist. Then, the problem can be formulated as the minimization problem of finding the smallest β such that $label_K(Pref_{\underline{\beta}})$ is not empty, *i.e.,* for which the degree of priority of the most important preference is low. In such a case, the best pessimistic utility corresponds to $n(\beta)$ (and for the commensurateness hypothesis it amounts to maximize α).

In the most general case, *i.e.,* when K contains certainty degrees and $Pref$ contains prioritized preferences, things get more complicated. In fact, in such a case, an optimal decision is a decision that allows to satisfy the maximum number of preferences with the most certain rules of K. Therefore, the problem may be formulated as the problem of finding a set of decisions Δ with α as high as possible and β as small as possible. By commensurateness hypothesis, the problem is converted in finding the maximum value of α such that $label_{K_\alpha}(Pref_{\underline{n(\alpha)}})$ is not empty.

We are now able to propose an algorithm (Algorithm 7) for computing an optimal (pessimistic) decision for the general case. The algorithm is based on successive computations of labels of $Pref$ which can be computed on the basis of Algorithm 5 (*i.e.,* based on a classical ASP computation).

The behavior of Algorithm 7 can be described as follows. Each time preferences belonging to $Pref_{\underline{n(\alpha)}}$ are satisfied, the set of potentially optimal decisions is updated by keeping only the decisions that satisfy all the literals in $Pref_{\underline{n(\alpha)}}$ considered so far. This is done, first considering all program rules and highest labelled preferences. If such label is not empty, then expectations are increased ($Inc(\alpha)$) trying to satisfy less preferred preferences by means of less knowledge. The procedure stops when a selected preference cannot be satisfied.

Example 10.6. As seen in Example 10.5, K and $Pref$ contain two layers (both scales are commensurate). First of all, according to function $Inc(\alpha)$, α is incremented to the lowest non-null value, *i.e.*, $\alpha = \min\{\lambda, n(\delta)\}$. Whatever the relative positions of λ and δ, $K_\alpha = K^*$. We have the following cases:

$\lambda > n(\delta)$: then $\alpha = n(\delta)$ and we have to compute $label_{K^*}(Pref_{\underline{\delta}})$. This means that $Pref_{\underline{\delta}} = \{\neg w\}$. Algorithm 5 will return the decision $\{u\}$ as label for this preference since the program[7]

$$P'_{dm} = \left\{ \begin{array}{llll} r_1: & \neg w & \leftarrow & u. \\ r_2: & w & \leftarrow & r, \neg u. \\ r_3: & r & \leftarrow & c. \\ r_4: & c & \leftarrow & \top. \\ r_5: & \neg ass(u) \times ass(u) & \leftarrow & \top. \end{array} \quad \begin{array}{llll} r_6: & \neg ass(\neg u) \times ass(\neg u) & \leftarrow & \top. \\ r_7: & u & \leftarrow & ass(u). \\ r_8: & \neg u & \leftarrow & ass(\neg u). \\ r_9: & \bot & \leftarrow & not\ \neg w. \end{array} \right\}$$

has $\{\neg w, u, ass(u), c, r, \neg ass(\neg u)\}$ as its most preferred answer set. As a next step, $\alpha = \lambda$, thus $Pref_{\underline{n(\lambda)}} = \{\neg w, \neg u\}$. But the computation of $label_{K^*}(Pref^*)$ is found to be empty (basically, it corresponds to the program built in Example 10.3). Therefore, the set of best pessimistic decisions is in this case $\mathcal{D} = \{u\}$ with utility $u_* = n(\delta)$.

$\lambda < n(\delta)$: then $\alpha = \lambda$ and we have to compute $label_{K^*}(Pref_{\underline{n(\lambda)}})$. As $n(\lambda) > \delta$, $Pref_{\underline{n(\lambda)}} = \{\neg w\}$, and $label_{K^*}(\neg w) = \{u\}$ (indeed, the program above P'_{dm} is constructed). A next step is performed where $\alpha = n(\delta)$ and $label_{K_{n(\delta)}}(Pref_{\underline{\delta}}) = \{u\}$ since the program $P'_{dm} \backslash r_3$ (r_3 is cut) has $\{\neg w, u, ass(u), c, \neg ass(\neg u)\}$ as its most preferred answer set.
We then have to perform a last step, where $\alpha = \mathbf{1}$, but the computation of $label_{K_1}(Pref^*)$ is equal to $\emptyset$. Therefore, the set of optimal decisions is in this case $\mathcal{D} = \{u\}$ with utility $u_* = n(\delta)$.

$\lambda = n(\delta)$: then $\alpha = \lambda = n(\delta)$ and we have to compute $label_{K^*}(Pref_{\underline{\delta}})$ which is equal to the computation of $label_{K^*}(\neg w)$ which returns $\{u\}$. Then a next step is performed where $\alpha = \mathbf{1}$ but $label_{K_1}(Pref^*) = \emptyset$.

Therefore, the best pessimistic solution of the running example is always to take an umbrella with utility $n(\delta)$.

Please observe that, in this case, the optimal pessimistic decision does not depend on the exact value of λ and δ, and even not on their relative positions. However, in the general case, only the positions of the priority and certainty levels matter.

[7] Please observe that preference $(\neg u, \delta)$ is cut. Consequently, it is not added to P'_{dm}.

Algorithm 8 computeOptimisticDecisionsUU(DMU) : $\langle \mathcal{D}, u^* \rangle$

Input: $\{ A\ DMU = \langle K, D, Pref \rangle$

Output: $\begin{cases} \mathcal{D} : \text{the set of best optimistic decisions} \\ u^* : \text{the utility of the best optimistic decisions} \end{cases}$

$\alpha \leftarrow \mathbf{0}$;
$\mathcal{D} \leftarrow computeOptimisticDecision(K_{\underline{\alpha}}, D, Pref_{\underline{\alpha}})$;
while $(\mathcal{D} = \{\})$ *and* $(\alpha < \mathbf{1})$ **do**
  $\{Inc'(\alpha)$ eliminates the least certain stratum remaining in K and $Pref\}$
  $\alpha \leftarrow Inc'(\alpha)$
  $\mathcal{D} \leftarrow computeOptimisticDecision(K_{\underline{\alpha}}, D, Pref_{\underline{\alpha}})$;
end while
return $\langle \mathcal{D}, n(\alpha) \rangle$

Computation of Optimistic Optimal Decisions

According to Algorithm 8, the computation of an optimistic optimal decision relies on Algorithm 6 and it proceeds in the following way. First of all, we try to compute an optimistic decision w.r.t. $K_{\underline{0}}$ and $Pref_{\underline{0}}$. If a decision is found, then the maximum utility, *i.e.*, $n(0) = 1$ is returned. Otherwise, we keep on increasing the level of α until a decision consistent with $K_{\underline{\alpha}}$ and $Pref_{\underline{\alpha}}$ is found (if it exists). The optimal optimistic utility is $n(\alpha)$.

Let us consider the behavior of Algorithm 8 in the computation of the optimal decision and the optimistic utility in our running example.

Example 10.7. At the beginning, we try to compute an optimistic decision w.r.t. $K_{\underline{0}}$ and $Pref_{\underline{0}}$. Since $K_{\underline{0}} = K^*$ and $Pref_{\underline{0}} = Pref^*$, the program built in Algorithm 8 is basically the same as the program built in Example 10.4. Since this program is inconsistent, we proceed with the next step. According to function $Inc'(\alpha)$, $\alpha = \min\{\lambda, \delta\}$. We have the following cases:

$\lambda > \delta$: then $\alpha = \delta$ and we have to compute an optimistic decision w.r.t. $K_{\underline{\delta}}$ and $Pref_{\underline{\delta}}$. This means that $K_{\underline{\delta}} = K^*$ and $Pref_{\underline{\delta}} = \{\neg w\}$. Algorithm 6 returns the decision $\{u\}$ as an optimal decision since the program:[8]

$$P_{dm'} = \left\{ \begin{array}{llllllll} r_1: & \neg w & \leftarrow & u. & r_6: & \neg ass(\neg u) \times ass(\neg u) & \leftarrow & \top. \\ r_2: & w & \leftarrow & r, \neg u. & r_7: & u & \leftarrow & ass(u). \\ r_3: & r & \leftarrow & c. & r_8: & \neg u & \leftarrow & ass(\neg u). \\ r_4: & c & \leftarrow & \top. & r_9: & \neg w & \leftarrow & \top. \\ r_5: & \neg ass(u) \times ass(u) & \leftarrow & \top. & & & & \end{array} \right\}$$

has $\{\neg w, u, ass(u), c, r, \neg ass(\neg u)\}$ as its most preferred answer set. Therefore, the best optimistic decision is $\{u\}$ with utility $u^* = n(\delta)$.

$\lambda < \delta$: then $\alpha = \lambda$ and we have to compute an optimistic decision w.r.t. $K_{\underline{\lambda}}$ and $Pref_{\underline{\lambda}}$. This means that $K_{\underline{\lambda}} = (K \backslash r_3)^*$ and $Pref_{\underline{\lambda}} = \{\neg w, \neg u\}$. Algorithm 6 returns the decision $\{\neg u\}$ as an optimal decision since the program:[9]

[8] Please observe that preference $(\neg u, \delta)$ is cut. Consequently, it is not added to $P_{dm'}$.
[9] Please observe that r_3 is cut. Consequently, it is not added to $P_{dm'}$.

$$P_{dm'} = \left\{ \begin{array}{llllllll} r_1: & \neg w & \leftarrow & u. & r_6: & \neg ass(\neg u) \times ass(\neg u) & \leftarrow & \top. \\ r_2: & w & \leftarrow & r, \neg u. & r_7: & u & \leftarrow & ass(u). \\ & & & & r_8: & \neg u & \leftarrow & ass(\neg u). \\ r_4: & c & \leftarrow & \top. & r_9: & \neg w & \leftarrow & \top. \\ r_5: & \neg ass(u) \times ass(u) & \leftarrow & \top. & r_9: & \neg u & \leftarrow & \top. \end{array} \right\}$$

is consistent (resp. inconsistent) when decision $\{\neg u\}$ (resp. decision $\{u\}$) is chosen. Therefore, the best optimistic decision is $\{\neg u\}$ with utility $u^* = n(\lambda)$.

$\lambda = \delta$: then $\alpha = \lambda = \delta$. We have to compute an optimistic decision w.r.t. $K_{\underline{\lambda}}$ and $Pref_{\underline{\lambda}}$. This means that $K_{\underline{\lambda}} = (K \backslash r_3)^*$ and $Pref_{\underline{\lambda}} = \{\neg w\}$. Algorithm 6 returns the decision $\{\neg u\}$ and $\{u\}$ as optimal decisions since the program:[10]

$$P_{dm'} = \left\{ \begin{array}{llllllll} r_1: & \neg w & \leftarrow & u. & r_6: & \neg ass(\neg u) \times ass(\neg u) & \leftarrow & \top. \\ r_2: & w & \leftarrow & r, \neg u. & r_7: & u & \leftarrow & ass(u). \\ & & & & r_8: & \neg u & \leftarrow & ass(\neg u). \\ r_4: & c & \leftarrow & \top. & r_9: & \neg w & \leftarrow & \top. \\ r_5: & \neg ass(u) \times ass(u) & \leftarrow & \top. & & & & \end{array} \right\}$$

is consistent for both decisions $\{\neg u\}$ and $\{u\}$. Therefore, the best optimistic decisions are $\{\neg u\}$ with utility $u^* = n(\lambda)$ and $\{u\}$ with utility $u^* = n(\delta)$.

According to the optimistic computation, optimistic optimal decisions are: $\langle u, n(\delta)\rangle$ and $\langle \neg u, n(\lambda)\rangle$. Therefore, the best decision in the optimistic case of this example depends on the values δ and λ.

Remarks

Finally, we can observe that the complexity for the computation of a pessimistic and optimistic optimal decision, when certainty and preference are matters of degrees, is Σ_2^P. Indeed, according to this classical ASP-based computation, Algorithm 7 and 8 both rely on a computation based on LPODs.

10.2.3 Possibilistic ASP-based Computation

In the previous section, we have provided a method for computing pessimistic and optimistic optimal decisions reducing the problem to a classical ASP computation. In general, the computation of pessimistic and optimistic optimal decisions can also be realized by means of an approach based on the LPPODs semantics.

Computation of Pessimistic Optimal Decisions

The computation of pessimistic optimal decisions based on PASP can be motivated by an important result of possibilistic logic. This result is the following.

[10] Please observe that r_3 and r_9 are cut. Consequently, they are not added to $P_{dm'}$.

Given a knowledge base K made of possibilistic formulas, *i.e.*, $K = \{(\varphi_i, \alpha_i) \mid 1 \leq i \leq n\}$, proving a possibilistic formula (p, α) from K can be done in two equivalent ways: according to possibilistic logic inference or according to classical logic inference. In fact, it holds that $K \vdash_{PL} (p, \alpha)$ if and only if $K_\alpha \vdash_{CL} p$, where $K_\alpha = \{\varphi \mid (\varphi, \beta) \in K \wedge \beta \geq \alpha\}$, and $\vdash_{PL}$ and $\vdash_{CL}$ are the possibilistic logic and classical inference respectively [Dubois et al., 1994].

This property allows to use possibilistic logic inference to prove a set of prioritized preferences $(p_1, \beta_1), \ldots, (p_n \beta_n)$ from a possibilistic knowledge base K and a decision d. In fact, proving a preference belonging to a stratum β_i amounts to check whether $K \wedge (d, 1) \vdash_{PL} (p, \beta_i)$ which amounts to prove, by refutation, that $K \wedge (d, 1) \wedge (p, 1) \vdash_{PL} (\bot, \gamma)$ with $\gamma \geq \beta_i$.

In the following, we intend to mimic the described methodology above in the context of possibilistic ASP. For this purpose, we first extend the notion of $\models_c$ to deal with sets of possibilistic literals as:

Definition 10.11. *Given a possibilistic logic program P and a set of possibilistic literals S, $P \models_p S$ holds, if there exists a possibilistic answer set M of P such that $S \sqsubseteq M$ where the relation between sets of possibilistic literals $\sqsubseteq$ is defined as:*
$S \sqsubseteq M \iff S^* \subseteq M^* \wedge \forall a, \alpha, \beta, (a, \alpha) \in S \wedge (a, \beta) \in M$ *then* $\alpha \leq \beta$.

Basically, the above definition is capturing the idea behind the *weight weakening* inference rule in possibilistic logic.[11]

We also extend Definition 10.5 to build an LPPOD given a qualitative DMU in the following way:

Definition 10.12. *Given $DMU = \langle K, D, Pref \rangle$ and $\alpha \in \mathcal{S}$, an LPPOD for DMU can be built as* $P_{dm_\alpha}(\langle K, D, Pref \rangle) = \{r \in K \mid Nec(r) \geq \alpha\} \cup \{\mathbf{1} : \neg ass(d) \times ass(d) \mid d \in D\} \cup \{\mathbf{1} : d \leftarrow ass(d) \mid d \in D\}$.

Based on the above definitions and the property of possibilistic logic described above, we are able to provide an LPPOD-based algorithm for computing the pessimistic optimal decision and the pessimistic utility.

Algorithm 9 describes an LPPOD-based procedure to compute the set of pessimistic decisions. $Inc(\alpha)$ is the function that returns the next level of α. At each iteration the expectation of satisfying as many preferences as possible given a level α is increased. P_{dm_α} is constructed by a method `decisionMakingToLPPOD(DMU,`α`)` which implements Definition 10.12. P_{dm_α} is constructed by taking all the rules with certainty higher than the current value of α. Preference satisfaction is checked by means of $\models_p$. `getDecisionLiterals` allows one to retrieve all (minimal) sets of decisions ($\mathcal{D}$) which are able to satisfy preferences contained in strata greater than $n(\alpha)$. The iteration keeps on until α has been maximized.

We exemplify the behavior of the algorithm by means of the following example.

Example 10.8. According to $Inc(\alpha)$, $\alpha = \min\{\lambda, n(\delta)\}$. We have the following cases:

[11] In possibilistic logic, for $\beta \leq \alpha$, $(\varphi, \alpha) \vdash_{PL} (\varphi, \beta)$.

Algorithm 9 computePessimisticDecisionsUU(DMU) : $\langle \mathcal{D}, u_* \rangle$

Input: $\begin{cases} A\ DMU = \langle K, D, Pref \rangle \end{cases}$

Output: $\begin{cases} \mathcal{D} : \text{the set of best pessimistic decisions} \\ u_* : \text{the utility of the best pessimistic decisions} \end{cases}$

$\mathcal{D} \leftarrow \emptyset$; $\alpha, u_* \leftarrow \mathbf{0}$; $finish \leftarrow false$
while $(not\ finish)$ **do**
  $\alpha \leftarrow Inc(\alpha)$;
  $P_{dm_\alpha} \leftarrow decisionMakingToLPPOD(DMU, \alpha)$
  if $P_{dm_\alpha} \models_p \{(p, \beta) \mid (p, \beta) \in Pref \text{ and } \beta > n(\alpha)\}$ **then**
    $u_* \leftarrow \alpha$;
    $\mathcal{D} \leftarrow getDecisionLiterals(SEM_{LPPOD}(P_{dm_\alpha}), p_\beta)$
  else
    $finish \leftarrow true$
  end if
end while
return $\langle D, u_* \rangle$

$\lambda > n(\delta)$: then $\alpha = n(\delta)$. This means that

$$P_{dm_{n(\delta)}} = \left\{ \begin{array}{llllllll} r_1 : \mathbf{1} : \neg w \leftarrow & u. & r_5 : \mathbf{1} : & \neg ass(u) \times ass(u) & \leftarrow & \top. \\ r_2 : \mathbf{1} : & w \leftarrow r, \neg u. & r_6 : \mathbf{1} : & \neg ass(\neg u) \times ass(\neg u) & \leftarrow & \top. \\ r_3 : \lambda : & r \leftarrow & c. & r_7 : \mathbf{1} : & u & \leftarrow & ass(u). \\ r_4 : \mathbf{1} : & c \leftarrow & \top. & r_8 : \mathbf{1} : & \neg u & \leftarrow & ass(\neg u). \end{array} \right\}$$

$P_{dm_{n(\delta)}}$ has two maximally preferred possibilistic answer sets, $M_1 = \{(\neg u, 1), (ass(\neg u), 1), (c, 1), (r, \lambda), (w, \lambda), (\neg ass(u), 1)\}$ and $M_2 = \{(u, 1), (ass(u), 1), (c, 1), (r, \lambda), (\neg w, 1), (\neg ass(\neg u), 1)\}$. The only preference to be considered is $(\neg w, 1)$. Then, $P_{dm_{n(\delta)}} \models_p (\neg w, 1)$, since $(\neg w, 1) \sqsubset M_2$. Therefore, $\mathcal{D} = \{u\}$ and $u_* = n(\delta)$. As a next step, $\alpha = \lambda$. Therefore, the preferences to be considered are $\{(\neg w, 1)(\neg u, \delta)\}$. But $P_{dm_{n(\delta)}} \not\models_p \{(\neg w, 1)(\neg u, \delta)\}$. The set of best pessimistic decisions is in this case $\mathcal{D} = \{u\}$ with utility $u_* = n(\delta)$.

$\lambda < n(\delta)$: then $\alpha = \lambda$. As $n(\lambda) > \delta$, the only preference to be considered is $\{\neg w\}$ and $P_{dm_\lambda} \models_p (\neg w, 1)$ (indeed, P_{dm_λ} is the same as the program above). A next step is performed with $\alpha = n(\delta)$. $P_{dm_\lambda} \backslash r_3$ (r_3 is cut) has $M_1 = \{(\neg u, 1), (ass(\neg u), 1), (c, 1), (\neg ass(u), 1)\}$ and $M_2 = \{(u, 1), (ass(u), 1), (c, 1), (\neg w, 1), (\neg ass(\neg u), 1)\}$ as its most preferred possibilistic answer sets. Thus, $P_{dm_\lambda} \backslash r_3 \models_p (\neg w, 1)$. We then have to perform a last step with $\alpha = \mathbf{1}$. Then, the preferences to be considered are $\{(\neg w, 1)(\neg u, \delta)\}$. But $P_{dm_\lambda} \backslash r_3 \not\models_p \{(\neg w, 1)(\neg u, \delta)\}$. Therefore, the set of optimal decisions is in this case $\mathcal{D} = \{u\}$ with utility $u_* = n(\delta)$.

$\lambda = n(\delta)$: then $\alpha = \lambda = n(\delta)$. The only preference to be considered is $(\neg w, 1)$. The LPPOD $P_{dm_{n(\delta)}}$ built is the same as above. Then, $P_{dm_{n(\delta)}} \models_p (\neg w, 1)$, since $(\neg w, 1) \sqsubset M_2$. A next step is performed with $\alpha = \mathbf{1}$, but $P_{dm_{n(\delta)}} \not\models_p \{(\neg w, 1)(\neg u, \delta)\}$.

As a consequence, the pessimistic optimal decision is $\mathcal{D} = \{u\}$ with an utility $u_* = n(\delta)$. This agrees, as expected, with the label-based computation presented in Section 10.2.2.

Algorithm 10 computeOptimisticDecisionsUU(DMU) : $\langle \mathcal{D}, u^* \rangle$

Input: $\{ A\ DMU = \langle K, D, Pref \rangle$

Output: $\begin{cases} \mathcal{D} : \text{the set of best optimistic decisions} \\ u^* : \text{the utility of the best optimistic decisions} \end{cases}$

$\alpha \leftarrow \mathbf{0}$; $\mathcal{D} \leftarrow \{\}$;
$P_{dm'} \leftarrow decisionMakingToLPPOD(DMU, \alpha)$
for all $(p_i, \beta_i) \in Pref$ **do**
  $P_{dm'} \leftarrow P_{dm'} \cup \{\beta_i : p \leftarrow \top\}$
end for
if $ConsCutDeg(P_{dm'}) > 0$ **then**
  $\alpha \leftarrow ConsCutDeg(P_{dm'})$
end if
if $(SEM_{LPPOD}(P_{dm'_{\underline{\alpha}}}) \neq \{\})$ **then**
  for all $M_i \in maxPreferredAS(SEM_{LPPOD}(P_{dm'_{\underline{\alpha}}}))$ **do**
    $\Delta_i \leftarrow getDecisionLiterals(D, M_i)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \Delta_i$
  end for
end if
return $\langle \mathcal{D}, n(\alpha) \rangle$

Computation of Optimistic Optimal Decisions

Concerning the computation of an optimistic decision, we can observe the following. The optimistic optimal criterion requires $K_{\underline{\alpha}} \wedge d \wedge P_{\underline{\alpha}}$ to be logically consistent. If it is not the case, consistency can be restored by trying to cut some knowledge contained below some computed threshold.

In order to restore the consistency of an inconsistent possibilistic knowledge base, possibilistic logic deletes the set of possibilistic formulas which are lower than the inconsistency degree [Dubois et al., 1994].

Based on this inconsistency degree, a methodology for computing the optimistic optimal decision can be defined in the possibilistic setting in a straightforward way. Indeed, since when we compute an optimistic optimal decision, we are looking for the maximum value of $n(\alpha)$ such that $K_{\underline{\alpha}} \wedge d \wedge P_{\underline{\alpha}}$ is consistent, the inconsistency degree of $K_{\underline{\alpha}} \wedge d \wedge P_{\underline{\alpha}}$ corresponds to the value of the optimistic utility (the decision can be obtained by iterating over all the decision contained in D).

By considering this idea, we can provide an alternative computation in the PASP setting. First, the concept of consistency degree of a logic program should be considered. Nicolas *et al.* [Nicolas et al., 2006] already defined the concept of α-cut for possibilistic logic programs (see Chapter 2, Section 2.5). Based on this definition, we define its respective generalization for our approach.

Definition 10.13. *Let P be an LPPOD, then we define:*

- $P_{\underline{\alpha}} = \{r \in P \mid Nec(r) > \alpha\}$
- *the consistency cut degree of P as*

$$ConsCutDeg(P) = \begin{cases} 0 \text{ if } P^* \text{ is consistent} \\ \min\{\alpha \mid P_{\underline{\alpha}} \text{ is consistent}\} \text{ otherwise} \end{cases}$$

Please observe that *ConsCutDeg* of an LPPOD identifies the minimum level of certainty for which $P_{\underline{\alpha}}$ is consistent.

Based on the above definition, Algorithm 10 computes the optimistic optimal decision given a DMU according to a PASP-based computation. First of all, we build an LPPOD $P_{dm'}$ by means of the construction of Definition 10.12 and by adding all the preferences in $Pref$ as possibilistic literals. If this possibilistic program is consistent, then the sets of decision literals are retrieved and the maximum value for the optimal optimistic utility is returned. Otherwise, the consistency degree of $P_{dm'}$ is computed. The consistency degree represents the optimal optimistic utility $n(\alpha)$ and it is returned together with the corresponding sets of decision literals.

Example 10.9. According to Algorithm 10, the computation of the DMU in Example 10.5 proceeds in the following way. $ConsCutDeg(P_{dm'}) > 0$ since $P_{dm'}$ is inconsistent. In such a case, the next value of α to be considered is $ConsCutDeg(P_{dm'})$. We have the following cases:

$\lambda > \delta$: then $ConsCutDeg(P_{dm'}) = \delta$ and we have to build the program $P_{dm'_{\underline{\delta}}}$:[12]

$$P_{dm'_{\underline{\delta}}} = \left\{ \begin{array}{llll} r_1 : \mathbf{1} : \neg w \leftarrow & u. & r_5 : \mathbf{1} : & \neg ass(u) \times ass(u) \leftarrow \top. \\ r_2 : \mathbf{1} : w \leftarrow & r, \neg u. & r_6 : \mathbf{1} : & \neg ass(\neg u) \times ass(\neg u) \leftarrow \top. \\ r_3 : \lambda : r \leftarrow & c. & r_7 : \mathbf{1} : & u \leftarrow ass(u). \\ r_4 : \mathbf{1} : c \leftarrow & \top. & r_8 : \mathbf{1} : & \neg u \leftarrow ass(\neg u). \\ & & r_9 : \mathbf{1} : & \neg w \leftarrow \top. \end{array} \right\}$$

which has $\{(\neg w, 1), (u, 1), (ass(u), 1), (c, 1), (r, \lambda), (\neg ass(\neg u), 1)\}$ as its most preferred possibilistic answer set. Therefore, the best optimistic decision is $\{u\}$ with utility $u^* = n(\delta)$.

$\lambda < \delta$: then $ConsCutDeg(P_{dm'}) = \lambda$ and we have to build the program $P_{dm'_{\underline{\lambda}}}$:[13]

$$P_{dm'_{\underline{\lambda}}} = \left\{ \begin{array}{llll} r_1 : \mathbf{1} : \neg w \leftarrow & u. & r_5 : \mathbf{1} : & \neg ass(u) \times ass(u) \leftarrow \top. \\ r_2 : \mathbf{1} : w \leftarrow & r, \neg u. & r_6 : \mathbf{1} : & \neg ass(\neg u) \times ass(\neg u) \leftarrow \top. \\ r_3 : \lambda : r \leftarrow & c. & r_7 : \mathbf{1} : & u \leftarrow ass(u). \\ & & r_8 : \mathbf{1} : & \neg u \leftarrow ass(\neg u). \\ r_9 : \mathbf{1} : \neg w \leftarrow & \top. & r_{10} : \delta : & \neg u \leftarrow \top. \end{array} \right\}$$

which has $\{(\neg w, 1), (\neg u, 1), (ass(\neg u), 1), (c, 1), (\neg ass(u), 1)\}$ as its most preferred possibilistic answer set. Therefore, the best optimistic decision is $\{\neg u\}$ with utility $u^* = n(\lambda)$.

$\lambda = \delta$: then $ConsCutDeg(P_{dm'}) = \lambda = \delta$ and we have to build the program $P_{dm'_{\underline{\lambda}}}$:[14]

[12] Please observe that the rule $\delta : \neg u \leftarrow \top$ is cut.
[13] Please observe that the rule $\lambda : r \leftarrow c$ is cut.
[14] Please observe that the rules $\lambda : r \leftarrow c$ and $\delta : \neg u \leftarrow \top$ are cut.

$$P_{dm'_{\underline{\lambda}}} = \left\{ \begin{array}{llllllll} r_1 : \mathbf{1} : \neg w \leftarrow & u. & r_5 : \mathbf{1} : & \neg ass(u) \times ass(u) & \leftarrow & \top. \\ r_2 : \mathbf{1} : & w \leftarrow r, \neg u. & r_6 : \mathbf{1} : & \neg ass(\neg u) \times ass(\neg u) & \leftarrow & \top. \\ r_3 : \lambda : & r \leftarrow & c. & r_7 : \mathbf{1} : & u & \leftarrow & ass(u). \\ & & & r_8 : \mathbf{1} : & \neg u & \leftarrow & ass(\neg u). \\ r_9 : \mathbf{1} : \neg w \leftarrow & \top. & & & & & \end{array} \right\}$$

which has $\{(\neg w, 1), (\neg u, 1), (ass(\neg u), 1), (c, 1), (\neg ass(u), 1)\}$ and $\{(\neg w, 1), (u, 1), (ass(u), 1), (c, 1), (\neg ass(\neg u), 1)\}$ as its most preferred possibilistic answer sets. Therefore, the best optimistic decision are $\{\neg u\}$ with utility $u^* = n(\lambda)$ and $\{u\}$ with utility $u^* = n(\delta)$.

Consequently, optimistic optimal decisions are $\{u\}$ with an utility $n(\delta)$ and $\{\neg u\}$ with an utility $n(\lambda)$. This agrees, as expected, with the computation presented in Section 10.2.2.

Remarks

Concerning the complexity of the computation of the pessimistic and optimistic optimal decision, we can observe the following. Since the employed formalism of LPPODs has complexity Σ_2^P (see discussion in Chapter 5, Section 5.5), the computation of Algorithms 9 and 10 still belongs to the same complexity class.

10.3 Related Work

The treatment of qualitative DMU we have presented in this chapter is inspired by the work in [Dubois and Prade, 1995] and it provides a syntactical counterpart to the semantical treatment of plausibility states and utility function.

When the possibilistic semantics for qualitative decision is considered (see Chapter 9, Section 9.2.1), optimal decisions are understood in terms of two possibilistic distributions which rank-order interpretations: a distribution π which orders interpretations according to their level of possibility (induced by the levels of certainty of the knowledge), and a utility function μ which orders interpretations according to their level of satisfaction (induced by the levels of priority). According to this semantical treatment, a pessimistic criterion then is to look for a decision that makes all the less satisfied consequences hardly plausible. In [Dubois and Prade, 1995], it is shown how the semantical treatment of the example presented in this chapter lead to the pessimistic utilities $u_*(u) = n(\delta)$ (and $u_*(\neg u) = 0$). The semantics of the optimistic criterion can be understood in a similar way.

Regarding logic programming, to the best of our knowledge, there are only few works in the literature about modeling qualitative decision problems in ASP [Brewka, 2005; Grabos, 2004]. These two approaches share the use the ordered disjunction connective $\times$ to represent preferences and to rank-order different possible states of the world represented as different answer sets. However, they differ in the way in which the uncertainty is handled.

In [Brewka, 2005], uncertainty is not explicitly represented. This method is based on the assumption that abnormal states of the world are totally disregarded and that all states which are taken into account are considered plausible. As a consequence, states can be

either negligible or plausible. But, in the latter case, no distinction between the degrees of plausibility of the states can be made, and no further distinctions between the generated answer sets are possible.

While in [Brewka, 2005] the existing method does not explicitly take into account an uncertainty ordering, Grabos in [Grabos, 2004] proposed to use $\times$ not only for modeling preferences but also for modeling the plausibility degrees of states. Depending whether a commensurability assumption between the two degrees of plausibility and of preferences is taken (resp. is not taken), decision rules give more importance (resp. is not not taken) to one of the degrees in order to select the best answer set according to the attitude of the decision maker w.r.t. the risk. Although this method offers a way to represent uncertainty, decision rules are empirical and they are not based on postulates like the possibilistic criteria. Moreover, although our commensurability assumption is a strong assumption, it has been observed in [Dubois et al., 2002] that working without it leads to an ineffective decision method.

10.4 Discussion and Concluding Remarks

In this chapter, we have presented an ASP-based and a PASP-based methodology to model qualitative DMU and to compute an optimal decision by considering two knowledge bases whose degrees of certainty and priority are commensurate. We have first shown how to encode fully certain knowledge and all-or-nothing preferences (Definition 10.2), and then, on top of that, how to compute a pessimistic optimal decision when uncertainty and preferences are matters of degrees (Definition 10.8).

The reader may be concerned why we have chosen not to take into account ASP optimization techniques (via objective functions) and to compute preferences at meta-level rather than inside LPODs and LPPODs. Our design choice can be motivated by the need of handling two separate knowledge bases and of having a formal handling of uncertainty (in terms of possibilistic logic). In this way, we have been able to provide a possibilistic ASP-based methodology which computes the same decisions of the label-based computation. This result agrees both with the classical and the possibilistic resolution views for computing optimal decisions in possibilistic logic.

For the sake of completion, let us see how the decision scenario proposed in this chapter looks like when cardinality constraints are used and preferences are directly encoded by means of LPPODs constructs. The use of cardinality constraints can be justified to provide the generation of the decision space (each decision literal will appear exactly once in one solution of the program). Preferences can be encoded by means of the $\times$ connector. Then, the following program can be obtained:[15]

[15] Although we have not considered cardinality constraints in LPPODs, we assume here that they are a short-hand for rules $\mathbf{1} : u \leftarrow not\ \neg u$ and $\mathbf{1} : \neg u \leftarrow not\ u$ [Simons et al., 2002].

$$P = \left\{ \begin{array}{lccl} r_1 = \mathbf{1}: & 1\{u, \neg u\}1. & & \\ r_2 = \mathbf{1}: & \neg w & \leftarrow & u. \\ r_3 = \mathbf{1}: & w & \leftarrow & r, \neg u. \\ r_4 = \lambda: & r & \leftarrow & c. \\ r_5 = \mathbf{1}: & c & \leftarrow & \top. \\ r_8 = \mathbf{1}: & \neg w \times \neg u & \leftarrow & \top. \end{array} \right\}$$

The above program has two possibilistic answer sets, *e.g.*, $\{(c,1), (r,\lambda)$,(u,1), $(\neg w,1)\}$, $\{(c,1), (r,\lambda), (\neg u,1), (w,\lambda)\}$. According to the preference relation induced by $\times$ (see Chapter 5, Section 5.4), the former possibilistic answer set is preferred to the latter. Although the result looks similar to what is obtained by the approach described in this chapter, it can be observed that the second alternative (not taking the umbrella) is not a valid solution in our approach. Indeed, it is a preference that it can never be satisfied and its pessimistic utility is equal to 0. The handling of uncertainty and preferences together is not aligned with the formulation of qualitative DMU as understood in the possibilistic logic setting (as also discussed at the end of Chapter 5).

Instead, the methodology in this chapter is aligned with the possibilistic logic view. It provides a computational solution for optimal decisions within two ASP-based frameworks: LPODs and its possibilistic extension LPPODs. Moreover, although we have not addressed any implementation issue, our approach can be implemented on top of two existing ASP-based solvers, *psmodels*[16] and *posPsmodels*[17] which provide a computation of the LPODs and LPPODs semantics respectively.

As general improvements, the decision method used is not able to identify decisions that may satisfy all the goals from the highest level to the lowest one, except one goal at some level β. In fact, the algorithm stops at the first unsatisfied preference and does not proceed with preferences at lower strata. The algorithm can be modified accordingly to deal with this case. Definition of $\models_c$ and $\models_p$ can also be extended to handle more complex preference expressions.

[16] `http://www.tcs.hut.fi/Software/smodels/priority/`
[17] `https://github.com/rconfalonieri/posPsmodels`

Part IV

Conclusion and Future Perspectives

11

Conclusion

Throughout the research presented in this thesis, we have aimed to understand the role played by preferences in logic programming in different domains. Indeed, logic programming, by supporting symbolic representation and computation of knowledge, can provide an effective tool to build the reasoning mechanisms needed by intelligent systems which assist users in the fulfillment of complex tasks. In order to support users and to understand user choices, such systems need a concise and processable representation of incomplete and uncertain information. In order to be able to choose among different options in a desirable way, they also need a concise and processable representation of preferences. Preferences can provide an effective way to choose among outcomes, whether these are the most plausible states of the world according to pieces of default knowledge, such as in nonmonotonic and uncertain reasoning, or the most satisfactory states when expressing user preferences in a given context, or decisions under uncertainty such as in qualitative DMU.

The overall objective of this thesis has turned to be an attempt to contribute on these active research trends in the field of logic programming. Therefore, our research has consisted of a collection of results on the study of the *representation of preferences* focusing on different domains such as *reasoning under incomplete and uncertain information*, *user preference modeling*, and *qualitative decision making under uncertainty*. From the research performed, we can conclude that preferences have a multi-faceted relationship with several knowledge representation domains and they have been understood in different ways in the literature. We believe that our proposed research analysis, frameworks, and methods might prove to be useful for the logic programming research towards the design of automated reasoning mechanisms for handling incomplete and uncertain knowledge, user preferences, and decision under uncertainty.

Apart of the individual discussions that we have provided at the end of each chapter, in the following section, we briefly discuss and summarize the achieved results. After that, we point out some future extensions and we provide some possible directions for future work.

11.1 Summary

In the first part of this thesis, our main results are focused on preferences in nonmonotonic and uncertain nonmonotonic reasoning (Part I).

In nonmonotonic reasoning, preferences are used to select default rules that support conflicting conclusions. In Chapter 3, we have seen how preferences in logic programming are used in two ways: *explicit* and *implicit*. Most approaches belong to the former category. Indeed, in logic programming, exceptions to default rules are encoded by means of negated-by-failure atoms in order to capture exceptional situations. Then, rules are selected by means of explicit preferences expressed in different ways: literal-oriented, rule-oriented or context-dependent such as in LPODs. On the other hand, implicit preferences can provide an effective mechanism for handling incomplete information in logic programming. Implicit preferences can emerge by considering the specificity of rules according to the information described by the defaults. While there are a lot of approaches for handling preferences in an explicit way in logic programming, only a few approaches have been proposed to handle preferences implicitly. Therefore, our first contribution has been:

C1: An approach to nonmonotonic reasoning using implicit preferences

In Chapter 4, we have proposed a logic programming procedure able to consider implicit preferences based on the specificity of defaults to handle incomplete information in extended definite programs. The specificity has been obtained by means of the Z-ordering of System Z [Pearl, 1990] adapted to logic program. Following the work in [Garcia et al., 2009], we have defined the tolerance of a rule w.r.t. other rules in terms of atom set inclusion rather than logical entailment (as it originally happens in System Z). Based on the partitions obtained by the Z-ordering, we have designed a rewriting procedure which converts default rules into strict rules in which exceptions are made explicit. In particular, we have been able to show that exceptions to logic programs rules can be caught in terms of strong negation and a set of completion rules aiming at completing the incomplete information in the program. This result strengthens an earlier result by Dimipoulos and Kakas [Dimopoulos and Kakas, 1995], showing how nonmonotonic reasoning can be handled with preferences rather than negation as failure. Furthermore, we have established that, under certain conditions, we can recover the results obtained by the method proposed in [Garcia et al., 2009] which encode exceptions by means of negation as failure (Proposition 4.3). This result is significant, since it suggests how negation as failure can be captured, in the class of stratified logic programs, by an operational approach which can provide a dual view to nonmonotonic reasoning in logic programs. The proposed approach has turned to provide a methodology to restore the consistency of inconsistent programs that implicitly involve a specificity ordering between the rules (Proposition 4.2). Restoring the consistency of a logic program is an interesting feature which can be used in different ways: in qualitative decision making (Chapter 10), when knowledge about the world is inconsistent, exceptions have to be made explicit before proceeding to the computation of an optimal decision; in a knowledge integration scenario, when the knowledge coming from different agents is inconsistent, the proposed method can be used to solve inconsistent situations (as also discussed in Section 4.4).

However, the representation of simple default statements encoded as extended definite rules is of quite limited applicability for representing user knowledge in an intelligent system. In particular, having only one option in each context does not give so much flexibility in modeling different choices. On the other hand, the LPODs framework [Brewka et al.,

Table 11.1. Summary of the results in the first part

Objs.	Contribution	Domain	Preference Type	Main Feature(s)	Chapter
1-2	**C1**	nonmonotonic reasoning	implicit / context-dependent	- exceptions handling - inconsistency handling	4
3-4	**C2**	uncertain nonmonotonic reasoning	explicit / context-dependent	- uncertainty handling - possibilistic answer sets selection	5

2004b] allows to specify more than one option in a given context and to select answer sets by means of the preference degrees induced by the ordered disjunction operator $\times$ (see Chapter 2, Section 2.2.4). Indeed, LPODs is designed to specify *context-dependent preferences* which can be used to do two things: to specify a preference order among exceptions for the selection of default rules (as we have considered in Chapter 5) and to specify a preference order among several alternatives encoding *user preferences* (as we have considered in Chapter 7).

Under the former view, when information is uncertain, the selection of exceptions to default rules can also be a matter of uncertainty. Indeed, one can be interested in selecting default rules based on the most certain rules in a logic program. However, most of the approaches in logic programming able to deal with uncertainty do not consider nonmonotonic reasoning or, if they do, no notion of explicit preference has been considered (Chapter 3). Therefore, our second contribution has been:

C2: A framework for dealing with explicit preferences and uncertainty

In Chapter 5, we have proposed a possibilistic logic programming framework, called Logic Programs with Possibilistic Ordered Disjunction (LPPODs), which supports the selection of uncertain default rules. The framework is able to model explicit preferences about rules having exceptions and certainty degrees in terms of necessity values according to possibilistic logic [Dubois et al., 1994]. LPPODs is the first logic programming specification able to consider explicit preferences and uncertainty together based on possibilistic logic and answer set semantics. LPPODs is a smooth extension of two existing logic programming frameworks: LPODs and possibilistic normal programs [Nicolas et al., 2006].

On one hand, from LPODs, the framework inherits the distinctive feature of expressing explicit context-dependent preferences among different exceptions to default rules (modeled as literals of a logic program). On the other hand, possibilistic normal programs allow to capture certainty degrees expressing to what extent a rule is certain (modeled as necessity values according to possibilistic logic) and to associate certainty degrees to possibilistic answer sets (in terms of possibilistic answer set semantics). The selection of the most plausible beliefs induces an order among the possibilistic answer sets of an LPPOD. In order to define such ordering, we have defined a possibilistic comparison criterion (Definition 5.17) which is able to consider both preference and certainty degrees. To be able to consider certainty degrees, the comparison criterion is applied to the normal form of an LPPOD which has been obtained by means of a set of transformation rules. Such transformation rules allow to propagate certainty values between program rules preserving the LPPODs seman-

tics (Lemma 5.1 and Theorem 5.1). The preference relation turns to properly generalize the comparison criterion for LPODs (Proposition 5.4). One important aspect of the LPPODs framework is that the its semantics is computable (Theorem 5.2) and that its complexity belongs to the same complexity class of its classical part, that is, LPODs. This has been an important result, since it has shown that LPPODs can yield a more expressive framework without increasing the complexity of the semantics we have defined.

Table 11.1 summarizes the results we have achieved in this first part.

On the other hand, preferences in logic programs are not only a way for selecting default rules implicitly or explicitly. In the modeling of user preferences (Part II), logic programs can offer an effective way for representing a preference model. Indeed, several logic programming extensions not only can encode user preferences in a compact way, but they can also draw an order between the outcomes of the preference model (Chapter 6). For instance, LPODs is a suitable specification for capturing context-dependent preferences and for specifying an order among several preferences of the users (modeled in terms of the literals of the answer sets of an LPOD). Capturing context-dependent preferences and drawing an order among the outcomes of a preference model are important features for building an effective preference handling method when modeling user preferences in context-aware (information) systems (Chapter 6). Indeed, these systems need to characterize user in a personalized way in order to be able to filter big amount of information and to provide personalized content to users. Personalization can be achieved by modeling user preferences in terms of user profiles. However, the modeling of user profiles bring several issues related to knowledge representation. User profiles should be rich enough to represent preferences that can depend both on contextual and incomplete information, but sufficiently compact to be processed in a fast way. Moreover, user preferences can change over the time. This means that preferences in some contexts can be more important than others or that some preferences can become obsolete. These concerns have suggested to explore for an enhancement of the personalization capabilities in a context-aware system by means of a logic programming-based preference model. Therefore, our third contribution has been:

C3: A logic programming-based preference model for handling user preferences in a context-aware system

In Chapter 7, we have proposed to model user preferences in a specific use-case of context-aware systems, Interactive Community Displays (ICDs), by means of a preference handling method based on the LPPODs framework (Chapter 5). Although LPPODs was introduced in the context of uncertain nonmonotonic reasoning, its preference representation capabilities have fit particularly well for building the preference model imposed by the requirements of the ICDs use-case (Section 7.1). First, most of the time, preferences depend on contextual information such as *date* and *time*, *user location*, *weather condition*, *etc*. For instance, the content suggested to a user who is staying in a given location at a given time, asking for a restaurant suggestion, has to be clearly different from the content suggested when (s)he is asking for a cinema suggestion. This context dependency has been captured in terms of possibilistic ordered disjunction rules (Section 7.5). Secondly, preferences can depend on incomplete information: normally, a user can have some preferences

Table 11.2. Summary of the results in the second part

Objs.	Contribution	Domain	Preference Type	Main Feature(s)	Chapter
5-6-7	**C3**	preference modeling / nonmonotonic reasoning	explicit / context-dependent / weighted	- user profile modeling	7
8-9	**C4**	preference modeling / nonmonotonic reasoning	explicit / context-dependent / nested	- rich syntax	8

unless some exceptional conditions are met. For instance, a user can prefer to go to a restaurant with a terrace rather to a restaurant without terrace unless the weather conditions are not favorable (*e.g., not rain*). This has been captured by negation as failure (Section 7.5). Thirdly, user profiles should be represented in a compact way and processed in a quick way. Users can have many preferences and preventing the user from having to wait too long is a main challenge to be considered in these systems. This has been achieved by representing user profiles in terms of LPPODs and by implementing the LPPODs semantics in the ASP-based solver *posPsmodels* (Section 7.6.1). Finally, preferences in a given context can be more important than others and a mechanism to measure preference importance was needed. Indeed, in such systems, users can have many preferences and an approach that considers all the specified preferences as equally important at the moment of selecting the content to suggest based on user preferences can be too rigid. Therefore, we have reinterpreted necessity values associated to LPPODs rules as weights for expressing the importance of preference rules, and, consequently, of the preferences contained in the rules (Section 7.6.2). If LPPODs has shown to provide an effective way to encode and process user preferences, it is unlikely that, at application level, preferences will be directly encoded in a symbolic way. Since the use of ontologies has become a common practice in knowledge system nowadays, we have integrated the LPPODs specification by means of a user profile and preference ontology (Section 7.3). Then, user profiles have been translated into the LPPODs symbolic representation and processed in a practical way by means of the *posPsmodels* solver.

However, as discussed in Section 7.9, preference expressions supported by LPPODs are quite limited. Indeed, since LPPODs generalizes LPODs (see Chapter 5), it inherits the expressiveness of LPODs which is limited to capture preference statements consisting of singleton preference literals. To be able to model more complex context-dependent preference expressions, our fourth contribution has consisted in:

C4: A framework for handling nested preferences

In Chapter 8, we have provided an extension of LPODs, called Nested Logic Programs with Ordered Disjunction (LPODs$^+$), which allows the formulation of nested preference expressions built by means of connectives $\{\vee, \wedge, \neg, not, \times\}$. We have shown how LPODs$^+$ can be defined in an easy way (Definition 8.4, 8.5 and 8.6) by reusing and extending some definitions related to the syntax and semantics of nested logic programs [Lifschitz et al.,

Table 11.3. Summary of the results in the third part

Objs.	Contribution	Domain	Preference Type	Main Feature(s)	Chapter
10-11-12	**C5**	qualitative decision making under uncertainty	meta-preferences	- computation of pessimistic and optimistic criteria	10

1999]. Moreover, LPODs$^+$ properly generalizes LPODs (Proposition 8.2, 8.3) and nested logic programs (Proposition 8.1). Since the syntax of LPOD$^+$ is too complex to be handled by existent answer set solvers, we have reduced the problem of inferring the answer sets of an OD$^+$-program to the problem of inferring the answer sets of an equivalent disjunctive logic programs by means of a faithful translation procedure (Theorem 8.1). This is an important result. In fact, despite the flexibility added at the syntactic level, the complexity of deciding whether an OD$^+$-program has at least one answer set corresponds to the complexity class of disjunctive logic programs, that is, Σ_2^P [Eiter and Gottlob, 1995]. In this way, complexity results of LPODs$^+$ can be lifted by the Σ_2^P-completeness of disjunctive logic programs.

Table 11.2 summarizes the results we have achieved in this second part.

User preferences can also be a matter for making decisions under uncertainty. Indeed, in the last part of this thesis (Part III), we have considered the notion of preference in qualitative DMU. Existing logic programming-based methodology for handling qualitative DMU lack a satisfactory handling of preference and uncertainty. In most of the approaches proposed, no explicit notion of uncertainty is considered. Furthermore, preferences are assumed to belong to the same program encoding the knowledge base at hand. However, in classical DMU, knowledge about the world and preferences are handled by two separate knowledge bases and the selection of an optimal decision is a matter of certainty and preference degrees. Therefore, our fifth contribution has been:

C5: An ASP-based methodology for computing decision under uncertainty

Contrary to what existing logic programming-based methodologies for handling qualitative decision proposed (see Chapter 9), in Chapter 10, we have associated preferences and knowledge about the world with priority and certainty degrees modeled in terms of necessity values (according to possibilistic logic) and we have treated them as two different knowledge bases. Indeed, in the proposed approach, optimal decisions are a matter of certainty and preference degrees and they can be ranked according to a pessimistic and optimistic decision criteria (already proposed in the possibilistic logic setting). We have provided a logic programming-based computation of these criteria. First, we have encoded a decision problem in terms abduction and LPODs. Then, we have generalized this result to the case in which knowledge and preferences are matters of degrees using the LPODs and LPPODs frameworks. The proposed methodology attempt to bridge the gap between logic programming and qualitative DMU since, to the best of our knowledge, any proposal with a satisfactory handling of preferences under uncertainty was made in this respect.

Table 11.3 summarizes the results we have achieved in this third, and last, part.

11.2 Possible Extensions and Future Work

As far as frameworks and methods presented in this thesis are concerned, the following extensions/improvements are possible:

- **Rule ranking**: the ranking of default rules we proposed in Chapter 4 is currently based on the Z-ordering. If the determination of the tolerance for few rules seems easy, it is not true in general. Computing such ordering is expensive and in the worst case, for a program P of n rules, it is necessary to isolate and stratify the 2^n subsets of P. This raises the question whether another ordering can be used to replace the Z-ordering.[1]
- **LPOD$^+$ framework**: concerning the computation of the optimal answer set of an OD$^+$-program, we have designed the implementation of a tester program in order to compare the answer sets of an OD$^+$-program. Although the tester design provides a first approximation, it does not look general enough to provide a general methodology for computing the satisfaction degree of an answer set w.r.t. $\{\vee, \wedge, not, \times\}$ formulas. This is due to the fact that, since formulas in an OD$^+$-program can contain expressions with the $\times$ arbitrarily nested, the nesting level of $\times$ is not known a priori. This suggests that the use a recursive construction for building the tester program can be valuable. We believe that recent advances on recursive aggregates in ASP [Faber et al., 2011] can provide the appropriate constructs for computing the satisfaction degree of a $\{\vee, \wedge, not, \times\}$ formula, and, as a consequence, for designing a more general tester program.
- **Preference expressions**: in the ASP model for qualitative DMU proposed in Chapter 10, user/agent preferences are limited to preference literals only. However, richer preference expressions need sometimes to be captured, especially when more realistic decision scenarios are addressed. Preference statements should be generalized not only to formulas built with disjunction, conjunction and implication, but also to more advanced constructs such as weighted disjunction and conjunction [Brewka et al., 2004b; Bosc et al., 2010]. Such constructs might improve the flexibility of the model in the encoding of preference statements.

As a result of the contributions of this work, we envision some future lines of research:

- **Merging Logic Programs**: the rewriting procedure proposed in Chapter 4 isolates conflicting default rules (belonging to different levels of specificity) in order to establish a consistency between all the rules contained in an extended definite logic program. From a wider perspective, the method can be thought as a procedure which can be used to merge different knowledge bases, modeled in terms of logic programs (as also discussed at the end of Chapter 4). Indeed, the problem of merging multiple, potentially conflicting pieces of information, arises in different contexts. For example, an agent may receive reports from differing sources of knowledge. In this case, the problem of combining logic programs that may be jointly inconsistent in order to get a consistent logic program is interesting. This observation naturally raises the question whether the merging procedure and merging operators, studied in the literature, can be related. The problem

[1] This is more an open issue rather than an extension.

of merging multiple, potentially conflicting bodies of information is a well studied problem in classical logics [Konieczny and Pérez, 2011, 1998; Grégoire and Konieczny, 2006]. Therefore, there are a lot of properties that could/should be investigated. However, there is one main difference from the classic work on merging and our approach, and that is the consideration of default rules. Merging and information fusion approaches mostly deal with classical logics. Thus, we believe that it would be interesting to investigate how the properties of classical merging can be applied to a logic programming setting with default rules. This should be an interesting future research direction, as there is not so much work on merging logic programs, apart of the works in [Hué et al., 2009; Delgrande et al., 2009]. This might also be interesting from another perspective, *i.e.,* belief revision in logic programming which is a similar problem [Delgrande et al., 2008; Krumpelmann and Kern-Isberner, 2010].

- **Argumentation-based Decision Making**: the qualitative DMU model proposed in Chapter 10 is based on the comparative evaluation of different alternatives by means of pessimistic and optimistic criteria. Therefore, the decision process is limited to decision criteria and it prevents the user from understanding why a decision is worse or better than another one. On the other hand, approaches based on argumentation theory [Fox et al., 2007; Dung et al., 2008; Amgoud and Prade, 2009; Matt, 2010] seem to be flexible enough for justifying/explaining rational decisions. In particular, the work in [Amgoud and Prade, 2009] provides an argumentation framework for explaining decisions in the possibilistic logic setting. It articulates optimistic and pessimistic decision criteria in terms of an argumentation process that consists of several steps: the construction of *pros* and *cons* arguments w.r.t. decisions; the evaluation of the strengths of these arguments; and the comparison of possible decisions on the basis of the relations between arguments. In this respect, we aim at extending the model proposed in Chapter 10 within an argumentation framework, in order to cast the work in [Amgoud and Prade, 2009] into a logic programming setting. This is promising since, recently, quite few works have been done on using ASP as a mechanism for computing extensions in argumentation [Toni and Sergot, 2011]. Moreover, possibilistic semantics for logic programming already exist. A first attempt in the direction of explaining decisions using an argumentation framework in a logic programming setting is our preliminary work in [Nieves and Confalonieri, 2011] based on possibilistic well-founded semantics. However, the selection of an optimal decision is due to preference relations over extensions and arguments supporting a decision, rather than optimistic and pessimistic decision criteria such as in [Amgoud and Prade, 2009].

Publications of the Author

[Confalonieri et al., 2009a] Confalonieri, R., Nieves, J., and Vázquez-Salceda, J. (2009). Pstable Semantics for Logic Programs with Possibilistic Ordered Disjunction. In Serra, R. and Cucchiara, R., editors, *Proceedings of the 11th International Conference of the Italian Association for Artificial Intelligence on Emergent Perspectives in Artificial Intelligence, (AI*IA '09)*, volume 5883 of *Lecture Notes in Artificial Intelligence*, pages 52–61. Springer Berlin, Berlin, Heidelberg.

[Confalonieri et al., 2009b] Confalonieri, R., Nieves, J. C., and Vázquez-Salceda, J. (2009). A Preference Meta-Model for Logic Programs with Possibilistic Ordered Disjunction. In Vos, M. D. and Schaub, T., editors, *Proceedings of the 2nd International Workshop on Software Engineering for Answer Set Programming (SEA'09)*, volume 546 of *CEUR Workshop Proceedings*, pages 19–33. CEUR-WS.org.

[Confalonieri et al., 2010] Confalonieri, R., Nieves, J. C., Osorio, M., and Vázquez-Salceda, J. (2010). Possibilistic Semantics for Logic Programs with Ordered Disjunction. In Link, S. and Prade, H., editors, *Proceedings of 6th International Symposium on Foundations of Information and Knowledge Systems, (FoIKS 2010)*, volume 5956 of *Lecture Notes in Computer Science*, pages 133–152. Springer-Verlag, Berlin, Heidelberg.

[Confalonieri et al., 2010a] Confalonieri, R., Nieves, J. C., Osorio, M., and Vázquez-Salceda, J. (2010). Possibilistic Semantics for Logic Programs with Ordered Disjunction. FoIKS 2010 Invited Paper, Special Issue of Annals of Mathematics and Artificial Intelligence, ISSN 1012-2443. Submitted in July 2010.

[Confalonieri and Nieves, 2010b] Confalonieri, R. and Nieves, J. C. (2010). Nested Logic Programs with Ordered Disjunction. In Osorio, M., Zepeda, C., Olmos, I., Carballido, J. L., and Josè Arrazola, C. M., editors, *Proceedings of the 6th Latin American Workshop on Non-Monotonic Reasoning (LANMR'10)*, volume 677 of *CEUR Workshop Proceedings*, pages 55–66. CEUR-WS.org.

[Confalonieri et al., 2011] Confalonieri, R., Prade, H., and Nieves, J. C. (2011). Handling Exceptions in Logic Programming without Negation as Failure. In Liu, W., editor, *Proceedings of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2011)*, volume 6717 of *Lecture Notes in Artificial Intelligence*, pages 509–520. Springer-Verlag, Berlin, Heidelberg.

[Confalonieri and Prade, 2011] Confalonieri, R. and Prade, H. (2011). Answer Set Programming for Computing Decisions Under Uncertainty. In Liu, W., editor, *Proceedings of the*

11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2011), volume 6717 of *Lecture Notes in Artificial Intelligence*, pages 485–496. Springer-Verlag, Berlin, Heidelberg.

[Nieves and Confalonieri, 2011] Nieves, J. C. and Confalonieri, R. (2011). A Possibilistic Argumentation Decision Making Framework with Default Reasoning. *Accepted in Fundamenta Informaticae*, ISSN 0169-2968.

[Confalonieri and Nieves, 2011] Confalonieri, R. and Nieves, J. C. (2011). Nested Preferences in Answer Set Programming. *Accepted in Fundamenta Informaticae*, ISSN 0169-2968.

A

Proofs

A.1 Proofs of Chapter 4

Proposition 4.1 *Algorithm 1 terminates.*

Proof. The proof will consist in showing that step 3, *i.e.*, the rewriting procedure, in algorithm 1 terminates, since all the other steps obviously terminate.

It can be noticed that algorithm 2 examines each rule of each stratum. For a rule of a stratum Δ_k, the algorithm executes at most two consistency tests with each rule of strata of rank greater or equal to $k+1$. Since each stratum is finite, the algorithm terminates.

□

Proposition 4.2 *The program $P_{\langle S,FC,CR\rangle}$ returned by Algorithm 1 is consistent.*

Proof. The proof will consist in showing that the set of rewritten rules S is consistent, since the consistency test done by Algorithm 3 returns a program $P_{\langle S,FC,CR\rangle}$ in which completion rules are consistent w.r.t. the set of rules S and the factual knowledge FC.

At the beginning, S is consistent since it is built on the set Δ_n of rules tolerated by the set $\Delta\backslash(\Delta_0,\ldots,\Delta_{n-1})$. At each step, a rule r_Δ is added to S only if its head is consistent with all the heads of the rules in S. Otherwise, for a rule $r_{\Delta_k} = h_{\Delta_k} \leftarrow b_{\Delta_k}$ from a stratum Δ_k, if it exists a rule $r_s = h_s \leftarrow b_s$ in S such that (for simplicity and without loss of generality we assume that the rules body of r_s and r_{Δ_k} are singleton literals) $\mathbf{cons}(\{h_{\Delta_k}\} \cup \{h_s\}) \ \wedge \ \mathbf{incons}(\{b_{\Delta_k}\} \cup \{b_s\})$, then r_{Δ_k} is replaced by a rule $r = l_{\Delta_k} \leftarrow l_{\Delta_k} \cup \varphi(l_s)$ (where φ is the mapping function we defined in Definition 4.2). Note that $l_{\Delta_k} \cup \varphi(l_s)$ is consistent since, by construction, every rule of Δ_{k+1} is tolerated by r. r modified by specifying all its exceptions is added to S only when it does not exist any rule in Δ_{k+1} such that its conclusion is inconsistent with b_{Δ_k}. Thus, S remains consistent.

□

Proposition 4.3 *Let Δ be a set of extended definite rules with an implicit inheritance relation, FC be a factual context, and AR be a pair of additional rules according to Definition 4.9. Let P' be the extended normal logic program obtained by the algorithm in [Garcia et al., 2009]. Let $\Delta' = \Delta \cup AR$,*

and $P_{\langle S,FC,CR\rangle}$ be the definite program obtained from Δ' by Algorithm 1. If M is answer set of $P' \cup FC$ and $M' = Cn(P_{\langle S,FC,CR\rangle})$ then $M \subseteq M'$.

Proof. Ab absurdo: the proof will consist in assuming that $M \not\subseteq M'$, *i.e.*, $M' \subset M$. First it can be noticed that since Δ contains a set of rules with an implicit inheritance relation, the program $P' \cup FC$ is stratified and it has a unique answer set [Baral, 2003]. For having $M' \subset M$, program $P' \cup FC$ must have an extra rule that is supported by the other rules such that its head can be added to the set of literals M. But this contradicts the hypothesis that the program $P' \cup FC$ and $P_{\langle S,FC,CR\rangle}$ are generated by the same set of extended definite rules Δ (and generally extra rules can be added for the generation of the program $P_{\langle S,FC,CR\rangle}$ only). Therefore, $M \subseteq M'$.

□

A.2 Proofs of Chapter 5

Proposition 5.2 *Let P be an LPPOD, and M^* be an answer set of P^*. If $\forall r \in P$, $Nec(r) = \mathbf{1}$, then $M = \{(\varphi, \mathbf{1}) \mid \varphi \in M^*\}$ is a possibilistic answer set of P.*

Proof. By Definition 5.5 we know that if M^* is answer set of P^* then $M = \Pi Cn(P_{\times}^{M^*})$ is a possibilistic answer set of P. We have to prove that the necessity values of the atoms of the possibilistic answer set of the possibilistic definite logic program $P_{\times}^{M^*}$ corresponds to 1. By the possibilistic consequence operator definition (Definition 5.4), we know that given a possibilistic set of atom M, the applicability degree β of a rule in M captures the necessity of the conclusion that the rule can produce w.r.t. M. Given a rule $r = 1 : c \leftarrow \mathcal{B}^+$, we consider the following cases:

Case (i) : $\mathcal{B}^+ = \emptyset$, then the rule is 1-applicable in M
Case (ii) : $\mathcal{B}^+ \not\subseteq M$, then the rule is 0-applicable in M.
Case (iii) : $\mathcal{B}^+ \subseteq M$, then the rule is 1-applicable in M since $min\{\alpha, \alpha_1, \ldots, \alpha_m\} = 1$ in which $\forall (b_i, \alpha_i) \in \mathcal{B}^+$, $1 \leq i \leq m$, $(b_i, \alpha_i) \subseteq M$ and $\alpha = \alpha_1 = \ldots = \alpha_m = 1$.

The necessity values of the atoms contained in the fix-point reached by the possibilistic consequence operator (Definition 5.4) are inferred by

$$\Pi T_P(M) = \{(\varphi, \delta) | \varphi \in head(P^*), App(P, M, \varphi) \neq \emptyset, \\ \delta = max_{r \in App(P,M,\varphi)}\{\beta | r \text{ is } \beta\text{-applicable in } M\}\}$$

All applicable rules are 1-applicable in M. Then, $\delta = 1$ and $M = \{(\varphi, 1) \mid \varphi \in M^*\}$ is a possibilistic answer set of P.

□

Proposition 5.3 *Let P be an LPPOD and M be a set of possibilistic atoms such that M^* is an answer set of P^*, then*

$$\Pi M(P_{\times}^{M^*}) = \{(\varphi, N_{P_{\times}^{M^*}}(\varphi)) | \varphi \in \mathcal{L}_{P_{\times}^{M^*}}, N_{P_{\times}^{M^*}}(\varphi) > 0\}$$

is a possibilistic answer set of P.

Remark A.1. $P_{\times}^{M^*}$ is a possibilistic definite logic program.

Proof. By Definition 5.5, given that M^* is an answer set of P^*, M is a possibilistic answer set of P if and only if $M = \Pi Cn(P_{\times}^{M^*})$. We have to prove that $\Pi(P_{\times}^{M^*}) = \Pi Cn(P_{\times}^{M^*})$. This follows in a straightforward way by Theorem 1 in [Nicolas et al., 2006], since $P_{\times}^{M^*}$ is a possibilistic definitive logic programs. Thus, M is a possibilistic answer set of P.

□

Lemma 5.1 *Let P and P' be two LPPODs related by any transformation in $\mathcal{CS}_{LPPOD}$. Then $SEM_{LPPOD}(P) \equiv_p SEM_{LPPOD}(P')$.*

Definition A.1 (Rewriting System for LPODs). [Confalonieri et al., 2010] *Let P be an LPOD and $\mathcal{CS}_{LPOD}$ be the rewriting system based on transformation rules $\{\rightarrow_{EC}, \rightarrow_{RED^+}, \rightarrow_{RED^-}, \rightarrow_{S}, \rightarrow_{F}, \rightarrow_{Loop}\}$, where each transformation is defined by redefining $\mathcal{CS}_{LPPOD}$ omitting the necessity values associated to programs rules in each transformation.*

Remark A.2. It can be noticed how $SEM_{LPPOD}(P)^* \equiv_p SEM_{LPPOD}(P')^*$ is equivalent to $SEM_{LPOD}(P^*) \equiv SEM_{LPOD}(P'^*)$, where SEM_{LPOD} is the LPODs semantics (Definition 2.5), and $\equiv$ denotes the equivalence relation w.r.t. the LPODs semantics.

Remark A.3. Let P be an LPPOD. Then $unf(P) = \mathcal{L}_{P^*} \backslash MM(def(ordis - nor(P)^*)$

Remark A.4. Let P be a definite logic program, the least model M of P can be computed as the least fix-point of the consequence operator $T_P : 2^{\mathcal{L}_P} \rightarrow 2^{\mathcal{L}_P}$ such that $T_P(M) = head(App(P, M)$.

Proof. To prove that $SEM_{LPPOD}(P) \equiv_p SEM_{LPPOD}(P')$ we have to show that given two LPPODs P, P' related by any transformation $\{\rightarrow_{PeC}, \rightarrow_{PRED^+}, \rightarrow_{PRED^-}, \rightarrow_{PS}, \rightarrow_{PF}, \rightarrow_{PLoop}\}$, $\Pi Cn(P_{\times}^{M^*}) = \Pi Cn(P_{\times}^{'M^*})$ holds. As the LPPODs semantics is a generalization of the LPOD semantics, we prove this lemma by first proving that the transformation rules for LPODs are invariant w.r.t. the LPODs semantics.

To this end, let us consider P^*, P'^*, and $\mathcal{CS}_{LPOD}$ (Definition A.1). By Remark A.2 we have to show that $SEM_{LPOD}(P_{\times}^{*M^*}) = SEM_{LPOD}((P')_{\times}^{*M^*}))$. We can observe that $P_{\times}^{*M^*}$ and $(P')_{\times}^{*M^*}$ are definite logic programs. From [Brewka et al., 1997], it is known that the stable semantics is closed w.r.t. the rewriting system $\mathcal{CS}'_{LPOD} = \mathcal{CS}_{LPOD} \backslash (\rightarrow_{Loop})$ and in [Osorio et al., 2001a] it has been proven that the transformation rule $\rightarrow_{DLoop}$ for disjunctive programs also preserves the stable semantics. In the case of LPODs, the stable semantics is also closed under $\rightarrow_{Loop}$. This is straightforward to see, since we can notice that $\forall M \in SEM_{LPOD}(P)$, $M \cap unf(P) = \emptyset$ by construction (Remark A.3), *i.e.*, only program rules which contain false atoms are removed by $\rightarrow_{Loop}$. Thus, the LPOD semantics is closed under $\mathcal{CS}_{LPOD}$.

Then, we can see in a straightforward way that the LPPODs semantics is closed under the transformation rules in the rewriting system $\mathcal{CS}'_{LPPOD} = \mathcal{CS}_{LPPOD} \backslash (\rightarrow_{PS})$, since the transformation rules $\{\rightarrow_{PeC}, \rightarrow_{PRED^+}, \rightarrow_{PRED^-}, \rightarrow_{PF}, \rightarrow_{PLoop}\}$ do not affect the necessity values associated to program rules.

The only transformation that affects necessity values is the Possibilistic Success. Hence, we have to prove that given two LPPODs P and P', $\rightarrow_{PS}$, and M^* answer set of P^*, $\Pi Cn(P_\times^{M^*}) = \Pi Cn(P_\times^{'M^*})$. For a better visibility, let us call $\Pi Cn(P_\times^{M^*}) = M_1$ and $\Pi Cn(P_\times^{'M^*}) = M_2$.

Now, let us suppose $P = P_1 \cup \{\alpha : \mathcal{C}^\times \leftarrow b. \,;\, \beta : b.\}$ and $P' = P_1 \cup \{min\{\alpha, \beta\} : \mathcal{C}^\times. \,;\, \beta : b.\}$. By applying the $\times$-possibilistic reduction using M^* to both P and P', we obtain $P_\times^{M^*} = P_{1_\times}^{M^*} \cup \{\alpha : c \leftarrow b. \,;\, \beta : b.\}$ and $P_\times^{'M^*} = P_{1_\times}^{M^*} \cup \{min\{\alpha, \beta\} : \mathcal{C}^\times. \,;\, \beta : b.\}$. It can be observed that $P_{1_\times}^{M^*}$ is the same subprogram.

Then, we can observe that given the answer set semantics definition in terms of fix-point of the immediate consequence operator (Remark A.4) and that $\rightarrow_S$ is closed under answer set semantics, since M^* is an answer set of P^*, then $(\Pi Cn(P_\times^{M^*}))^* = (\Pi Cn(P_\times^{'M^*}))^*$, *i.e.*, $M_1^* = M_2^*$.

To prove $\Pi Cn(P_\times^{M^*}) = \Pi Cn(P_\times^{'M^*})$, let us suppose *ab absurdo* that $M_1 \neq M_2$. Since $M_1^* = M_2^*$, this basically means that M_1 and M_2 have the same atoms but they can differ at least in one of the necessity values associated to their atoms. Let us consider P^* and P'^*. Without loss of generality, let us assume that $P_1 = \emptyset$ and $P_1^* = \emptyset$. This means that $M_1^* = M_2^* = \{(c, b)\}$. However, by assuming $M_1 \neq M_2$, $M_1 = \{(c, \delta), (b, \beta)\}$ and $M_2 = \{(c, \delta'), (b, \beta)\}$ with $\delta \neq \delta'$.

By applying the fix-point operator to $P_\times^{M^*}$, we obtain:

$\Pi T^0_{P_\times^M} = \emptyset$

$\Pi T^1_{P_\times^M} = \Pi T_{P_\times^M}(\emptyset) = \{(b, \beta)\}$

$\Pi T^2_{P_\times^M} = \Pi T_{P_\times^M}(\{(b, \beta)\}) = \{(b, \beta), (c, min\{\alpha, \beta\})\}$

$\Pi T^3_{P_\times^M} = \Pi T_{P_\times^M}(\{(b, \beta), (c, min\{\alpha, \beta\})\}) = \{(b, \beta), (c, min\{\alpha, \beta\})\}$

Then, by applying the fix-point operator to $P_\times^{'M^*}$, we obtain:

$\Pi T^0_{P_\times^{'M}} = \emptyset$

$\Pi T^1_{P_\times^{'M}} = \Pi T_{P_\times^M}(\emptyset) = \{(b, \beta)\}$

$\Pi T^2_{P_\times^{'M}} = \Pi T_{P_\times^M}(\{(b, \beta)\}) = \{(b, \beta), (c, min\{\alpha, \beta, min\{\alpha, \beta\}\})\}$

$\Pi T^3_{P_\times^{'M}} = \Pi T_{P_\times^M}(\{(b, \beta), (c, min\{\alpha, \beta, min\{\alpha, \beta\}\})\}) = \{(b, \beta), (c, min\{\alpha, \beta, min\{\alpha, \beta\}\})\}$

Thus, $M_1 = \{(b, \beta), (c, min\{\alpha, \beta\})\}$ and $M_2 = \{(b, \beta), (c, min\{\alpha, \beta, min\{\alpha, \beta\}\})\}$. But the necessity values associated to the atoms are the same and no other rules can have influenced the necessity values of the atoms, since we have assumed that $P_1 = \emptyset$. This contradicts the hypothesis $M_1 \neq M_2$.

Thus, the LPPODs semantics is closed under CS_{LPPOD}.

□

Theorem 5.1 *Let P be an LPPOD. Then $\mathcal{CS}_{LPPOD}$ is confluent, noetherian and $norm_{\mathcal{CS}_{LPPOD}}(P)$ is unique.*

Remark A.5. Let T_1 and T_2 be two transformation rules, and P, P_1, P_2, and P_3 be logic programs. T_1 and T_2 *commute*, if $P \rightarrow_{T_1} P_1$ and $P \rightarrow_{T_2} P_2$ then $P_1 \rightarrow_{T_2} P_3$ and $P_2 \rightarrow_{T_1} P_3$.

Remark A.6. Let T_1 and T_2 be two transformation rules, and P, P_1, P_2 be logic programs. T_1 *absorves* T_2, if $P \rightarrow_{T_1} P_1$ and $P \rightarrow_{T_2} P_2$ then $P_2 \rightarrow_{T_1} P_1$.

Proof. We have to prove only that $\mathcal{CS}_{LPPOD}$ is confluent and noetherian, since by rewriting system theory it is known that if a rewriting system is confluent and noetherian then its normal form always exists and it is unique (see Chapter 2, Section 2.3). Thus, we have to prove that:

- $\mathcal{CS}_{LPPOD}$ is noetherian
- $\mathcal{CS}_{LPPOD}$ is confluent

It is easy to see that $\mathcal{CS}_{LPPOD}$ is noetherian, since all our program transformations decrease the size of an LPPOD.

The confluence of $\mathcal{CS}_{LPPOD}$ can be proved using local confluence as stated by Newman's lemma [Newman, 1942]. According to [Newman, 1942], a noetherian rewriting system is confluent if it is locally confluent. As we know that $\mathcal{CS}_{LPPOD}$ terminates, it is a long but easy exercise to verify the local confluence of $\mathcal{CS}_{LPPOD}$. For doing this, we have to show that for each pair of transformations in $\mathcal{CS}_LPPOD$, either two transformations commute (Remark A.5) or one transformation absorbs the other one (Remark A.6) (commutation and absorption are special cases of local confluence). In the following, we present some cases and we generalize this result for all the other combinations.

Let us suppose that we have an LPPOD P and we apply a transformation to get P_1. Let us also suppose that from P it is possible to apply another transformation to get P_2. Thus, we need to show that it exists P_3 such that P_1 can arrive to P_3 by the use of some transformations (possibly none) and also P_2 can arrive to P_3 by the use of some transformations (possibly none). Let us consider some cases.

Case 1: $\rightarrow_{PeC}$ and $\rightarrow_{PS}$ commute. Let P be:

P: $\alpha_1 : Head_1 \leftarrow c, not\ c,\ Body_1.$
$\alpha_2 : Head_2 \leftarrow e,\ Body_2.$
$\alpha_3 : e.$
P'

in which $\alpha_1 > \alpha_2 > \alpha_3$ are necessity values in $\mathcal{S} = \{\alpha_3, \alpha_2, \alpha_1\}$. P' denotes the rest of the rules in P, $Head_1$ and $Head_2$ denote rules heads, and $Body_1$, $Body_2$ denote the rest of the rules body.
If we apply $\rightarrow_{PS}$ to P we obtain P_1, and if we apply $\rightarrow_{PeC}$ to P we obtain P_2:

P_1: $\alpha_1 : Head_1 \leftarrow c, not\ c,\ Body_1.$ P_2:
$\alpha_3 : Head_2 \leftarrow Body_2.$ $\alpha_2 : Head_2 \leftarrow e,\ Body_2.$
$\alpha_3 : e.$ $\alpha_3 : e.$
P' P'

Now, if we apply $\rightarrow_{PeC}$ to P_1 we obtain P_3, and if we apply $\rightarrow_{PS}$ to P_2 we obtain P_3:

P_3: P_3:
$\alpha_3 : Head_2 \leftarrow Body_2.$ $\alpha_3 : Head_2 \leftarrow Body_2.$
$\alpha_3 : e.$ $\alpha_3 : e.$
P' P'

Thus, for this case both transformation rules $\rightarrow_{PeC}$ and $\rightarrow_{PS}$ commute.

Case 2: $\rightarrow_{PeC}$ and $\rightarrow_{PRED^+}$ commute. Let P be:

P: $\alpha_1 : Head_1 \leftarrow c, not\ c,\ Body_1.$
$\alpha_2 : Head_2 \leftarrow not\ e,\ Body_2.$
$\alpha_3 : e.$
P'

Thus, if we apply $\rightarrow_{PeC}$ to P we obtain P_1, and if we apply $\rightarrow_{PRED^+}$ to P we obtain P_2:

P_1: $\alpha_2 : Head_2 \leftarrow not\ e,\ Body_2.$ $\alpha_3 : e.$ P'

P_2: $\alpha_1 : Head_1 \leftarrow c, not\ c,\ Body_1.$ $\alpha_3 : e.$ P'

Now, if we apply $\rightarrow_{PRED^+}$ to P_1 we obtain P_3, and if we apply $\rightarrow_{PeC}$ to P_2 we obtain P_3:

P_3: $\alpha_3 : e.$ P'

P_3: $\alpha_3 : e.$ P'

Thus, for this second case both transformation rules $\rightarrow_{PeC}$ and $\rightarrow_{PRED^+}$ commute.

Case 3: $\rightarrow_{PLoop}$ absorbs $\rightarrow_{PeC}$. Let P be:

P: $\alpha_1 : a \leftarrow b, not\ b.$
$\alpha_2 : c \leftarrow c.$
$\alpha_3 : d.$

Thus, if we apply $\rightarrow_{PLoop}$ to P we obtain P_1 (since $unf(P) = \{a, b, c\}$), and if we apply $\rightarrow_{PeC}$ to P we obtain P_2:

P_1: $\alpha_3 : d.$

P_2: $\alpha_2 : c \leftarrow c.$ $\alpha_3 : d.$

Now, if we apply $\rightarrow_{PLoop}$ to P_2 we obtain P_1 (since $unf(P) = \{c\}$):

P_1: $\alpha_3 : d.$

P_1: $\alpha_3 : d.$

Thus, $\rightarrow_{PLoop}$ absorbs $\rightarrow_{PeC}$ (it can be shown how $\rightarrow_{PLoop}$ and $\rightarrow_{PeC}$ also commute).

Case 4: $\rightarrow_{PRED^+}$ absorbs $\rightarrow_{PRED^-}$. Let P be:

P: $\alpha_1 : Head_1 \leftarrow not\ b, not\ p,\ Body_1.$
$\alpha_2 : p.$
P'

Let us suppose that $b \notin HEAD(P)$. If we apply $\rightarrow_{PRED^+}$ to P we obtain P_1 and if we apply $\rightarrow_{PRED^-}$ to P we obtain P_2:

P_1: P_2: $\alpha_1 : Head_1 \leftarrow not\ p,\ Body_1$
$\alpha_2 : p.$ $\alpha_2 : p.$
P' P'

Now, if we apply $\rightarrow_{PRED^+}$ to P_2 we obtain P_1:

P_1: P_1:
$\alpha_2 : p$ $\alpha_2 : p.$
P' P'

Thus, $\rightarrow_{PRED^+}$ absorbs $\rightarrow_{PRED^-}$ (it can be shown how $\rightarrow_{PRED^+}$ and $\rightarrow_{PRED^-}$ also commute).
In a similar way, all the following pairs of transformations commute and/or absorb:

Case 5 : $\rightarrow_{PeC}$ and $\rightarrow_{PRED^-}$.
Case 6 : $\rightarrow_{PeC}$ and $\rightarrow_{PF}$.
Case 7 : $\rightarrow_{PeC}$ and $\rightarrow_{PF}$.
Case 8 : $\rightarrow_{PRED^+}$ and $\rightarrow_{PS}$.
Case 9 : $\rightarrow_{PRED^+}$ and $\rightarrow_{PLoop}$.
Case 10 : $\rightarrow_{PRED^-}$ and $\rightarrow_{PF}$.
Case 11 : $\rightarrow_{PRED^-}$ and $\rightarrow_{PS}$.
Case 12 : $\rightarrow_{PRED^-}$ and $\rightarrow_{PLoop}$.
Case 13 : $\rightarrow_{PF}$ and $\rightarrow_{PS}$.
Case 14 : $\rightarrow_{PLoop}$ and $\rightarrow_{PF}$.
Case 15 : $\rightarrow_{PS}$ and $\rightarrow_{PLoop}$.

As CS_{LPPOD} is noetherian and as all pair of transformations in CS_{LPPOD} are locally confluent, then CS_{LPPOD} is confluent.

□

Proposition 5.4 *Let M_1 and M_2 be possibilistic answer sets of an LPPOD P, $\succ_p$ be the preference relation based on rules satisfaction degrees only, and $\succ_{pp}$ be the possibilistic preference relation. If $M_1^* \succ_p M_2^*$ then $M_1 \succ_{pp} M_2$.*

Remark A.7. $M_1^* >_p M_2^*$ iff $\exists\ r \in P^*$ such that $deg_{M_1^*}(r) < deg_{M_2^*}(r)$, and $\nexists r' \in P^*$ such that $deg_{M_2^*}(r') < deg_{M_1^*}(r')$ (Definition 5.16).

Remark A.8. $norm_{\mathcal{LPPOD}}(P)^* = norm_{\mathcal{LPOD}}(P^*)$

Proof. Ab absurdo: let us suppose that $M_1 \not\succ_{pp} M_2$. We have two cases: either (i) $M_2 >_{pp} M_1$ or (ii) $M_2 \not>_{pp} M_1$ (and $M_1 \not\succ_{pp} M_2$).

Case (i): $M_2 >_{pp} M_1$. If $M_2 >_{pp} M_1$, then $\exists\ r \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_2^*}(r^*) < deg_{M_1}(r^*)$, and $\nexists r' \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_1^*}(r'^*) < deg_{M_2^*}(r'^*)$ and $n(r) < n(r')$. By Remark A.8 this would mean that $\exists\ r^* \in norm_{\mathcal{LPOD}}(P^*)$ such that $deg_{M_2^*}(r^*) < deg_{M_1^*}(r^*)$, and $\nexists r'^* \in norm_{\mathcal{LPOD}}(P^*)$ such that $deg_{M_1^*}(r'^*) < deg_{M_2^*}(r'^*)$, *i.e.*, $M_2^* >_p M_1^*$. This contradicts the hypothesis $M_1^* >_p M_2^*$ (Remark A.7).

Case (ii): $M_2 \not>_{pp} M_1$. If $M_2 \not>_{pp} M_1$ (and $M_1 \not>_{pp} M_2$), then neither $\exists\, r \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_2^*}(r^*) < deg_{M_1^*}(r^*)$, and $\nexists r' \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_1^*}(r'^*) < deg_{M_2}(r'^*)$ and $n(r) < n(r')$, nor $\exists\ r \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_1^*}(r^*) < deg_{M_2^*}(r^*)$, and $\nexists r' \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_2^*}(r'^*) < deg_{M_1^*}(r'^*)$ and $n(r) < n(r')$. This is only true if $\forall r \in P$, $deg_{M_1^*}(r^*) = deg_{M_2^*}(r^*)$ and $\exists r' \in P$ such that $deg_{M_1^*}(r^*) = deg_{M_2^*}(r^*)$ and $n(r) = n(r')$. However by Remark A.8 this would mean that $\forall r^* \in norm_{\mathcal{LPOD}}(P^*)$, $deg_{M_1^*}(r^*) = deg_{M_2^*}(r^*)$. Thus, $M_2^* \not>_p M_1^*$ and $M_1^* \not>_p M_2^*$. This contradicts the hypothesis $M_1^* >_p M_2^*$ (Remark A.7).

□

Theorem 5.2 *Let P be an LPPOD, such that $P \subseteq Prog_{\mathcal{A}-\mathcal{S}}$, where $Prog_{\mathcal{A}-\mathcal{S}}$ is the set of all LPPODs which can be generated by a finite set of atoms $\mathcal{A}$ and a finite set of necessity values $\mathcal{S}$. Then, the LPPODs semantics is computed by the function $SEM_{LPPOD} : P \to \mathcal{PS}$.*

Proof. The proof is straightforward, since Algorithm 4 computes the LPPODs semantics.

□

A.3 Proofs of Chapter 8

Proposition 8.1 *Let P be an OD^+-program such that $\forall r \in P$, $r = H \leftarrow B$, H and B be formulas $\times$ free, and M be a set of literals. $M \in SEM_{LPOD^+}(P)$ if and only if $M \in SEM_{NLP}(P)$.*

Remark A.9. Let P be a nested logic program and M be a set of literals. M is an answer set of P if it is an answer set of P^M ([Lifschitz et al., 1999]).

Proof. The proof follows in a straightforward way, since it can be noticed that when $\forall r \in P$, H and B are formulas $\times$ free, $P^M_{\times^+}$ coincides with P^M of nested logic program. Therefore, the $LPODs^+$ semantics (Definition 8.6) and NLP semantics (Remark A.9) coincide as well.

□

Proposition 8.2 *Let P be an OD^+-program such that $\forall r \in P$, $r = A_1 \times \ldots \times A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_{m+n}$ where A_i's ($1 \leq i \leq k$) and B_j's ($1 \leq j \leq m+n$) are literals and M be a set of literals. $M \in SEM_{LPOD^+}(P)$ if and only if $M \in SEM_{LPOD}(P)$.*

Remark A.10. The $P^M_\times$ reduction is defined as $\bigcup_{r \in P} r^M_\times$, where $r^M_\times := \{A_i \leftarrow B_1 \wedge \ldots \wedge B_m | A_i \in M \text{ and } M \cap (\{A_1, \ldots, A_{i-1}\} \cup \{B_{m+1}, \ldots, B_{m+n}\}) = \emptyset\}$ ([Brewka et al., 2004b]).

Remark A.11. Let P be an LPOD, and M be a set of literals. M is an answer set of P ($M \in SEM_{LPOD}(P)$) if and only if a) M is a minimal model of $P^M_\times$ and b) it is closed under P ([Brewka et al., 2004b]).

Proof. The proof consists in showing the two implications:

Case(i): If $M \in SEM_{LPOD^+}(P)$ then $M \in SEM_{LPOD}(P)$. By Definition 8.6, M is answer set of $P^M_{\times^+}$, *i.e.*, M is minimal among the sets of atoms closed under $P^M_{\times^+}$. Moreover, if $M \in SEM_{LPOD^+}(P)$, then M is closed $\forall r \in P$. Thus, condition b) in Remark A.11 is satisfied by definition. It can also be observed that, since M is an answer set of P, $P^M_{\times^+}$ and $P^M_{\times}$ coincide. Since M is a minimal model of $P^M_{\times^+}$ and, the set of rules reduced by $P^M_{\times^+}$ and $P^M_{\times}$ are the same, then M is a minimal model of $P^M_{\times}$ as well.

Case(ii): If $M \in SEM_{LPOD}(P)$ then $M \in SEM_{LPOD^+}(P)$. By Remark A.11, M is a minimal model of $P^M_{\times}$ and it is closed under P. Since M is closed under P, M is closed under $P^M_{\times^+}$, and it is a minimal model of $P^M_{\times^+}$ since, as M is an answer set of P, the two reductions coincide.

□

Proposition 8.3 *Let P be an OD^+-program such that $\forall r \in P$, $r = A_1 \times \ldots \times A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_{m+n}$ where A_i's ($1 \leq i \leq k$) and B_j's ($1 \leq j \leq m+n$) are literals and M be an answer set of P. Then, $deg^+_M(r) = deg_M(r)$.*

Proof. Given an answer set M of an OD^+ program P two situations can occur:

Case (i): the body of the rule r is not satisfied by M. Then, by Definition 8.8, $deg^+_M(r) = 1$ and, by Definition 2.6, $deg_M(r) = 1$. Therefore, $deg^+_M(r) = deg_M(r)$.

Case (ii): the body of the rule r is satisfied by M. Then, by Definition 2.6, $deg_M(r) = min\{j \mid A_j \in M, 1 \leq j \leq k\}$. By Definition 8.8, $deg^+_M(r) = opt_M(A_1 \times \ldots \times A_k)$, in which:

$$opt_M(A_1 \times \ldots \times A_k) \begin{cases} 1, & \text{if } M \models A_1^M \\ 2, & \text{if } M \not\models A_1^M \wedge M \models A_2^M \\ \ldots & \\ k, & \text{if } M \not\models A_1^M \wedge M \not\models A_2^M \wedge \ldots \wedge M \models A_k^M \end{cases}$$

Thus, $deg^+_M(r) = min\{j' \mid M \models A_{j'}, 1 \leq j' \leq k\}$. Since $A_j = A'_j$, then $j = j'$ and $deg^+_M(r) = deg_M(r)$.

□

Proposition 8.4 *For any formula A, B:*

$$A \times B \equiv A \vee (not\ A \wedge B)$$

Proof. For doing this proof we have to show that:

$$X \models (A \times B)^Y \text{ iff } X \models (A \vee (not\ A \wedge B))^Y$$

Then:

$$
\begin{array}{ll}
X \models (A \times B)^Y & \text{iff} \\
(A \times B)^Y = \top & \text{iff} \\
(Y \models A^Y) \vee (Y \not\models A^Y \wedge Y \models B^Y) & \text{iff} \\
A^Y = \top, (not\ A)^Y = \top, B^Y = \top & \text{iff} \\
X \models A^Y \vee (X \models not\ A^Y \wedge X \models B^Y) & \text{iff} \\
X \models A^Y \vee X \models (not\ A^Y \wedge B^Y) & \text{iff} \\
X \models A^Y \vee X \models (not\ A \wedge B)^Y & \text{iff} \\
X \models A^Y \vee (not\ A \wedge B)^Y & \text{iff} \\
X \models (A \vee (not\ A \wedge B))^Y &
\end{array}
$$

□

Proposition 8.5 *For any formula A, B, C,*

1. $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$ *and* $(A \vee B) \vee C \equiv A \vee (B \vee C)$
2. $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
3. $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
4. $not\ (A \wedge B) \equiv not\ A \vee not\ B$
5. $not\ (A \vee B) \equiv not\ A \wedge not\ B$
6. $not\ not\ not\ A \equiv not\ A$
7. $A \leftarrow B \wedge not\ not\ C \equiv A \vee not\ B \leftarrow C$
8. $A \vee not\ not\ B \leftarrow C \equiv A \leftarrow not\ B \wedge C$
9. $A \wedge B \leftarrow C \equiv \begin{cases} A \leftarrow C \\ B \leftarrow C \end{cases}$
10. $A \leftarrow B \vee C \equiv \begin{cases} A \leftarrow B \\ B \leftarrow C \end{cases}$

Proof. The proof of this proposition follows by Proposition 4 and Proposition 6 in [Lifschitz et al., 1999].

□

Lemma 8.1 *Any OD^+-program $P \in LPODs^+_{\mathcal{L}}$ is equivalent to a generalized disjunctive program $P^{\langle not, \vee \rangle} \in GDLPs_{\mathcal{L}}$.*

Proof. As the transformation in Proposition 8.4 transforms nested ordered disjunction formulas to disjunction and conjunction formulas, and since all the other transformations in Proposition 8.5 correspond to those presented in [Lifschitz et al., 1999], this proposition follows from Proposition 7 in [Lifschitz et al., 1999] in a straightforward way.

□

Lemma 8.2 *Any generalized disjunctive logic program $P^{\langle not, \vee \rangle} \in GDLPs_{\mathcal{L}}$ is equivalent to a disjunctive logic program $P^{\langle \vee \rangle} \in DLPs_{\mathcal{L} \cup \mathcal{L}'}$.*

Proof. Since the mapping defined in Definition 8.12 is faithful, *i.e.,* it preserves the semantics of a logic program [Janhunen, 2001; Pearce et al., 2002], it translates a $GDLP_{\mathcal{L}}$ into an equivalent $DLP_{\mathcal{L} \cup \mathcal{L}'}$.

□

Theorem 8.1 *Let $P \in LPODs^{+}_{\mathcal{L}}$ and $P^{\langle\vee\rangle} \in DLPs_{\mathcal{L}\cup\mathcal{L}'}$ the disjunctive logic program obtained by transforming P. If M' is an answer set of $P^{\langle\vee\rangle}$ then $M = M' \cap \mathcal{L}$ is answer set of P.*

Proof. This theorem follows in a straightforward way by transitivity from Lemma 8.1 and Lemma 8.2.

□

A.4 Proofs of Chapter 10

Proposition 10.2 *Let $DM = \langle K, D, Pref\rangle$ be a decision making problem and let P_{dm} be the LPOD generated by Algorithm 5. Then, $\Delta \in label_K(Pref)$ is an optimal pessimistic decision if and only if there exists a consistent maximally preferred answer set M of P_{dm} such that $\Delta = \{d \in D \mid ass(d) \in M\}$.*

Proof. The proof will consist in showing that $K \cup \Delta \models_c Pref$ and that Δ is minimal. The proof for this proposition is similar to the proof provided for Proposition 4.1 in [Brewka, 2006].

$K \cup \Delta \models_c Pref$. $P_{dm} \cup \Delta \models_c Pref$ if and only if it exists a consistent answer set M of $K \cup \Delta \cup \{not\ p \mid p \in Pref\}$. This is true if and only if $\{ass(d) \mid d \in \Delta\} \cup \{d \leftarrow ass(d) \mid d \in D\} \cup K \cup \{not\ p \mid p \in Pref\}$ has a consistent answer set $M \cup \{ass(d) \mid d \in \Delta\}$ and ass does not appear in M, but this can happen if and only if $K \cup \{not\ p \mid p \in Pref\} \cup \{d \leftarrow ass(d) \mid d \in D\} \cup \{\neg ass(d) \times ass(d) \mid d \in D\}$ has a consistent answer set $M \cup \{ass(d) \mid d \in \Delta\} \cup \{\neg ass(d) \mid d \in D \backslash \Delta\}$.

The minimality of Δ can be proved observing that Δ is minimal if and only if $M \cup \{ass(d) \mid d \in \Delta\} \cup \{\neg ass(d) \mid d \in D \backslash \Delta\}$ is a maximally preferred answer set of P_{dm}. This is true since making decision of the form $ass(d)$ is non-preferred according to the ordered disjunction-based construction in P_{dm}.

□

References

[Adams, 1975] Adams, E. W. (1975). *The Logic of Conditionals: An Application of Probability to Deductive Logic*. Reifdel Publishing Company, Dordvecht, Holland.

[Adomavicius and Tuzhilin, 2005] Adomavicius, G. and Tuzhilin, A. (2005). Personalization technologies: a process-oriented perspective. *Communications of the ACM*, 48(10):83–90.

[Alsinet and Godo, 2002] Alsinet, T. and Godo, L. (2002). Towards an Automated Deduction System for First-Order Possibilistic Logic Programming with Fuzzy Constants. *International Journal of Intelligent Systems*, 17(9):887–924.

[Amgoud and Prade, 2009] Amgoud, L. and Prade, H. (2009). Using arguments for making and explaining decisions. *Artificial Intelligence*, 173(3-4):413–436.

[Baldauf et al., 2007] Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.

[Balduccini and Mellarkod, 2003] Balduccini, M. and Mellarkod, V. S. (2003). CR-Prolog with Ordered Disjunction. In Vos, M. D. and Provetti, A., editors, *Proceedings of the 2nd International Workshop on Answer Set Programming, Advances in Theory and Implementation*, volume 78 of *CEUR Workshop Proceedings*, pages 98–112. CEUR-WS.org.

[Baldwin, 1987] Baldwin, J. F. (1987). Evidential support logic programming. *Fuzzy Sets and Systems*, 24(1):1–26.

[Baral, 2003] Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

[Baral and Gelfond, 1994] Baral, C. and Gelfond, M. (1994). Logic Programming and Knowledge Representation. *Journal of Logic Programming*, 19/20:73–148.

[Baral et al., 2009] Baral, C., Gelfond, M., and Rushton, N. (2009). Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming*, 9(1):57–144.

[Bauters et al., 2010a] Bauters, K., Schockaert, S., Cock, M. D., and Vermeir, D. (2010). Possibilistic Answer Set Programming Revisited. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence, (UAI'10)*. AUAI Press.

[Bauters et al., 2010b] Bauters, K., Schockaert, S., Cock, M. D., and Vermeir, D. (2010). Towards Possibilistic Fuzzy Answer Set Programming. In *Proceedings of the 13th International Workshop on Non-monotonic reasoning, (NMR'10)*.

[Bauters et al., 2011] Bauters, K., Schockaert, S., Cock, M. D., and Vermeir, D. (2011). Weak and strong disjunction in possibilistic ASP. In *Proceedings of the 11th International Conference on Scalable Uncertainty Management (SUM 2011), To appear*.

[Benferhat et al., 2004] Benferhat, S., Brewka, G., and Berre, D. L. (2004). On the relation between qualitative choice logic and possibilistic logic. In *Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty (IPMU 2004)*, pages 951–957.

[Benferhat et al., 2002] Benferhat, S., Dubois, D., Kaci, S., and Prade, H. (2002). Possibilistic logic representation of preferences: relating prioritized goals and satisfaction levels expressions. In van Harmelen, F., editor, *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, (ECAI'2002)*, pages 685–689. IOS Press.

[Benferhat et al., 2006] Benferhat, S., Dubois, D., Kaci, S., and Prade, H. (2006). Bipolar possibility theory in preference modeling: Representation, fusion and optimal solutions. *Information Fusion*, 7(1):135–150.

[Benferhat et al., 1992] Benferhat, S., Dubois, D., and Prade, H. (1992). Representing Default Rules in Possibilistic Logic. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning, (KR'92)*, pages 673–684. Cambridge, MA.

[Benferhat et al., 1997] Benferhat, S., Dubois, D., and Prade, H. (1997). Nonmonotonic reasoning, conditional objects and possibility theory. *Artificial Intelligence*, 92(1-2):259–276.

[Benferhat et al., 1998] Benferhat, S., Dubois, D., and Prade, H. (1998). Practical Handling of Exception-Tainted Rules and Independence Information in Possibilistic Logic. *Applied Intelligence*, 9(2):101–127.

[Benferhat et al., 2001] Benferhat, S., Dubois, D., and Prade, H. (2001). Towards a Possibilistic Logic Handling of Preferences. *Applied Intelligence*, 14(3):303–317.

[Benferhat and Sedki, 2008] Benferhat, S. and Sedki, K. (2008). Two alternatives for handling preferences in qualitative choice logic. *Fuzzy Sets and Systems*, 159(15):1889–1912.

[Bienvenu et al., 2010] Bienvenu, M., Lang, J., and Wilson, N. (2010). From preference logics to preference languages, and back. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning, (KR 2010)*. AAAI, Toronto.

[Bonet and Geffner, 1996] Bonet, B. and Geffner, H. (1996). Arguing for Decisions: A Qualitative Model of Decision Making. In Horwitz, E., editor, *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI'96)*, pages 98–105. Morgan Kaufmann.

[Börzsönyi et al., 2001] Börzsönyi, S., Kossmann, D., and Stocker, K. (2001). The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, pages 421–430. IEEE Computer Society, Washington, DC, USA.

[Bosc and Pivert, 1992] Bosc, P. and Pivert, O. (1992). Some approaches for relational databases flexible querying. *Journal of Intelligent Information Systems*, 1(1):323–354.

[Bosc et al., 2009] Bosc, P., Pivert, O., and Prade, H. (2009). A Model Based on Possibilistic Certainty Levels for Incomplete Databases. In Godo, L. and Pugliese, A., editors, *Proceedings of the 3rd International Conference on Scalable Uncertainty Management (SUM'09)*, volume 5785 of *Lecture Notes in Computer Science*, pages 80–94. Springer Berlin - Heidelberg.

[Bosc et al., 2010] Bosc, P., Pivert, O., and Prade, H. (2010). A Possibilistic Logic View of Preference Queries to an Uncertain Database. In *Proceedings of 19th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'10)*, pages 581–595.

[Boutilier, 1994] Boutilier, C. (1994). Toward a Logic for Qualitative Decision Theory. In Doyle, J. and Sandewall, E., editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, (KR'94)*, pages 75–86. Morgan Kaufmann.

[Boutilier et al., 2004] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., and Poole, D. (2004). CP-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21(1):135–191.

[Bouveret et al., 2009] Bouveret, S., Endriss, U., and Lang, J. (2009). Conditional importance networks: a graphical language for representing ordinal, monotonic preferences over sets of goods. In Kitano, H., editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 67–72. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Brafman and Domshlak, 2004] Brafman, R. I. and Domshlak, C. (2004). Database preference queries revisited. Technical Report TR2004-1934, Cornell University - Computing and Information Science.

[Brafman and Domshlak, 2009] Brafman, R. I. and Domshlak, C. (2009). Preference Handling - An Introductory Tutorial. *AI Magazine*, 30(1):58–86.

[Brafman et al., 2006] Brafman, R. I., Domshlak, C., and Shimony, S. E. (2006). On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research*, 25(1):389–424.

[Brafman and Tennenholtz, 1996] Brafman, R. I. and Tennenholtz, M. (1996). On the foundations of qualitative decision theory. In *Proceedings of the 13th National Conference on Artificial Intelligence - Volume 2*, AAAI'96, pages 1291–1296. AAAI Press.

[Brass and Dix, 1995] Brass, S. and Dix, J. (1995). Characterizations of the Stable Semantics by Partial Evaluation. In Marek, V., Nerode, A., and Truszczyński, M., editors, *Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning, (LPNMR'95)*, volume 928 of *Lecture Notes in Computer Science*, pages 85–98. Springer-Verlag, London, UK.

[Brass and Dix, 1999] Brass, S. and Dix, J. (1999). Semantics of (disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 40(1):1–46.

[Brass et al., 2001] Brass, S., Dix, J., Freitag, B., and Zukowski, U. (2001). Transformation-based bottom-up computation of the well-founded model. *Theory and Practice of Logic Programming*, 1(5):497–538.

[Brewka, 1994] Brewka, G. (1994). Adding Priorities and Specificity to Default Logic. In MacNish, C., Pearce, D., and Pereira, L., editors, *Proceedings of the European Workshop on Logics in Artificial Intelligence, (JELIA'94)*, volume 838 of *Lecture Notes in Computer Science*, pages 247–260. Springer-Verlag, London, UK.

[Brewka, 2002] Brewka, G. (2002). Logic programming with ordered disjunction. In *Proceedings of 18th National Conference on Artificial intelligence, (AAAI-02)*, pages 100–105. AAAI Press, Menlo Park, CA, USA.

[Brewka, 2005] Brewka, G. (2005). Answer Sets and Qualitative Decision Making. *Synthese*, 146(1-2):171–187.

[Brewka, 2006] Brewka, G. (2006). Answer Sets and Qualitative Optimization. *Logic Journal of the IGPL*, 14(3):413–433.

[Brewka et al., 2004a] Brewka, G., Benferhat, S., and Le Berre, D. (2004). Qualitative Choice Logic. *Artificial Intelligence*, 157(1-2):203–237.

[Brewka et al., 1993] Brewka, G., Dix, J., and Konolige, K. (1993). A Tutorial on Nonmonotonic Reasoning. In Brewka, G., Jantke, K., and Schmitt, P., editors, *Proceedings of the 2nd International Workshop on Nonmonotonic and Inductive Logic*, volume 659 of *Lecture Note in Computer Science*, pages 1–88. Springer-Verlag, London, UK.

[Brewka et al., 1997] Brewka, G., Dix, J., and Konolige, K. (1997). *Nonmonotonic Reasoning: An Overview*. CSLI Lecture Notes 73. CSLI Publications, Stanford, CA.

[Brewka and Eiter, 1999] Brewka, G. and Eiter, T. (1999). Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356.

[Brewka et al., 2002] Brewka, G., Niemelä, I., and Syrjänen, T. (2002). Implementing Ordered Disjunction Using Answer Set Solvers for Normal Programs. In Flesca, S., Greco, S., Ianni, G., and Leone, N., editors, *Proceedings of the European Conference on Logics in Artificial Intelligence, (JELIA'02)*, volume 2424 of *Lecture Notes in Computer Science*, pages 444–456. Springer Verlag, London, UK.

[Brewka et al., 2004b] Brewka, G., Niemelä, I., and Syrjänen, T. (2004). Logic Programs with Ordered Disjunction. *Computational Intelligence*, 20(2):333–357.

[Brewka et al., 2003] Brewka, G., Niemelä, I., and Truszczyński, M. (2003). Answer Set Optimization. In Gottlob, G. and Walsh, T., editors, *Proceedings of 18th International Joint Conference on Artificial Intelligence, (IJCAI'03)*, pages 867–872. Morgan Kaufmann.

[Brewka et al., 2005] Brewka, G., Niemelä, I., and Truszczyński, M. (2005). Prioritized component systems. In Veloso, M. M. and Kambhampati, S., editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference*, pages 596–601. AAAI Press / The MIT Press.

[Brewka et al., 2008] Brewka, G., Niemelä, I., and Truszczyński, M. (2008). Preferences and Nonmonotonic Reasoning. *AI Magazine*, 29(4):69–78.

[Cabalar, 2011] Cabalar, P. (2011). A Logical Characterisation of Ordered Disjunction. *AI Communications*, 24(2):165–175.

[Ceccaroni et al., 2009] Ceccaroni, L., Codina, V., Palau, M., and Pous, M. (2009). PaTac: Urban, Ubiquitous, Personalized Services for Citizens and Tourists. In *Proceedings of the Third International Conference on Digital Society*, pages 7–12. IEEE Computer Society, Washington, DC, USA.

[Chen and Pu, 2004] Chen, L. and Pu, P. (2004). Survey of Preference Elicitation Methods. Technical Report IC/200467, Swiss Federal Institute of Technology in Lausanne (EPFL), Lausanne, Switzerland.

[Chomicki, 2003] Chomicki, J. (2003). Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):427–466.

[Codina, 2009] Codina, V. (2009). Design, development and deployment of an intelligent, personalised recommendation system. Master's thesis, Universitat Politècnica de Catalunya, Barcelona, Spain.

[Confalonieri et al., 2010c] Confalonieri, R., Nieves, J. C., and Vázquez-Salceda, J. (2010). Towards the Implementation of a Preference- and Uncertain-Aware Solver Using Answer

Set Programming. Technical Report LSI-10-16-R, Universitat Politècnica de Catalunya, Barcelona, Spain.

[Costantini and Formisano, 2009] Costantini, S. and Formisano, A. (2009). Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of Algorithms*, 64(1):3–15.

[Costantini and Formisano, 2011] Costantini, S. and Formisano, A. (2011). Weight Constraints with Preferences in ASP. In Delgrande, J. and Faber, W., editors, *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 229–235. Springer-Verlag, Berlin, Heidelberg.

[Delgrande et al., 2004] Delgrande, J., Schaub, T., Tompits, H., and Wang, K. (2004). A classification and Survey of Preference Handling Approaches in Nonmonotonic Reasoning. *Computational Intelligence*, 20(2):308–334.

[Delgrande and Schaub, 2000] Delgrande, J. P. and Schaub, T. (2000). Expressing preferences in default logic. *Artificial Intelligence*, 123(1-2):41–87.

[Delgrande et al., 2003] Delgrande, J. P., Schaub, T., and Tompits, H. (2003). A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129–187.

[Delgrande et al., 2008] Delgrande, J. P., Schaub, T., Tompits, H., and Woltran, S. (2008). Belief Revision of Logic Programs under Answer Set Semantics. In Brewka, G. and Lang, J., editors, *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 411–421. AAAI Press.

[Delgrande et al., 2009] Delgrande, J. P., Schaub, T., Tompits, H., and Woltran, S. (2009). Merging Logic Programs under Answer Set Semantics. In Hill, P. and Warren, D., editors, *Proceedings of 25th International Conference on Logic Programming (ICLP 2009)*, volume 5649 of *Lecture Notes in Computer Science*, pages 160–174. Springer Berlin / Heidelberg.

[Delgrande and Schaub, 1997a] Delgrande, J. P. and Schaub, T. H. (1997). Compiling reasoning with and about preferences into default logic. In *Proceedings of the 15th International Joint Conference on Artifical intelligence - Volume 1, (IJCAI'97)*, pages 168–174. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Delgrande and Schaub, 1997b] Delgrande, J. P. and Schaub, T. H. (1997). Compiling specificity into approaches to nonmonotonic reasoning. *Artificial Intelligence*, 90(1-2):301–348.

[Denecker and Kakas, 2002] Denecker, M. and Kakas, A. C. (2002). Abduction in Logic Programming. In Kakas, A. C. and Sadri, F., editors, *Computational Logic: Logic Programming and Beyond*, volume 2407 of *Lecture Notes in Computer Science*, pages 402–436. Springer Berlin / Heidelberg.

[Dershowitz and Plaisted, 2001] Dershowitz, N. and Plaisted, D. A. (2001). Rewriting. In Robinson, J. A. and Voronkov, A., editors, *Handbook of Automated Reasoning*, pages 535–610. Elsevier and MIT Press.

[Dimopoulos and Kakas, 1995] Dimopoulos, Y. and Kakas, A. C. (1995). Logic Programming without Negation as Failure. In Lloyd, J. W., editor, *Proceedings of the 1995 International Symposium on Logic Programming*, pages 369–383. MIT Press.

[Dix et al., 2001] Dix, J., Osorio, M., and Zepeda, C. (2001). A General Theory of Confluent Rewriting Systems for Logic Programming and its Applications. *Annals of Pure and Applied Logic*, 108(1–3):153–188.

[Domshlak et al., 2011] Domshlak, C., Hüllermeier, E., Kaci, S., and Prade, H. (2011). Preferences in AI: An overview. *Artifical Intelligence*, 175(7-8):1037–1052.

[Doyle and Thomason, 1999] Doyle, J. and Thomason, R. H. (1999). Background to Qualitative Decision Theory. *AI Magazine*, 20(2):55–68.

[Dubois et al., 2002] Dubois, D., Fargier, H., Prade, H., and Perny, P. (2002). Qualitative decision theory: from Savage's axioms to nonmonotonic reasoning. *Journal of the ACM*, 49(4):455–495.

[Dubois et al., 1991] Dubois, D., Lang, J., and Prade, H. (1991). Towards Possibilistic Logic Programming. In Furukawa, K., editor, *Proceedings of International Conference on Logic Programming, (ICLP'91)*, pages 581–595. MIT Press.

[Dubois et al., 1994] Dubois, D., Lang, J., and Prade, H. (1994). Possibilistic logic. In Gabbay, D. M., Hogger, C. J., Robinson, J. A., and Siekmann, J. H., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming - Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, pages 439–513. Oxford University Press, Inc., New York, NY, USA.

[Dubois et al., 1999] Dubois, D., Le Berre, D., Prade, H., and Sabbadin, R. (1999). Using Possibilistic Logic for Modeling Qualitative Decision: ATMS-based Algorithms. *Fundamenta Informaticae*, 37(1-2):1–30.

[Dubois and Prade, 1988] Dubois, D. and Prade, H. (1988). *Possibility Theory*. Plenum Press, New York.

[Dubois and Prade, 1992] Dubois, D. and Prade, H. (1992). Possibility theory as a basis for preference propagation in automated reasoning. In *Proceedings of the 1st IEEE International Conference on Fuzzy Systems*, pages 821–832.

[Dubois and Prade, 1994] Dubois, D. and Prade, H. (1994). Conditional Objects as Nonmonotonic Consequence Relations. In Doyle, J., Sandewall, E., and Torasso, P., editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, (KR'94)*, pages 170–177. Morgan Kaufmann.

[Dubois and Prade, 1995] Dubois, D. and Prade, H. (1995). Possibility theory as a basis for qualitative decision theory. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, (IJCAI'95)*, pages 1924–1930. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Dubois and Prade, 1997] Dubois, D. and Prade, H. (1997). Using fuzzy sets in flexible querying: Why and how? In *Flexible query answering systems*, pages 45–60. Kluwer Academic Publishers, Norwell, MA, USA.

[Dubois and Prade, 2004] Dubois, D. and Prade, H. (2004). Possibilistic logic: a retrospective and prospective view. *Fuzzy Sets and Systems*, 144(1):3–23.

[Dubois et al., 1997] Dubois, D., Prade, H., and Sabbadin, R. (1997). A Possibilistic Logic Machinery for Qualitative Decision. In *Proceedings of AAAI Spring Symposium on Qualitative Preferences in Deliberation and Practical Reasoning*, pages 47–54. AAAI Press.

[Dubois et al., 2001] Dubois, D., Prade, H., and Sabbadin, R. (2001). Decision-theoretic foundations of qualitative possibility theory. *European Journal of Operational Research*, 128(3):459–478.

[Dung et al., 2008] Dung, P. M., Thang, P. M., and Toni, F. (2008). Towards argumentation-based contract negotiation. In Besnard, P., Doutre, S., , and Hunter, A., editors, *Proceedings of 2nd International Conference on Computational Models of Argument (COMMA'08)*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 134–146. IOS Press, Amsterdam, The Netherlands, The Netherlands.

[Dupin de Saint-Cyr and Prade, 2008] Dupin de Saint-Cyr, F. and Prade, H. (2008). Handling uncertainty and defeasibility in a possibilistic logic setting. *International Journal of Approximate Reasoning*, 49(1):67–82.

[Eiter et al., 1999] Eiter, T., Faber, W., Leone, N., and Pfeifer, G. (1999). The Diagnosis Frontend of the dlv system. *AI Communication*, 12(1-2):99–111.

[Eiter and Gottlob, 1995] Eiter, T. and Gottlob, G. (1995). On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3):289–323.

[Eiter et al., 1997] Eiter, T., Gottlob, G., and Leone, N. (1997). Abduction from logic program: semantics and complexity. *Theoretical Computer Science*, 189(1-2):129–177.

[Faber et al., 2011] Faber, W., Pfeifer, G., and Leone, N. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298.

[Fitting, 1991] Fitting, M. (1991). Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11(2):91–116.

[Fox et al., 2007] Fox, J., Glasspool, D., Grecu, D., Modgil, S., South, M., and Patkar, V. (2007). Argumentation-Based Inference and Decision Making–A Medical Perspective. *IEEE Intelligence Systems*, 22(6):34–41.

[García and Simari, 2004] García, A. J. and Simari, G. R. (2004). Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming*, 4(2):95–138.

[Garcia et al., 2000] Garcia, B. B., Lopes, J. G. P., and Varejão, F. (2000). Compiling Default Theory into Extended Logic Programming. In Monard, M. and Sichman, J., editors, *Proceedings of 7th Ibero-American International Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence, (IBERAMIA-SBIA '00)*, volume 1952 of *Lecture Notes in Computer Science*, pages 207–216. Springer Berlin / Heidelberg.

[Garcia et al., 2009] Garcia, L., Ngoma, S., and Nicolas, P. (2009). Dealing Automatically with Exceptions by Introducing Specificity in ASP. In Sossai, C. and Chemello, G., editors, *Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, (ECSQARU'99)*, volume 5590 of *Lecture Notes in Artificial Intelligence*, pages 614–625. Springer-Verlag, Berlin, Heidelberg.

[Gärdenfors and Makinson, 1994] Gärdenfors, P. and Makinson, D. (1994). Nonmonotonic inference based on expectations. *Artificial Intelligence*, 65(2):197–245.

[Gebser et al., 2011] Gebser, M., Kaminski, R., König, A., and Schaub, T. (2011). Advances in *gringo* series 3. In Delgrande, J. P. and Faber, W., editors, *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning, (LPNMR'11)*, volume

6645 of *Lecture Notes in Computer Science*, pages 345–351. Springer-Verlag, Berlin, Heidelberg.

[Gelfond, 1994] Gelfond, M. (1994). Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence*, 12(1):89–116.

[Gelfond and Lifschitz, 1988] Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming, (ICLP'88)*, pages 1070–1080. The MIT Press.

[Gelfond and Lifschitz, 1991] Gelfond, M. and Lifschitz, V. (1991). Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3/4):365–386.

[Gelfond and Son, 1998] Gelfond, M. and Son, T. C. (1998). Reasoning with Prioritized Defaults. In Dix, J., Pereira, L., and Przymusinski, T., editors, *Selected papers from the Third International Workshop on Logic Programming and Knowledge Representation, (LPKR '97)*, volume 1417 of *Lecture Notes in Computer Science*, pages 164–223. Springer-Verlag, London, UK.

[Godfrey and Ning, 2004] Godfrey, P. and Ning, W. (2004). Relational Preference Queries via Stable Skyline. Technical Report CS-2004-03, York University.

[Goldzsmidt et al., 1993] Goldzsmidt, M., Morris, P., and Pearl, J. (1993). A Maximum Entropy Approach to Nonmonotonic Reasoning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):220–232.

[Gómez-Sebastià et al., 2009] Gómez-Sebastià, I., Palau, M., Nieves, J. C., Vázquez-Salceda, J., and Ceccaroni, L. (2009). Dynamic Orchestration of Distributed Services on Interactive Community Displays: The ALIVE Approach. In Demazeau, Y., PavÃşn, J., Corchado, J., and Bajo, J., editors, *Proceedings of the 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, volume 55 of *Advances in Soft Computing*, pages 450–459. Springer Berlin / Heidelberg.

[Grabos, 2004] Grabos, R. (2004). Qualitative Model of Decision Making. In Bussler, C. and Fensel, D., editors, *Artificial Intelligence: Methodology, Systems, and Applications*, volume 3192 of *Lecture Notes in Artificial Intelligence*, pages 480–489. Springer Berlin / Heidelberg.

[Greenfield, 2006] Greenfield, A. (2006). *Everyware: The Dawning Age of Ubiquitous Computing*. New Riders, Berkeley, CA, USA.

[Grégoire and Konieczny, 2006] Grégoire, E. and Konieczny, S. (2006). Logic-based approaches to information fusion. *Information Fusion*, 7(1):4–18.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition - Special issue: Current issues in knowledge modeling*, 5(2):199–220.

[Hadjali et al., 2008] Hadjali, A., Kaci, S., and Prade, H. (2008). Database Preferences Queries - A Possibilistic Logic Approach with Symbolic Priorities. In Hartmann, S. and Kern-Isberner, G., editors, *Proceedings of 5th International Symposium on Foundations of Information and Knowledge Systems, (FoIKS 2008)*, volume 4932 of *Lecture Notes in Computer Science*, pages 291–310. Springer Berlin / Heidelberg.

[Halldén, 1957] Halldén, S. (1957). *On the Logic of 'Better'*. Lund, Sweden, Gleerup, no. 2 library of theoria edition.

[Hansson, 2001] Hansson, S. O. (2001). Preference Logic. In Gabbay, D. M. and Guenthner, F., editors, *Handbook of Philosophical Logic, 2nd Edition, Volume 4*, pages 319–394. Kluwer Academic Publishers, Dortrecht, Holland.

[Horty, 1994] Horty, J. F. (1994). Some direct theories of nonmonotonic inheritance. pages 111–187. Oxford University Press, Inc., New York, NY, USA.

[Horty, 2002] Horty, J. F. (2002). Skepticism and floating conclusions. *Artificial Intelligence*, 135(1-2):55–72.

[Hué et al., 2009] Hué, J., Papini, O., and Würbel, E. (2009). Merging Belief Bases Represented by Logic Programs. In Sossai, C. and Chemello, G., editors, *Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2009)*, volume 5590 of *Lecture Notes in Artificial Intelligence*, pages 371–382. Springer-Verlag, Berlin, Heidelberg.

[Inoue and Sakama, 1998] Inoue, K. and Sakama, C. (1998). Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78.

[Janhunen, 2001] Janhunen, T. (2001). On the Effect of Default Negation on the Expressiveness of Disjunctive Rules. In Eiter, T., Faber, W., and Truszczynski, M., editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning, (LPNMR'01)*, volume 2173 of *Lecture Notes In Computer Science*, pages 93–106. Springer-Verlag, London, UK.

[Kaci, 2011] Kaci, S. (2011). *Working with Preferences: Less Is More*. Springer-Verlag.

[Kaci et al., 2006] Kaci, S., Dubois, D., and Prade, H. (2006). Representing preferences in the possibilistic setting. In Bosi, G., Brafman, R. I., Chomicki, J., and Kießing, W., editors, *Preferences: Specification, Inference, Applications*, number 04271 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl, Dagstuhl, Germany.

[Kaci and Prade, 2007] Kaci, S. and Prade, H. (2007). Relaxing Ceteris Paribus Preferences with Partially Ordered Priorities. In Mellouli, K., editor, *Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, (ECSQARU'07)*, volume 4724 of *Lecture Notes in Artificial Intelligence*, pages 660–671. Springer-Verlag, Berlin, Heidelberg.

[Kaci and Prade, 2008] Kaci, S. and Prade, H. (2008). Mastering the Processing of Preferences by Using Symbolic Priorities in Possibilistic Logic. In *Proceeding of the 2008 conference on ECAI 2008*, pages 376–380. IOS Press, Amsterdam, The Netherlands, The Netherlands.

[Kakas et al., 1992] Kakas, A. C., Kowalski, R. A., and Toni, F. (1992). Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770.

[Kärger et al., 2008] Kärger, P., Lopes, N., Olmedilla, D., and Polleres, A. (2008). Towards Logic Programs with Ordered and Unordered Disjunction. In *Proceedings of Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP2008), 24th International Conference on Logic Programming (ICLP 2008)*, pages 46–60.

[Kern-Isberner and Lukasiewicz, 2004] Kern-Isberner, G. and Lukasiewicz, T. (2004). Combining probabilistic logic programming with the power of maximum entropy. *Artificial Intelligence*, 157(1-2):139–202.

[Kießling, 2002] Kießling, W. (2002). Foundations of preferences in database systems. In *Proceedings of the 28th international conference on Very Large Data Bases, (VLDB'02)*, pages 311–

322. Morgan Kaufmann.

[Kifer and Subrahmanian, 1992] Kifer, M. and Subrahmanian, V. S. (1992). Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12(4):335–367.

[Konieczny and Pérez, 1998] Konieczny, S. and Pérez, R. P. (1998). On the Logic of Merging. In Cohn, A. G., Schubert, L. K., and Shapiro, S. C., editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 488–498. Morgan Kaufmann.

[Konieczny and Pérez, 2011] Konieczny, S. and Pérez, R. P. (2011). Logic Based Merging. *Journal of Philosophical Logic*, 40(2):239–270.

[Konolige, 1989] Konolige, K. (1989). Hierarchic Autoepistemic Theories for Non-monotonic Reasoning: Preliminary Report. In Reinfrank, M., de Kleer, J., Ginsberg, M., and Sandewall, E., editors, *Proceedings of the 2nd International Workshop on Non-monotonic Reasoning*, volume 346 of *Lecture Notes in Computer Science*, pages 42–59. Springer-Verlag New York, Inc., New York, NY, USA.

[Kowalski and Sadri, 1991] Kowalski, R. and Sadri, F. (1991). Logic Programs with Exceptions. *New Generation Computing*, 9(3):387–400.

[Kraus et al., 1990] Kraus, S., Lehmann, D., and Magidor, M. (1990). Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44(1-2):167–207.

[Krumpelmann and Kern-Isberner, 2010] Krumpelmann, P. and Kern-Isberner, G. (2010). On belief dynamics of dependancy relations for extended logic programs. In *Proceedings of the 13th International Workshop on Non-monotonic Reasoning*.

[Lacroix and Lavency, 1987] Lacroix, M. and Lavency, P. (1987). Preferences; Putting More Knowledge into Queries. In Stocker, P. M., Kent, W., and Hammersley, P., editors, *Proceedings of the 13th International Conference on Very Large Data Bases, (VLDB '87)*, pages 217–225. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Lam et al., 2009] Lam, J. S. C., Vasconcelos, W. W., Guerin, F., Corsar, D., Chorley, A., Norman, T. J., Vázquez-Salceda, J., Panagiotidi, S., Confalonieri, R., Gomez, I., Hidalgo, S., Álvarez-Napagao, S., Nieves, J. C., Roig, M. P., Ceccaroni, L., Aldewereld, H., Dignum, V., Dignum, F., Penserini, L., Padget, J. A., Vos, M. D., Andreou, D., Cliffe, O., Staikopoulos, A., Popescu, R., Clarke, S., Sergeant, P., Reed, C., Quillinan, T., and Nieuwenhuis, K. (2009). ALIVE: A Framework for Flexible and Adaptive Service Coordination. In Aldewereld, H., Dignum, V., and Picard, G., editors, *Proceedings of the 10th International Workshop on Engineering Societies in the Agents World, (ESAW 2009)*, volume 5881 of *Lecture Notes in Computer Science*, pages 236–239. Springer.

[Lehmann and Magidor, 1992] Lehmann, D. and Magidor, M. (1992). What does a conditional knowledge base entail? *Artificial Intelligence*, 55(1):1–60.

[Leone et al., 2006] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006). The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562.

[Lichtenstein and Slovic, 2006] Lichtenstein, S. and Slovic, P. (2006). *The Construction of Preference*. Cambridge University Press, New York, NY, USA.

[Lifschitz, 1985] Lifschitz, V. (1985). Computing circumscription. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 2, (IJCAI'85)*, pages 121–127. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Lifschitz, 1996] Lifschitz, V. (1996). Foundations of logic programming. pages 69–127. Center for the Study of Language and Information, Stanford, CA, USA.

[Lifschitz et al., 2001] Lifschitz, V., Pearce, D., and Valverde, A. (2001). Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541.

[Lifschitz et al., 1999] Lifschitz, V., Tang, L. R., and Turner, H. (1999). Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389.

[Lifschitz and Woo, 1992] Lifschitz, V. and Woo, T. Y. C. (1992). Answer sets in general nonmonotonic reasoning (preliminary report). In Nebel, B., Rich, C., and Swartout, W. R., editors, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning, (KR'92)*, pages 603–614.

[Liu, 2010] Liu, F. (2010). Von Wright's *"The Logic of Preference"* revisited. *Synthese*, 175(1):69–88.

[Liu et al., 2007] Liu, P., Nie, G., and Chen, D. (2007). Exploiting Semantic Descriptions of Products and User Profiles for Recommender Systems. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, (CIDM 2007)*, pages 179–185. IEEE.

[Lloyd, 1987] Lloyd, J. W. (1987). *Foundations of logic programming*. Springer-Verlag New York, Inc., New York, NY, USA, 2nd extended edition edition.

[Lukasiewicz, 1998] Lukasiewicz, T. (1998). Probabilistic Logic Programming. In Prade, H., editor, *Proceedings of 13th European Conference on Artificial Intelligence, (ECAI 1998)*, pages 388–392. John Wiley and Sons, Chichester.

[Lukasiewicz, 2005] Lukasiewicz, T. (2005). Weak nonmonotonic probabilistic logics. *Artificial Intelligence*, 168(1-2):119–161.

[Matt, 2010] Matt, P.-A. (2010). *Argumentation as a practical foundation for decision theory*. PhD thesis, Department of Computing, Imperial College London.

[McCarthy, 1980] McCarthy, J. (1980). Circumscription - A form of non-monotonic reasoning. *Artificial Intelligence*, 13(1-2):27–39.

[McCarthy, 1986] McCarthy, J. (1986). Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28(1):89–116.

[McCarthy and Hayes, 1969] McCarthy, J. and Hayes, P. J. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B. and Michie, D., editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press.

[McDermott and Doyle, 1980] McDermott, D. V. and Doyle, J. (1980). Nonmonotonic logic I. *Artificial Intelligence*, 13(1-2):41–72.

[Mellor et al., 2004] Mellor, S., Scott, K., Uhl, A., and Weise, D. (2004). *MDA Distilled: Principles of Model-Driven Architecture*. Addison Wesley.

[Metzger, 2005] Metzger, A. (2005). *Model-Driven Software Development*, chapter A Systematic Look at Model Transformations, pages 19–33. Computer Science. Springer Berlin Heidelberg.

[Moore, 1985] Moore, R. C. (1985). Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94.

[Newman, 1942] Newman, M. H. A. (1942). On Theories with a Combinatorial Definition of Equivalence. *The Annals of Mathematics*, 43(2):223–243.

[Ng and Subrahmanian, 1992] Ng, R. T. and Subrahmanian, V. S. (1992). Probabilistic Logic Programming. *Information and Computation*, 101(2):150–201.

[Nicolas et al., 2003] Nicolas, J. B., Farcet, N., Jézéquel, J. M., Langlois, B., and Pollet, D. (2003). Reflective Model Driven Engineering. In Stevens, P., Whittle, J., and Booch, G., editors, *UML 2003 - The Unified Modeling Language. Modeling Languages and Applications*, volume 2863 of *Lecture Note in Computer Science*, pages 175–189. Springer Berlin / Heidelber.

[Nicolas et al., 2006] Nicolas, P., Garcia, L., Stéphan, I., and Lefèvre, C. (2006). Possibilistic uncertainty handling for answer set programming. *Annals of Mathematics and Artificial Intelligence*, 47(1-2):139–181.

[Niemelä and Simons, 1997] Niemelä, I. and Simons, P. (1997). Smodels - An Implementation of the Stable Model and Well-Founded Semantics for Normal LP. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceeding of 4th International Conference on Logic Programming and Nonmonotonic Reasoning, (LPNMR'97)*, volume 1265 of *Lecture Notes in Computer Science*, pages 421–430. Springer-Verlag, London, UK.

[Nieuwenborgh et al., 2007] Nieuwenborgh, D., Cock, M., and Vermeir, D. (2007). An introduction to fuzzy answer set programming. *Annals of Mathematics and Artificial Intelligence*, 50(3-4):363–388.

[Nieves et al., 2011] Nieves, J. C., Osorio, M., and Cortés, U. (2011). Semantics for Possibilistic Disjunctive Programs. *Theory and Practice of Logic Programming*, abs/1106.0776.

[Nilsson, 1994] Nilsson, N. J. (1994). Probabilistic logic revisited. pages 39–42. MIT Press, Cambridge, MA, USA.

[Osorio et al., 2001a] Osorio, M., Navarro, J. A., and Arrazola, J. (2001). Equivalence in Answer Set Programming. In Pettorossi, A., editor, *Selected Papers from the 11th International Workshop on Logic Based Program Synthesis and Transformation, (LOPSTR 2001)*, volume 2372 of *Lecture Notes in Computer Science*, pages 57–75. Springer-Verlag, London, UK.

[Osorio and Nieves, 2007] Osorio, M. and Nieves, J. C. (2007). Pstable semantics for possibilistic logic programs. In Gelbukh, A. and Morales, A. F. K., editors, *Proceedings of the artificial intelligence 6th Mexican international conference on Advances in artificial intelligence*, MICAI'07, pages 294–304. Springer-Verlag, Berlin, Heidelberg.

[Osorio et al., 2001b] Osorio, M., Nieves, J. C., and Giannella, C. (2001). Useful Transformations in Answer Set Programming. In Provetti, A. and Son, T. C., editors, *Proceedings of the 1st International Answer Set Programming Workshop, Towards Efficient and Scalable Knowledge Representation and Reasoning*, pages 146–152. AAAI Press, Stanford, E.U.

[Osorio et al., 2004] Osorio, M., Ortiz, M., and Zepeda, C. (2004). Using CR-Rules for evacuation planning. In Ita-Luna, G. D., Fuentes-Chávez, O., and Osorio-Galindo, M., editors, *Proceedings of the IX Ibero-american Workshops on Artificial Inteligence*, pages 56–63. Puebla, Mexico.

[Osorio et al., 2006] Osorio, M., Pérez, J. A. N., Ramírez, J. R. A., and Macías, V. B. (2006). Logics with Common Weak Completions. *Journal of Logic and Computation*, 16(6):867–890.

[Osorio and Zepeda, 2006] Osorio, M. and Zepeda, C. (2006). Minimal Extended Generalized Answer Sets and their Applications. In Dávila, R., Osorio, M., and Zepeda, C., editors,

Proceedings of the Workshop in Logic, Language and Computation (LoLaCOM06), Fifth Mexican Internantional Conference on Artificial Intelligence, volume 220 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Osorio et al., 2011] Osorio, M., Zepeda, C., and Carballido, J. L. (2011). Extended ordered disjunction programs to model preferences. *Special issue of Journal Research on Computing Science: Advances in Computer Science and Applications*, 53:211–220.

[Palau, 2010] Palau, M. (2010). Combining Coordination and Organisation Mechanisms for the Development of a Dynamic Context-aware Information System Personalised by means of Logic-based Preference Methods. Master's thesis, Universitat Politècnica de Catalunya, Barcelona, Spain.

[Palau et al., 2010] Palau, M., Ceccaroni, L., Gómez-Sebastià, I., Vázquez-Salceda, J., and Nieves, J. C. (2010). Coordination and Organisational Mechanisms Applied to the Development of a Dynamic, Context-aware Information Service. In Filipe, J., Fred, A. L. N., and Sharp, B., editors, *Proceedings of the International Conference on Agents and Artificial Intelligence, Volume 2, (ICAART 2010)*, pages 88–95. INSTICC Press.

[Park et al., 2009] Park, K.-L., Yoon, U. H., and Kim, S.-D. (2009). Personalized Service Discovery in Ubiquitous Computing Environments. *IEEE Pervasive Computing*, 8(1):58–65.

[Pearce et al., 2002] Pearce, D., Sarsakov, V., Schaub, T., Tompits, H., and Woltran, S. (2002). A Polynomial Translation of Logic Programs with Nested Expressions into Disjunctive Logic Programs: Preliminary Report. In Stuckey, P., editor, *Proceedings of the 18th International Conference on Logic Programming, (ICLP'02)*, volume 2401 of *Lecture Notes in Computer Science*, pages 386–404. Springer-Verlag, London, UK.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Pearl, 1990] Pearl, J. (1990). System z: a natural ordering of defaults with tractable applications to nonmonotonic reasoning. In *Proceedings of the 3rd conference on Theoretical Aspects of Reasoning about Knowledge, (TARK '90)*, pages 121–135. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Prade, 2009] Prade, H. (2009). Current Research Trends in Possibilistic Logic- Multiple Agent Reasoning, Preference Representation, and Uncertain Databases. In Ras, Z. and Dardzinska, A., editors, *Advances in Data Management*, volume 223 of *Studies in Computational Intelligence*, pages 311–330. Springer Berlin / Heidelberg.

[Rahwan and Simari, 2009] Rahwan, I. and Simari, G. R. (2009). *Argumentation in Artificial Intelligence*. Springer.

[Reichenbach, 1949] Reichenbach, H. (1949). *The Theory of Probability*. University of California Press, Berkeley.

[Reiter, 1980] Reiter, R. (1980). A Logic for Default Reasoning. *Artificial Intelligence*, 13(1-2):81–137.

[Reiter and Criscuolo, 1981] Reiter, R. and Criscuolo, G. (1981). On interacting defaults. In Hayes, P. J., editor, *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 1, (IJCAI'81)*, pages 270–276. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Rintanen, 1994] Rintanen, J. (1994). Prioritized Autoepistemic Logic. In MacNish, C., Pearce, D., and Pereira, L. M., editors, *Proceedings of the European Workshop on Logics in Artificial Intelligence, (JELIA'94)*, volume 838 of *Lecture Notes In Artificial Intelligence*, pages 232–246. Springer-Verlag, London, UK.

[Rintanen, 1995] Rintanen, J. (1995). On specificity in default logic. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, (IJCAI'95)*, pages 1474–1479. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Rintanen, 1998] Rintanen, J. (1998). Lexicographic priorities in default logic. *Artificial Intelligence*, 106(2):221–265.

[Saad and Pontelli, 2005] Saad, E. and Pontelli, E. (2005). Hybrid Probabilistic Logic Programs with Non-monotonic Negation. In Gabbrielli, M. and Gupta, G., editors, *Proceedings of 21st International Conference on Logic Programming, (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 204–220. Springer Berlin / Heidelberg, New York, US.

[Sabbadin, 1998] Sabbadin, R. (1998). Decision As Abduction? In Prade, H., editor, *Proceedings of 13th European Conference on Artificial Intelligence, (ECAI 1998)*, pages 600–604. John Wiley and Sons, Chichester.

[Sakama and Inoue, 2000] Sakama, C. and Inoue, K. (2000). Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123(1-2):185–222.

[Savage, 1972] Savage, L. J. (1972). *The Foundations of Statistics*. Dover, New York.

[Schaub and Wang, 2001] Schaub, T. and Wang, K. (2001). A comparative study of logic programs with preference. In Nebel, B., editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1, (IJCAI'01)*, pages 597–602. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Shafer, 1976] Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

[Simons et al., 2002] Simons, P., Niemelá, I., and Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234.

[Sombé, 1990] Sombé, L. (1990). *Reasoning under incomplete information in artificial intelligence: a comparison of formalisms using a single example*. John Wiley & Sons, Inc., New York, NY, USA.

[Subrahmanian, 2007] Subrahmanian, V. S. (2007). Uncertainty in logic programming. *Association for Logic Programming (ALP), Newsletter*, 20(2).

[Tan and Pearl, 1994a] Tan, S.-W. and Pearl, J. (1994). Qualitative decision theory. In *Proceedings of the 12th National Conference on Artificial Intelligence (vol. 2), (AAAI'94)*, pages 928–933. American Association for Artificial Intelligence, Menlo Park, CA, USA.

[Tan and Pearl, 1994b] Tan, S.-W. and Pearl, J. (1994). Specification and Evaluation of Preferences Under Uncertainty. In Doyle, J., Sandewall, E., and Torasso, P., editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, (KR'94)*, pages 530–539. Morgan Kaufmann.

[Toni and Sergot, 2011] Toni, F. and Sergot, M. (2011). Argumentation and Answer Set Programming. In Balduccini, M. and Son, T., editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *Lecture Notes in Computer Science*, pages 164–180. Springer Berlin / Heidelberg.

[Touretzky, 1986] Touretzky, D. S. (1986). *The mathematics of inheritance systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Touretzky et al., 1987] Touretzky, D. S., Horty, J. F., and Thomason, R. H. (1987). A clash of intuitions: the current state of nonmonotonic multiple inheritance systems. In McDermott, J. P., editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 1, (IJCAI'87)*, pages 476–482. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Tselentis et al., 2009] Tselentis, G., Domingue, J., Galis, A., Gavras, A., and Hausheer, D. (2009). *Towards the Future Internet: A European Research Perspective*. IOS Press, Amsterdam, The Netherlands, The Netherlands.

[van Benthem et al., 2009] van Benthem, J., Girard, P., and Roy, O. (2009). Everything else being equal: A modal logic approach to ceteris paribus preferences. *Journal of Philosophical Logic*, 38:83–125.

[van Dalen, 1994] van Dalen, D. (1994). *Logic and Structure*. Springer-Verlag, Berlin, 3rd augmented edition edition.

[Van Gelder et al., 1991] Van Gelder, A., Ross, K. A., and Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):619–649.

[Van Nieuwenborgh and Vermeir, 2006] Van Nieuwenborgh, D. and Vermeir, D. (2006). Preferred answer sets for ordered logic programs. *Theory and Practice of Logic Programming*, 6(1-2):107–167.

[Vázquez-Salceda et al., 2010] Vázquez-Salceda, J., Ceccaroni, L., Dignum, F., Vasconcelos, W., Padget, J., Clarke, S., Sergeant, P., and Nieuwenhuis, K. (2010). Combining Organisational and Coordination Theory with Model Driven Approaches to Develop Dynamic, Flexible, Distributed Business Systems. In Akan, O., Bellavista, P., Cao, J., Dressler, F., Ferrari, D., Gerla, M., Kobayashi, H., Palazzo, S., Sahni, S., Shen, X. S., Stan, M., Xiaohua, J., Zomaya, A., Coulson, G., Telesca, L., Stanoevska-Slabeva, K., and Rakocevic, V., editors, *Digital Business*, volume 21 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 175–184. Springer Berlin Heidelberg.

[von Neumann and Morgenstern, 1944] von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ.

[von Wright, 1963] von Wright, G. H. (1963). *The Logic of Preference: An Essay*. Edinburgh University Press, Scotland, UK.

[Walley, 1991] Walley, P. (1991). *Statistical Reasoning with Imprecise Probabilities*. (Chapman and Hall, London.

[Wan, 2009] Wan, H. (2009). Belief Logic Programming. In Hill, P. and Warren, D., editors, *Logic Programming*, volume 5649 of *Lecture Notes in Computer Science*, pages 547–548. Springer Berlin / Heidelberg.

[Wang and Kong, 2007] Wang, R.-Q. and Kong, F.-S. (2007). Semantic-Enhanced Personalized Recommender System. In *Proceedings of the 6th International Conference on Machine Learning and Cybernetics*, volume 7, pages 4069–4074.

[Wilson, 2004] Wilson, N. (2004). Extending CP-nets with stronger conditional preference statements. In Cohn, A. G., editor, *Proceedings of the 19th National Conference on Artifi-*

cial Intelligence, AAAI'04, pages 735–741. AAAI Press.

[You et al., 1999] You, J.-H., Wang, X., and Yuan, L. Y. (1999). Compiling defeasible inheritance networks to general logic programs. *Artificial Intelligence*, 113(1-2):247–268.

[Zadeh, 1975] Zadeh, L. A. (1975). Fuzzy logic and approximate reasoning. *Synthese*, 30(3):407–428.

[Zadeh, 1978] Zadeh, L. A. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3–28.

[Zepeda, 2005] Zepeda, C. (2005). *Evacuation Planning using Answer Set Programming*. PhD thesis, Universidad De Las Américas - Puebla and Institut National des Sciences Appliquées de Lyon.

[Zhang and Foo, 1997] Zhang, Y. and Foo, N. Y. (1997). Answer sets for prioritized logic programs. In Maluszynski, J., editor, *Proceedings of the 1997 International Symposium on Logic Programming, (ILPS '97)*, pages 69–83. MIT Press, Cambridge, MA, USA.

Index

answer set programming, 17
  answer set semantics, 19
  definite logic programs, 18
  disjunctive logic programs, 21
  extended normal logic programs, 18
  fix-point, 20
  generalized disjunctive logic programs, 22
  LPODs, 23
  nested logic programs, 126

classical decision theory, 139

default rules
  autoepistemic logic, 36
  circumscription, 35
  default logic, 34
  exceptions rewriting, 51
  logic programming, 39
  possibilistic logic, 37

LPODs$^+$, 124
  disjunctive logic programs mapping, 129
  semantics, 126
  syntax, 125
LPPODs, 65
  computation, 79
  fix-point possibilistic semantics, 67
  possibilistic answer set selection, 76
  possibility distribution semantics, 71
  solver, 112
  syntax, 66
  transformation rules, 73

nonmonotonic logics, 33
  autoepistemic logic, 36
  circumscription, 35
  default logic, 34
nonmonotonic reasoning, 33
  autoepistemic logic, 36
  blocking inheritance, 38, 40, 56, 59
  circumscription, 35
  default logic, 34
  explicit preferences, 41
  floating conclusions, 40, 56
  implicit preferences, 40
  irrelevance, 37
  logic programming, 39
  LPODs, 42
  negation as failure, 39
  possibilistic logic, 37
  System P, 37
nonmonotonic reasoning and uncertainty
  fuzzy answer set programming, 43
  possibilistic answer set programming, 28, 44
  probabilistic answer set programming, 44

ontology, 106
  LPPODs, 109
  user preferences, 108
  user profile, 107

personalization, 98
  context-aware system, 99
    Interactive Community Displays, 103
possibilistic answer set programming, 28
possibilistic logic, 26
  possibilistic entailment, 28
  possibilistic modus ponens, 27

qualitative decision making, 140
  ASP, 148
  decision criteria, 141, 143

logic programming, 144
LPODs, 154
LPPODs, 158
possibilistic ASP, 153
possibilistic semantics, 141
propositional bases, 142
stratified propositional bases, 143

rewriting systems
transformation rules, 25

specificity, 40, 50
Z-ordering, 37, 50

user preference handling, 89
CP-nets, 92
database preference queries, 118
logic programming, 94
ASO programs, 96, 135
DLPODs, 95, 134
LPODs, 94
LPODs$^+$, 124
LPPODs, 84, 85, 106, 109, 112
possibilistic database preference queries, 119
LPPODs, 119
possibilistic logic, 91
QCL, 89