Wissensverarbeitung und Informationssysteme
Institut für Informatik
Universität Potsdam

Preferences in Answer Set Programming

DISSERTATION

zur Erlangung des akademischen Grades
“Doctor rerum naturalium”
(Dr. rer. nat.)
in der Wissenschaftsdisziplin Informatik

unter der Leitung von
Prof. Dr. Torsten Schaub
Wissensverarbeitung und Informationssysteme
Institut für Informatik

eingereicht an der

Mathematisch-Naturwissenschaftlichen Fakultät
der
Universität Potsdam

von

Kathrin Konczak

Für Papa

Acknowledgments

First of all, I would like to thank my supervisor Prof. Dr. Torsten Schaub for his invaluable support. I would also like to thank my co-authors Wolfgang Faber, Jerome Lang, Thomas Linke, and Ralf Vogel for the successful collaboration, and my colleagues at the University of Potsdam, especially Christian Anger and Martin Gebser for their helpful comments on my papers, as well as Jürgen Brandt-Mihram for the technical support. Furthermore, I would like to thank Philippe Besnard and Kewen Wang for their helpful assistance, as well as André Neumann and Susanne Grell for their assistance with the implementation of our software and for providing experimental results.

My work has been supported by the German Science Foundation (Deutsche Forschungsgemeinschaft) under grant SCHA 550/6-4, TP C, and by the European Commission, FET (”Future Emerging Technologies”) initiative, under project IST-2001-37004 WASP.

Last but not least, I thank my family and friends for their encouragement and considerateness.

Kurzfassung

Die Antwortmengenprogrammierung entwickelte sich in den späten 90er Jahren als neues Paradigma der logischen Programmierung und ist in den Gebieten des nicht-monotonen Schließens und der deduktiven Datenbanken verwurzelt. Dabei wird eine Problemstellung als logisches Programm repräsentiert, dessen Lösungen, die so genannten Antwortmengen, genau den Lösungen des ursprünglichen Problems entsprechen. Die Antwortmengenprogrammierung bildet ein geeignetes Fundament zur Repräsentation und zum Lösen von Entscheidungs- und Suchproblemen in der Komplexitätsklasse NP. Anwendungen finden wir unter anderem in der Produktkonfiguration, Diagnose und bei graphen-theoretischen Problemen, z.B. der Suche nach Hamiltonschen Kreisen.

In den letzten Jahren wurden viele Erweiterungen der Antwortmengenprogrammierung betrachtet. Die am meisten untersuchte Erweiterung ist die Modellierung von Präferenzen. Diese bilden eine natürliche und effektive Möglichkeit, unter einer Vielzahl von Lösungen eines Problems bevorzugte Lösungen zu selektieren. Präferenzen finden beispielsweise in der Stundenplanung, bei Auktionen und bei Produktkonfigurationen ihre Anwendung.

Der Schwerpunkt dieser Arbeit liegt in der Modellierung, Implementierung und Anwendung von Präferenzen in der Antwortmengenprogrammierung. Da es verschiedene Ansätze gibt, um Präferenzen darzustellen, konzentrieren wir uns auf geordnete logische Programme, wobei Präferenzen als partielle Ordnung der Regeln eines logischen Programms ausgedrückt werden. Dabei betrachten wir drei verschiedene Semantiken zur Interpretation dieser Präferenzen. Im Vorfeld wurden für diese Semantiken die bevorzugten Antwortmengen durch einen Compiler oder durch Meta-Interpretation berechnet. Da Präferenzen Lösungen selektieren, stellt sich die Frage, ob es möglich ist, diese direkt in den Berechnungsprozeß von präferenzierten Antwortmengen zu integrieren, so dass die bevorzugten Antwortmengen ohne Zwischenschritte berechnet werden können. Dazu entwickeln wir zuerst ein auf Graphen basierendes Gerüst zur Berechnung von Antwortmengen. Anschließend werden wir darin Präferenzen integrieren, so dass bevorzugte Antwortmengen ohne Compiler oder Meta-Interpretation berechnet werden. Es stellt sich heraus, dass die integrative Methode auf den meisten betrachteten Problemklassen wesentlich leistungsfähiger ist als der Compiler oder Meta-Interpretation.

Ein weiterer Schwerpunkt dieser Arbeit liegt in der Frage, inwieweit sich geordnete logische Programme vereinfachen lassen. Dazu steht die Methodik der strengen Äquivalenz von logischen Programmen zur Verfügung. Wenn ein logisches Programm streng äquivalent zu einem seiner Teilprogramme ist, so kann man dieses durch das entsprechende Teilprogramm ersetzen, ohne dass sich die zugrunde liegende Semantik ändert. Bisher wurden strenge Äquivalenzen nicht für logische Programme mit Präferenzen untersucht. In dieser Arbeit definieren wir erstmalig strenge Äquivalenzen für geordnete logische Programme. Wir geben notwendige und hinreichende Bedingungen für die strenge Äquivalenz zweier geordneter logischer Programme an. Des Weiteren werden wir auch die Frage beantworten, inwieweit geordnete logische Programme und deren Präferenzstrukturen vereinfacht werden können.

Abschließend präsentieren wir zwei neue Anwendungsbereiche von Präferenzen in der Antwortmengenprogrammierung. Zuerst definieren wir neue Prozeduren zur Entscheidungsfindung innerhalb von Gruppenprozessen. Diese integrieren wir anschließend in das Problem der Planung eines Treffens für eine Gruppe. Als zweite neue Anwendung rekonstruieren wir mit Hilfe der Antwortmengenprogrammierung eine linguistische Problemstellung, die in deutschen Dialekten auftritt. Momentan wird im Bereich der Linguistik darüber diskutiert, ob Regelsysteme von (menschlichen) Sprachen einzigartig sind oder nicht. Die Rekonstruktion von grammatikalischen Regularitäten mit Werkzeugen aus der Informatik erlaubt die Unterstützung der These, dass linguistische Regelsysteme Gemeinsamkeiten zu anderen nicht-linguistischen Regelsystemen besitzen.

Abstract

Answer Set Programming (ASP) emerged in the late 1990s as a new logic programming paradigm, having its roots in nonmonotonic reasoning, deductive databases, and logic programming with negation as failure. The basic idea of ASP is to represent a computational problem as a logic program whose answer sets correspond to solutions, and then to use an answer set solver for finding answer sets of the program. ASP is particularly suited for solving NP-complete search problems. Among these, we find applications to product configuration, diagnosis, and graph-theoretical problems, e.g. finding Hamiltonian cycles.

On different lines of ASP research, many extensions of the basic formalism have been proposed. The most intensively studied one is the modelling of preferences in ASP. They constitute a natural and effective way of selecting preferred solutions among a plethora of solutions for a problem. For example, preferences have been successfully used for timetabling, auctioning, and product configuration.

In this thesis, we concentrate on preferences within answer set programming. Among several formalisms and semantics for preference handling in ASP, we concentrate on ordered logic programs with the underlying D-, W-, and B-semantics. In this setting, preferences are defined among rules of a logic program. They select preferred answer sets among (standard) answer sets of the underlying logic program. Up to now, those preferred answer sets have been computed either via a compilation method or by meta-interpretation. Hence, the question comes up, *whether* and *how* preferences can be integrated into an existing ASP solver. To solve this question, we develop an operational graph-based framework for the computation of answer sets of logic programs. Then, we integrate preferences into this operational approach. We empirically observe that our integrative approach performs in most cases better than the compilation method or meta-interpretation.

Another research issue in ASP are optimization methods that remove redundancies, as also found in database query optimizers. For these purposes, the rather recently suggested notion of strong equivalence for ASP can be used. If a program is strongly equivalent to a subprogram of itself, then one can always use the subprogram instead of the original program, a technique which serves as an effective optimization method. Up to now, strong equivalence has not been considered for logic programs with preferences. In this thesis, we tackle this issue and generalize the notion of strong equivalence to ordered logic programs. We give necessary and sufficient conditions for the strong equivalence of two ordered logic programs. Furthermore, we provide program transformations for ordered logic programs and show in how far preferences can be simplified.

Finally, we present two new applications for preferences within answer set programming. First, we define new procedures for group decision making, which we apply to the problem of scheduling a group meeting. As a second new application, we reconstruct a linguistic problem appearing in German dialects within ASP. Regarding linguistic studies, there is an ongoing debate about how unique the rule systems of language are in human cognition. The reconstruction of grammatical regularities with tools from computer science has consequences for this debate: if grammars can be modelled this way, then they share core properties with other non-linguistic rule systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Answer Set Programming (ASP) emerged in the late 1990s as a new logic programming paradigm [100, 147, 149, 132], having its roots in nonmonotonic reasoning, deductive databases, and logic programming with negation as failure. Since its beginning, it has been regarded as the computational embodiment of nonmonotonic reasoning and a primary candidate for an effective knowledge representation tool. This view has been increased by the emergence of highly efficient solvers for ASP [181, 73, 152]. It now seems hard to dispute that ASP brought new life to logic programming and nonmonotonic reasoning research and has become a major driving force for these two fields.

The basic idea of ASP is to represent a computational problem as a logic program whose answer sets correspond to solutions, and then use an answer set solver for finding answer sets of the program. This approach is closely related to the one pursued in propositional satisfiability checking (SAT), where problems are encoded as propositional theories whose models represent solutions. Even though, syntactically, ASP programs look like Prolog programs, they are treated by rather different computational mechanisms. Indeed, the usage of model generation instead of query evaluation can be seen as a recent trend in the encompassing field of knowledge representation and reasoning. From a more formal point of view, ASP is particularly suited for solving combinatorial search problems lying in NP or NP^{NP}. Among these, we find applications to plan generation, product configuration, diagnosis, and graph-theoretical problems.

On different lines of ASP research, many extensions of the basic formalism have been proposed. Perhaps the most intensively studied one is the modeling of preferences in ASP, cf. [65]. Strongly rooted in the research of nonmonotonic formalisms, the ability to specify preferences is acknowledged to be particularly beneficial to ASP, since they constitute a natural and effective way of resolving indeterminate solutions. For example, preferences have been successfully used for timetabling [91], auctioning [12], and configuration [35]. A sophisticated application for information site selection is described in [79].

In this thesis, we concentrate on preferences within answer set programming. Among the numerous formalisms and semantics for preference handling in ASP, we limited ourselves to ordered logic programs with the underlying D-, W-, and B-semantics [62, 194, 34]. There, preferences are defined among rules of a logic program and select preferred answer sets from (standard) answer sets of the underlying logic program. Up to now, those preferred answer sets have been computed either by compilation methods or by meta-interpretation. Hence, the question comes up, *whether* and *how* those preferences can be integrated into an existing ASP solver. To respond to this question, we develop first an operational graph-based framework for the computation of answer sets of a logic programs. Then, we integrate preferences into this operational approach. As a result, we obtain that preferred answer sets can directly be computed by this graph-based approach, without the need of any compilation method or meta-interpretation. One major subject is the question whether an integrative approach is better than a compilation or meta-interpretation method for the computation of preferred answer sets for ordered logic programs. We show that our integrative approach performs in most cases better than the compilation method or meta-interpretation.

Another research issue in ASP are optimization methods which remove redundancies, as also found in database query optimizers. This is motivated by the fact that ASP code is often generated automatically, since the availability of efficient solvers has stimulated the use of ASP in increasingly large practical applications. Those applications require features for modular programming. For these purposes the rather recently suggested notion of strong equivalence for ASP [133, 188] can be used. Indeed, if two ASP programs are strongly equivalent, they can be used interchangeably in any context. This gives a handle on showing the equivalence of ASP modules. Moreover, if a program is strongly equivalent to a subprogram of itself, then one can always use the subprogram instead of the original program, a technique which serves as an effective optimization method. Up to now, strong equivalence has never been considered for logic programs with preferences. In this thesis, we tackle this issue and generalize the notion of strong equivalence to ordered logic programs with the underlying D-, W-, and B-semantics [62, 194, 34].

To summarize our contributions, we answer three main questions within this thesis:

1. How can preferences be integrated into an existing ASP solver?

2. Which method for the computation of preferred answer sets of ordered logic programs is better; the integration of preferences into an ASP solver or a compilation or meta-interpretation method for preferences?

3. How should ordered logic programs be handled by optimization methods? Under which conditions can we simplify ordered logic programs and their underlying preferences?

Beside these questions regarding ordered logic programs, we present two new applications for preferences within answer set programming. First, we define new procedures for group decision making, which we apply to the problem of scheduling a meeting for a group. As a second new application, we reconstruct a linguistic problem appearing in German dialects within ASP.

This thesis is organized as follows: Chapter 2 provides a brief overview over background of this work. We summarize the syntax and semantics of answer set programming, which is used throughout this thesis [5]. We briefly introduce the method of abduction, mainly used in Chapter 6, the notion of orders as basic concept for the modeling of preferences and ordered logic programs, used in Chapter 4 and 5. Finally, we consider program equivalences and program transformations, which are extended to logic programs with preferences in Chapter 5, and provide some background on computational complexity.

In Chapter 3, we investigate the usage of rule dependency graphs and their colorings for characterizing and computing standard answer sets of logic programs. This approach provides us with insights into the interplay between rules when inducing answer sets. We start with different characterizations of answer sets in terms of totally colored dependency graphs. We then develop operational characterizations of answer sets in terms of operators on partial colorings. In analogy to the notion of a derivation in proof theory, our operational characterizations are expressed as (non-deterministically formed) sequences of colorings, turning an uncolored graph into a totally colored one. In this way, we obtain an operational framework in which different combinations of operators result in different formal properties. Among others, we identify the basic strategy employed by the `noMoRe` system [141, 153] and justify its algorithmic approach. Furthermore, we distinguish operations corresponding to Fitting's operator as well as to well-founded semantics. These contributions have been published in [124, 123] and abridged reports in [121, 122]. A Prolog implementation of the operational characterization can be found at [97]. This contribution also provides the basic techniques for the ASP solver`nomore++` [152, 3, 2].

For the integration of preferences into answer set programming, we extend in Chapter 4 the operational characterization presented in Chapter 3 by preferences. We elaborate upon rule dependency graphs and their colorings for characterizing different preference handling strategies found in the literature. We start with characterizing (three types of) preferred answer sets in terms of totally colored dependency graphs. In particular, we demonstrate that this approach allows us to capture all three preference semantics in a uniform setting by means of the concept of a height function. In turn, we exemplarily develop an operational

characterization of preferred answer sets in terms of operators on partial colorings for one particular strategy. In analogy to the notion of a derivation in proof theory, our operational characterization is expressed as a (non-deterministically formed) sequence of colorings, gradually turning an uncolored graph into a totally colored one. Finally, we describe the C++ implementation of this approach, referred to as `nomore`$^<$ system, as a branch of the `nomore++` system [152]. Furthermore, we compare this integrative approach for handling preferences with a meta-interpretation and a compilation method. For these purposes, we develop benchmarks for logic programs with preferences. These contributions have been published in [125, 126, 103, 98].

Chapter 5 concentrates on notions of equivalence for logic programs with preferences. Since logic programs are often generated automatically by so-called frontends, optimization methods are important to remove redundancies. Such optimization methods are also found in database query optimizers. For our purposes the recently suggested notion of strong equivalence for ASP [133, 188] can be used. Indeed, if two ASP programs are strongly equivalent, they can be used interchangeably in any context. This gives a handle on showing the equivalence of ASP modules. Moreover, if a program is strongly equivalent to a subprogram of itself, then one can use the subprogram instead of the original program, a technique which serves as an effective optimization method. In Chapter 5 we tackle this issue and generalize the notion of strong equivalence to ASP with preferences. In detail, we consider ordered programs with three underlying semantics for preference handling. We define several notions of equivalence for ordered programs and their characterizations as well as program simplifications. Furthermore, we study computational complexity issues. These contributions have been published in [89, 90, 119].

In Chapter 6 we present two new applications of preferences within answer set programming coming from other fields of artificial intelligence and linguistics. First, we define new procedures for group decision making problems. We apply these new voting procedures to the problem of scheduling a meeting for a group. Furthermore, we include diagnostic reasoning into the scheduling problem. As a second new application, we associate linguistic optimality theory with abduction and preference handling within ASP. We present linguistic problems that appear in the study of dialects as new application of abduction and preference handling. We consider differences in German dialects, which originate from different rankings of linguistic constraints which determine the well-formedness of expressions within a language. We introduce a framework for analyzing differences in German dialects by abduction of preferences. More precisely, we will take the perspective of a linguist and reconstruct dialectal variation as an abduction problem: Given an observation that a sentence is found as grammatically correct, abduce the underlying constraint ranking. For this, we give a new definition for the determination of optimal candidates for total orders with indifferences. These contributions have been published in [118, 120, 127, 128].

We conclude with Chapter 7. For a better readability, we provide all proofs in the appendix, which also includes the bibliography and the index.

Chapter 2

Background

In this chapter, we provide the basics of answer set programming and preferences. In Section 2.1, we give the basic syntax and semantics for answer sets. Furthermore, we consider related semantics, e.g. well-founded semantics, and language extensions, e.g. disjunction, aggregate functions. In Section 2.2, we briefly consider abduction, which is used later in Chapter 6. One semantics for preference handling, which we consider in Chapter 4 and 5, is described in Section 2.4, after briefly recalling preference relations in Section 2.3. An overview about other preference semantics within answer set programming follows in Section 4.4. In Section 2.5, we consider basic concepts of program equivalences and simplifications, which are extended to preferences in Chapter 5. Lastly, we briefly recall some complexity classes in Section 2.6.

2.1 Answer Set Programming

In this section, we provide the formal definition of syntax and semantics of answer sets for logic programs. Furthermore, we consider the well-founded semantics and language extensions. For this, we assume a familiarity of the reader with logic programming, which has been intensively studied in [131, 144, 10].

2.1.1 Syntax

A *language* $\mathcal{L}$ for a logic program consists of variables, constants, function symbols of arity n, predicate symbols of arity m, and of the symbols "$\leftarrow$", "not", "(", ")", ",", and ".". A *term* is inductively defined as follows: First, variables and constants are terms. Second, if f is a function of arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term. A term is called *variable-free* (or ground) whenever it contains no variables. An *atom* is an expression $p(t_1, \ldots, t_m)$ for the predicate p with arity m and the terms $t_1, \ldots, t_m$. Also, an atom is variable-free if it contains no variables. The *Herbrand Universe* of $\mathcal{L}$, denoted with $HU_{\mathcal{L}}$, is the set of all variable-free terms in language $\mathcal{L}$. The *Herbrand Base* of the language $\mathcal{L}$, denoted by $HB_{\mathcal{L}}$, is the set of all variable-free atoms in language $\mathcal{L}$. In this work, we consider only finite Herbrand Universes. Regarding infinite Herbrand Universes we refer the reader to [16].

Given a language $\mathcal{L}$, we can define the notion of a rule.

Definition 2.1.1 *A (normal) rule r is an expression of the form*

$$p_0 \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n$$

where $n \geq m \geq 0$ and each $p_i, 0 \leq i \leq n$ is an atom.

Given a normal rule r, we denote with $body(r)$ the *body*, $\{p_1, \ldots\ p_m,\ not\ p_{m+1}, \ldots, not\ p_n\}$, of r, and with $head(r)$ the *head*, p_0, of r. To distinguish between the positive and the negative part of the body, we

define $body^+(r) = \{p_1, \ldots, p_m\}$ for the positive body and $body^-(r) = \{p_{m+1}, \ldots, p_n\}$ for the negative body. The intuitive reading of such a rule is as follows: If all atoms in $body^+(r)$ are derived and no atom in $body^-(r)$ is derivable, then $head(r)$ can be derived. Here, the negative information (indicated by the symbol not) is implicitly provided through the closed world assumption, also called negation-as-failure, [165].

Rules with an empty body, i.e. $head(r) \leftarrow$ are referred to as *facts*. We define a *(normal) logic program* Π as a set of normal rules. A logic program is called *basic* if $body^-(r) = \emptyset$ for all its rules. Furthermore, we denote with Atm, or sometimes $Atm(\Pi)$, the set of all atoms appearing in a logic program Π. Whenever a rule r contains variables, we can obtain the *ground instantiation* by applying all possible substitutions σ from the variables in r to elements of the Herbrand Universe $HU_{\mathcal{L}}$.

Example 1 *Let $\mathcal{L}$ be the language with constants $\{a, b, c\}$, variables X and Y, and the predicate symbols p and q both having arity 2. Furthermore, let Π be the following logic program:*

$$p(a, b) \leftarrow \qquad p(c, a) \leftarrow \qquad q(X, Y) \leftarrow p(Y, X)$$

Then, the ground instantiation is as follows:

$$\begin{array}{llll} p(a,b) \leftarrow & p(c,a) \leftarrow & q(a,a) \leftarrow p(a,a) & q(a,b) \leftarrow p(b,a) \\ q(a,c) \leftarrow p(c,a) & q(b,a) \leftarrow p(a,b) & q(b,b) \leftarrow p(b,b) & q(b,c) \leftarrow p(c,b) \\ q(c,a) \leftarrow p(a,c) & q(c,b) \leftarrow p(b,c) & q(c,c) \leftarrow p(c,c) & \end{array}$$

2.1.2 Semantics

In the following, we consider semantics for logic programs. First, we have a closer look at the answer set semantics and then we examine the well-founded semantics.

2.1.2.1 Answer Sets

In this section, we describe the semantics of answer sets, which was originally defined in [99, 100].

Answer sets as such are defined via a reduction to negation-as-failure-free programs: A logic program is called *basic* if $body^-(r) = \emptyset$ for all its rules. A set of atoms X is *closed under* a basic program Π if for any $r \in \Pi$, we have $head(r) \in X$ whenever $body^+(r) \subseteq X$. The smallest set of atoms which is closed under a basic program Π is denoted by $Cn(\Pi)$. The Gelfond-Lifschitz *reduct* of a normal logic program Π *relative to* a set X of atoms:

$$\Pi^X = \{head(r) \leftarrow body^+(r) \mid r \in \Pi, body^-(r) \cap X = \emptyset\}. \tag{2.1}$$

With these formalities at hand, we can define *answer set semantics* for logic programs.

Definition 2.1.2 *A set X of atoms is an* answer set *of a normal logic program Π if $Cn(\Pi^X) = X$.*

Furthermore, we use $AS(\Pi)$ for denoting the set of all answer sets of a program Π. This definition is due to [100], where the term *stable model* is used; the idea traces back to [166]. In fact, one may regard an answer set as a model of a program Π that is somehow "stable" under Π. In other words, an answer set is closed under the rules of Π, and it is "supported by Π", that is, each of its atoms has a derivation using "applicable" rules from Π.

An alternative inductive characterization for operator Cn can be obtained by appeal to an *immediate consequence operator* [144]. Let Π be a basic program and X a set of atoms. The operator T_Π is defined as follows:

$$T_\Pi(X) = \{head(r) \mid r \in \Pi, body(r) \subseteq X\}\ . \tag{2.2}$$

Iterated applications of T_Π are written as T_Π^j for $j \geq 0$, where $T_\Pi^0(X) = X$ and $T_\Pi^i(X) = T_\Pi(T_\Pi^{i-1}(X))$ for $i \geq 1$. It is well-known that $Cn(\Pi) = \bigcup_{i \geq 0} T_\Pi^i(\emptyset)$ for any basic program Π. Also, for any answer set X of program Π, it holds that $X = \bigcup_{i \geq 0} T_{\Pi^X}^i(\emptyset)$.

Another important concept is that of the *generating rules* of an answer set. The set $R_\Pi(X)$ of generating rules of a set X of atoms from program Π is defined as

$$R_\Pi(X) = \{r \in \Pi \mid body^+(r) \subseteq X, body^-(r) \cap X = \emptyset\} \ . \tag{2.3}$$

In fact, one can show that a set of atoms X is an answer set of a program Π iff $X = Cn((R_\Pi(X))^\emptyset)$ (see Theorem A.1.2 on page 141; note that $\Pi^\emptyset = \{head(r) \leftarrow body^+(r) \mid r \in \Pi\}$ for any program Π).

Example 2 *For illustration of answer sets, consider the program*

$$\Pi_1 = \{p \leftarrow p,\ q \leftarrow not\ p\} \ .$$

Among the four candidate sets, we find a single answer set, $\{q\}$, as can be verified by means of the following table:

X	Π_1^X	$Cn(\Pi_1^X)$
$\emptyset$	$p \leftarrow p$	$\{q\}$
	$q \leftarrow$	
$\{p\}$	$p \leftarrow p$	$\emptyset$
$\{q\}$	$p \leftarrow p$	$\{q\}$
	$q \leftarrow$	
$\{p, q\}$	$p \leftarrow p$	$\emptyset$

A noteworthy fact is that posing the query p or q to Π_1 in a Prolog system leads to a non-terminating situation due to its top-down approach.

Analogously, we may check that the program

$$\Pi_2 = \{p \leftarrow not\ q,\ q \leftarrow not\ p\} \tag{2.4}$$

has the two answer sets $\{p\}$ and $\{q\}$.

X	Π_2^X	$Cn(\Pi_2^X)$
$\emptyset$	$p \leftarrow$	$\{p, q\}$
	$q \leftarrow$	
$\{p\}$	$p \leftarrow$	$\{p\}$
$\{q\}$	$q \leftarrow$	$\{q\}$
$\{p, q\}$		$\emptyset$

The two rules in Π_2 are mutually exclusive and capture an indefinite situation: p can be added unless q has been added and vice versa.

Unlike the previous examples,

$$\Pi_3 = \{p \leftarrow not\ p\} \tag{2.5}$$

admits no answer set, which can be verified by the following table:

X	Π_3^X	$Cn(\Pi_3^X)$
$\emptyset$	$p \leftarrow$	$\{p\}$
$\{p\}$	$\emptyset$	$\emptyset$

Interestingly, Π_3 offers a straightforward way to model *integrity constraints*, which are rules with a head atom. This can be done by introducing a new atom f, which is inserted into the negative part of the body and into the head of the rule, i.e. the integrity constraint $\leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n$ can be replaced by the rule

$$f \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n,\ not\ f$$

Whenever an integrity constraint is violated by a candidate set, this set is eliminated by the effect observed on Program Π_3 in Equation (2.5). The usefulness of integrity constraints can be observed by adding $\leftarrow p$ to Program Π_2 in Equation (2.4). In fact, the integrity constraints $\leftarrow p$ eliminates the original answer set $\{p\}$ of Π_2, so that $\Pi_2 \cup \{\leftarrow p\}$ yields a single answer set $\{q\}$.

Regarding complexity [1], we have the following results: Given a normal logic program Π and an atom p, deciding whether p is true in some answer set of Π is NP-complete. Given a normal logic program Π and an atom p, deciding whether p is true in all answer sets of Π is co-NP-complete. Given a normal logic program Π and a set X of atoms, deciding whether X is an answer set of Π is in P.

2.1.2.2 Fitting's and Well-founded semantics

Unlike answer sets semantics, other approaches rely on 3-valued models (or partial models). Such a model consists of a set of *true* atoms, a set of *false* atoms, and a set of atoms, where there truth value is *unknown*. Given that the union of these three sets is the set of all atoms Atm appearing in a normal logic program, it is sufficient to specify two of the three sets for determining a 3-valued interpretation. Accordingly, a 3-valued interpretation I is presented by a pair (X, Y) where X and Y are sets of atoms with $X \cap Y = \emptyset$. For an atom $a \in Atm$, $a \in X$ means that a is true in I, while $a \in Y$ means that a is false in I. Otherwise, a is considered to be unknown in I.

The most prominent semantics are due to Fitting [95, 167] and Van Gelder [190]. In contrast to answer sets semantics, both aim at characterizing conclusions comprised in a single 3-valued model of the underlying program. Interestingly, such 3-valued models provide an approximation of answer sets semantics in the sense that all atoms true in such a 3-valued model belong to all answer sets of a given program, and no false atom is contained in any answer sets.

Among both 3-valued semantics, less conclusions are obtained in Fitting's semantics. It can be defined by means of the following operator.

Definition 2.1.3 *Let Π be a logic program and let X, Y be sets of atoms.*
We define

$$\begin{aligned}\Phi^+_\Pi(X,Y) &= \{head(r) \mid r \in \Pi, body^+(r) \subseteq X, body^-(r) \subseteq Y\}\\ \Phi^-_\Pi(X,Y) &= \{q \mid \textit{for all } r \in \Pi, \textit{ if } head(r) = q,\\ &\qquad \textit{then } (body^+(r) \cap Y \neq \emptyset \textit{ or } body^-(r) \cap X \neq \emptyset)\}\ .\end{aligned}$$

The pair mapping $\Phi_\Pi(X,Y) = (\Phi^+_\Pi(X,Y), \Phi^-_\Pi(X,Y))$ is often referred to as Fitting's operator [94]. Furthermore, Φ_Π is monotonic. Iterated applications of Φ_Π are written as Φ^i_Π for $i \geq 0$, where $\Phi^0_\Pi(X,Y) = (X,Y)$ and $\Phi^{i+1}_\Pi(X,Y) = \Phi_\Pi\Phi^i_\Pi(X,Y)$ for $i \geq 0$. We denote the least fixpoint by $lfp(\Phi_\Pi) = \bigcup_{i\geq 0} \Phi^i_\Pi$ and refer to it as *Fitting semantics* for the program Π. We have $lfp(\Phi_\Pi) = \Phi^{n+1}_\Pi(\emptyset,\emptyset) = \Phi^n_\Pi(\emptyset,\emptyset)$ for some finite n since Π is finite.

For capturing other semantics, the construction $Cn(\Pi^X)$ is sometimes regarded as an operator $C_\Pi(X)$. The anti-monotonicity of C_Π implies that C^2_Π is monotonic. As shown in [189], different semantics are obtained by distinguishing different groups of (alternating) fixpoints of $C^2_\Pi(X)$. For instance, given a program Π, the least fixed point of C^2_Π is known to amount to its *well-founded model*. Answer sets of Π are simply fixed points of C^2_Π that are also fixed points of C_Π. The well-founded model can be characterized in

[1] Complexity Classes are described in Section 2.6.

terms of the least fixpoint of operator C^2_Π. That is, the well-founded model of a program Π is given by the 3-valued interpretation $(lfp(C^2_\Pi), Atm \setminus C_\Pi lfp(C^2_\Pi))$. Hence, it is sufficient to consider the least fixpoint of C^2_Π, since it determines the well-founded model. We therefore refer to the least fixpoint of C^2_Π as the *well-founded set* of Π. The set $Atm \setminus C_\Pi lfp(C^2_\Pi)$ is usually referred to as the *unfounded* set of Π.

Concerning 3-valued interpretations, we obtain the following definition of unfounded sets [190].

Definition 2.1.4 *Let Π be a logic program and let Z be a set of atoms.*

Furthermore, let (X, Y) be a 3-valued interpretation.

Then, Z is an unfounded set *of Π w.r.t. (X, Y) if each $q \in Z$ satisfies the following condition: For each $r \in \Pi$ with $head(r) = q$, one of the following conditions hold:*

1. *$body^+(r) \cap Y \neq \emptyset$ or $body^-(r) \cap X \neq \emptyset$;*
2. *there exists a $p \in body^+(r)$ such that $p \in Z$.*

The *greatest unfounded set* of Π w.r.t. (X, Y), denoted $\mathbf{U}_\Pi(X, Y)$, is an unfounded set, which is the union of all sets that are unfounded w.r.t. (X, Y).

Alternatively, the well-founded model can then be obtained by iteratively applying Fitting's operator and computing greatest unfounded sets. Iterated applications of Φ_Π and $\mathbf{U}_\Pi$ are written as $\mathcal{W}^{i+1}_\Pi = \Phi_\Pi(\mathcal{W}^i_\Pi) \cup \mathbf{U}_\Pi(\mathcal{W}^i_\Pi)$, where $\mathcal{W}^0_\Pi = (\emptyset, \emptyset)$, for $i \geq 0$. The least fixpoint is then the well-founded model of Π and defined as $\mathcal{W}^\omega_\Pi = \cup_{i \geq 0} \mathcal{W}^i_\Pi$. There always exists a well-founded model of Π. If the well-founded model is total, then the set of true atoms is the unique answer set of Π. Furthermore, the well-founded model is a subset of every answer set, i.e. all true atoms are true in all answer sets and no false atoms of the well-founded model is contained in any answer sets of Π.

Example 3 *Let us consider the following logic program:*

$$\Pi = \left\{ \begin{array}{lll} a & \leftarrow & \\ b & \leftarrow & a \\ c & \leftarrow & a, not\ d \\ d & \leftarrow & not\ c \\ e & \leftarrow & f \\ f & \leftarrow & e \end{array} \right\}$$

The answer sets of Π are $\{a, b, c\}$ and $\{a, b, d\}$. Applying Fitting's operator applied to $(\emptyset, \emptyset)$ yields $\{a, b\}$ as true and none atom as false. Furthermore, we obtain that $\{e, f\}$ is the greatest unfounded set w.r.t. $(\emptyset, \emptyset)$. Hence, we get the 3-valued model $(\{a, b\}, \{e, f\})$. Repeated applications of Φ_Π and $\mathbf{U}_\Pi$ result again in $(\{a, b\}, \{e, f\})$. Hence, Π has the well-founded model $(\{a, b\}, \{e, f\})$.

Regarding program $\Pi = \{p \leftarrow not\ p\}$, we get $(\emptyset, \emptyset)$ as well-founded model, whereas Π has no answer set. Hence, the well-founded semantics leaves every atom as undefined in this conflicting situation, where no answer set exists.

Considering complexity, the Fitting semantics can be computed in linear time in size of Π and the well-founded model in quadratic time in size of Π.

2.1.3 Language Extensions

In this section, we consider language extensions of normal logic programs. First, we show how easily normal logic programs can be extended by classical negation and other Boolean expressions. Second, we consider language extensions of logic programs by weight and aggregate functions, e.g. counting elements in a set, finding maximal or minimal elements.

2.1.3.1 Classical negation

Normal logic programs provide negative information implicitly through the closed world assumption [165]. Let us consider the following rule: $cross \leftarrow\ not\ train$. If $train$ is not derivable, the atom $cross$ becomes true. But this may lead to a disaster because we have no explicit information that there in fact is no train. An alternative would be to use an explicit negation operator $\neg$. Then we can express the previous rule as follows: $cross \leftarrow\ \neg train$.

An atom p or a (classical) negated atom $\neg p$ is called a *literal*. Logic programs with literals are called *extended logic programs* [100]. An extended logic program is *contradictory* [10] if complementary literals, e.g. $train$ and $\neg train$, are enforced to be in an answer set. In that case, one obtains exactly one answer set, viz the set of all literals. If a program is not contradictory, the definition of answer sets of extended logic programs carries over from normal ones.

Classical negation can be eliminated by a polynomial transformation, replacing each negated atom $\neg p$ by a new atom p' and adding the rules,

$$q \leftarrow p, p' \quad \text{and} \quad q' \leftarrow p, p' \,, \tag{2.6}$$

for each atom p and q. The rules in (2.6) generate the set of all literals in case of contradictory programs. For preserving only consistent answer sets, we may add integrity constraints $\leftarrow p, p'$ for each atom p, instead of the rules in (2.6).

2.1.3.2 Disjunction

Disjunctive logic programs extend normal or extended logic programs by disjunctive information in the head of a rule [100]. More precisely, the head of a rule is a disjunction $q_0; \ldots; q_k$ of atoms or literals q_i respectively, where $0 \leq i \leq k$. E.g. $p; q$ expresses that "p is *true* or q is *true*". Letting $head(r) = \{q_0, \ldots, q_k\}$, a set of atoms X is closed under a basic program Π if for any $r \in \Pi$, $head(r) \cap X \neq \emptyset$ whenever $body^+(r) \subseteq X$. The definition of Π^X carries over from normal programs. An answer set X of a disjunctive logic program is a $\subseteq$- minimal set of atoms being closed under Π^X. For example, the disjunctive logic program $\Pi = \{p; q \leftarrow\}$ has the answer sets $\{p\}$ and $\{q\}$. The set $\{p, q\}$ is closed under $\Pi^{\{p,q\}}$, but it is not an answer set of Π since it is not $\subseteq$-minimal. Observe that adding the rules $p \leftarrow$ and $q \leftarrow$ to Π makes $\{p, q\}$ the only answer set of $\Pi \cup \{p \leftarrow, q \leftarrow\}$.

The usage of disjunction raises the complexity of the underlying decision problems, e.g. deciding whether there exists an answer set X for a given atom p such that $p \in X$ is Σ_2^P-complete [83]. Answer sets of disjunctive logic programs can be computed by using the `DLV` system [73, 130].

2.1.3.3 Weight Constraints

A *weight constraint* [180] is of the form

$$l \leq \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, not\ b_1 = w_{b_1}, \ldots, not\ b_n = w_{b_n}\} \leq u$$

where each a_i, b_j is an atom. Each atom or negated atom has an associated weight, e.g. w_{b_1} is the weight of $not\ b_1$. The numbers l and u give $lower$ and $upper$ bounds, respectively, of the constraint. The weights can be real numbers, where also negative weights are allowed. Intuitively, a weight constraint is satisfied by a set of atoms S if the sum of weights of those atoms of $\{a_1, \ldots, a_n\}$ that are in S and those atoms of $\{b_1, \ldots, b_m\}$ that are not in S is between the bounds l and u. Weight constraints may appear in the head or in the positive body of rules.

Special cases of weights constraint are cardinality constraints and choice rules. *Cardinality constraints* are of the form $l\ \{a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m\}\ u$, where each atom a_i and each negated atom $not\ b_j$ has weight 1. For the special case, where we do not have negated atoms, i.e. $l\ \{a_1, \ldots, a_n\}\ u$, the cardinality constraint "selects" subsets of $\{a_1, \ldots, a_n\}$ of size between l and u.

Either of the bounds l and u can also be omitted in which case a missing lower bound is taken to be $-\infty$ and a missing upper bound is taken to be ∞. As a special case, we get so called *choice rules*, which are of the form

$$\{h_1, \ldots, h_k\} \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots not\ b_n.$$

That is, whenever the body of a choice rule is satisfied, we can derive an arbitrary subset of $\{h_1, \ldots, h_k\}$.

Interestingly, one can also model integrity constraints $\leftarrow C_1, \ldots, C_n$ with weight rules by taking an unsatisfiable constraint, e.g. $1 \leq \{\}$, as head of such a rule.

Another important feature of weight constraints is the modeling of optimization statements, which allow to select subsets with a minimal or maximal weight, respectively. A *minimization* statement M is of the form

$$minimize\{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, not\ b_1 = w_{b_1}, \ldots, not\ b_n = w_{b_n}\}$$

declaring that we want to find a stable model S with the smallest weight

$$w(M, S) = \sum_{a_i \in S} w_{a_i} + \sum_{b_j \notin S} w_{b_j}.$$

Analogously, we have *maximize statements* of the form

$$maximize\{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, not\ b_1 = w_{b_1}, \ldots, not\ b_n = w_{b_n}\}.$$

Regarding complexity, we have the following for extended logic programs with weight constraints, where we assume that all weights and bounds are integers: Let Π be a finite set of weight rules without optimization statements and S be a set of atoms. Then it can be decided in polynomial time whether S is an answer set of Π. Deciding whether a set of weight rules without optimization statements has an answer set is NP-complete. Computing an answer set of a set of weight rules without optimize statements is FNP-complete. [2] However, adding optimization statements raises the complexity, since we are looking for optimal models. Deciding whether a set of weight rules has an optimal model containing a given atom is Δ_2^P-hard. Let Π be a set of weight rules with optimization statements. Then computing an optimal answer set of Π is $\mathsf{FP}^{\mathsf{NP}}$-complete. [3]

Weight constraints as language extensions are supported by the `smodels` system [181, 150].

2.1.3.4 Aggregate functions and Templates

The `DLV` system supports, besides disjunctive programs, arithmetic operators (like sum, times, count) [66, 67, 93, 42, 73, 88] and templates [107, 44].

Aggregate predicates allow to express properties over a set of elements. They can occur in the bodies of rules and constraints, possibly negated using negation-as-failure. Aggregates often allow to describe problems in a more natural way.

Since aggregate functions are defined over sets, we first present the notion of a symbolic set [66]. A *symbolic set* is a pair $\{Vars : Conj\}$, where $Vars$ is a list of variables, and $Conj$ is a conjunction of standard literals. A ground set is a set of pairs of the form $\langle \bar{t} : Conj \rangle$, where $\bar{t}$ is a list of constants and $Conj$ is a variable-free conjunction of literals. For example, assume that $p(a, 1), p(c, 1)$ and $p(c, 2)$ holds. Then, the symbolic set $\{X : p(X, 1)\}$ is interpreted as $\{\langle a : p(a, 1)\rangle, \langle c : p(c, 1)\rangle\}$. That is, the symbolic set is the ground instantiation for variable X such that $p(X, 1)$ holds.

The `DLV` system provides the following aggregate functions [67, 73]:

- $\#count$, which counts the number of elements in a set:

[2] FNP is the class of all function problems associated with languages in NP.

[3] $\mathsf{FP}^{\mathsf{NP}}$ is the class of all functions from strings to strings that can be computed by a polynomial-time Turing machine with a `NP` oracle.

- $\#sum$, which sums up a set of numbers;
- $\#times$, which calculates the product of a set of number; and
- $\#min$ and $\#max$, which select the minimum/maximum element in a set, where the lexicographic ordering is considered when the set contains strings.

Example 4 *Let* Π *be the following logic program.* $\Pi = \{p(1,1) \leftarrow,\ p(2,1) \leftarrow,\ p(5,1) \leftarrow,\ p(2,2) \leftarrow,\ p(4,2) \leftarrow,\ p(7,3) \leftarrow\} \cup \{i(N) \leftarrow |\ N = 1,2,\ldots,7\}$*, where the predicate* $i(.)$ *provides the range of* $p(.,.)$ *ensuring safety of aggregate functions [67]. Then, the aggregate function*

$$\#sum\{X : p(X,1)\}$$

is instantiated by $\#sum\{\langle 1 : p(1,1)\rangle, \langle 2 : p(2,1)\rangle, \langle 5 : p(5,1)\rangle\}$ *and returns the value* $8 = 1 + 2 + 5$*, which can be obtained by writing* $N = \#sum\{X : p(X,1)\}$*. Analogously,* $N = \#count\{X : p(X,1)\}$ *gives* $N = 3$*.* $N = \#times\{X : p(X,1)\}$ *gives* $N = 10$*. The maximal element w.r.t. the first component of predicate* p *can be obtained by* $N = \#max\{X : p(X,Y), i(Y)\}$*, which returns* $N = 7$*. Note that in the expression* $\#max\{X : p(X,Y), i(Y)\}$ *the variable* Y *is instantiated by using the predicate* $i(.)$*. Analogously, the minimal element of the set, where the second component of predicate* $p(.,.)$ *is* 2 *is obtained by* $N = \#min\{X : p(X,2)\}$ *which yields* $N = 2$*.*

The well-founded semantics for aggregate functions has been defined in [42].

Furthermore, DLV supports a language extension with "template" predicates, which are somehow similar to Object Oriented Logic Programming [59, 107]. For example, the template

$$\begin{array}{l}\#template\\ subset[p(1)](1)\{subset(X) \vee \neg subset(X) \leftarrow p(X)\}\end{array}$$

defines subsets of the domain of a given predicate $p(.)$. For more details about templates, see [107, 44].

2.2 Abduction

Consider a situation in which a boat can be used to cross a river if it is not leaking or, if it is leaking, there is a bucket to scoop the water out of the boat [136]. This situation can be formalized by the following rules:

$$\begin{array}{rcl} canCross & \leftarrow & boat, not\, leaking \\ canCross & \leftarrow & boat, leaking, hasBucket \end{array}$$

Now, you observe somebody who crosses the river with a boat. But, how can you explain that? One explanation for $canCross$ is $\{boat\}$, meaning that you have a boat and that is it not leaking. Other explanations are $\{boat, leaking, hasBucket\}$ as well as $\{boat, hasBucket\}$. Problems, where we search for explanations for an observation, are called *abductive problems*.

An abductive framework [96, 68, 84, 116] is a triple $\langle \Pi, H, O\rangle$, where Π is a logic program, H is a set of facts, referred to as *hypotheses*, and O is a set of atoms, referred to as *observations*. A set $\Delta \subseteq H$ is an *explanation* of O w.r.t. Π if there is an answer sets of $\Pi \cup \Delta$ which contains O (brave reasoning). Note that in the literature some definitions of abduction are based on cautious reasoning, i.e. O must be contained in *all* answer sets of $\Pi \cup \Delta$. An explanation Δ_1 is minimal, if for every other explanation Δ_2 of O, $\Delta_2 \not\subset \Delta_1$ holds. We call Δ a *single* explanation if $|\Delta| = 1$. In the $canCross$ example, $\{boat\}$, is the minimal (and single) explanation.

Hence, abduction offers a very natural inference process of forming a hypothesis that explains observed phenomena. The diagnosis front-end of the DLV system [73, 76] offers a comfortable way for computing explanations within answer set programming.

The usefulness of abduction [68, 114] has been demonstrated in a variety of applications, e.g. in diagnosis of electrical circuitry [76], air crew-assignment [68], and database updates [115].

Complexity results for abduction are provided in [84].

2.3 Orders

Next, we give some terminology and notations about preference relations.

A *(partial) order* R on a set X is a reflexive, transitive and antisymmetric relation on X (recall that R is antisymmetric iff for all $x, y \in X$, $R(x, y)$ and $R(y, x)$ implies $x = y$.) $R(x, y)$ is also denoted by $x \succeq_R y$. $\succ_R$ denotes the strict relation induced from R, defined by $x \succ_R x'$ iff $x \succeq_R x'$ and not $(x' \succeq_R x)$[4]. An order R is *total* (or *complete*, *linear*) iff $R(x, x')$ or $R(x', x)$ holds for all $x, x' \in X$. Let R, R' be two orders on X. R' *extends* R iff $R \subseteq R'$, that is, $R(x, x')$ implies $R'(x, x')$ for all $x, x' \in X$. Let R be an order. Then, a total order T is an *complete extension* of R iff T extends R. $Ext(R)$ denotes the set of all complete extensions of R.

2.4 Ordered Logic Programs

In this section, we consider preferences among rules of a logic program and the underlying semantics. A logic program Π is said to be *ordered* if we have a set $\mathcal{N}$ of terms serving as *names* for rules and a *preference relation* $< \subseteq \mathcal{N} \times \mathcal{N}$ as a strict partial order among rules, where we write $s < t$ for the names $s, t \in \mathcal{N}$ of rules in Π. [5] Furthermore, we assume a bijective function $n(.)$ assigning to each rule $r \in \Pi$ a name $n(r) \in \mathcal{N}$. To simplify our notation, we usually write r_i instead of $n(r_i)$ for some $r_i \in \Pi$. Given $r_i, r_j \in \Pi$, $r_i < r_j$ states that r_j has higher priority than r_i. Formally, an ordered (logic) program can be understood as a quadruple $(\Pi, \mathcal{N}, n, <)$, where Π is a logic program, n is a bijective function between Π and the set of names $\mathcal{N}$, and $<$ is a set of preference relations over Π. Whenever possible, we leave $\mathcal{N}$ and n implicit and just write $(\Pi, <)$ for ordered programs. Moreover, we write $<= \emptyset$ if the partial order is empty, that is, $(\Pi, \emptyset)$ denotes a logic program. The interpretation that one rule has higher priority than another rule can be made precise in different ways. In what follows, we consider three such interpretations: D– [62], B– [34], and W– preference [194]. Given $(\Pi, <)$, all of them use $<$ for selecting (different) preferred answer sets among the standard answer sets of Π. We now recall the definitions of these semantics. [6]

Definition 2.4.1 *Let $(\Pi, <)$ be an ordered logic program and let X be an answer set of Π.*

Then, X is called $<^D$–preferred, if an enumeration $\langle r_i \rangle_{i \in I}$ of Π exists such that for every $i, j \in I$ we have

1. *if $r_i < r_j$, then $j < i$, and*
2. *if $r_i \in R_\Pi(X)$ then $body^+(r_i) \subseteq \{head(r_j) \mid r_j \in R_\Pi(X), j < i\}$, and*
3. *if $r_i \in \Pi \setminus R_\Pi(X)$ then*
   (a) *$body^+(r_i) \not\subseteq X$ or*
   (b) *$body^-(r_i) \cap \{head(r_j) \mid r_j \in R_\Pi(X), j < i\} \neq \emptyset$.*

[4] Or equivalently, since R is antisymmetric: $x \succ_R x'$ iff $x \succeq_R x'$ and $x' \neq x$).

[5] Within this notation we don't allow rules with an empty head. Such integrity constraints $\leftarrow$ *body* have to be written as $f \leftarrow body, not\ f$ for f being a new atom.

[6] In the following, I is an index set $I = \{1, \ldots, n\}$ for $|\Pi| = n$.

Condition 1 stipulates that the enumeration of Π is compatible with $<$. Condition 2 makes the property of supportedness explicit. Although any answer set is generated by a supported sequence of rules, in D–preferences, rules cannot be supported by lower-ranked ones. Condition 3a separates the handling of unsupported rules from preference handling. Condition 3b guarantees that rules can never be blocked by lower-ranked ones. [7] Let us consider the following examples.

Example 5 *The ordered program*

$$(\Pi_1, <_1) = \left\{ \begin{array}{ll} r_1 : a \leftarrow & \\ r_2 : b \leftarrow a & r_1 <_1 r_2 \end{array} \right\}$$

has no $<^D$-preferred answer set, since before applying r_2 we have to apply r_1 (Condition 2 in Definition 2.4.1), which is contradictory to the given preference relation $r_1 <_1 r_2$, where the more preferred rule r_2 has to be derived before the lower ranked rule r_1 (Condition 1 in Definition 2.4.1).

Example 6 *The ordered program*

$$(\Pi_2, <_2) = \left\{ \begin{array}{ll} r_1 : b \leftarrow a, not\ c & r_3 <_2 r_2 \\ r_2 : c \leftarrow not\ b & r_2 <_2 r_1 \\ r_3 : a \leftarrow & \end{array} \right\}$$

has the standard answer sets $\{a, b\}$ and $\{a, c\}$, where both are not $<^D$- preferred. For answer set $\{a, b\}$, the rule r_1 is applicable only after having applied the lower ranked rule r_3, which is not allowed in this semantics. For answer set $\{a, c\}$, the rule r_1 is blocked by the lower ranked rule r_2, i.e. $c \in body^-(r_1)$ is derived by using lower ranked rule r_2 (Condition 3b in Definition 2.4.1).

Next, we consider the W-semantics which weakens the concept of order preservation in Condition 2 and 3 of the D-semantics.

Definition 2.4.2 *Let $(\Pi, <)$ be an ordered logic program and let X be an answer set of Π.*

Then, X is called $<^W$–preferred, if an enumeration $\langle r_i \rangle_{i \in I}$ of Π exists such that for every $i, j \in I$ we have

1. *if $r_i < r_j$, then $j < i$, and*
2. *if $r_i \in R_\Pi(X)$ then*
   (a) *$body^+(r_i) \subseteq \{head(r_j) \mid r_j \in R_\Pi(X), j < i\}$ or*
   (b) *$head(r_i) \in \{head(r_j) \mid r_j \in R_\Pi(X), j < i\}$, and*
3. *if $r_i \in \Pi \setminus R_\Pi(X)$ then*
   (a) *$body^+(r_i) \not\subseteq X$ or*
   (b) *$body^-(r_i) \cap \{head(r_j) \mid r_j \in R_\Pi(X), j < i\} \neq \emptyset$ or*
   (c) *$head(r_i) \in \{head(r_j) \mid r_j \in R_\Pi(X), j < i\}$.*

For W–preference, the concept of order preservation of D–preference is weakened in Condition 2 and 3 for suspending both conditions, whenever the head of a preferred rule is derivable in an alternative way. [8]

[7] Note that in Definition 2.4.1 we do not necessarily have to require that X has to be an answer set, since conditions 2 and 3 capture the conditions for X being an answer set.

[8] As in Definition 2.4.1, the condition that X has to be an answer set is not necessarily required.

Example 7 *Let us reconsider the examples given above. For the program* $(\Pi_1, <_1)$ *given in Example 5 we have that* $\{a, b\}$ *is not a* $<^W$*- preferred answer set. But adding the rule* $b \leftarrow$ *yields* $\{a, b\}$ *as* $<^W$*- preferred answer set of* $(\Pi_1 \cup \{b \leftarrow\}, <_1)$ *since the head of* r_2 *is derivable in an alternative way, namely by the non-preferred rule* $b \leftarrow$.

For the ordered program $(\Pi_2, <_2)$ *given in Example 6, we also have that no answer set is* $<^W$*-preferred since the heads are not derivable in an alternative way.*

Finally, we consider the B-semantics.

Definition 2.4.3 *Let* $(\Pi, <)$ *be an ordered logic program and let* X *be an answer set of* Π.

Then, X *is called* $<^B$*–preferred, if an enumeration* $\langle r_i \rangle_{i \in I}$ *of* Π *exists such that for every* $i, j \in I$ *we have*

1. *if* $r_i < r_j$*, then* $j < i$*, and*
2. *if* $r_i \in \Pi \setminus R_\Pi(X)$ *then*
   (a) $body^+(r_i) \not\subseteq X$ *or*
   (b) $body^-(r_i) \cap \{head(r_j) \mid r_j \in R_\Pi(X), j < i\} \neq \emptyset$ *or*
   (c) $head(r_i) \in X$.

B–preference drops Condition 2; thus decoupling preference handling from the order induced by consecutive rule applications. [9]

Example 8 *For the ordered program* $(\Pi_1, <_1)$ *given in Example 5, we can see that* $\{a, b\}$ *is a* $<^B$*-preferred answer set since preference handling is decoupled from rule application.*

Furthermore, for program $(\Pi_2, <_2)$ *given in Example 6, we get* $\{a, b\}$ *as* $<^B$*-preferred answer set. Like with the other semantics, answer set* $\{a, c\}$ *is non-preferred also in the* B*-semantics, since rule* r_1 *is blocked by the less preferred rule* r_1 *and* $head(r_1)$ *is not in the corresponding answer set.*

For $\sigma \in \{D, W, B\}$, we define $AS^\sigma((\Pi, <))$ as the set of all $<^\sigma$-preferred answer sets of ordered logic program $(\Pi, <)$. As shown in [177], the three strategies yield an increasing number of preferred answer sets. That is, D–preference is stronger than W–preference, which is stronger than B–preference, which is stronger than no preference. More precisely, we have

$$AS^D((\Pi, <)) \subseteq AS^W((\Pi, <)) \subseteq AS^B((\Pi, <)) \subseteq AS(\Pi).$$

Whenever no preferences are specified, the preferred answer sets are exactly the standard answer sets, i.e. $AS^\sigma((\Pi, \emptyset)) = AS(\Pi)$ for $\sigma \in \{D, W, B\}$. In the following, we sometimes just write preferred answer sets whenever we mean $<^\sigma$- preferred, or $<^\sigma$-preserving answer set, where σ holds for D, W, and B.

As seen in Example 5, neither of the three semantics guarantee the existence of a preferred answer set. For this reason, the notion of *weakly preferred answer sets* has been defined in [34].

The given approaches preserve all anti-monotonicity. That is, increasing the preference relations imply a decreasing number of preferred answer sets [177].

Theorem 2.4.1 *Let* $(\Pi, <_1)$ *and* $(\Pi, <_2)$ *be ordered logic programs and* $\sigma \in \{D, W, B\}$.
If $<_1 \subseteq <_2$ *then* $AS^\sigma((\Pi, <_2)) \subseteq AS^\sigma((\Pi, <_1))$.

An important property of these preference approaches is that they fulfill Brewka's and Eiter's Principles for priorities [34]:

[9] For this Definition, we have to require that X is an answer set, since B- preferences pay no attention to supportness of rules.

Principle I. Let $(\Pi, <)$ be an ordered logic program and let X_1 and X_2 be two standard answer sets of Π generated by $R_\Pi(X_1) = R \cup \{r_1\}$ and $R_\Pi(X_2) = R \cup \{r_2\}$, respectively, where $r_1, r_2 \notin R$. If $r_2 < r_1$, then X_2 is not a preferred answer set of Π.

Principle II. Let $(\Pi, <)$ be an ordered logic program and let X be a preferred answer set of Π. Furthermore, let $r \notin \Pi$ be a rule such that $body^+(r) \not\subseteq X$. Then, X is a preferred answer set of $(\Pi \cup \{r\}, <)$.

The first principle states the main intuition of preferences among rules. Intuitively, whenever two rules r_1 and r_2, where r_1 is preferred over r_2, are "responsible" for generating two different answer sets, the answer set generated by r_2 should not be preferred. The second principle expresses that adding non-applicable rules, where the positive body is not derivable, should not change the preferredness of answer sets.

The presented preference relations have also been extended to the dynamic case in [62]. More precisely, a dynamic preference is of the form $n(r_i) < n(r_j) \leftarrow body$ expressing that the preference relation $n(r_i) < n(r_j)$ becomes active whenever the $body$ of the rule is true. In what follows, we consider only the static case $n(r_i) < n(r_j) \leftarrow$, which is interpreted as order among rules.

The different three semantics have been implemented by meta-interpretation [77, 74] and by the `plp` system, a pre-compilation front-end [62, 161] for existing answer set programming solvers. An integration into an answer set programming solver is be presented in Chapter 4. There, we provide an operational characterization for the computation of preferred answer sets together with a C++ implementation, including benchmarks for ordered programs and a comparison to the `plp` system and the meta-interpreter. Computational complexity issues for the D-, W-, and B-semantics are given in Chapter 5.

The `plp` system associated with the D-semantics [61] has successfully been used for information-site selection [79]. A prototypical environment of a movie domain has been developed, which comprises (i) basic domain knowledge, (ii) XML sources containing movie data wrapped from the Internet Movie Database [108] and other movie related data sources, and (iii) suitable site descriptions. Queries are formulated in XML-QL [70], and can be executed after site selection on the respective source. Rule-based preferences are then used to select sites such that the utility of the answer (in terms of quality of the result and other criteria) is as large as possible for the user.

As another example, the D-semantics can be used within the problem of searching Hamiltonian cycles in graphs. There, preferences are used to "guide" the search for a cycle, e.g. vertex v should be visited before vertex v'. Also, for the coloring problem of a graph, preferences can be used to prefer special colors for certain vertices. For more details about including preferences in benchmarks see Section 4.2.

The D-semantics builds upon previous work presented in [60] has been used to link default logic with autoepistemic logic in [85]. There, classical default theories have been treated as ordered theories and it is given a simple translation of classical default logic into an autoepistemic system with language constructs representing different degrees of confidence.

2.5 Program Equivalences

Since answer set programming code is often generated automatically by so-called front-ends, one needs optimization methods removing redundancies, as also found in database query optimizers. For these purposes the recently suggested notion of strong equivalence for answer set programming [133, 188] can be used. Indeed, if two ASP programs are strongly equivalent, they can be used interchangeably in any context. This gives a handle on showing the equivalence of ASP modules. If a program is strongly equivalent to a subprogram of itself, then one can always use the subprogram instead of the original program, a technique which serves as an effective optimization method.

2.5.1 Strong and Uniform Equivalence

In the following, we will consider some notions of program equivalences under answer set semantics, which have been intensively studied in [154, 133, 80, 78, 188, 196, 157]. For the sake of completeness, we give first the notion of *equivalence*.

Definition 2.5.1 *Let* Π_1 *and* Π_2 *be logic programs.*
Then, Π_1 *and* Π_2 *are* equivalent*, denoted by* $\Pi_1 \equiv \Pi_2$*, if* $AS(\Pi_1) = AS(\Pi_2)$.

Inspired by database techniques, we give next the definition of uniform equivalence.

Definition 2.5.2 *Let* Π_1 *and* Π_2 *be logic programs.*
Then, Π_1 *and* Π_2 *are* uniformly equivalent*, denoted by* $\Pi_1 \equiv_u \Pi_2$*, if for any set of facts* F *we have* $AS(\Pi_1 \cup F) = AS(\Pi_2 \cup F)$.

More precisely, two programs are uniformly equivalent if no matter which facts we add to a program, we get the same results. More precisely, if a program Π is uniformly equivalent to one of its subprograms $\Pi' \subseteq \Pi$, then we can replace Π by Π' and we will still get the same answer sets when adding arbitrary facts to Π' and Π, respectively.

Instead of adding facts, of course, one can consider the addition of arbitrary programs.

Definition 2.5.3 *Let* Π_1 *and* Π_2 *be logic programs.*
Then, Π_1 *and* Π_2 *are* strongly equivalent*, denoted by* $\Pi_1 \equiv_s \Pi_2$*, if for any set of rules* R *we have* $AS(\Pi_1 \cup R) = AS(\Pi_2 \cup R)$.

That is, no matter which program R is added to Π_1 or Π_2, $\Pi_1 \cup R$ and $\Pi_2 \cup R$ yield the same answer sets if they are strongly equivalent. Obviously, whenever two programs are strongly equivalent, they are uniformly equivalent but not vice versa. As shown in [133], strong equivalence is closely related to the non-classical logic of *here-and-there*, which was adapted to logic-programming terms in [133, 188]: Let Π be a logic program, and let $X, Y \subseteq A$ such that $X \subseteq Y$. The pair (X, Y) is an *SE-model* of Π, if $Y \models \Pi$ and $X \models \Pi^Y$. By $SE(\Pi)$ we denote the set of all SE-models of Π. Usually the set A is left implicit and one just writes $SE(\Pi)$. Then, for any logic programs Π_1 and Π_2, Π_1 and Π_2 are strongly equivalent iff they coincide on their set of SE-models [188]. Deciding whether two normal programs are strongly or uniformly equivalent is co-NP-complete [134, 158, 82].

As discussed in [159], uniform and strong equivalence are essentially the only concepts of equivalence obtained by varying the logical form of the program extension. Current research concentrates on relativized notions of strong and uniform equivalence. Instead of considering *all* possible program extensions, one considers only programs built from a restricted alphabet or, orthogonally, compare answer sets only on a subset of the alphabet [196, 82], which seems to be very natural from the practical point of view. Other works focus on the extension of notions of equivalence for the non-ground case [81, 134].

2.5.2 Program Simplifications

Using notions of program equivalences, one can define program simplifications removing redundancies as done in [25, 193, 157, 154, 80]. In the following, we will concentrate on program simplifications under strong equivalence as defined in [80, 154], restricted to the case of (normal) logic programs.

Theorem 2.5.1 *[80] Let* Π *be a logic program.*

TAUT: *Let* $r \in \Pi$ *such that* $head(r) \in body^+(r)$*. Then,* $\Pi \equiv_s \Pi \setminus \{r\}$.

RED$^-$: *Let* $r_1, r_2 \in \Pi$ *such that* $head(r_2) \in body^-(r_1)$, $body(r_2) = \emptyset$*. Then,* $\Pi \equiv_s \Pi \setminus \{r_1\}$.

NONMIN: *Let* $r_1, r_2 \in \Pi$ *such that* $head(r_1) = head(r_2)$ *and* $body(r_2) \subseteq body(r_1)$. *Then,* $\Pi \equiv_s \Pi \setminus \{r_1\}$.

WGPPE: *Let* $r_1 \in \Pi$, $a \in body^+(r_1)$, $G_a = \{r_2 \in \Pi \mid head(r_2) = a\}$ *and* $G_a \neq \emptyset$. *Then,* $\Pi \equiv_s \Pi \cup G'_a$ *for* $G'_a = \{head(r_1) \leftarrow (body^+(r_1) \setminus \{a\}) \cup not\ body^-(r_1) \cup body(r_2) \mid r_2 \in G_a\}$.

CONTRA: *Let* $r \in \Pi$ *such that* $body^+(r) \cap body^-(r) = \emptyset$. *Then,* $\Pi \equiv_s \Pi \setminus \{r\}$.

S-IMP*: *Let* $r, r' \in \Pi$ *such that there exists an* $A \subseteq body^-(r')$ *such that* $head(r) \in A$, $body^-(r) \subseteq body^-(r') \setminus A$ *and* $body^+(r) \subseteq body^+(r')$. *Then,* $\Pi \equiv_s \Pi \setminus \{r'\}$.

Note that, *WGPPE* adds rules to a program to enable other transformations like *TAUT*,*RED*$^-$, *NONMIN*, *CONTRA*, or *S-IMP**. Hence, the simplifications given above remove redundancies, and one can replace a program by one of its a subprogram described above without changing semantics.

In Chapter 5, we consider program equivalences and simplification of ordered programs (defined in Section 2.4 on page 13), which have never been studied before.

2.6 Complexity Classes

In this section, we briefly introduce relevant complexity classes [156]. P is the class of decision problems deterministically solvable in polynomial time. NP is the class of decision problems solvable by a nondeterministic polynomial-time Turing machine. A decision problem A is in coNP if the complement of A is in NP. The classes Σ_k^P, Π_k^P, and Δ_k^P of the Polynomial Hierarchy are defined as follows

$$\begin{array}{l}\Delta_0^P = \Sigma_0^P = \Pi_0^P = \mathsf{P} \text{ and} \\ \text{for all } k \geq 1, \Delta_k^P = \mathsf{P}^{\Sigma_{k-1}^P}, \Sigma_k^P = \mathsf{NP}^{\Sigma_{k-1}^P}, \Pi_k^P = \mathsf{co\text{-}\Sigma_k^P},\end{array}$$

where NP^C denotes the class of decision problems that are solvable in polynomial time by a nondeterministic Turing machine with an oracle for any decision problem π in the class C, and P^C the class of decision problems that are solvable in polynomial time with an oracle for any decision problem π in the class C. In particular, we have $\mathsf{NP} = \Sigma_1^P$, $\mathsf{co\text{-}NP} = \Pi_1^P$, and $\Delta_2^P = \mathsf{P}^{\mathsf{NP}}$. Let C be a complexity class and L be a decision problem in C. Then, L is *C-complete* if any problem $L' \in C$ can be reduced to L.

Chapter 3

Graphs and Colorings in ASP

Graphs constitute a fundamental tool within computing science, in particular, in programming languages, where graphs are often used for analyzing a program's behavior. Clearly, this also applies to logic programming. For instance, Prolog's procedural semantics is intimately connected to the concept of SLD-trees [144]. For further analysis, like profiling, other types of graphs, such as call graphs, play an important role during program development. Similarly, in alternative semantics of logic programming, like *answer set programming* [99, 100], graphs have been used for deciding whether answer sets exist [94, 11].

We take the application of graphs even further and elaborate in this chapter upon an approach to using graphs as the underlying computational model for computing answer sets. To this end, we build upon and largely extend the theoretical foundations introduced in [139, 4]. Our approach has its roots in default logic [166], where extensions are often characterized through their (unique) set of generating default rules. Accordingly, we are interested in characterizing answer sets by means of their set of generating rules. For determining whether a rule belongs to this set, we must verify that each positive body atom is derivable and that no negative body atom is derivable. In fact, an atom is derivable if the set of generating rules includes a rule having the atom as its head; or conversely, an atom is not derivable if there is no rule among the generating rules that has the atom as its head. Consequently, the formation of the set of generating rules amounts to resolving positive and negative dependencies among rules. For capturing these dependencies, we take advantage of the concept of a *rule dependency graph*, wherein each node represents a rule of the underlying program and two types of edges stand for the aforementioned positive and negative rule dependencies, respectively. [1] For expressing the applicability status of rules, that is, whether a rule belongs to a set of generating rules or not, we label, or as we say *color*, the respective nodes in the graph. In this way, an answer set can be expressed by a total coloring of the rule dependency graph. Of course, in what follows, we are mainly interested in the inverse, that is, when does a graph coloring correspond to an answer set of the underlying program; and, in particular, how can we compute such a total coloring.

Generally speaking, graphs provide a formal device for making structural properties of an underlying problem explicit. In this guise, we start by identifying graph structures that capture structural properties of logic programs and their answer sets. As a result, we obtain several characterizations of answer sets in terms of totally colored dependency graphs that differ in graph-theoretical aspects. To a turn, we build upon these characterizations in order to develop an operational framework for answer set formation. The idea is to start from an uncolored rule dependency graph and to employ specific operators that turn a partially colored graph gradually into a totally colored one that represents an answer set. This approach is strongly inspired by the concept of a derivation, in particular, by that of an SLD-derivation [144]. Accordingly, a program has a certain answer set iff there is a sequence of operations turning the uncolored graph into a totally colored one that provably corresponds to the answer set.

[1] This type of graph was called "block graph" in [139].

This chapter is organized as follows. Section 3.1 lays the formal foundations of our approach by introducing its basic graph-theoretical instruments. While the following Section 3.2 addresses characterizations of answer sets through totally colored graphs, Section 3.3 deals with operational characterizations of answer sets. Section 3.4 identifies relationships with Fitting's and well-founded semantics. Section 3.5 discusses the approach, in particular in the light of related work. We conclude with Section 3.6.

3.1 Graphs and colorings

A *graph* is a pair (V, E) where V is a set of *vertices* and $E \subseteq V \times V$ a set of (directed) *edges*. A *path* from x to y in (V, E) for $x, y \in V$ is a sequence $x_1, \ldots, x_n$ such that $x = x_1, y = x_n$, $(x_i, x_{i+1}) \in E$ for $1 \leq i < n$, and the elements x_i are pairwise disjoint. A set of edges E contains a cycle if there is a nonempty set $\{x_i \mid i \in \{0, \ldots, n\}\}$ of vertices such that $(x_i, x_{i+1}) \in E$ for $i \in \{0, \ldots, n-1\}$ and $(x_n, x_0) \in E$. A graph (V, E) is *acyclic* if E contains no cycles. For $W \subseteq V$, we denote $E \cap (W \times W)$ by $E|_W$. Also, we abbreviate the induced subgraph $G = (V \cap W, E|_W)$ of (V, E) by $G|_W$. A *labeled graph* is a graph with an associated labeling function $\ell : E \to L$ for some set of labels L. In view of our small label set $L = \{0, 1\}$ (see below), we leave ℓ and L implicit and denote such labeled graphs by triples (V, E_0, E_1), where $E_i = \{e \in E \mid \ell(e) = i\}$ for $i = 0, 1$. An *i-subgraph* of (V, E_0, E_1) is a graph (W, F) such that $W \subseteq V$ and $F \subseteq E_i|_W$ for $i = 0, 1$. [2] An *i-path* from x to y in (V, E_0, E_1) is a path from x to y in (V, E_i) for $x, y \in V$ and $i = 0, 1$.

In the context of logic programming, we are interested in graphs reflecting dependencies among rules.

Definition 3.1.1 *Let Π be a logic program.*

The rule dependency graph (RDG) $\Gamma_\Pi = (\Pi, E_0, E_1)$ of Π is a labeled graph with

$$\begin{aligned} E_0 &= \left\{(r, r') \mid r, r' \in \Pi, head(r) \in body^+(r')\right\}; \\ E_1 &= \left\{(r, r') \mid r, r' \in \Pi, head(r) \in body^-(r')\right\}. \end{aligned}$$

We omit the subscript Π from Γ_Π whenever the underlying program is clear from the context. We follow [155] in distinguishing between 0- and 1-edges. Observe that several programs may have isomorphic *RDG*s. For example, $\Pi = \{a \leftarrow b, c \leftarrow a\}$ and $\Pi' = \{a \leftarrow, c \leftarrow a\}$ have isomorphic *RDG*s.

Example 9 *Consider the logic program $\Pi_9 = \{r_1, \ldots, r_6\}$, comprising the following rules:*

$$\begin{array}{lrcl} r_1: & p & \leftarrow & \\ r_2: & b & \leftarrow & p \\ r_3: & f & \leftarrow & b, not\ f' \\ r_4: & f' & \leftarrow & p, not\ f \\ r_5: & b & \leftarrow & m \\ r_6: & x & \leftarrow & f, f', not\ x \end{array}$$

The RDG of Π_9 is given as follows:

$$\Gamma_{\Pi_9} = (\ \Pi_9,\ \{(r_1, r_2), (r_1, r_4), (r_2, r_3), (r_3, r_6), (r_4, r_6), (r_5, r_3)\},\ \{(r_3, r_4), (r_4, r_3), (r_6, r_6)\}\ )$$

It is depicted graphically in Figure 3.1a. For instance, $(\{r_1, r_2, r_3, r_4\}, \{(r_1, r_2)\})$ is a 0-subgraph of Γ_{Π_9} and $(\{r_5, r_6\}, \{(r_6, r_6)\})$ is a 1-subgraph of Γ_{Π_9}.

We call C a *(partial) coloring* of Γ_Π if C is a partial mapping $C : \Pi \to \{\oplus, \ominus\}$. We call C a *total* coloring, if C is a total mapping. Intuitively, the colors $\oplus$ and $\ominus$ indicate whether a rule is supposedly

[2] Note that an i-subgraph is not an induced graph.

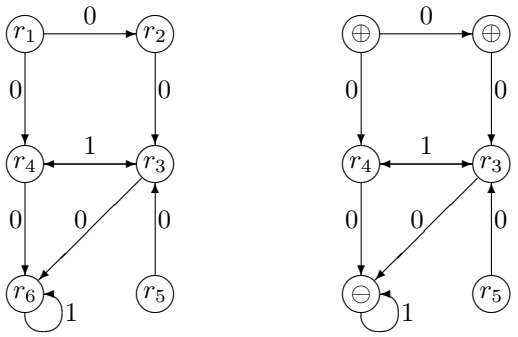

Figure 3.1: (a) *RDG* of logic program Π_9 (b) (partially) colored *RDG* $(\Gamma_{\Pi_9}, C_{3.1})$.

applied or blocked, respectively. We sometimes denote the set of all vertices colored with $\oplus$ or $\ominus$ by $C_\oplus$ or $C_\ominus$, respectively. That is, $C_\oplus = \{r \mid C(r) = \oplus\}$ and $C_\ominus = \{r \mid C(r) = \ominus\}$. If C is total, $(C_\oplus, C_\ominus)$ is a binary partition of Π. That is, $\Pi = C_\oplus \cup C_\ominus$ and $C_\oplus \cap C_\ominus = \emptyset$. Accordingly, we often identify a coloring C with the pair $(C_\oplus, C_\ominus)$. A partial coloring C induces a pair $(C_\oplus, C_\ominus)$ of sets such that $C_\oplus \cup C_\ominus \subseteq \Pi$ and $C_\oplus \cap C_\ominus = \emptyset$. For comparing partial colorings, C and C', we define $C \sqsubseteq C'$, if $C_\oplus \subseteq C'_\oplus$ and $C_\ominus \subseteq C'_\ominus$. The "empty" coloring $(\emptyset, \emptyset)$ is the $\sqsubseteq$-smallest coloring. Accordingly, we define $C \sqcup C'$ as $(C_\oplus \cup C'_\oplus, C_\ominus \cup C'_\ominus)$.[3] We denote the set of all partial colorings of a *RDG* Γ_Π by $\mathbb{C}_{\Gamma_\Pi}$. For readability, we often omit the index Γ_Π and simply write $\mathbb{C}$, whenever this is clear from the context.

If C is a coloring of Γ_Π, we call the pair (Γ_Π, C) a *colored RDG*. For example, "coloring" the *RDG* of Π_9 from Example 9 with [4]

$$C_{3.1} = (\{r_1, r_2\},\ \{r_6\}) \tag{3.1}$$

yields the colored graph given in Figure 3.1b. For simplicity, when coloring, we replace the label of a node by the respective color. Observe that our conception of coloring is nonstandard insofar that adjacent vertices may be colored with the same color. We are sometimes interested in the subgraph $\Gamma_\Pi|_{C_\oplus \cup C_\ominus}$ induced by the colored nodes. Restricting Γ_{Π_9} to the nodes colored in Figure 3.1b, yields the *RDG* $(\{r_1, r_2, r_6\}, \{(r_1, r_2)\}, \{(r_6, r_6)\})$.

The central question addressed in this chapter is how to characterize and compute the total colorings of *RDG*s that correspond to the answer sets of an underlying program. In fact, the colorings of interest can be distinguished in a straightforward way. Let Π be a logic program along with its *RDG* Γ. Then, for every answer set X of Π, define an *admissible coloring*[5] C of Γ as

$$C = (R_\Pi(X), \Pi \setminus R_\Pi(X))\ . \tag{3.2}$$

By way of the respective generating rules, we associate with any program a set of admissible colorings whose members are in one-to-one correspondence with its answer sets. Any admissible coloring is total; furthermore, we have $X = head(C_\oplus)$. We use $AC(\Pi)$ for denoting the set of all admissible colorings of a *RDG* Γ_Π.

For a partial coloring C, we define $AC_\Pi(C)$ as the set of all admissible colorings of Γ_Π compatible with C. Formally, given the *RDG* Γ of a logic program Π and a partial coloring C of Γ, define

$$AC_\Pi(C) = \{C' \in AC(\Pi) \mid C \sqsubseteq C'\}\ . \tag{3.3}$$

[3] We use "squared" relation symbols, like $\sqsubseteq$ or $\sqcup$ when dealing with partial colorings.

[4] For the sake of uniqueness, we label the coloring with the equation number.

[5] The term "admissible coloring" was coined in [39]; they were referred to as "application colorings" in [139]. Note that both colorings concepts are originally defined in purely graph-theoretical terms. Here, we simply adopt this term for distinguishing colorings corresponding to answer sets of an underlying program (cf. Section 3.5).

Clearly, $C_1 \sqsubseteq C_2$ implies $AC_\Pi(C_1) \supseteq AC_\Pi(C_2)$. Observe also that a partial coloring C is extensible to an admissible one C', that is, $C \sqsubseteq C'$, iff $AC_\Pi(C)$ is non-empty. For a total coloring C, $AC_\Pi(C)$ is either empty or singleton. Regarding program Π_9 and coloring $C_{3.1}$, we get $AC_{\Pi_9}(C_{3.1}) = AC(\Pi_9) = \{(\{r_1, r_2, r_3\}, \{r_4, r_5, r_6\}), (\{r_1, r_2, r_4\}, \{r_3, r_5, r_6\})\}$ (see also Figure 3.3 below).

Accordingly, we define $AS_\Pi(C)$ as the set of all answer sets X of Π compatible with partial coloring C.

$$AS_\Pi(C) = \{X \in AS(\Pi) \mid C_\oplus \subseteq R_\Pi(X) \text{ and } C_\ominus \cap R_\Pi(X) = \emptyset\}. \tag{3.4}$$

Note that $head(C_\oplus) \subseteq X$ for any answer set $X \in AS_\Pi(C)$ (see Theorem A.1.4 on page 142). Otherwise, similar considerations apply to $AS_\Pi(C)$ as made above for $AC_\Pi(C)$. As regards program Π_9 and coloring $C_{3.1}$, we get $AS_{\Pi_9}(C_{3.1}) = AS(\Pi_9) = \{\{b, p, f\}, \{b, p, f'\}\}$. It is noteworthy that due to the one-to-one correspondence between $AS_\Pi(C)$ and $AC_\Pi(C)$ (see Theorem A.1.1 on page 141), one can replace one by the other in most subsequent results. Often it is simply a matter of simplicity which formulation is used.

We need the following concepts for describing a rule's status of applicability in a colored *RDG*.

Definition 3.1.2 *Let $\Gamma = (\Pi, E_0, E_1)$ be the RDG of logic program Π and C be a partial coloring of Γ. For $r \in \Pi$, we define:*

1. *r is supported in (Γ, C), if $body^+(r) \subseteq \{head(r') \mid (r', r) \in E_0, r' \in C_\oplus\}$;*
2. *r is unsupported in (Γ, C), if $\{r' \mid (r', r) \in E_0, head(r') = q\} \subseteq C_\ominus$ for some $q \in body^+(r)$;*
3. *r is blocked in (Γ, C), if $r' \in C_\oplus$ for some $(r', r) \in E_1$;*
4. *r is unblocked in (Γ, C), if $r' \in C_\ominus$ for all $(r', r) \in E_1$.*

Conditions 1–4 express standard concepts of logic programming. Condition 1 and 2 tell us whether the atoms in $body^+(r)$ are established within C or not; Condition 3 and 4 do the same for $body^-(r)$. Since C is partial, there are usually rules whose status of support and blockage is (yet) undecided. Note that support- and unsupportedness are only complementary if C is total; likewise with blockage. For r and r' as given in Condition 3, we say that r is blocked by r'. Whenever C is total, a rule is unsupported or unblocked iff it is not supported or not blocked, respectively. Note that the qualification $(r', r) \in E_0$ could be safely removed from Condition 1 and 2; we left it in for stressing the symmetry among the first two and the last two conditions. Observe that all four properties are decidable by looking at the immediate predecessors in the graph. With a slightly extended graph structure, they can be expressed in purely graph-theoretical terms, without any reference to the heads and bodies of the underlying rules.[6]

We use $S(\Gamma, C), \overline{S}(\Gamma, C), B(\Gamma, C)$, and $\overline{B}(\Gamma, C)$ for denoting the sets of all supported, unsupported, blocked, and unblocked rules in (Γ, C). For a total coloring C, we have $\overline{S}(\Gamma, C) = \Pi \setminus S(\Gamma, C)$ and $\overline{B}(\Gamma, C) = \Pi \setminus B(\Gamma, C)$. Furthermore, $S(\Gamma, C)$ and $\overline{S}(\Gamma, C)$ as well as $B(\Gamma, C)$ and $\overline{B}(\Gamma, C)$, respectively, are disjoint.

For illustration, consider the sets obtained regarding the colored *RDG* $(\Gamma_{\Pi_9}, C_{3.1})$, given in Figure 3.1b.

$$\begin{array}{lcllcl} S(\Gamma_{\Pi_9}, C_{3.1}) & = & \{r_1, r_2, r_3, r_4\} & \overline{S}(\Gamma_{\Pi_9}, C_{3.1}) & = & \{r_5\} \\ B(\Gamma_{\Pi_9}, C_{3.1}) & = & \emptyset & \overline{B}(\Gamma_{\Pi_9}, C_{3.1}) & = & \{r_1, r_2, r_5, r_6\} \end{array} \tag{3.5}$$

The following theorem shows the correspondence between properties of rules in a logic program and properties of vertices of a *RDG*, in the presence of an existing answer set.

Theorem 3.1.1 *Let Γ be the RDG of logic program Π, C be a partial coloring of Γ and $X \in AS_\Pi(C)$. For $r \in \Pi$, we have*

[6] For details on these pure graph-theoretical characterization, we refer the reader to a companion paper [141], dealing with the system `noMoRe`.

1. $body^+(r) \subseteq X$, *if* $r \in S(\Gamma, C)$;
2. $body^+(r) \not\subseteq X$, *if* $r \in \overline{S}(\Gamma, C)$;
3. $body^-(r) \cap X \neq \emptyset$, *if* $r \in B(\Gamma, C)$;
4. $body^-(r) \cap X = \emptyset$, *if* $r \in \overline{B}(\Gamma, C)$.

For admissible colorings, we may turn the above "if" statements into "iff".

Corollary 3.1.2 *Let* Γ *be the RDG of logic program* Π, C *be an admissible coloring of* Γ *and* $\{X\} = AS_\Pi(C)$.
For $r \in \Pi$, *we have*

1. $body^+(r) \subseteq X$ *iff* $r \in S(\Gamma, C)$;
2. $body^+(r) \not\subseteq X$ *iff* $r \in \overline{S}(\Gamma, C)$;
3. $body^-(r) \cap X \neq \emptyset$ *iff* $r \in B(\Gamma, C)$;
4. $body^-(r) \cap X = \emptyset$ *iff* $r \in \overline{B}(\Gamma, C)$.

The next results are important for understanding the idea of our approach.

Theorem 3.1.3 *Let* Γ *be the RDG of logic program* Π *and* C *be a partial coloring of* Γ.
Then, for every $X \in AS_\Pi(C)$ *we have that*

1. $S(\Gamma, C) \cap \overline{B}(\Gamma, C) \subseteq R_\Pi(X)$;
2. $\overline{S}(\Gamma, C) \cup B(\Gamma, C) \subseteq \Pi \setminus R_\Pi(X)$.

If C *is admissible, we have for* $\{X\} = AS_\Pi(C)$ *that*

3. $S(\Gamma, C) \cap \overline{B}(\Gamma, C) = R_\Pi(X)$;
4. $\overline{S}(\Gamma, C) \cup B(\Gamma, C) = \Pi \setminus R_\Pi(X)$.

In fact, the last two equations are equivalent since C is total. Each of them can be understood as a necessary yet insufficient condition for characterizing answer sets. We elaborate upon sufficient graph-theoretical conditions in the next section.

Let us reconsider the partially colored RDG $(\Gamma_{\Pi_9}, C_{3.1})$ in Figure 3.1b. For every $X \in AS_{\Pi_9}(C_{3.1}) = \{\{b, p, f\}, \{b, p, f'\}\}$, we have

$$\begin{array}{lcl} S(\Gamma_{\Pi_9}, C_{3.1}) \cap \overline{B}(\Gamma_{\Pi_9}, C_{3.1}) & = & \{r_1, r_2, r_3, r_4\} \cap \{r_1, r_2, r_5, r_6\} \\ & = & \{r_1, r_2\} \\ & \subseteq & R_{\Pi_9}(X); \\ \overline{S}(\Gamma_{\Pi_9}, C_{3.1}) \cup B(\Gamma_{\Pi_9}, C_{3.1}) & = & \{r_5\} \cup \emptyset \\ & = & \{r_5\} \\ & \subseteq & \Pi \setminus R_{\Pi_9}(X). \end{array}$$

Regarding $\Pi = \{a \leftarrow not\ a\}$, it is instructive to observe the instance of Condition 3 in Theorem 3.1.3 for total coloring $C = (\{a \leftarrow not\ a\}, \emptyset)$ and set $X = \{a\}$:

$$S(\Gamma_\Pi, C) \cap \overline{B}(\Gamma_\Pi, C) = \{a \leftarrow not\ a\} \cap \emptyset = R_\Pi(X)\ .$$

This demonstrates that Condition 3 is insufficient for characterizing answer sets. In fact, observe that $C_\oplus \neq S(\Gamma_\Pi, C) \cap \overline{B}(\Gamma_\Pi, C)$.

3.2 Deciding answersetship from colored graphs

The result given in Theorem 3.1.3 started from an existing answer set induced from a given coloring. We now develop concepts that allow us to decide whether a (total) coloring represents an answer set by purely graph-theoretical means.

3.2.1 Graph-based characterization

To begin with, we define a graph structure accounting for the notion of recursive support.

Definition 3.2.1 *Let Γ be the RDG of logic program Π.*

We define a support graph *of Γ as an acyclic 0-subgraph (V, E) of Γ such that $body^+(r) \subseteq \{head(r') \mid (r', r) \in E\}$ for all $r \in V$.*

Intuitively, support graphs constitute the graph-theoretical counterpart of operator *Cn*. That is, a support graph comprises dependencies among heads and positive bodies on which potential applications of the T_Π operator rely.

Example 10 *Consider program Π_{10} consisting of rules*

$$\begin{array}{lll} r_1: & a & \leftarrow \\ r_2: & b & \leftarrow not\ a \\ r_3: & c & \leftarrow b \\ r_4: & b & \leftarrow c\,. \end{array}$$

Among others, the RDG of Π_{10} has support graphs

$$(\emptyset, \emptyset), (\{r_1, r_2\}, \emptyset), (\{r_2, r_3\}, \{(r_2, r_3)\}), \text{ and } (\Pi_{10}, \{(r_2, r_3), (r_3, r_4)\}).$$

Observe that the empty graph $(\emptyset, \emptyset)$ is a support graph of any (uncolored) graph. Self-supportedness is avoided due to the acyclicity of support graphs. $(\Pi_{10}, \{(r_4, r_3), (r_3, r_4)\})$ is cyclic and hence no support graph.

Every *RDG* has a unique support graph possessing a largest set of vertices.

Theorem 3.2.1 *Let Γ be the RDG of logic program Π.*

Then, there exists a support graph (V, E) of Γ such that $V' \subseteq V$ for all support graphs (V', E') of Γ.

For simplicity, we refer to such support graphs as *maximal support graphs*; all of them share the same set of vertices. This set of vertices corresponds to the generating rules of $Cn(\Pi^\emptyset)$. Different maximal support graphs comprise different sets of edges, reflecting the intuition that atoms may be derivable in different ways. Given that the empty graph is a support graph of any (uncolored) graph, there is always a maximal support graph.

For example, the maximal support graph of the *RDG* of logic program Π_9, given in Figure 3.1a, is depicted in Figure 3.2a. The latter contains except for (r_5, r_3) all 0-edges of the former, viz $(\Gamma_{\Pi_9}, C_{3.1})$; also r_5 is excluded since it cannot be supported (recursively). The only maximal support graph of $\Gamma_{\Pi_{10}}$ is $(\Pi_{10}, \{(r_2, r_3), (r_3, r_4)\})$ (cf. Example 10). Extending Π_{10} by $r_5 : b \leftarrow$ yields three maximal support graphs

$$\begin{array}{l} (\Pi_{10} \cup \{r_5\}, \{(r_2, r_3), (r_3, r_4)\}), \\ (\Pi_{10} \cup \{r_5\}, \{(r_5, r_3), (r_3, r_4)\}), \text{ and} \\ (\Pi_{10} \cup \{r_5\}, \{(r_2, r_3), (r_5, r_3), (r_3, r_4)\}). \end{array}$$

All of them share the same set of vertices but differ in the set of edges.

The concept of a support graph is extended to colored graphs in the following way.

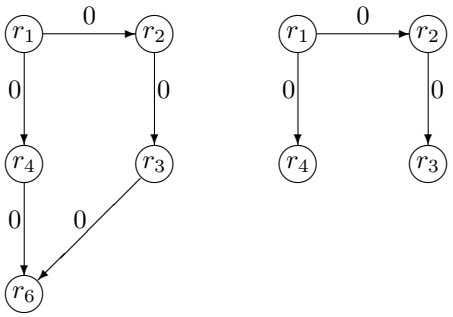

Figure 3.2: The maximal support graph of (a) Γ_{Π_9} (b) $(\Gamma_{\Pi_9}, C_{3.1})$.

Definition 3.2.2 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*

We define a support graph *of (Γ, C) as a support graph (V, E) of Γ such that $C_\oplus \subseteq V$ and $C_\ominus \cap V = \emptyset$ for some $E \subseteq (\Pi \times \Pi)$.*

Recall that E consists of 0-arcs only. Also, note that Definition 3.2.1 and 3.2.2 coincide whenever C is the empty coloring. In general, the support graphs of (Γ, C) are exactly those support graphs of Γ whose vertex set includes $C_\oplus$ and excludes $C_\ominus$. Intuitively, a support graph of a *colored RDG* (Γ, C) takes the applicability status of the rules expressed by C into account. That is, it contains all rules whose positive body is derivable, given that all rules in $C_\oplus$ are applicable and all rules in $C_\ominus$ are inapplicable.

For example, the maximal support graph of the colored *RDG* $(\Gamma_{\Pi_9}, C_{3.1})$, given in Figure 3.1b, is depicted in Figure 3.2b. The latter must include all positively colored and exclude all negatively colored nodes of the former. Given program Π_{10} from Example 10, a "bad" coloring, like $C = (\{b \leftarrow c\}, \{b \leftarrow not\ a\})$, may deny the existence of a support graph of (Γ, C).

Given an arbitrary coloring C, there is a priori no relationship among the set of rules supported by a colored graph, viz $S(\Gamma, C)$, and its support graphs. To see this, consider the support graph $\Gamma_{\Pi_{10}}$ of program Π_{10} along with coloring $C = (\{r_1, r_3, r_4\}, \{r_2\})$. While we have $S(\Gamma_{\Pi_{10}}, C) = \Pi_{10}$, there is no support graph of $(\Gamma_{\Pi_{10}}, C)$. For one thing, a support graph is denied since r_3 and r_4 form a circular support. For another thing, r_2 is always supported, no matter which coloring is considered, whereas it cannot belong to a support graph once it is blocked (i.e. colored with $\ominus$). This illustrates two things. First, support graphs provide a global, recursive structure tracing the support of a rule over 0-paths back to rules with empty positive bodies. Unlike this, $S(\Gamma, C)$ relies only on 0-edges, capturing thus a rather local notion of support. Second, support graphs take the complete applicability status expressed by a coloring into account. Unlike this, $S(\Gamma, C)$ may contain putatively blocked rules from $C_\ominus$.

As above, we distinguish maximal support graphs of colored graphs through their maximal set of vertices.

For colored graphs, we have the following conditions guaranteeing the existence of (maximal) support graphs.

Theorem 3.2.2 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*

Then, there is a (maximal) support graph of (Γ, C), if one of the following conditions holds.

1. *$AC_\Pi(C) \neq \emptyset$;*
2. *$(C_\oplus, E)$ is a support graph of $\Gamma|_{C_\oplus \cup C_\ominus}$ for some $E \subseteq (\Pi \times \Pi)$.*

The existence of a support graph implies the existence of a maximal one. This is why we put maximal in parentheses in the preamble of Theorem 3.2.2.

As with Property 3 or 4 in Theorem 3.1.3, respectively, the existence of a support graph can be understood as a necessary condition for characterizing answer sets:

Corollary 3.2.3 *Let Γ be the RDG of logic program Π and C be an admissible coloring of Γ.*
Then, $(C_\oplus, E)$ is a support graph of (Γ, C) for some $E \subseteq (\Pi \times \Pi)$.

In fact, taken the last result together with Property 3 or 4 in Theorem 3.1.3, respectively, we obtain a sufficient characterization of admissible colorings (along with their underlying answer sets).

Theorem 3.2.4 (Answer set characterization, I) *Let Γ be the RDG of logic program Π and let C be a total coloring of Γ.*
Then, the following statements are equivalent.

1. *C is an admissible coloring of Γ;*
2. *$C_\oplus = S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ and there is a support graph of (Γ, C);*
3. *$C_\ominus = \overline{S}(\Gamma, C) \cup B(\Gamma, C)$ and there is a support graph of (Γ, C).*

Interestingly, this characterization shows that once we have established a support graph, it doesn't matter whether we focus exclusively on the applicable (as in Statement 2.) or on the inapplicable rules (as in 3.) for characterizing admissible colorings. In both cases, $C_\oplus$ provides the vertices of the maximal support graphs.

For illustration, let us consider the two admissible colorings of *RDG* Γ_{Π_9}, corresponding to the two answer sets of program Π_9 (given in Example 9):

$$C_{3.6} = (\{r_1, r_2, r_3\},\ \{r_4, r_5, r_6\})\ ; \tag{3.6}$$

$$C_{3.7} = (\{r_1, r_2, r_4\},\ \{r_3, r_5, r_6\}). \tag{3.7}$$

The resulting colored *RDG*s are depicted in Figure 3.3. (Cf. Figure 3.1a for the underlying uncolored graph.)

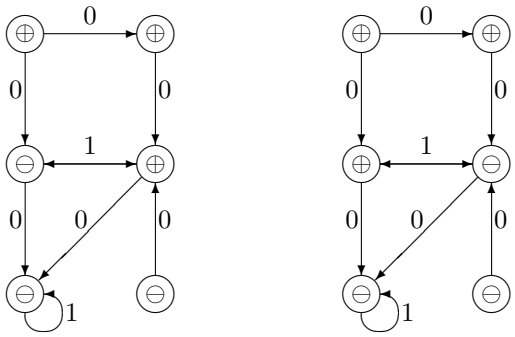

Figure 3.3: The totally colored *RDG*s $(\Gamma_{\Pi_9}, C_{3.6})$ and $(\Gamma_{\Pi_9}, C_{3.7})$.

Let us detail the case of $C_{3.6}$. We get:

$$\begin{aligned} S(\Gamma_{\Pi_9}, C_{3.6}) \cap \overline{B}(\Gamma_{\Pi_9}, C_{3.6}) &= \{r_1, r_2, r_3, r_4\} \cap \{r_1, r_2, r_3, r_5, r_6\} \\ &= \{r_1, r_2, r_3\} \\ &= (C_{3.6})_\oplus; \\ \overline{S}(\Gamma_{\Pi_9}, C_{3.6}) \cup B(\Gamma_{\Pi_9}, C_{3.6}) &= \{r_5, r_6\} \cup \{r_4\} \\ &= \{r_4, r_5, r_6\} \\ &= (C_{3.6})_\ominus. \end{aligned}$$

The maximal support graph of $(\Gamma_{\Pi_9}, C_{3.6})$ is given by $((C_{3.6})_\oplus, \{(r_1, r_2), (r_2, r_3)\})$; it is depicted below in Figure 3.4.

3.2.2 Capturing original concepts

It is interesting to see how the original definition of an answer set X, that is, $X = Cn(\Pi^X)$, along with its underling constructions, viz reduction Π^X and the Cn operator, can be captured within our graph-based setting.

Clearly, Π^X amounts to the set of unblocked rules.

Theorem 3.2.5 *Let Γ be the RDG of logic program Π.*
Furthermore, let C be a total coloring of Γ and X be a set of atoms such that $X = head(C_\oplus)$.
Then, $head(r) \leftarrow body^+(r) \in \Pi^X$ iff $r \in \overline{B}(\Gamma, C)$ for $r \in \Pi$.

The next result fixes the relationship of maximal support graphs to the consequence operator of basic programs.

Theorem 3.2.6 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*
If (V, E) is a maximal support graph of (Γ, C), then $head(V) = Cn((\Pi \setminus C_\ominus)^\emptyset)$.

Hence, the vertices of the maximal support graph correspond directly to the consequences w.r.t. the given partial coloring. For the "empty" coloring $C = (\emptyset, \emptyset)$, we have $head(V) = Cn(\Pi^\emptyset)$.

Taking the graph-theoretical counterparts of the reduct Π^X and the Cn operator yields the following graph-theoretical characterization of answer sets:

Theorem 3.2.7 (Answer set characterization, II) *Let Γ be the RDG of logic program Π and let C be a total coloring of Γ.*
Then, C is an admissible coloring of Γ iff $(C_\oplus, E)$ is a maximal support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$ for some $E \subseteq (\Pi \times \Pi)$.

Recall that $\Gamma|_{\overline{B}(\Gamma,C)}$ is the restriction of Γ to the set of unblocked rules in (Γ, C).[7] In view of Theorem 3.2.5, the graph $\Gamma|_{\overline{B}(\Gamma,C)}$ amounts to a reduced (basic) program Π^X whose closure $Cn(\Pi^X)$ is characterized by means of a maximal support graph (cf. Theorem 3.2.6).

3.2.3 Subgraph-based characterization

Using "unblocked" rules as done in the previous characterization, refers only implicitly to the blockage relations expressed by 1-edges. In analogy to Definition 3.2.1, this structure can be made explicit by the notion of a *blockage graph*. For this, we use π_2 for projecting the second argument of a relation. [8]

Definition 3.2.3 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*
We define a blockage graph *of (Γ, C) as a 1-subgraph (V, E) of Γ such that*

1. $E \cap (C_\oplus \times C_\oplus) = \emptyset$;
2. $\pi_2(E \cap (C_\oplus \times C_\ominus)) = C_\ominus \cap V$.

In other words, the first condition says that there is no $(r, r') \in E$ such that $r, r' \in C_\oplus$, while the second one stipulates that for all $r \in C_\ominus \cap V$ there exists an $r' \in C_\oplus$ such that $(r', r) \in E$.

Let us briefly compare the definitions of support and blockage graphs. Support graphs capture a recursive concept, stipulating that 0-edges are contained in 0-paths, tracing the support of rules back to rules with empty positive body. Unlike this, blockage graphs aim at characterizing rather local dependencies, based on 1-edge-wise constraints. That is, while the acyclicity of a support graph cannot be checked locally, we may verify whether a graph is a blockage graph by inspecting one of its 1-edges after the other.

Together, both concepts provide the following characterization of answer sets.

[7] In fact, $\Gamma|_{\overline{B}(\Gamma,C)}$ could be replaced in Theorem 3.2.7 by $\Gamma|_{S(\Gamma,C)\cap\overline{B}(\Gamma,C)}$ without changing its validity.
[8] That is, $\pi_2(R) = \{r_2 \mid (r_1, r_2) \in R\}$ for a binary relation R.

Theorem 3.2.8 (Answer set characterization, III) *Let $\Gamma = (\Pi, E_0, E_1)$ be the RDG of logic program Π and let C be a total coloring of Γ.*

Then, C is an admissible coloring of Γ iff

1. *there is some support graph of (Γ, C) and*
2. *$(S(\Gamma, C), E_1|_{S(\Gamma,C)})$ is a blockage graph of (Γ, C).*

Observe that $(C_\oplus, E)$ is a support graph of (Γ, C) for some $E \subseteq \Pi \times \Pi$. Condition 2 stipulates, among other things, that all supported yet inapplicable rules are properly blocked (cf. Condition 2 in Definition 3.2.3). The restriction to supported rules is necessary in order to eliminate rules that are inapplicable since they are unsupported. Note that the blockage graph in Condition 2 can also be written as $(C, E_1)|_{S(\Gamma,C)}$. For example, the support and the blockage graph of the colored RDG $(\Gamma_{\Pi_9}, C_{3.6})$, given on the left hand side in Figure 3.3, are depicted in Figure 3.4. This figure nicely illustrates the subset relationship between the

Figure 3.4: Support and blockage graph of $(\Gamma_{\Pi_9}, C_{3.6})$.

vertices of the support and the blockage graph.

Without explicit mention of the blockage graph, a similar characterization can be given in the following way.

Corollary 3.2.9 (Answer set characterization, III′) *Let $\Gamma = (\Pi, E_0, E_1)$ be the RDG of logic program Π and let C be a total coloring of Γ.*

Then, C is an admissible coloring of Γ iff the following conditions hold.

1. *$(C_\oplus, E)$ is a support graph of (Γ, C) for some $E \subseteq \Pi \times \Pi$;*
2. *for all $r \in (C_\ominus \cap S(\Gamma, C))$ there exists an $r' \in C_\oplus$ such that $(r', r) \in E_1$;*
3. *for all $r, r' \in C_\oplus$ we have $(r, r') \notin E_1$.*

3.3 Operational characterizations

The goal of this section is to provide an operational characterization of answer sets, based on the concepts introduced in the last section. The idea is to start with the "empty" coloring $(\emptyset, \emptyset)$ and to successively apply operators that turn a partial coloring C into another one C' such that $C \sqsubseteq C'$. This is done until finally an admissible coloring, yielding an answer set, is obtained.

3.3.1 Deterministic operators

We concentrate first on operations extending partial colorings in a deterministic way.

Definition 3.3.1 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*

Then, define $\mathcal{P}_\Gamma : \mathbb{C} \to \mathbb{C}$ as $\mathcal{P}_\Gamma(C) = C \sqcup (S(\Gamma, C) \cap \overline{B}(\Gamma, C),\ \overline{S}(\Gamma, C) \cup B(\Gamma, C))$.

A partial coloring C is closed under $\mathcal{P}_\Gamma$, if $C = \mathcal{P}_\Gamma(C)$. Note that $\mathcal{P}_\Gamma(C)$ does not always exist. To see this, observe that $\mathcal{P}_\Gamma((\{a \leftarrow not\ a\}, \emptyset))$ would be $(\{a \leftarrow not\ a\}, \{a \leftarrow not\ a\})$, which is no mapping and thus no partial coloring. Interestingly, $\mathcal{P}_\Gamma$ exists on colorings expressing answer sets.

Theorem 3.3.1 *Let Γ be the RDG of logic program Π and C a partial coloring of Γ.*
If $AC_\Pi(C) \neq \emptyset$, then $\mathcal{P}_\Gamma(C)$ exists.

Note that $\mathcal{P}_\Gamma(C)$ may exist although $AC_\Pi(C) = \emptyset$. [9] To see this, consider the program $\Pi = \{a \leftarrow, c \leftarrow a, not\ c\}$. Clearly, $AS(\Pi) = \emptyset$. However, $\mathcal{P}_\Gamma((\emptyset, \emptyset)) = (\{a \leftarrow\}, \emptyset)$ exists.

Now, we can define our principal propagation operator in the following way.

Definition 3.3.2 *Let Γ be the RDG of logic program Π and C a partial coloring of Γ.*
Then, define $\mathcal{P}^*_\Gamma : \mathbb{C} \to \mathbb{C}$ where $\mathcal{P}^*_\Gamma(C)$ is the $\sqsubseteq$-smallest partial coloring containing C and being closed under $\mathcal{P}_\Gamma$.

Essentially, $\mathcal{P}^*_\Gamma(C)$ amounts to computing the "immediate consequences" from a given partial coloring C. [10]

Also, like $\mathcal{P}_\Gamma(C)$, $\mathcal{P}^*_\Gamma(C)$ is not necessarily defined. This situation is made precise next.

Theorem 3.3.2 *Let Γ be the RDG of logic program Π and C a partial coloring of Γ.*
If $AC_\Pi(C) \neq \emptyset$, then $\mathcal{P}^*_\Gamma(C)$ exists.

Under the previous conditions, we may actually characterize $\mathcal{P}^*_\Gamma$ in terms of iterated applications of $\mathcal{P}_\Gamma$; this is detailed in Section A.2 and used in the proofs. In fact, the non-existence of $\mathcal{P}^*_\Gamma$ is an important feature since an undefined application of $\mathcal{P}^*_\Gamma$ amounts to a backtracking situation at the implementation level. Note that $\mathcal{P}^*_\Gamma((\emptyset, \emptyset))$ always exists, even though we may have $AC_\Pi((\emptyset, \emptyset)) = \emptyset$ (because of $AS(\Pi) = \emptyset$).

We have the following result.

Corollary 3.3.3 *Let Γ be the RDG of logic program Π. Then, $\mathcal{P}^*_\Gamma((\emptyset, \emptyset))$ exists.*

For illustration, consider program Π_9 in Example 9. We get:

$$\begin{aligned}
\mathcal{P}_\Gamma((\emptyset,\emptyset)) &= (\emptyset,\emptyset) \sqcup (\{r_1\} \cap \{r_1, r_2, r_5\}, \{r_5\} \cup \emptyset) \\
&= (\{r_1\}, \{r_5\}) \\
\mathcal{P}_\Gamma((\{r_1\}, \{r_5\})) &= (\{r_1\}, \{r_5\}) \sqcup (\{r_1, r_2, r_4\} \cap \{r_1, r_2, r_5\}, \{r_5\} \cup \emptyset) \\
&= (\{r_1, r_2\}, \{r_5\}) \\
\mathcal{P}_\Gamma((\{r_1, r_2\}, \{r_5\})) &= (\{r_1, r_2\}, \{r_5\}) \sqcup (\{r_1, r_2, r_3, r_4\} \cap \{r_1, r_2, r_5\}, \{r_5\} \cup \emptyset) \\
&= (\{r_1, r_2\}, \{r_5\})
\end{aligned}$$

Hence, we obtain

$$\mathcal{P}^*_\Gamma((\emptyset,\emptyset)) = (\{r_1, r_2\}, \{r_5\})\ .$$

Let us now elaborate more upon the formal properties of $\mathcal{P}_\Gamma$ and $\mathcal{P}^*_\Gamma$. First, we observe that both are reflexive, that is, $C \sqsubseteq \mathcal{P}_\Gamma(C)$ and $C \sqsubseteq \mathcal{P}^*_\Gamma(C)$ (provided that $\mathcal{P}_\Gamma(C)$ and $\mathcal{P}^*_\Gamma(C)$ exists). Furthermore, both operators are monotonic in the following sense.

Theorem 3.3.4 *Let Γ be the RDG of logic program Π and let C and C' be partial colorings of Γ such that $AC_\Pi(C') \neq \emptyset$.*

[9] Since our goal is to compute members of $AC_\Gamma(C)$, the precondition $AC_\Pi(C) \neq \emptyset$ is sufficient for our purposes. It remains future work to identify a necessary *and* sufficient condition guaranteeing the existence of $\mathcal{P}_\Gamma(C)$.

[10] In Section 3.4, this is related to Fitting's semantics.

1. *If* $C \sqsubseteq C'$, *then* $\mathcal{P}_\Gamma(C) \sqsubseteq \mathcal{P}_\Gamma(C')$;
2. *If* $C \sqsubseteq C'$, *then* $\mathcal{P}^*_\Gamma(C) \sqsubseteq \mathcal{P}^*_\Gamma(C')$.

Given that $\mathcal{P}_\Gamma$ is reflexive, the last result implies that $C \sqsubseteq \mathcal{P}_\Gamma(C) \sqsubseteq \mathcal{P}_\Gamma(\mathcal{P}_\Gamma(C))$ whenever $AC_\Pi(C) \neq \emptyset$. In addition, $\mathcal{P}^*_\Gamma$ clearly enjoys a restricted idempotency property, that is, $\mathcal{P}^*_\Gamma(C) = \mathcal{P}^*_\Gamma(\mathcal{P}^*_\Gamma(C))$ provided that $AC_\Pi(C) \neq \emptyset$.

Our next result shows that $\mathcal{P}_\Gamma$ and $\mathcal{P}^*_\Gamma$ are answer set preserving.

Theorem 3.3.5 *Let* Γ *be the RDG of logic program* Π *and* C *be a partial coloring of* Γ.
Then, we have

1. $AC_\Pi(C) = AC_\Pi(\mathcal{P}_\Gamma(C))$;
2. $AC_\Pi(C) = AC_\Pi(\mathcal{P}^*_\Gamma(C))$.

That is, $X \in AS_\Pi(C)$ iff $X \in AS_\Pi(\mathcal{P}_\Gamma(C))$ iff $X \in AS_\Pi(\mathcal{P}^*_\Gamma(C))$.

A similar result holds for the underlying support and blockage graphs.

Theorem 3.3.6 *Let* $\Gamma = (\Pi, E_0, E_1)$ *be the RDG of logic program* Π *and* C *be a partial coloring of* Γ.
For $C' = \mathcal{P}^*_\Gamma(C)$, *we have*

1. *if* $(C_\oplus, E)$ *is a support graph of* (Γ, C), *then* $(C'_\oplus, E')$ *is a support graph of* (Γ, C') *for some* $E, E' \subseteq E_0$;
2. *if* $(C_\oplus \cup C_\ominus, E_1)|_{S(\Gamma,C)}$ *is a blockage graph of* (Γ, C) *and* $C_\oplus \subseteq S(\Gamma, C)$, *then* $(C'_\oplus \cup C'_\ominus, E_1)|_{S(\Gamma,C')}$ *is a blockage graph of* (Γ, C').

A similar result can be shown for $\mathcal{P}_\Gamma$.

Finally, $\mathcal{P}_\Gamma$ can be used for deciding answersetship in the following way.

Corollary 3.3.7 (Answer set characterization, I′) *Let* Γ *be the RDG of logic program* Π *and let* C *be a total coloring of* Γ.
Then, C *is an admissible coloring of* Γ *iff* $\mathcal{P}_\Gamma(C) = C$ *and there is a support graph of* (Γ, C).

This result directly follows from Theorem 3.2.4. Clearly, the result is also valid when replacing $\mathcal{P}_\Gamma$ by $\mathcal{P}^*_\Gamma$.

The following operation draws upon the maximal support graph of colored *RDG*s.

Definition 3.3.3 *Let* Γ *be the RDG of logic program* Π *and* C *be a partial coloring of* Γ.
Furthermore, let (V, E) *be a maximal support graph of* (Γ, C) *for some* $E \subseteq (\Pi \times \Pi)$.
Then, define $\mathcal{U}_\Gamma : \mathbb{C} \to \mathbb{C}$ *as* $\mathcal{U}_\Gamma(C) = (C_\oplus, \Pi \setminus V)$.

This operator allows for coloring rules with $\ominus$ whenever it is clear from the given partial coloring that they will remain unsupported.[11] Observe that $\Pi \setminus V = C_\ominus \cup (\Pi \setminus V)$. As with $\mathcal{P}^*_\Gamma$, operator $\mathcal{U}_\Gamma(C)$ is an extension of C. To be more precise, we have $C_\oplus = (\mathcal{U}_\Gamma(C))_\oplus$ and $C_\ominus \cup \overline{S}(\Gamma, C) \subseteq (\mathcal{U}_\Gamma(C))_\ominus$. Unlike $\mathcal{P}^*_\Gamma$, however, operator $\mathcal{U}_\Gamma$ allows for coloring nodes unconnected with the already colored part of the graph.

As regards program Π_9 in Example 9, for instance, we obtain $\mathcal{U}_\Gamma((\emptyset, \emptyset)) = (\emptyset, \{r_5\})$. While this information on r_5 can also be supplied by $\mathcal{P}_\Gamma$, it is not obtainable for "self-supporting 0-loops", as in $\Pi = \{p \leftarrow q, q \leftarrow p\}$. In this case, we obtain $\mathcal{U}_\Gamma((\emptyset, \emptyset)) = (\emptyset, \{p \leftarrow q, q \leftarrow p\})$, which is not obtainable through $\mathcal{P}_\Gamma$ (and $\mathcal{P}^*_\Gamma$, respectively).

Although $\mathcal{U}_\Gamma$ cannot be defined in general, it is defined on colorings satisfying the conditions of Theorem 3.2.2, guaranteeing the existence of support graphs.

[11] The relation to unfounded sets is described in Corollary 3.4.3.

Corollary 3.3.8 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*
If (Γ, C) has a support graph, then $\mathcal{U}_\Gamma(C)$ exists.

We may actually characterize $\mathcal{U}_\Gamma(C)$ in terms of iterated applications of an operator, similar to T_Π; this is detailed in Section 3.3.4.2 as well as in Section A.2.

As with $\mathcal{P}^*_\Gamma$, operator $\mathcal{U}_\Gamma$ is reflexive, idempotent, monotonic, and answer set preserving.

Theorem 3.3.9 *Let Γ be the RDG of logic program Π and let C and C' be partial colorings of Γ such that $AC_\Pi(C) \neq \emptyset$ and $AC_\Pi(C') \neq \emptyset$.*
Then, we have the following properties.

1. *$C \sqsubseteq \mathcal{U}_\Gamma(C)$;*
2. *$\mathcal{U}_\Gamma(C) = \mathcal{U}_\Gamma(\mathcal{U}_\Gamma(C))$;*
3. *if $C \sqsubseteq C'$, then $\mathcal{U}_\Gamma(C) \sqsubseteq \mathcal{U}_\Gamma(C')$.*

Theorem 3.3.10 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*
Then, we have $AC_\Pi(C) = AC_\Pi(\mathcal{U}_\Gamma(C))$.

Note that unlike $\mathcal{P}_\Gamma$, operator $\mathcal{U}_\Gamma$ leaves the support graph of (Γ, C) unaffected. Since, according to Theorem 3.2.8, the essential blockage graph is composed of supported rules only, the same applies to this graph as well.

Theorem 3.3.11 *Let $\Gamma = (\Pi, E_0, E_1)$ be the RDG of logic program Π and C be a partial coloring of Γ.*
For $C' = \mathcal{U}_\Gamma(C)$, we have

1. *if $(C_\oplus, E)$ is a support graph of (Γ, C), then $(C_\oplus, E)$ is a support graph of (Γ, C') for some $E \subseteq E_0$;*
2. *if $(C_\oplus \cup C_\ominus, E_1)|_{S(\Gamma,C)}$ is a blockage graph of (Γ, C), then $(C_\oplus \cup C_\ominus, E_1)|_{S(\Gamma,C)}$ is a blockage graph of (Γ, C').*

In fact, we have in the latter case that $(C_\oplus \cup C_\ominus, E_1)|_{S(\Gamma,C)} = (C'_\oplus \cup C'_\ominus, E_1)|_{S(\Gamma,C')}$.

Because $\mathcal{U}_\Gamma$ implicitly enforces the existence of a support graph, our operators furnish yet another characterization of answer sets.

Corollary 3.3.12 (Answer set characterization, I'') *Let Γ be the RDG of logic program Π and let C be a total coloring of Γ.*
Then, C is an admissible coloring of Γ iff $C = \mathcal{P}_\Gamma(C)$ and $C = \mathcal{U}_\Gamma(C)$.

Clearly, this result is also valid when replacing $\mathcal{P}_\Gamma$ by $\mathcal{P}^*_\Gamma$.

Note that the last result is obtained from Corollary 3.3.7 by replacing the requirement of the existence of a support graph by $C = \mathcal{U}_\Gamma(C)$. However, the last condition cannot guarantee that all supported unblocked rules belong to $C_\oplus$. For instance, $(\emptyset, \{a \leftarrow\})$ has an empty support graph; hence $(\emptyset, \{a \leftarrow\}) = \mathcal{U}_\Gamma((\emptyset, \{a \leftarrow\}))$. That is, the trivially supported fact $a \leftarrow$ remains in $C_\ominus$. In our setting, such a miscoloring is detected by operator $\mathcal{P}_\Gamma$. That is, $\mathcal{P}_\Gamma((\emptyset, \{a \leftarrow\}))$ does not exist, since it would yield $(\{a \leftarrow\}, \{a \leftarrow\})$, which is no partial coloring.

3.3.2 Basic operational characterization

We start by providing a very general operational characterization that possesses a maximum degree of freedom.

To this end, we observe that Corollary 3.3.7 and 3.3.12, respectively, can serve as a straightforward *check* for deciding whether a given total coloring constitutes an answer set. A corresponding *guess* can be provided through an operator capturing a non-deterministic (don't know) choice.

Definition 3.3.4 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*

For $\circ \in \{\oplus, \ominus\}$, define $\mathcal{C}^\circ_\Gamma : \mathbb{C} \to \mathbb{C}$ as

1. $\mathcal{C}^\oplus_\Gamma(C) = (C_\oplus \cup \{r\}, C_\ominus)$ *for some* $r \in \Pi \setminus (C_\oplus \cup C_\ominus)$;
2. $\mathcal{C}^\ominus_\Gamma(C) = (C_\oplus, C_\ominus \cup \{r\})$ *for some* $r \in \Pi \setminus (C_\oplus \cup C_\ominus)$.

We use $\mathcal{C}^\circ_\Gamma$ whenever the distinction between $\mathcal{C}^\oplus_\Gamma(C)$ and $\mathcal{C}^\ominus_\Gamma(C)$ is of no importance. Strictly speaking, $\mathcal{C}^\circ_\Gamma$ is also parametrized with r; we leave this implicit to abstract from the actual choice. In fact, whenever both operators $\mathcal{C}^\oplus_\Gamma(C)$ and $\mathcal{C}^\ominus_\Gamma(C)$ are available, the choice of r is only a *"don't care"* choice, while that among $\oplus$ and $\ominus$ is the crucial *"don't know"* choice. Intuitively, this is because all rules must be colored either way; it is the attributed color that is of prime importance for the existence of an answer set.

Combining the previous guess and check operators yields our first operational characterization of admissible colorings (along with its underlying answer sets).

Theorem 3.3.13 (Operational answer set characterization, I) *Let Γ be the RDG of logic program Π and let C be a total coloring of Γ.*

Then, C is an admissible coloring of Γ iff there exists a sequence $(C^i)_{0 \leq i \leq n}$ with the following properties:

1. $C^0 = (\emptyset, \emptyset)$;
2. $C^{i+1} = \mathcal{C}^\circ_\Gamma(C^i)$ *for some* $\circ \in \{\oplus, \ominus\}$ *and* $0 \leq i < n$;
3. $C^n = \mathcal{P}_\Gamma(C^n)$;
4. $C^n = \mathcal{U}_\Gamma(C^n)$;
5. $C^n = C$.

In what follows, we refer to such sequences also as *coloring sequences*. Note that all sequences satisfying conditions 1-5 of Theorem 3.3.13 are *successful* in the sense that their last element corresponds to an existing answer set. If a program has no answer set, then no such sequence exists.

Although this straightforward guess and check approach may not be of great implementation value, it supplies us with an initial skeleton for the coloring process that we refine in the sequel. In particular, this characterization stresses the basic fact that we possess complete freedom in forming a coloring sequence as long as we can guarantee that the resulting coloring is a fixed point of $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$. It is worth mentioning that this simple approach is inapplicable when fixing $\circ$ to either $\oplus$ or $\ominus$ (cf. Section 3.3.3 below).

We observe the following properties.

Theorem 3.3.14 *Given the same prerequisites as in Theorem 3.3.13, let $(C^i)_{0 \leq i \leq n}$ be a sequence satisfying conditions 1-5 in Theorem 3.3.13.*

Then, we have the following properties for $0 \leq i \leq n$.

1. C^i *is a partial coloring;*
2. $C^i \sqsubseteq C^{i+1}$;

3. $AC_\Pi(C^i) \supseteq AC_\Pi(C^{i+1})$;

4. $AC_\Pi(C^i) \neq \emptyset$;

5. (Γ, C^i) *has a (maximal) support graph.*

All these properties represent invariants of the consecutive colorings. While the first three properties are provided by operator $\mathcal{C}^\circ_\Gamma$ in choosing among uncolored rules only, the last two properties are actually enforced by the final coloring C^n, that is, the "check" expressed by conditions 3–5 in Theorem 3.3.13. In fact, sequences only enjoying conditions 1 and 2 in Theorem 3.3.13, fail to satisfy Property 4 and 5 in general. In practical terms, this means that computations of successful sequences may be led numerous times on the "garden path" before termination.

As it is well-known, the number of choices can be significantly reduced by applying deterministic operators. To this end, given a partial coloring C, define $(\mathcal{PU})^*_\Gamma(C)$ as the $\sqsubseteq$-smallest partial coloring containing C and being closed under $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$. [12]

Theorem 3.3.15 (Operational answer set characterization, II) *Let* Γ *be the RDG of logic program* Π *and let* C *be a total coloring of* Γ.

Then, C *is an admissible coloring of* Γ *iff there exists a sequence* $(C^i)_{0\leq i\leq n}$ *with the following properties:*

1. $C^0 = (\mathcal{PU})^*_\Gamma((\emptyset, \emptyset))$;

2. $C^{i+1} = (\mathcal{PU})^*_\Gamma(\mathcal{C}^\circ_\Gamma(C^i))$ *for some* $\circ \in \{\oplus, \ominus\}$ *and* $0 \leq i < n$;

3. $C^n = C$.

On the one hand, the continuous applications of $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$ extend colorings after each choice. On the other hand, this proceeding guarantees that each partial coloring C^i is closed under $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$.

Regarding correctness and completeness, however, it is clear in view of Theorem 3.3.13 that any number of iterations of $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$ can be executed after $\mathcal{C}^\circ_\Gamma$ as long as $(\mathcal{PU})^*_\Gamma$ is the final operation leading to C^n in Theorem 3.3.15.

For illustration, consider the coloring sequence (C^0, C^1) obtained for answer set $\{b, p, f'\}$ of program Π_9 in Example 9:

$$\begin{array}{lcll} C^0 & = & (\mathcal{PU})^*_\Gamma((\emptyset,\emptyset)) & = & (\{p \leftarrow, b \leftarrow p\}, \{b \leftarrow m\}) \\ C^1 & = & (\mathcal{PU})^*_\Gamma(\mathcal{C}^\oplus_\Gamma(C^0)) & = & (\{p \leftarrow, b \leftarrow p, f' \leftarrow p, not\ f\}, \\ & & & & \quad \{b \leftarrow m, f \leftarrow b, not\ f', x \leftarrow f, f', not\ x\}) \end{array}$$

The decisive operation in this sequence is the application of $\mathcal{C}^\oplus_\Gamma$ leading to $C(\{f \leftarrow b, not\ f'\}) = \oplus$. Note that in this simple example all propagation is accomplished by operator $\mathcal{P}_\Gamma$. We have illustrated the formation of the sequence in Figure 3.5. The same final result is obtained when choosing $\mathcal{C}^\ominus_\Gamma$ such that $C(\{f' \leftarrow p, not\ f\}) = \ominus$. This illustrates that several coloring sequences may lead to the same answer set.

As with Corollary 3.3.3, we have the following result.

Corollary 3.3.16 *Let* Γ *be the RDG of logic program* Π. *Then,* $(\mathcal{PU})^*_\Gamma((\emptyset, \emptyset))$ *exists.*

As we will see in Section 3.4, this operator sequence directly corresponds to the well-founded model of a program.

The usage of continuous propagations leads to further invariant properties.

[12] An iterative characterization of $(\mathcal{PU})^*_\Gamma(C)$ is given in Section A.2

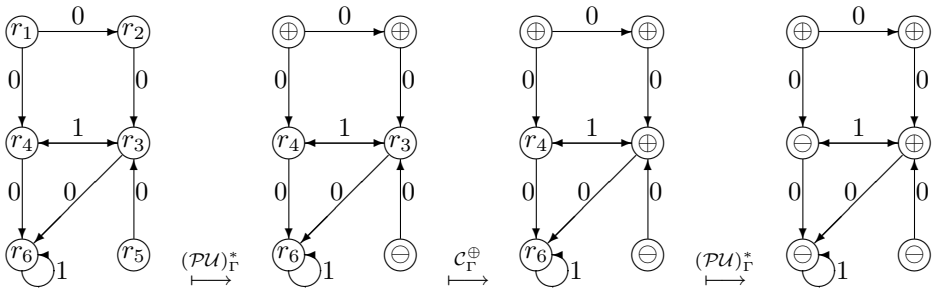

Figure 3.5: A coloring sequence for program Π_9.

Theorem 3.3.17 *Given the same prerequisites as in Theorem 3.3.15, let $(C^i)_{0\leq i\leq n}$ be a sequence satisfying conditions 1-3 in Theorem 3.3.15.*

Then, we have the following:

1.–5. as given in Theorem 3.3.14;

6. $C^{i+1}_\oplus \supseteq S(\Gamma, C^i) \cap \overline{B}(\Gamma, C^i)$;

7. $C^{i+1}_\ominus \supseteq \overline{S}(\Gamma, C^i) \cup B(\Gamma, C^i)$.

Taking the last two properties together with Condition 5 from Theorem 3.3.14, we see that propagation gradually enforces exactly the attributes on partial colorings, expressed in Theorem 3.2.4 for admissible colorings.

Given that we obtain only two additional properties, one may wonder whether exhaustive propagation truly pays off. In fact, its great value becomes apparent when looking at the properties of prefix sequences, not necessarily leading to a successful end.

Theorem 3.3.18 *Given the same prerequisites as in Theorem 3.3.15, let $(C^j)_{0\leq j\leq m}$ be a sequence satisfying Condition 1 and 2 in Theorem 3.3.15.*

Then, we have the following.

1.–3., 5. as given in Theorem 3.3.14;

6.–7. as given in Theorem 3.3.17.

Using exhaustive propagations, we observe that except for Property 4 all properties possessed by successful sequences, are shared by (possibly unsuccessful) prefix sequences.

The next results make the aforementioned claim on the effect of deterministic operators on admissible prefix sequences more precise:

Theorem 3.3.19 *Let Γ be the RDG of logic program Π.*

Let m be the number of sequences over $\mathbb{C}$ satisfying conditions 1-4 in Theorem 3.3.13 and let n be the number of sequences over $\mathbb{C}$ satisfying conditions 1 and 2 in Theorem 3.3.15.

Then, we have $n \leq m$.

Moreover, successful sequences are usually shorter when using propagation.

Theorem 3.3.20 *Let Γ be the RDG of logic program Π and C be an admissible coloring of Γ.*

Let $(C^i)_{0\leq i\leq m}$ and $(C^j)_{0\leq j\leq n}$ be the shortest sequences obtained for C according to Theorem 3.3.13 and Theorem 3.3.15, respectively.

Then, we have $n \leq m$.

The same result can be shown for the longest sequences.

3.3.3 Unicoloring operational characterization

As mentioned above, the use of two coloring operators is essential for our initial operational characterization given in Theorem 3.3.13. In fact, this is obsolete when using continuous propagation, as done in Theorem 3.3.15. That is, rather than using two coloring operators, we may actually use only one of them, and leave the attribution of the complementary color to propagation operators. To be more precise, this amounts to replacing Condition 2 in Theorem 3.3.15 either by

$$2.^{+}\ \ C^{i+1} = (\mathcal{PU})^{*}_{\Gamma}(\mathcal{C}^{\oplus}_{\Gamma}(C^{i})) \quad \textit{or} \qquad 2.^{-}\ \ C^{i+1} = (\mathcal{PU})^{*}_{\Gamma}(\mathcal{C}^{\ominus}_{\Gamma}(C^{i})) \qquad \textit{for } 0 \leq i < n.$$

Then, we have the following result.

Corollary 3.3.21 (Operational answer set characterization, II^{+}/II^{-}) *Theorem 3.3.15 still holds, when replacing Condition 2 either by 2.$^{+}$ or 2.$^{-}$.*

Note that the possible set of (prefix) sequences is smaller than that obtained from the "bi-coloring" in Theorem 3.3.15, simply, because choices are restricted to a single color. On the other hand, we see no advantage of either approach regarding the length of successful sequences. Also, since the choice of the color is fixed, the choice of the rule to be colored becomes a "don't know" choice.

The last type of characterization is based on exhaustive propagation. Hence, we are interested in the question how much propagation is sufficient for compensating the possibility of choosing among two colors. The next result gives an answer for the case of unicoloring with $\mathcal{C}^{\oplus}_{\Gamma}$.

Theorem 3.3.22 (Operational answer set characterization, III^{+}) *Let Γ be the RDG of logic program Π and let C be a total coloring of Γ.*

Then, C is an admissible coloring of Γ iff there exists a sequence $(C^{i})_{0\leq i\leq n}$ with the following properties:

1. $C^{0} = (\emptyset, \emptyset)$;
2. $C^{i+1} = \mathcal{C}^{\oplus}_{\Gamma}(C^{i})$ *for* $0 \leq i < n-1$;
3. $C^{n} = \mathcal{U}_{\Gamma}(\mathcal{P}^{*}_{\Gamma}(C^{n-1}))$;
4. $C^{n} = C$.

The actual propagation is done in Condition 3. That is, the consecutive application of $\mathcal{U}_{\Gamma}$ and $\mathcal{P}^{*}_{\Gamma}$ allows for coloring all remaining rules in C^{n-1} with $\ominus$.

Note that the application order of $\mathcal{P}^{*}_{\Gamma}$ and $\mathcal{U}_{\Gamma}$ in Condition 3 cannot be reversed. To see this, reconsider program Π_{10} in Example 10 on page 24, consisting of rules

$$\begin{array}{lll} r_1: & a & \leftarrow \\ r_2: & b & \leftarrow \mathit{not}\ a \\ r_3: & c & \leftarrow b \\ r_4: & b & \leftarrow c\,. \end{array}$$

and observe that its only answer set $\{a\}$ corresponds to coloring $(\{r_1\}, \{r_2, r_3, r_4\})$. In order to obtain this according to Theorem 3.3.22, we must color rule r_1 with $\oplus$. We then get $\mathcal{P}^{*}_{\Gamma}((\{r_1\}, \emptyset)) = (\{r_1\}, \{r_2\})$ and finally $\mathcal{U}_{\Gamma}((\{r_1\}, \{r_2\})) = (\{r_1\}, \{r_2, r_3, r_4\})$. Switching the last operators yields $\mathcal{U}_{\Gamma}((\{r_1\}, \emptyset)) = (\{r_1\}, \emptyset)$ and $\mathcal{P}^{*}_{\Gamma}((\{r_1\}, \emptyset)) = (\{r_1\}, \{r_2\})$ and we fail to obtain the desired result.

Interestingly, the usage of the operator $\mathcal{P}^{*}_{\Gamma}$ is sufficient when coloring with $\ominus$ only.

Theorem 3.3.23 (Operational answer set characterization, III^{-}) *Let Γ be the RDG of logic program Π and let C be a total coloring of Γ.*

Then, C is an admissible coloring of Γ iff there exists a sequence $(C^{i})_{0\leq i\leq n}$ with the following properties:

1. $C^0 = (\emptyset, \emptyset)$;
2. $C^{i+1} = \mathcal{C}_\Gamma^\ominus(C^i)$ *for* $0 \leq i < n-1$;
3. $C^n = \mathcal{P}_\Gamma^*(C^{n-1})$;
4. $C^n = C$.

Unicoloring offers another perspective on strategies employing bicoloring: The coloration with the second color may be regarded as some sort of "lemmatization" avoiding duplicate solutions rather than a genuine choice.

3.3.4 Support-driven operational characterization

The number of possible choices encountered during a computation is of crucial importance. We have already seen above that propagation has great computational advantages. What else may cut down the number of choices?

Looking at the graph structures underlying an admissible coloring (cf. Theorem 3.2.8), we observe that support graphs capture a global — since recursive — structure, while blockage graphs aim at a rather local structure, based on arc-wise constraints. Consequently, it seems advisable to prefer choices maintaining support structures over those maintaining blockage relations, since the former have more global repercussions than the latter.

To this end, we develop in this section a strategy that is based on a choice operation restricted to supported rules.

Definition 3.3.5 *Let* Γ *be the RDG of logic program* Π *and* C *be a partial coloring of* Γ.
For $\circ \in \{\oplus, \ominus\}$, *define* $\mathcal{D}_\Gamma^\circ : \mathbb{C} \to \mathbb{C}$ *as*

1. $\mathcal{D}_\Gamma^\oplus(C) = (C_\oplus \cup \{r\}, C_\ominus)$ *for some* $r \in S(\Gamma, C) \setminus (C_\oplus \cup C_\ominus)$;
2. $\mathcal{D}_\Gamma^\ominus(C) = (C_\oplus, C_\ominus \cup \{r\})$ *for some* $r \in S(\Gamma, C) \setminus (C_\oplus \cup C_\ominus)$.

Compared to operators $\mathcal{C}_\Gamma^\circ$, the latter restrict their choice to supported rules. Verifying whether a rule is supported can then be done in a local fashion by looking at the immediate predecessors in the *RDG*. With this little additional effort, the number of colorable rules is smaller than that encountered when applying $\mathcal{C}_\Gamma^\circ$. The benefit of support-driven characterizations is that the length of coloring sequences is bound by the number of supported rules. Depending on how the non-determinism of $\mathcal{D}_\Gamma^\circ$ is dealt with algorithmically, this may either lead to a reduced depth of the search tree or a reduced branching factor.

In fact, in a successful coloring sequence $(C^i)_{0 \leq i \leq n}$, all rules in $(C^n)_\oplus$ must belong to an encompassing support graph and thus be supported. Hence, by means of $\mathcal{D}_\Gamma^\oplus(C)$ (along with $\mathcal{P}_\Gamma^*$) the supportedness of each set $C_\oplus^i$ can be made invariant. Consequently, such a proceeding allows for establishing the existence of support graphs, as stipulated in Theorem 3.2.4, so to speak "on the fly". To a turn, this allows for a much simpler approach to the task(s) previously accomplished by operator $\mathcal{U}_\Gamma$. We discuss two such simplifications in what follows.

3.3.4.1 Support-driven operational characterization I

Given that the existence of support graphs is guaranteed, one may actually completely dispose of operator $\mathcal{U}_\Gamma$ and color in a final step all uncolored rules with $\ominus$.

Definition 3.3.6 *Let* Γ *be the RDG of logic program* Π *and* C *a partial coloring of* Γ.
Then, define $\mathcal{N}_\Gamma : \mathbb{C} \to \mathbb{C}$ *as* $\mathcal{N}_\Gamma(C) = (C_\oplus, \Pi \setminus C_\oplus)$.

Roughly speaking, the idea is then to "actively" color only supported rules and rules blocked by supported rules; all remaining rules are then unsupported and "thrown" into $C_\ominus$ in a final step. This is made precise in the following characterization.

Theorem 3.3.24 (Operational answer set characterization, IV) *Let Γ be the RDG of logic program Π and let C be a total coloring of Γ.*

Then, C is an admissible coloring of Γ iff there exists a sequence $(C^i)_{0\leq i\leq n}$ with the following properties:

1. $C^0 = (\emptyset, \emptyset)$;
2. $C^{i+1} = \mathcal{D}^\circ_\Gamma(C^i)$ *where* $\circ \in \{\oplus, \ominus\}$ *and* $0 \leq i < n-1$;
3. $C^n = \mathcal{N}_\Gamma(C^{n-1})$;
4. $C^n = \mathcal{P}_\Gamma(C^n)$;
5. $C^n = C$.

We note that there is a little price to pay for turning $\mathcal{U}_\Gamma$ into $\mathcal{N}_\Gamma$, expressed by the test on the final total coloring in Condition 4. Without it, one could use $\mathcal{N}_\Gamma$ to obtain a total coloring by coloring rules with $\ominus$ in an arbitrary way.

We obtain the following properties for the previous type of coloring sequences.

Theorem 3.3.25 *Given the same prerequisites as in Theorem 3.3.24, let $(C^i)_{0\leq i\leq n}$ be a sequence satisfying conditions 1-5 in Theorem 3.3.24.*

Then, we have the following.

1.–5. as given in Theorem 3.3.14;

8. $(C^i_\oplus, E)$ *is a support graph of* (Γ, C^i) *for some* $E \subseteq \Pi \times \Pi$.

Condition 8 makes the aforementioned claim on the supportedness of each rule in $C^i_\oplus$ explicit. In contrast to the coloring sequences enjoying Condition 5 only, the sequences formed by means of $\mathcal{D}^\circ_\Gamma$ guarantee that each $C^i_\oplus$ forms a support graph.

In fact, there is some overlap among operator $\mathcal{D}^\ominus_\Gamma$ and $\mathcal{N}_\Gamma$. To see this, consider $\Pi = \{a \leftarrow, b \leftarrow not\ a\}$. We must initially apply $\mathcal{D}^\oplus_\Gamma$ to obtain $(\{a \leftarrow\}, \emptyset)$ from the empty coloring. Then, however, there are two possibilities for obtaining total coloring $(\{a\}, \{b \leftarrow not\ a\})$, either by applying $\mathcal{D}^\ominus_\Gamma$ or by applying $\mathcal{N}_\Gamma$. In fact, in view of this, $\mathcal{N}_\Gamma$ allows us to dispose of $\mathcal{D}^\ominus_\Gamma$:

Corollary 3.3.26 (Operational answer set characterization, IV$^+$) *Theorem 3.3.24 still holds, when replacing Condition 2 by*

2.$^+$ $C^{i+1} = \mathcal{D}^\oplus_\Gamma(C^i)$ *where* $0 \leq i < n-1$.

Observe that there is no characterization using $\mathcal{D}^\ominus_\Gamma$ and $\mathcal{N}_\Gamma$ because this leaves no possibility for coloring rules with $\oplus$.

Theorem 3.3.27 *Given the same prerequisites as in Corollary 3.3.26, let $(C^i)_{0\leq i\leq n}$ be a sequence satisfying conditions 1-5 in Corollary 3.3.26.*

Then, we have the following.

1.–5. as given in Theorem 3.3.14;

8. as given in Theorem 3.3.25;

9. $(C^i_\oplus \cup C^i_\ominus, E)|_{S(\Gamma,C^i)}$ *is a blockage graph of* (Γ, C^i).

Interestingly, only support-driven unicoloring by means of $\mathcal{D}^\oplus_\Gamma$ can guarantee the consecutive existence of blockage graphs. This is because $\mathcal{D}^\oplus_\Gamma$ warrants, first, that $C^i_\oplus \subseteq S(\Gamma, C^i)$ and thus all blocking rules are taken into account and, second, that rules are not arbitrarily colored with $\ominus$ but rather guarded by operator $\mathcal{P}_\Gamma$. [13]

As discussed above, there is some overlap in the characterization expressed in Theorem 3.3.24. Interestingly, this can be eliminated by adding propagation operator $\mathcal{P}^*_\Gamma$ to the previous characterization. This results in coloring sequences corresponding to the basic strategy used in the `noMoRe` system [4].

Theorem 3.3.28 (Operational answer set characterization, V) *Let* Γ *be the RDG of logic program* Π *and let* C *be a total coloring of* Γ.

Then, C *is an admissible coloring of* Γ *iff there exists a sequence* $(C^i)_{0\leq i\leq n}$ *with the following properties:*

1. $C^0 = \mathcal{P}^*_\Gamma((\emptyset, \emptyset))$;
2. $C^{i+1} = \mathcal{P}^*_\Gamma(\mathcal{D}^\circ_\Gamma(C^i))$ *where* $\circ \in \{\oplus, \ominus\}$ *and* $0 \leq i < n-1$;
3. $C^n = \mathcal{N}_\Gamma(C^{n-1})$;
4. $C^n = \mathcal{P}_\Gamma(C^n)$;
5. $C^n = C$.

For illustration, consider the coloring sequence (C^0, C^1, C^2) obtained for answer set $\{b, p, f'\}$ of program Π_9 in Example 9:

$$\begin{array}{lcll} C^0 & = & \mathcal{P}^*_\Gamma((\emptyset,\emptyset)) & = & (\{p \leftarrow, b \leftarrow p\}, \{b \leftarrow m\}) \\ C^1 & = & \mathcal{P}^*_\Gamma(\mathcal{D}^\oplus_\Gamma(C^0)) & = & (\{p \leftarrow, b \leftarrow p, f' \leftarrow p, not\ f\}, \\ & & & & \quad \{b \leftarrow m, f \leftarrow b, not\ f', x \leftarrow f, f', not\ x\}) \\ C^2 & = & \mathcal{N}_\Gamma(C^1) & = & C^1 \end{array}$$

This sequence is similar to the one obtained for Π_9 with the characterization given in Theorem 3.3.15. All propagation is accomplished by operator $\mathcal{P}_\cdot$. However, operator $\mathcal{D}^\oplus_\Gamma$ is faced with less choices than $\mathcal{C}^\oplus_\Gamma$ (in Theorem 3.3.15) because only two among the three uncolored rules are supported.

For another example, consider program Π_{10} in Example 10 on page 24. We get a coloring sequence (C^0, C^1), where

$$\begin{array}{lcll} C^0 & = & \mathcal{P}^*_\Gamma((\emptyset,\emptyset)) & = & (\{a \leftarrow\}, \{b \leftarrow not\ a\}) \\ C^1 & = & \mathcal{N}_\Gamma((\{a \leftarrow\}, \{b \leftarrow not\ a\})) & = & (\{a \leftarrow\}, \{b \leftarrow not\ a, c \leftarrow b, b \leftarrow c\})\ . \end{array}$$

Note that operator $\mathcal{D}^\circ_\Gamma$ is inapplicable to C^0, since $S(\Gamma, C^0) \setminus (C^0_\oplus \cup C^0_\ominus)$ is empty. In this situation, $\mathcal{C}^\circ_\Gamma$ would be applicable to color either of the two uncolored rules in $\Pi \setminus (C^0_\oplus \cup C^0_\ominus)$. In the final step, the two unfounded rules are directly colored by operator $\mathcal{N}_\Gamma$ without any further efforts.

Indeed, the strategy of `noMoRe` applies $\mathcal{P}^*_\Gamma \circ \mathcal{D}^\circ_\Gamma$ as long as there are supported rules. Once no more uncolored supported rules exist, operator $\mathcal{N}_\Gamma$ is called. Finally, $\mathcal{P}_\Gamma$ is applied but only to those rules colored previously by $\mathcal{N}_\Gamma$. At first sight, this approach may seem to correspond to a subclass of the coloring sequences described above, in the sense that `noMoRe` enforces a maximum number of transitions described in Condition 2. To see that this is not the case, we observe the following property.

[13] This is more apparent in Corollary 3.3.31 below, when using $\mathcal{P}^*_\Gamma$ for propagation as well.

Theorem 3.3.29 *Given the same prerequisites as in Theorem 3.3.28, let $(C^i)_{0\leq i\leq n}$ be a sequence satisfying conditions 1-5 in Theorem 3.3.28.*

Then, we have $(\mathcal{N}_\Gamma(C^{n-1})_\ominus \setminus C_\ominus^{n-1}) \subseteq \overline{S}(\Gamma, C)$.

That is, no matter which (supported) rules are colored $\ominus$ by $\mathcal{D}_\Gamma^\ominus$, operator $\mathcal{N}_\Gamma$ only applies to unsupported ones. It is thus no restriction to enforce the consecutive application of $\mathcal{P}_\Gamma^*$ and $\mathcal{D}_\Gamma^\circ$ until no more supported rules are available. In fact, it is the interplay of the two last operators that guarantees this property. For instance, looking at $\Pi = \{a, b \leftarrow \mathit{not}\ a\}$, we see that we directly obtain the final total coloring because $(\{a\}, \{b \leftarrow \mathit{not}\ a\}) = \mathcal{P}_\Gamma^*(\mathcal{D}_\Gamma^\oplus((\emptyset, \emptyset)))$, without any appeal to $\mathcal{N}_\Gamma$. Rather it is $\mathcal{P}_\Gamma^*$ that detects that $b \leftarrow \mathit{not}\ a$ belongs to the set of blocked rules. Generally speaking, $\mathcal{D}_\Gamma^\oplus$ consecutively chooses the generating rules of an answer set, finally gathered in $C_\oplus = S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. Every rule in $B(\Gamma, C)$ is blocked by some rule in $C_\oplus$. So whenever a rule r is added by $\mathcal{D}_\Gamma^\oplus$ to $C_\oplus$, operator $\mathcal{P}_\Gamma^*$ adds all rules blocked by r to $C_\ominus$. In this way, $\mathcal{P}_\Gamma^*$ and $\mathcal{D}_\Gamma^\oplus$ gradually color all rules in $S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ and $B(\Gamma, C)$, so that all remaining uncolored rules, subsequently treated by $\mathcal{N}_\Gamma$, must belong to $\overline{S}(\Gamma, C)$. [14]

Furthermore, we obtain the following properties.

Theorem 3.3.30 *Given the same prerequisites as in Theorem 3.3.28, let $(C^i)_{0\leq i\leq n}$ be a sequence satisfying conditions 1-5 in Theorem 3.3.28.*

Then, we have the following.

1.–5. as given in Theorem 3.3.14;

6.–7. as given in Theorem 3.3.17;

8. as given in Theorem 3.3.25.

Finally, for the sake of completeness, we give the unicoloring variant along with its properties.

Corollary 3.3.31 (Operational answer set characterization, $\mathbf{V}^+$) *Theorem 3.3.28 still holds, when replacing Condition 2 by*

2.$^+$ $C^{i+1} = \mathcal{P}_\Gamma^*(\mathcal{D}_\Gamma^\oplus(C^i))$ where $0 \leq i < n-1$.

Theorem 3.3.32 *Given the same prerequisites as in Corollary 3.3.31, let $(C^i)_{0\leq i\leq n}$ be a sequence satisfying conditions 1-5 in Corollary 3.3.31.*

Then, we have the following.

1.–5. as given in Theorem 3.3.14;

6.–7. as given in Theorem 3.3.17;

8.–9. as given in Theorem 3.3.25 and 3.3.27.

The previous characterization also satisfies Theorem 3.3.29.

3.3.4.2 Support-driven operational characterization II

Although the previous approach has turned out to be of practical value as the core inference strategy of the `noMoRe` system, it is still improvable. This is because the early detection of unsupported rules may allow

[14] Some rules of $\overline{S}(\Gamma, C)$ are already detected and added to $C_\ominus$ by $\mathcal{P}_\Gamma^*$.

for better propagation results and thus fewer choices. To see this, consider program $\Pi_{3.8} = \{r_1, r_2, r_3\}$, where [15]

$$\begin{array}{lllll} r_1: & p & \leftarrow & not\ q \\ r_2: & q & \leftarrow & r, not\ p \\ r_3: & r & \leftarrow & q. \end{array} \tag{3.8}$$

This program has the answer set $\{p\}$ represented by the admissible coloring $(\{r_1\}, \{r_2, r_3\})$. Without an operator like $\mathcal{U}_\Gamma$ one would need a choice operation for detecting the unfounded rules r_2 and r_3. Following Theorem 3.3.28, we get the coloring sequence:

$$\begin{array}{lllll} C^0 & = & \mathcal{P}^*_\Gamma((\emptyset,\emptyset)) & = & (\emptyset,\emptyset) \\ C^1 & = & \mathcal{P}^*_\Gamma(\mathcal{D}^\oplus_\Gamma(C^0)) & = & (\{r_1\},\emptyset) \\ C^2 & = & \mathcal{N}_\Gamma(C^1) & = & (\{r_1\},\{r_2,r_3\}) \end{array}$$

Although the support-driven choice operator $\mathcal{D}^\circ_\Gamma$ is only faced with a single alternative, as opposed to the three alternatives encountered by operator $\mathcal{C}^\circ_\Gamma$, it would be clearly advantageous to solve this example by propagation only. This motivates a variant of operator $\mathcal{U}_\Gamma$ that takes advantage of the support-driven strategy pursued by $\mathcal{D}^\circ_\Gamma$.

Given the *RDG* Γ of a logic program Π and a partial coloring C of Γ, define $\mathcal{T}_\Gamma : \mathbb{C} \to \mathbb{C}$ as

$$\mathcal{T}_\Gamma(C) = (C_\oplus \cup (S(\Gamma, C) \setminus C_\ominus), C_\ominus)$$

and define $\mathcal{T}^*_\Gamma(C)$ as the $\sqsubseteq$-smallest partial coloring containing C and being closed under $\mathcal{T}_\Gamma$.

The next result shows that $\mathcal{T}^*_\Gamma$ extends a given support graph to a maximal one.

Theorem 3.3.33 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*

If $(C_\oplus, E)$ is a support graph of (Γ, C), then $((\mathcal{T}^*_\Gamma(C))_\oplus, E')$ is a maximal support graph of (Γ, C) for some $E, E' \subseteq (\Pi \times \Pi)$.

Recall that by definition $C \sqsubseteq \mathcal{T}^*_\Gamma(C)$.

With this, we can define the following incremental (and constructive) variant of $\mathcal{U}_\Gamma$:

Definition 3.3.7 *Let Γ be the RDG of logic program Π and C a partial coloring of Γ.*

Then, define $\mathcal{V}_\Gamma : \mathbb{C} \to \mathbb{C}$ as $\mathcal{V}_\Gamma(C) = (C_\oplus, \Pi \setminus V)$ where $V = \mathcal{T}^*_\Gamma(C)_\oplus$.

Observe that $\Pi \setminus V = C_\ominus \cup (\Pi \setminus V)$. It is instructive to compare the latter definition with that of $\mathcal{U}_\Gamma$, given in Definition 3.3.3. In fact, $\mathcal{U}_\Gamma$ can be obtained by means of $\mathcal{T}^*_\Gamma$ by defining V in Definition 3.3.3 as $V = \mathcal{T}^*_\Gamma((\emptyset,\emptyset))_\oplus$ subject to the condition $C_\oplus \subseteq V$ and $C_\ominus \cap V = \emptyset$.

The next result tells us when both operators coincide.

Corollary 3.3.34 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ.*

If $(C_\oplus, E)$ is a support graph of (Γ, C) for some $E \subseteq (\Pi \times \Pi)$, then $\mathcal{U}_\Gamma(C) = \mathcal{V}_\Gamma(C)$.

Given that $\mathcal{D}^\circ_\Gamma$ adds only supported rules to a coloring C, it gradually extends the underlying support graph around $C_\oplus$. Hence, we may replace operator $\mathcal{U}_\Gamma$ by $\mathcal{V}_\Gamma$ whenever using choice operator $\mathcal{D}^\circ_\Gamma$. This is impossible when using operator $\mathcal{C}^\circ_\Gamma$, since its choice may not be supported and thus destroy the invariant support property expressed in Property 8 in Theorem 3.3.25. Unlike $\mathcal{U}_\Gamma$, operator $\mathcal{V}_\Gamma$ takes the support status of all rules in $C_\oplus$ for granted. This allows $\mathcal{V}_\Gamma$ to focus on uncolored rules, viz $\Pi \setminus (C_\oplus \cup C_\ominus)$. Such an assumption is not made by $\mathcal{U}_\Gamma$, which (possibly) reestablishes the support status of rules in $C_\oplus$; it is thus proned to consider all rules in $\Pi \setminus C_\ominus$. Technically, this is reflected by the fact that the computation of $\mathcal{V}_\Gamma$ by means of $\mathcal{T}_\Gamma$ may start from C, while the one of $\mathcal{U}_\Gamma$ by $\mathcal{T}_\Gamma$ must start out with the empty coloring. In all,

[15] For the sake of uniqueness, we label the program with the equation number.

$\mathcal{V}_\Gamma$ does not lead to fewer choices than $\mathcal{U}_\Gamma$, it has rather the computational advantage of avoiding redundant computations.

Now, we are ready to give support-oriented counterparts of the operational characterizations given in the two previous subsections. Most of them are obtainable by simply replacing operators $\mathcal{C}^\circ_\Gamma$ and $\mathcal{U}_\Gamma$ by $\mathcal{D}^\circ_\Gamma$ and $\mathcal{V}_\Gamma$, respectively. The first exception is Theorem 3.3.13 where the replacement of $\mathcal{C}^\circ_\Gamma$ by $\mathcal{D}^\circ_\Gamma$ leaves no way of coloring unsupported rules with $\ominus$. (This provides further evidence for the necessity of $\mathcal{N}_\Gamma$ in Theorem 3.3.24.)

For defining a support-oriented counterpart of Theorem 3.3.15, we first need the following propagation operator: As with $(\mathcal{PU})^*_\Gamma$, given a partial coloring C, we define $(\mathcal{PV})^*_\Gamma(C)$ as the $\sqsubseteq$-smallest partial coloring containing C and being closed under $\mathcal{P}_\Gamma$ and $\mathcal{V}_\Gamma$. [16]

Theorem 3.3.35 (Operational answer set characterization, VI) *Let Γ be the RDG of logic program Π and let C be a total coloring of Γ.*

Then, C is an admissible coloring of Γ iff there exists a sequence $(C^i)_{0\leq i\leq n}$ with the following properties:

1. $C^0 = (\mathcal{PV})^*_\Gamma((\emptyset,\emptyset))$;
2. $C^{i+1} = (\mathcal{PV})^*_\Gamma(\mathcal{D}^\circ_\Gamma(C^i))$ *where* $\circ \in \{\oplus,\ominus\}$ *and* $0 \leq i < n$;
3. $C^n = C$.

For illustration, consider program $\Pi_{3.8}$ in (3.8). We get a coloring sequence (C^0), where

$$C^0 \quad = \quad (\mathcal{PV})^*_\Gamma((\emptyset,\emptyset)) \quad = \quad (\{p \leftarrow not\ q\}, \{q \leftarrow r, not\ p, r \leftarrow q\})\ .$$

Or to be more precise,

$$\begin{array}{rcll} & & \mathcal{V}_\Gamma((\emptyset,\emptyset)) & = & (\emptyset, \{q \leftarrow r, not\ p, r \leftarrow q\}) \\ C^0 & = & \mathcal{P}_\Gamma((\emptyset, \{q \leftarrow r, not\ p, r \leftarrow q\})) & = & (\{p \leftarrow not\ q\}, \{q \leftarrow r, not\ p, r \leftarrow q\})\ . \end{array}$$

Analogously, we obtain for program Π_{10} in Example 10 on page 24 a singular coloring sequence (C^0), where

$$C^0 \quad = \quad (\mathcal{PV})^*_\Gamma((\emptyset,\emptyset)) \quad = \quad (\{a \leftarrow\}, \{b \leftarrow not\ a, c \leftarrow b, b \leftarrow c\})\ .$$

We note the following invariant properties of the sequence.

Theorem 3.3.36 *Given the same prerequisites as in Theorem 3.3.35, let $(C^i)_{0\leq i\leq n}$ be a sequence satisfying conditions 1-3 in Theorem 3.3.35.*

Then, we have the following.

1.–5. as given in Theorem 3.3.14;

6.–7. as given in Theorem 3.3.17;

8. as given in Theorem 3.3.32.

In analogy to what we have shown in Section 3.3.3, we may replace Condition 2 in Theorem 3.3.35 either by

$$2.^+\ C^{i+1} = (\mathcal{PV})^*_\Gamma(\mathcal{D}^\oplus_\Gamma(C^i)) \quad \textit{or} \qquad 2.^-\ C^{i+1} = (\mathcal{PV})^*_\Gamma(\mathcal{D}^\ominus_\Gamma(C^i)) \qquad \textit{for } 0 \leq i < n.$$

Then, we have the following.

Corollary 3.3.37 (Operational answer set characterization, VI$^+$/VI$^-$) *Theorem 3.3.35 still holds, when replacing Condition 2 either by $2.^+$ or $2.^-$.*

Also, all results obtained in Section 3.3.4.1 remain true, when replacing $\mathcal{N}_\Gamma$ by $\mathcal{V}_\Gamma$.

[16] A characterization in terms of iterated applications is given in Section A.2 on page 144.

3.3.5 Summary

In view of the many different operational characterizations, we summarize in Table 3.1 their approach to the formation of coloring sequences and their major properties. For this, we distinguish between the formation of coloring sequences $(C^i)_{0\leq i\leq n}$ and the test on their final element C^n. For describing the formation process in a compact way, we confine ourselves to operators and denote their composition by $\circ$. To be precise, given operators O_1 and O_2, we write $(O_1 \circ O_2)$ instead of $(\lambda x.O_1(O_2(x)))$. Furthermore, we use O^k to denote k consecutive applications of O for some arbitrary positive integer k.

For instance, Theorem 3.3.13 captures sequences that are obtained by $k = n$ applications of $\mathcal{C}^\circ_\Gamma$; this is indicated in Table 3.1 by $[\mathcal{C}^\circ_\Gamma]^k$. The two final tests on C^n, viz $C^n = \mathcal{P}_\Gamma(C^n)$ and $C^n = \mathcal{U}_\Gamma(C^n)$, are pointed out by $\mathcal{P}_\Gamma, \mathcal{U}_\Gamma$ (in column "Check").

	Section	Theorem	Formation Process	Check	Properties	Properties Prefix sequences
I	3.3.2	3.3.13	$[\mathcal{C}^\circ_\Gamma]^k$	$\mathcal{P}_\Gamma, \mathcal{U}_\Gamma$	1–5	1–3
II		3.3.15	$[(\mathcal{PU})^*_\Gamma \circ \mathcal{C}^\circ_\Gamma]^k \circ (\mathcal{PU})^*_\Gamma$	–	1–7	1–3,5–7
II$^+$	3.3.3	3.3.21	$[(\mathcal{PU})^*_\Gamma \circ \mathcal{C}^\oplus_\Gamma]^k \circ (\mathcal{PU})^*_\Gamma$	–	1–7	1–3,5–7
II$^-$		3.3.21	$[(\mathcal{PU})^*_\Gamma \circ \mathcal{C}^\ominus_\Gamma]^k \circ (\mathcal{PU})^*_\Gamma$	–	1–7	1–3,5–7
III$^+$		3.3.22	$\mathcal{U}_\Gamma \circ \mathcal{P}^*_\Gamma \circ [\mathcal{C}^\oplus_\Gamma]^k$	–	1–5	1–3,5
III$^-$		3.3.23	$\mathcal{P}^*_\Gamma \circ [\mathcal{C}^\ominus_\Gamma]^k$	–	1–5	1–3,5
IV	3.3.4.1	3.3.24	$\mathcal{N}_\Gamma \circ [\mathcal{D}^\circ_\Gamma]^k$	$\mathcal{P}_\Gamma$	1–5,8	1–3,5,8
IV$^+$		3.3.26	$\mathcal{N}_\Gamma \circ [\mathcal{D}^\oplus_\Gamma]^k$	$\mathcal{P}_\Gamma$	1–5,8–9	1–3,5,8–9
V		3.3.28	$\mathcal{N}_\Gamma \circ [\mathcal{P}^*_\Gamma \circ \mathcal{D}^\circ_\Gamma]^k \circ \mathcal{P}^*_\Gamma$	$\mathcal{P}_\Gamma$	1–8	1–3,5–8
V$^+$		3.3.31	$\mathcal{N}_\Gamma \circ [\mathcal{P}^*_\Gamma \circ \mathcal{D}^\oplus_\Gamma]^k \circ \mathcal{P}^*_\Gamma$	$\mathcal{P}_\Gamma$	1–9	1–3,5–9
VI	3.3.4.2	3.3.35	$[(\mathcal{PV})^*_\Gamma \circ \mathcal{D}^\circ_\Gamma]^k \circ (\mathcal{PV})^*_\Gamma$	–	1–8	1–3,5–8
VI$^+$		3.3.37	$[(\mathcal{PV})^*_\Gamma \circ \mathcal{D}^\oplus_\Gamma]^k \circ (\mathcal{PV})^*_\Gamma$	–	1–9	1–3,5–9
VI$^-$		3.3.37	$[(\mathcal{PV})^*_\Gamma \circ \mathcal{D}^\ominus_\Gamma]^k \circ (\mathcal{PV})^*_\Gamma$	–	1–8	1–3,5–8

Table 3.1: Summary of operational characterizations.

We observe that all successful coloring sequences enjoy properties 1-5. Properties 6 and 7 rely on exhaustive propagations, at least with operator $\mathcal{P}^*_\Gamma$. While Property 8 is guaranteed in all support-driven characterizations, Property 9 is only warranted when unicoloring with $\mathcal{D}^\oplus_\Gamma$. In fact, we see that exhaustive propagations moreover enforce that the respective properties (except for Property 4 and in one case Property 5) are also enjoyed by prefix sequences. That is, sequences satisfying the first two conditions, thus sharing the format $[O_2]^k \circ O_1$ for some combination of operators O_i for $i = 1, 2$. This is interesting from a computational point of view, since the more properties are enforced on partial colorings, the smaller is the overall search space. For brevity, we refrain from giving explicit theorems on prefix sequences (in addition to Theorem 3.3.18), since their proofs are obtained in a straightforward way.

Finally, let us summarize the computational complexity of the various operators.

Theorem 3.3.38 *Let Γ be the RDG of logic program Π and let C be a partial coloring of Γ.*
If n is the number of rules in Π, then

1. $\mathcal{P}_\Gamma(C)$ *is computable in* $\mathcal{O}(n)$,
2. $\mathcal{P}^*_\Gamma(C)$ *is computable in* $\mathcal{O}(n)$,
3. $\mathcal{U}_\Gamma(C)$ *is computable in* $\mathcal{O}(n)$,
4. $(\mathcal{PU})^*_\Gamma(C)$ *is computable in* $\mathcal{O}(n^2)$,
5. $\mathcal{V}_\Gamma(C)$ *is computable in* $\mathcal{O}(n)$,
6. $(\mathcal{PV})^*_\Gamma(C)$ *is computable in* $\mathcal{O}(n^2)$,
7. $\mathcal{N}_\Gamma(C)$ *is computable in* $\mathcal{O}(n)$.

In view of this result, we can decide in polynomial time whether a coloring sequence is in accord with a particular characterization. Hence, a successful coloring sequence can be generated in nondeterministic polynomial time.

3.4 Fitting and Well-founded Semantics

For relating Fitting's semantics and the well-founded semantics (see Section 2.1.2.2 on page 8) to the operators on *RDG*s, we need the following relationship between atom-based models and colorings.

Definition 3.4.1 *Let* Γ *be the RDG of logic program* Π *and let* C *be a partial coloring of* Γ.
We define

$$\begin{aligned} X_C &= \{head(r) \mid r \in C_\oplus\}, \\ Y_C &= \{q \mid \textit{for all } r \in \Pi, \textit{ if } head(r) = q, \textit{ then } r \in C_\ominus\}. \end{aligned}$$

The pair (X_C, Y_C) is a 3-valued interpretation of Π.
We have the following result.

Theorem 3.4.1 *Let* Γ *be the RDG of logic program* Π.
If $C = \mathcal{P}^*_\Gamma((\emptyset,\emptyset))$, *then* $\Phi^\omega_\Pi(\emptyset,\emptyset) = (X_C, Y_C)$.

Note that $\mathcal{P}^*_\Gamma((\emptyset,\emptyset))$ as well as $\Phi^\omega_\Pi(\emptyset,\emptyset)$ always exists (Cf. Corollary 3.3.3).
For the well-founded semantics, we fix the relationship among greatest unfounded sets and maximal support graphs.

Theorem 3.4.2 *Let* Γ *be the RDG of logic program* Π *and* C *be a partial coloring of* Γ *such that* $C_\ominus \subseteq \overline{S}(\Gamma, C) \cup B(\Gamma, C)$.
Furthermore, let (V, E) *be a maximal support graph of* (Γ, C) *for some* $E \subseteq \Pi \times \Pi$.
Then, $(Atm \setminus head(V))$ *is the greatest unfounded set of* Π *w.r.t.* (X_C, Y_C).

This result can be expressed in terms of operator $\mathcal{U}_\Gamma$.

Corollary 3.4.3 *Let* Γ *be the RDG of logic program* Π *and* C *be a partial coloring of* Γ *such that* $C_\ominus \subseteq \overline{S}(\Gamma, C) \cup B(\Gamma, C)$.
If $C' = \mathcal{U}_\Gamma(C)$, *then* $(Atm \setminus head(\Pi \setminus C'_\ominus))$ *is the greatest unfounded set of* Π *w.r.t.* (X_C, Y_C).

Finally, we can express the well-founded semantics in terms of our operators in the following way.

Theorem 3.4.4 *Let* Γ *be the RDG of logic program* Π.
If $C = (\mathcal{PU})^*_\Gamma((\emptyset,\emptyset))$, *then* (X_C, Y_C) *is the well-founded model of* Π.

Hence, by Theorem 3.3.38, the well-founded model is computable in our approach in quadratic time in size of Π.

3.5 Discussion and related work

The approach presented in this chapter has its roots in earlier work [142, 143], proposing *block graphs* as a tool for query-answering in default logic [166] and the underlying existence of extensions problem. Roughly speaking, these graphs are closely related to the blockage graphs introduced in Section 3.2 because both possess a single type of edges indicating blockage relations. Inspired by the distinction between supporting and blocking arcs made in [155], this has led to our approach to characterizing and computing answer sets; it provides the theoretical foundations of the `noMoRe` answer set programming system [4].

In this chapter, we put forward the simple concept of a rule dependency graph (*RDG*) for capturing the interplay between rules inducing answer sets.[17,18] Many other forms of dependency graphs can be found in the literature. For instance, dependency graphs (DGs) among predicate symbols were proposed in [6] for defining *stratified* programs. DGs among atoms can be defined analogously [163]: The nodes of such graphs are atoms appearing in a program Π; edges are distinguished similar to Definition 3.1.1, viz $E_0 = \{(head(r), p) \mid r \in \Pi, p \in body^+(r)\}$ and $E_1 = \{(head(r), p) \mid r \in \Pi, p \in body^-(r)\}$. Originally, these edges are referred to as being positive and negative, respectively. Accordingly, a cycle is said to be negative if it contains some negative edge. With this, a program is *stratified* if its DG does not contain a negative cycle. Stratified logic programs have a unique answer set [99]. Many other properties, obtained from the structure of the DG, were identified for investigating the consistency of Clark's completion [46]. In fact, it is shown in [94] that most of them also guarantee the existence of answer sets. As discussed in [52], DGs do not allow for capturing answer set semantics of logic programs. The difficulty is that there are syntactically and semantically different programs having the same DG.

Among the more recent literature, we find [72], where rule dependency graphs are defined for *reduced negative* programs. A program is *negative* if it includes only rules r where $body^+(r) = \emptyset$. Informally, a program is *reduced* if different rules $h_1 \leftarrow B, \ldots, h_k \leftarrow B$ with same body B are merged into one rule $h_1 \wedge \cdots \wedge h_k \leftarrow B$ where the head is a conjunction of atoms. When restricting our attention to negative programs with unique bodies, the graphs of [72] amount to *RDG*s restricted to 1-edges. The following interesting results are shown in [72] for reduced programs: Stable models [99], partial stable models [170], and well-founded semantics [190] of reduced negative programs correspond to kernels, semi-kernels and the initial acyclic part of the corresponding *RDG*, respectively. General programs are dealt with by program transformations, turning general programs to reduced negative ones.

Another interesting and closely related graph-theoretical approach is described in [39, 53, 54]. Although their primary focus lies on special negative programs, referred to as *kernel programs* (see below) their dependency graph can be defined in a general way. This approach relies on extended dependency graphs (*EDG*), whose nodes are given by the multi-set of rule heads together with atoms not appearing as heads. For a program Π, the set of vertices amounts to the set $\{(head(r), r) \mid r \in \Pi\} \cup \{(u, u) \mid u \notin head(\Pi)\}$. There is a positive edge (u, v) in the *EDG* if $u \in body^+(r)$ and $v = head(r)$ for some rule $r \in \Pi$; there is a negative edge (u, v) in the *EDG* if $u \in body^-(r)$. Hence *EDG*s and *RDG*s are generally different. For example, take program $\{r_a, r_b\} = \{a \leftarrow not\ b,\ b \leftarrow not\ a, not\ c\}$. Then the *EDG* of this program has three nodes a, b and c and three negative edges (b, a), (a, b) and (c, b), whereas the *RDG* has two nodes r_a and r_b and two edges (r_a, r_b) and (r_b, r_a). *Kernel programs* are negative programs subject to the condition that each head atom must also appear as a body atom in the program and all atoms in the program must be undefined in its well-founded model. According to [39], every normal program can be transformed into some equivalent kernel program. In analogy to [72], general programs are then dealt with through program transformations. Interestingly, it is shown in [53] that *EDG*s and *RDG*s are isomorphic for kernel programs. *EDG*s are used in [39] to study properties of logic programs, like existence of answer sets. Furthermore, colorings of *EDG*s are used in [39] for characterizing answer sets of kernel programs. For kernel programs,

[17] The definition of the *RDG* differs from that in [139], whose practically motivated restrictions turn out to be superfluous from a theoretical perspective.

[18] Due to the aforementioned historical development, the *RDG* was in [139, 141] still referred to as "block graph". We abandon the latter term in order to give the same status to support and blockage relations.

these colorings correspond to the ones studied in Section 3.2 (because of the aforecited isomorphism). Interestingly, [39] defines admissible colorings in terms of their "complements". That is, informally in our terminology, a coloring C is *non-admissible* if for some u, either (i) $u \in C_{\oplus}$, and for some $(u, v) \in E_1$, $v \in C_{\oplus}$, or (ii) $u \in C_{\ominus}$ and for all $(u, v) \in E_1$, $v \in C_{\ominus}$. Then, C is admissible if it is not non-admissible. This definition is only concerned with blockage and thus applies to negative programs only. On the other hand, it is closely related to the concept of a blockage graph (cf. Definition 3.2.3). That is, in the case of negative programs, the blockage graph amounts to an admissibly colored *EDG*. To see this, compare the negation of Condition 2 and 3 in Corollary 3.2.9 with Condition (i) and (ii) above (while setting $S(\Gamma, C)$ to Π). A blockage graph may thus be regarded as a natural extension of the concept of blockage used in [39] from negative to general logic programs.

In all, the major difference between the two latter approaches [72, 39] and ours boils down to the indirect or direct treatment of positive body atoms, respectively. While our techniques are developed for full-fledged logic programs, [72, 39] advocate an initial transformation to negative programs, on which their methods are primarily defined. (Note that general logic programs cannot be reduced to negative ones in a modular way [112].) Rather all our characterizations of answer sets in Section 3.2 stress the duality between supporting and blocking relations among rules. Finally, it is noteworthy that all three approaches address several rather different problems. While we are primarily interested in operational characterizations, the two other approaches address fundamental problems such as the existence of answer sets. In particular, they elaborate upon the graph-theoretical concept of negative programs. In this way, all approaches are nicely complementary to each other and therefore do largely benefit from each other. An overview over different graphs associated with logic programs can be found in [52].

We have introduced support graphs for capturing the inferential dependencies among rule heads and positive body atoms. In fact, support graphs may be seen as a (rule-oriented) materialization of the notion of a *well-supported* interpretation, or more precisely, its underlying well-founded partial order (cf. [94]): An interpretation X is well-supported if there is a strict well-founded partial order $\prec$ on X such that for every $p \in X$ there is some $r \in R_{\Pi}(X)$ with $p = head(r)$ and $q \prec p$ for every $q \in body^+(r)$. An edge (r_1, r_2) in a support graph of a colored *RDG* corresponds to the pairs $head(r_2) \prec head(r_1)$ for $q \in body^+(r_2)$.

A major goal of this work is to provide operational characterizations of answer sets that allow us to bridge the gap between formal yet static characterizations of answer sets and algorithms for computing them. For instance, in the seminal paper [150] describing the approach underlying the `smodels` system, the characterization of answer sets is given in terms of so-called *full-sets* and their computation is directly expressed in terms of procedural algorithms. Our operational semantics aims at offering an intermediate stage that facilitates the formal elaboration of computational approaches. Our approach is strongly inspired by the concept of a derivation, in particular, that of an SLD-derivation [144]. This attributes our coloring sequences the flavor of a derivation in a family of calculi, whose respective set of inference rules correspond to the selection of operators. A resolution calculus for skeptical stable model semantics is given in [17]. Interestingly, this calculus is not derived from credulous inference; also, it does not need the given program to be instantiated before reasoning. Gentzen-style calculi for default logic (and thus implicitly also for logic programming) can be found in [15, 18].

Regarding our modeling of operations, it is worth mentioning that one could also use total operations instead of partial ones. For instance, instead of defining $\mathbb{C}$ as a set of partial mappings, one could consider the set of a binary partitions of Π *plus* (Π, Π) as the representative for inconsistent colorings. For instance, $\mathcal{P}_\Gamma$ could then be defined as a mapping $\mathcal{P}_\Gamma : \mathbb{C} \to \mathbb{C} \cup \{(\Pi, \Pi)\}$, where (Π, Π) is obtained whenever $\mathcal{P}_\Gamma$ "detects" an inconsistency. Although such an approach seems natural in a logical setting, involving deductive closure, we put forward partial mappings in our abstract operational setting. In this way, an answer set exists iff there *exists* a corresponding coloring sequence. Accordingly, there is no answer set iff there is no coloring sequence. This nicely corresponds to the concept of a derivation and notably avoids the distinction between coloring sequences leading to admissible and inconsistent colorings.

We have furthermore shown in Section 3.4 that particular operations correspond to Fitting's and well-founded semantics [95, 190, 164, 69]. A stepwise characterization of both semantics was proposed in [26]

by defining a confluent rewriting system. The rewrite of a program corresponds to a 3-valued interpretation in which all facts in the rewritten program are true and all atoms not appearing among the heads of the rewrite are false. All other atoms are undefined. The program transformations $\mapsto_P$ and $\mapsto_S$ delete atoms from the positive and negative part of the body, if they are true in the associated 3-valued interpretation. If a rule is transformed into a fact through these transformations, then this corresponds in our approach to coloring this rule with $\oplus$ by $\mathcal{P}_\Gamma$. Analogously, the transformations $\mapsto_N$ and $\mapsto_F$ delete rules which have atoms in their body being false in the 3-valued interpretation. These transformations correspond to coloring non-applicable rules with $\ominus$. In this way, the iterative application of these 4 transformations yields the least fixpoint of Fitting's operator. In view of Theorem 3.4.1, these 4 transformations have the same effect as operator $\mathcal{P}_\Gamma$. Another program transformation, viz $\mapsto_L$, is introduced in [26] for 0-loop detection; thus allowing for computing the greatest unfounded set. This transformation is similar to operator $\mathcal{U}_\Gamma$. Taken together, $\mapsto_P, \mapsto_S, \mapsto_N, \mapsto_F, \mapsto_L$ form a confluent rewriting system, whose final rewrite corresponds to the well-founded model of the initial logic program.

The operational characterization, presented in this chapter, establishes the basics for the `nomore++` system [152, 3, 2], the successor of `noMoRe`. Instead of rule dependency graphs, the `nomore++` system treats heads and bodies equitably as computational objects. Analogously to colorings or rule dependency graphs, body-head dependency graphs were defined and partial mappings for heads and bodies reflecting applicability and non-applicability of heads and bodies. The propagation operators $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$ have been adapted to heads and bodies and extended to backward propagation. Choices are also support-driven, but restricted on bodies. Furthermore, `nomore++` has been extended by a hybrid lookahead strategy, incorporating heads and bodies of rules.

Although we leave algorithmic and implementation issues to further work, in particular those, dealing with our system `noMoRe`, some remarks relating our approach to the ones underlying the answer set programming systems `DLV` [76, 73, 130] and `smodels` [181, 150, 180] are in order. A principal difference manifests itself in how choices are performed. While the two latter's choice is based on atoms occurring (negatively) in the underlying program, our choices are based on its rules. The former approach is per se advantageous whenever multiple rules share a common head. This is compensated in the `noMoRe` system by additional propagation rules eliminating a rule from the inference process, once its head has been derived in an alternative way. From a general perspective, a rule-based choice can be regarded as a compound choice on atoms. That is, assigning a rule r a positive applicability status (via $\oplus$) corresponds to assigning all atoms in $head(r) \cup body^+(r)$ the value true and all atoms in $body^-(r)$ the value false. Conversely, assigning r a negative status of applicability (viz $\ominus$) corresponds to assigning at least one atom in $body^+(r)$ the value false or one in $body^-(r)$ the value true. Only if all rules with the same head are colored with $\ominus$, a rule head can be assigned false. While this type of choice is realized by $\mathcal{C}^\circ_\Gamma$, the one by $\mathcal{D}^\circ_\Gamma$ is more restrictive since all atoms in $body^+(r)$ are known to be true. An advantage of the approach based on choice operator $\mathcal{D}^\circ_\Gamma$ is that we can guarantee the support of rules on the fly. The elimination of unsupported rules can then either be restricted to uncolored rules, as done with operator $\mathcal{V}_\Gamma$, or even done in a final step without further detection efforts by appeal to operator $\mathcal{N}_\Gamma$. Interestingly, the choice operator of `DLV` has also a support-driven flavor: When choosing a (negative) body literal q, one of the qualifying conditions is the existence of a rule r such that $q \in body^-(r)$ and $body^+(r) \subseteq I$, where I is the current partial assignment.[19,20] Unlike this, support checking is a recurring operation in the `smodels` system, similar to operator $\mathcal{U}_\Gamma$. On the other hand, this approach ensures that the `smodels` algorithm runs in linear space complexity, while a graph-based approach needs quadratic space in the worst case (due to its number of edges). Interestingly, `smodels`' implementation relies on a rule-head dependency graph, in which rules and atoms are connected via pointers. Such an investment in space pays off once one is able to exploit the additional structural information offered by a graph. First steps in this direction are made in [140], where

[19] Formally, I is a four-valued interpretation. Two further conditions qualify the (disjunctive) head and the negative body literals of the rule r depending on their truth values.

[20] No support is taken into account by `DLV` when choosing a (positive) head literal.

graph compressions are described that allow for conflating entire subgraphs into single nodes. Propagation is more or less done similarly in all three approaches. That is, all these systems follow the strategy coined in [184], namely "Answer Sets = Well-founded Semantics + Branch and Bound". `smodels` relies on computing well-founded semantics, whereas `DLV` uses Fitting's or well-founded semantics, depending on whether (in our terminology) the program's *RDG* contains 0-cycles or not [43]. Also, both systems use back-propagation mechanisms. In `DLV`, this allows to mark atoms as being eventually true. Operators capturing `DLV`'s propagation operations are given in [87]. Among them, operator T_Π amounts to our Operator $\mathcal{P}_\Pi$; others address the aforementioned back-propagation and propagation of "eventually true" atoms. In addition to the propagation operators discussed in Section 3.3, `noMoRe` also uses different types of back-propagation [141], including special treatment of integrity constraints, as well as operations for ignoring rules once their head has been established. What truly distinguishes `noMoRe`'s propagation operations is their support-preserving way in conjunction with choice operator $\mathcal{D}^\circ_\Gamma$ (cf. Property 8 in Theorem 3.3.25, 3.3.27, 3.3.30, 3.3.32, and 3.3.36).

	noMoRe									smodels		
	V			VI			VI+h_{sm}					
n	chs	ass	time	chs	ass	time	chs	ass	time	chs	ass	time
3	1	39	0.0	1	39	0.0	1	175	0.0	1	45	0.001
4	13	240	0.0	12	241	0.01	5	710	0.03	5	381	0.0
5	74	1430	0.06	64	1420	0.08	23	3918	0.23	26	2757	0.002
6	468	9935	0.59	385	9687	0.71	119	24046	1.88	305	34202	0.019
7	3370	78803	5.77	2676	75664	6.88	719	0.18M	16.98	4814	0.53M	0.319
8	27480	0.70M	61.84	21259	0.67M	72.99	5039	1.47M	173.81	86364	9.17M	6.29
9	0.25M	6.97M	730	0.19M	6.55M	849	40319	14M	3639	1.86M	197M	159

Table 3.2: Results for computing **all** answer sets of the Hamiltonian cycle problem on complete graphs with n nodes.

	noMoRe									smodels		
	V			VI			VI+h_{sm}					
n	chs	ass	time	chs	ass	time	chs	ass	time	chs	ass	time
7	15	386	0.03	15	386	0.04	5	1075	0.13	30	4701	0.005
8	21	639	0.06	21	639	0.07	6	1626	0.22	8	2941	0.003
9	28	1000	0.11	28	1000	0.14	7	2368	0.4	48	12555	0.009
10	36	1494	0.19	36	1494	0.24	8	3337	0.75	1107	193287	0.155
11	45	2148	0.31	45	2148	0.57	9	4571	1.45	18118	2.81M	2.613
12	55	2991	0.53	55	2991	1.01	10	6110	2.47	0.39M	56.6M	60
13	66	4054	1.13	66	4054	1.62	11	7996	3.87	5.30M	721M	866
14	78	5370	1.92	78	5370	2.47	12	10273	5.62	—	—	>2h
15	91	6974	2.83	91	6974	3.47	13	12987	8.06	—	—	>2h
16	105	8903	4.01	105	8903	4.86	14	16186	11.07	—	—	>2h
17	120	11196	5.49	120	11196	6.58	15	19920	14.79	—	—	>2h
18	136	13894	7.15	136	13894	8.62	16	24241	19.74	—	—	>2h

Table 3.3: Results for computing **one** answer set of the Hamiltonian cycle problem on complete graphs with n nodes

Finally, let us underpin the potential of our approach by some indicative empirical results. For this purpose, we have chosen the Hamiltonian cycle problem on two different types of graphs, namely *complete* and so-called *clumpy graphs*, as put forward in [195]. This choice is motivated by the fact that — unlike most of the other known benchmark examples — Hamiltonian problems naturally lead to non-tight encodings and thus comprise more complex support structures.[21] Each clumpy graph has a given number of clumps

[21] Tight encodings for Hamiltonian problems were proposed in [137].

	noMoRe									smodels		
	V			VI			VI+h_{sm}					
n	chs	ass	time	chs	ass	time	chs	ass	time	chs	ass	time
4	289	5132	0.22	56	1357	0.09	8	2006	0.16	2	1532	0.001
4	18	395	0.02	14	332	0.02	2	515	0.03	1	735	0.001
4	396	5468	0.29	179	3776	0.28	11	1834	0.18	18	10393	0.007
4	22	366	0.02	21	364	0.02	6	1000	0.08	3	1489	0.002
4	118	2223	0.14	47	1227	0.11	13	2763	0.29	20	13525	0.008
5	3765	54264	3.06	37	733	0.08	14	2721	0.29	6	4291	0.004
5	1113	13535	1.08	93	1779	0.22	32	4929	1.1	2306	1.08M	0.775
5	207	3340	0.17	74	1450	0.14	11	2611	0.29	4	2956	0.003
5	1195	14780	1.14	191	5390	0.79	29	7773	1.64	201	98546	0.069
5	4535	72129	4.91	505	15690	2.12	54	17425	3.66	82	60317	0.038
6	359	6563	0.52	89	2174	0.47	346	76915	25.46	0.17M	95M	76
6	1228	20970	1.64	261	6238	1.38	4419	1.06M	335	0.24M	214M	152
6	1.71M	33M	3278	0.10M	3.18M	935	12608	4.2M	1537	–	–	>2h
6	233	4937	0.38	158	3908	0.96	1161	0.27M	94	14	11841	0.01
6	3237	41286	2.78	499	8336	1.94	57	9162	2.62	38	38277	0.025

Table 3.4: Results for computing **one** answer set of the Hamiltonian cycle problem on clumpy graphs with n clumps and different instances

(sets of nodes) where nodes are connected with more edges than between clumps. That is, edges in clumpy graphs are distributed less uniform and solving Hamiltonian cycle problems becomes more difficult. All benchmark examples are included in the distribution of the noMoRe system [153].

All presented experiments have been done under Linux (kernel 2.6) on a Intel Pentium 4 processor with 2.26GHz and 512MB main memory. We have used noMoRe V1.0 [153] under Eclipse Prolog with *RDG*s (flag asp_r set) and smodels version 2.27 [181]. Although we do not report it here, we have run the whole test series with DLV as well. We have not included the results because DLV outperforms smodels as well as noMoRe on all problem instances as regards time. Furthermore, DLV does not report assignments and it has a different concept of choices than smodels and noMoRe.

Tables 3.2 and 3.3 summarize the results for computing *all* and *one* answer set for Hamiltonian cycle problems on complete graphs with n nodes, respectively. Each table reports the number of choices (chs), the number of assignments[22] (ass) and the consumed time in seconds (time) for each operational characterization of the noMoRe and smodels systems. M abbreviates millions. Observe that, although the Hamiltonian cycle problem for complete graphs is easy solvable for humans, it is difficult for ASP solvers. The computation of all solutions reflects the system behavior on excessive backtracking.[23] Table 3.4 gives results for computing Hamiltonian cycles on clumpy graphs with n clumps; we have tested five different instances for each n. The first two operational characterizations V and VI correspond to the respective rows in Table 3.1, whereas VI+h_{sm} indicates a modified version of VI comprising an smodels-like heuristic including lookahead. Observe that we compare a Prolog and a C++ implementation and thus the resulting time measurements should not be overrated. Instead the number of needed choices and assignments give a better indication for the relation of noMoRe and smodels. Concerning choices and assignments, Table 3.2 shows that for computing all Hamiltonian cycles of complete graphs the noMoRe strategy VI+h_{sm} performs better than smodels. Further evidence for this is given in Table 3.3 where we are able to observe this phenomenon even by looking at time measurements (for $n \leq 11$).

The results for clumpy graphs in Table 3.4 demonstrate that noMoRe behaves more uniform on those examples than smodels. For examples in the first three instances with six clumps noMoRe needs at

[22] For each change of the truth value of a atom and for each change of the color of a node in a *RDG* one assignment is counted in smodels and noMoRe, respectively.

[23] We also ran test series on non-answer-set instances, namely, Hamiltonian cycle problems on bipartite graphs; the overall result was the same as obtained with the series reported here.

most 4419 choices whereas smodels needs at least 170000 choices. Furthermore, on the third instance smodels did not even get an answer set after more than two hours. On the other hand, even if there are examples where smodels needs less choices than noMoRe we did not recognize such extreme outliers for noMoRe as we did for smodels during our tests.

Comparing the three noMoRe strategies among each other, we observe that the extension of strategy VI by an smodels-like heuristics including lookahead usually decreases the number of choices (except for clumpy graphs with $n = 6$). However, this does not necessarily lead to better performance in time, since the lookahead increases the number of assignments. Similarly, we observe that strategy VI always makes fewer choices than strategy V. Again, this advantage does not always pay off as regards time, since the verification of support by operator $\mathcal{V}_\Gamma$ is more time consuming than applying operator $\mathcal{N}_\Gamma$.

We stress that these experiments have a purely indicative character; a systematic experimental evaluation is left for further work. Nonetheless our experiments show prospects insofar as even a moderately optimized Prolog implementation of our strategies outperforms a state-of-the-art system on certain benchmarks. Even though many other benchmark problems are still solved much faster by the state-of-the-art solvers, a comparison of the number of choices reveals that our approach has no substantial disadvantages.

The fact that DLV deals with disjunctive programs makes many of its special features inapplicable in our setting of normal logic programs. Alternative approaches can be found in [138, 47], where answer sets are computed by means of SAT solvers. Algorithms and implementation techniques for computing well-founded semantics can be found among others in [171, 145].

3.6 Conclusion

We have elaborated upon rule dependency graphs (*RDG*s) and their colorings for characterizing and computing answer sets of logic programs. While *RDG*s determine the possible interplay among rules inducing answer sets, its colorings fix their concrete application status.

We have started by identifying graph structures that capture structural properties of logic programs and their answer sets. As a result, we obtain several characterizations of answer sets in terms of totally colored dependency graphs. All characterizations reflect the dichotomy among the notions of support and blockage. In fact, once a "recursive support" is established, this dichotomy allows for characterizing answer sets in terms of their generating or their non-generating rules. The notion of "recursive support" is captured by the graph-theoretical concept of a support graph, whose counterpart is given by the blockage graph. Unlike the basic set-theoretic concepts, these subgraphs do not reflect the aforementioned dichotomy in a fully symmetric way. This is because support graphs capture a global — since recursive — structure, whilst blockage graphs aim at a rather local structure, based on arc-wise constraints. Taken together, both subgraphs provide another characterization of answer sets. Interestingly, their existence is incrementally enforced whenever appropriate propagation operations are used during the coloring process.

To a turn, we build upon these basic graph-theoretical characterizations for developing an operational framework for non-deterministic answer set formation. The goal of this framework is to offer an intermediate stage between declarative characterizations of answer sets and corresponding algorithmic specifications. We believe that this greatly facilitates the formal elaboration of computational approaches. The general idea is to start from an uncolored *RDG* and to employ specific operators that turn a partially colored graph gradually in a totally colored one, finally representing an answer set. To this end, we have developed a variety of deterministic and non-deterministic operators. Different coloring sequences (enjoying different formal properties) are obtained by selecting different combinations of operators. Among others, we distinguish unicoloring and support-driven operational characterizations. In particular, we have identified the basic strategies employed by the noMoRe system as well as operations yielding Fitting's and well-founded semantics. Taken together, the last results show that noMoRe's principal propagation operation amount to applying Fitting's operator, when using strategy IV in Table 3.1 or computing well-founded semantics, when applying strategy VI. Notably, the explicit detection of 0-loops can be avoided by employing

a support-driven choice operation. More recent developments within noMoRe, such as back-propagation, heuristics, and implementation details are left for further work. Generally speaking, noMoRe is conceived as a parametric system that allows for choosing different strategies. The noMoRe system is available at http://www.cs.uni-potsdam.de/~linke/nomore.

In fact, our operational framework can be seen as a "theoretical toolbox" that allows for assembling specific strategies for answer set formation. The algorithmic realization of our coloring sequences in terms of backtracking algorithms is rather straightforward. The interesting step is of course the implementation of choice operators. In principle, a choice is made from a set of rules, each of which may be attributed a (different) color. This leaves room for different implementations, inducing differently shaped search trees. A prototypical platform offering the described spectrum of operations can be downloaded at http://www.cs.uni-potsdam.de/~konczak/system/gcasp. In all, our elaboration has laid the basic formal foundation for computing answer sets by means of *RDG*s and their colorings. Current and future work mainly deals with further exploitation of the structural information offered by a graph-based approach. In Chapter 4, we show how preferences among rules are easily incorporated as a third type of edges. Other work includes graph compressions allowing for collapsing entire subgraphs into single nodes [140]. Last but not least, our approach seems to be well-suited for debugging and profiling purposes. First, given that it relies on rules, the objects of computation are the same as the descriptive objects within the problem specification. Second, the underlying graph allows for a very natural visualization of computations. This has already led to a noMoRe-specific profiler, described in [19].

Chapter 4

Ordered Programs in Answer Set Programming

On different lines of ASP research, many extensions of the basic formalism have been proposed. Perhaps the most intensively studied one is the modeling of preferences in ASP, cf. [65], Section 2.4 on page 13, and Section 4.4 on page 70. Strongly rooted in the research of nonmonotonic formalisms, the ability to specify preferences is acknowledged to be particularly beneficial to ASP, since they constitute a very natural and effective way of resolving indeterminate solutions. Up to now, these preference semantics where incorporated into answer set solvers either by meta-interpretation [77] or by compilation methods [62, 161]; therefore, preferences were never integrated into the solvers themselves. This is where our contribution comes in. In Section 4.1, we extend the graph-based approach presented in Chapter 3 by preference information. We provide an operational characterization, where preferences are directly integrated in the computation of preferred answer sets.

In Section 4.2, we provide new benchmarks for logic programs with preferences, which are used for experiments in the following Section 4.3. There, we present the new `nomore`$^<$ (pronunciation: nomorepref) system, a C++ implementation of the graph-based approach described in Section 4.1. One of the main motivations for integration preferences into ASP solver is the question whether an integrative approach is better than a compilation method. Therefore, we compare the `nomore`$^<$ system with a meta-interpreter and the `plp` system. More precisely, we compare the integration of preference information into an ASP solver with compilation methods for preferences acting as front-ends for ASP solvers.

Section 4.4 contains an overview of currently existing semantics for preferences in answer set programming. We conclude this chapter in Section 4.5.

4.1 Graphs and Colorings with Preferences

Up to now, the rule preferences described in Section 2.4 on page 13 where incorporated into answer set solvers either by meta-interpretation [77] or by pre-compilation front-ends [62]; therefore, preferences were never integrated into the solvers themselves. This is where our contribution comes in. We argue that the graph-based approaches presented in Chapter 3 provide an appropriate model for integrating preferences into answer set programming and the corresponding solvers. We underpin this claim, first, by showing how three among the most prominent preference handling approaches presented in Section 2.4 on page 13 can be characterized by graph-oriented methods in a uniform way and, second, by showing how this can be realized by means of an operational semantics. This is usable for extending graph-based answer set solvers, such as `noMoRe` [4]. Following Chapter 3, our idea is to start from an uncolored rule dependency graph and to employ specific operators that turn a partially colored graph gradually into a totally colored one that

represents a preferred answer set. Accordingly, a program has a certain preferred answer set iff there is a sequence of operations turning the uncolored graph into a totally colored one, expressing the answer set.

4.1.1 Graphs and colorings with preferences

On page 20 in Section 3.1, we have defined rule dependency graphs for normal logic program. Next, we extend this definition to ordered programs by adding a third type of edges reflecting dependencies among rules.

Definition 4.1.1 *Let $(\Pi, <)$ be an ordered logic program. The ordered rule dependency graph (RDG) $\Psi_{(\Pi,<)} = (\Pi, E_0, E_1, E_2)$ of $(\Pi, <)$ is a labeled directed graph with*

$$\begin{aligned} E_0 &= \left\{(r, r') \mid r, r' \in \Pi, \mathit{head}(r) \in \mathit{body}^+(r')\right\}; \\ E_1 &= \left\{(r, r') \mid r, r' \in \Pi, \mathit{head}(r) \in \mathit{body}^-(r')\right\}; \\ E_2 &= \left\{(r, r') \mid r, r' \in \Pi, r' < r\right\}. \end{aligned}$$

For illustration, consider the ordered program $(\Pi_{4.1}, <) = (\{r_1, \ldots, r_4\}, <)$, where:

$$\begin{array}{llllll} r_1: & p & \leftarrow & & r_3: & f \leftarrow b, not\ f' \\ r_2: & b & \leftarrow & p & r_4: & f' \leftarrow p, not\ f \end{array} \qquad r_3 < r_4 \tag{4.1}$$

Among the two standard answer sets of $\Pi_{4.1}$, $\{p, b, f\}$, and $\{p, b, f'\}$, the preference $r_3 < r_4$ selects the latter. That is, $AS^D((\Pi_{4.1}, <)) = \{\{p, b, f'\}\}$.[1] The *RDG* of $(\Pi_{4.1}, <)$ is depicted in Figure 4.1a.

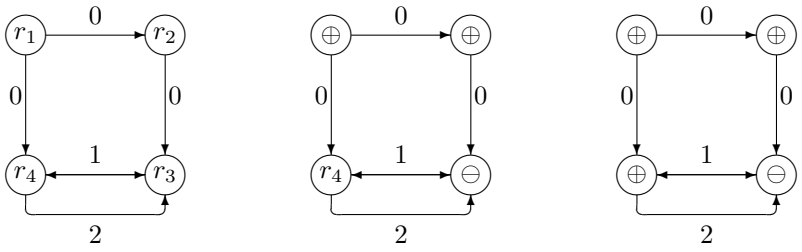

Figure 4.1: **(a)**: The *RDG* of ordered program $(\Pi_{4.1}, <)$; **(b)**: The (partially) colored *RDG* $(\Psi_{(\Pi_{4.1},<)}, C_{4.2})$; **(c)** The totally colored *RDG* $(\Psi_{(\Pi_{4.1},<)}, C_{4.4})$.

The definition of colorings on rule dependency graphs for ordered programs does directly carry over from standard programs (see on page 20, Section 3.1). For example, "coloring" the *RDG* of $(\Pi_{4.1}, <)$ with

$$C_{4.2} = (\{r_1, r_2\},\ \{r_3\}) \tag{4.2}$$

yields the colored graph given in Figure 4.1b. The central question addressed in this chapter is how to compute the total colorings of *RDG*s that correspond to the preferred answer sets of an underlying program. In fact, the colorings of interest can be distinguished in a straightforward way. Following the approach presented in Chapter 3, we associate with each answer set an admissible coloring via the set of generating rules. Let $(\Pi, <)$ be an ordered logic program along with its *RDG* Ψ. Analogously, for every $<^\sigma$–preserving answer set X of Π, where $\sigma \in \{D, B, W\}$, we define an *$<^\sigma$–preserving admissible coloring* C of Ψ as

$$C = (R_\Pi(X), \Pi \setminus R_\Pi(X)).$$

[1] Note that actually $AS^\sigma((\Pi_{4.1}, <)) = \{\{p, b, f'\}\}$ for every $\sigma \in \{D, B, W\}$.

In this way, we associate with any program a set of $<^\sigma$–preserving admissible colorings whose members are in one-to-one correspondence with its $<^\sigma$–preserving answer sets. Clearly, any $<^\sigma$–preserving admissible coloring is total; also, we have $X = head(C_\oplus)$. We use $AC^\sigma((\Pi, <))$ for denoting the set of all $<^\sigma$–preserving admissible colorings of a *RDG* $\Psi_{(\Pi,<)}$. For a partial coloring C, we define

$$AC^\sigma_{(\Pi,<)}(C) = \{C' \in AC^\sigma((\Pi, <)) \mid C \sqsubseteq C'\} \tag{4.3}$$

as the set of all $<^\sigma$–preserving admissible colorings of $\Psi_{(\Pi,<)}$ compatible with C. [2] Clearly, $C_1 \sqsubseteq C_2$ implies $AC^\sigma_{(\Pi,<)}(C_1) \supseteq AC^\sigma_{(\Pi,<)}(C_2)$. Observe that a partial coloring C is extensible to a $<^\sigma$–preserving admissible one C' (that is $C \sqsubseteq C'$) iff $AC^\sigma_{(\Pi,<)}(C)$ is non-empty. For a total coloring C, $AC^\sigma_{(\Pi,<)}(C)$ is either empty or singleton. Regarding program $(\Pi_{4.1}, <)$ and coloring $C_{4.2}$, we get

$$AC^\sigma_{(\Pi_{4.1},<)}(C_{4.2}) = AC^\sigma((\Pi_{4.1}, <)) = \{(\{r_1, r_2, r_4\}, \{r_3\})\}, \tag{4.4}$$

for $\sigma \in \{D, B, W\}$. Accordingly, we define $AS^\sigma_{(\Pi,<)}(C)$ as the set of all *$<^\sigma$–preserving answer sets* X of $(\Pi, <)$ compatible with partial coloring C:

$$AS^\sigma_{(\Pi,<)}(C) = \{X \in AS^\sigma((\Pi, <)) \mid C_\oplus \subseteq R_\Pi(X) \text{ and } C_\ominus \cap R_\Pi(X) = \emptyset\}. \tag{4.5}$$

Note that $head(C_\oplus) \subseteq X$ for any $<^\sigma$–preserving answer set $X \in AS_\Pi(C)$. As regards program $(\Pi_{4.1}, <)$ and coloring $C_{4.2}$, we get $AS^\sigma_{(\Pi_{4.1},<)}(C_{4.2}) = \{\{b, p, f'\}\}$. We call a coloring simply *admissible*, if X is an answer set of Π. Also, if X is an answer set of Π, we omit the superscript σ in the above defined sets. Furthermore, we use $AC_{(\Pi,<)}(C)$ for denoting the set of all admissible colorings of $\Psi_{(\Pi,<)}$ compatible with C.

Analogously to Definition 3.1.2 on page 22, we need the following concepts for describing a rule's status of applicability.

Definition 4.1.2 *Let $\Psi = (\Pi, E_0, E_1, E_2)$ be the RDG of ordered program $(\Pi, <)$ and C be a partial coloring of Ψ. For $r \in \Pi$, we define:*

1. *r is supported in (Ψ, C), if $body^+(r) \subseteq \{head(r') \mid (r', r) \in E_0, r' \in C_\oplus\}$;*
2. *r is unsupported in (Ψ, C), if $\{r' \mid (r', r) \in E_0, head(r') = q\} \subseteq C_\ominus$ for some $q \in body^+(r)$;*
3. *r is blocked in (Ψ, C), if $r' \in C_\oplus$ for some $(r', r) \in E_1$;*
4. *r is unblocked in (Ψ, C), if $r' \in C_\ominus$ for all $(r', r) \in E_1$;*
5. *r is maximal in (Ψ, C) if $\{r' \mid (r', r) \in E_2\} \subseteq (C_\oplus \cup C_\ominus)$.*

Condition 1–4 are analogous to those in Definition 3.1.2 on page 22. The concept expressed in Condition 5 allows for distinguishing rules, all of which more preferred rules have either been found to be applicable or blocked. Such rules are *maximal* insofar as they are not dominated by any preferred rules having an undecided status of applicability.

In what follows, we use $S(\Psi, C)$, $\overline{S}(\Psi, C)$, $B(\Psi, C)$, $\overline{B}(\Psi, C)$, and $M(\Psi, C)$ for denoting the sets of all supported, unsupported, blocked, unblocked, and maximal rules in (Ψ, C), respectively. For illustration, consider the sets obtained regarding $(\Psi_{(\Pi_{4.1},<)}, C_{4.2})$, given in Figure 4.1b.

$$\begin{array}{llll} S(\Psi_{(\Pi_{4.1},<)}, C_{4.2}) & = & \{r_1, r_2, r_3, r_4\} & \overline{S}(\Psi_{(\Pi_{4.1},<)}, C_{4.2}) = \emptyset \\ B(\Psi_{(\Pi_{4.1}, <)}, C_{4.2}) & = & \emptyset & \overline{B}(\Psi_{(\Pi_{4.1}, <)}, C_{4.2}) = \{r_1, r_2, r_4\} \\ M(\Psi_{(\Pi_{4.1},<)}, C_{4.2}) & = & \{r_1, r_2, r_4\} & \end{array} \tag{4.6}$$

Rule r_3 is not maximal in $(\Psi_{(\Pi_{4.1},<)}, C_{4.2})$ because the higher preferred rule r_4 is uncolored and thus not known to be blocked or applied.

[2] Regarding notation, observe that $AC^\sigma((\Pi, <))$ stands for the admissible colorings associated with program $(\Pi, <)$, while $AC^\sigma_{(\Pi,<)}(C)$ stands for admissible colorings associated with partial coloring C.

4.1.2 Deciding preferred answersetship

We now develop concepts that allow us to decide whether a (total) coloring represents an order preserving admissible coloring by purely graph-theoretical means. For this purpose, we use the definition of support graphs and admissible colorings from Section 3.2 (Definition 3.2.2 and Theorem 3.2.4), which are transferred to rule dependency graphs for ordered programs.

For capturing preferences, we propose the concept of a *height function*. To begin with, we develop this concept for the D–semantics.

Definition 4.1.3 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$, C be a total coloring of Ψ and let (V, E_0, E_1, E_2) be a subgraph of Ψ.*

We define a D–height function of (V, E_0, E_1, E_2) as a function $h : V \to \mathbb{N}$ such that for all $r \in V$, we have

1. *if $(r', r) \in E_2$, then $h(r') < h(r)$,*
2. *if $(r', r) \in E_0$ and $r, r' \in C_\oplus$, then $h(r') < h(r)$, and*
3. *if $r \in C_\ominus \cap V$, then there exists an $r' \in C_\oplus$ and $(r', r) \in E_1$ such that $h(r') < h(r)$.*

The values attributed by a height function reflect a possible order of rule consideration (not necessarily application). Rules with a lower h-value must be considered before rules with a higher h-value. In this respect, Condition 1 stipulates that higher ranked rules must be considered before lower ranked rules; in this way, h respects the preferences from $<$.[3] If (V, E_0) forms a support graph of (Ψ, C), then Condition 2 ensures that rules are never supported by rules having a greater h-value. Condition 3 expresses that rules colored with $\ominus$, must be blocked by rules with a smaller h-value (that is, intuitively, already applied rules). It is instructive to observe that every height function induces a partial order on Π, extending the given partial order $<$. Furthermore, this induced order is always extensible to a total order of Π (cf. Definition 2.4.1 on page 13).

Taking the concept of a D–height function together with Theorem 3.2.4 on page 26 for admissible colorings, we obtain a characterization of $<^D$–preserving admissible colorings.

Theorem 4.1.1 *Let $\Psi = (\Pi, E_0, E_1, E_2)$ be the RDG of ordered logic program $(\Pi, <)$ and C be a total coloring of Ψ.*

Then, C is a $<^D$–preserving admissible coloring iff

1. *$C_\oplus = S(\Psi, C) \cap \overline{B}(\Psi, C)$ and*
2. *for some $E_0' \subseteq E_0$, we have*
   (a) *$(C_\oplus, E_0')$ is a support graph of (Ψ, C) and*
   (b) *there exists a D–height function of $(S(\Psi, C), E_0', E_1|_{S(\Psi,C)}, E_2|_{S(\Psi,C)})$.*

Conditions 1 and 2a are the ones found on page 26 in Theorem 3.2.4 for standard admissible colorings, while Condition 2b selects the $<^D$–preserving ones by means of a D–height function. For this, only supported rules are taken into account; unsupported rules are inapplicable anyway. Now, the height function ties the arcs in E_0' of the support graph to the ones reflecting blockage $E_1|_{S(\Psi,C)}$ and preference $E_2|_{S(\Psi,C)}$. This guarantees that the underlying answer set can be formed in an order preserving way.

For illustration, consider $(\Pi_{4.1}, <)$. For the admissible coloring $(\{r_1, r_2, r_4\}, \{r_3\})$, we have the following D–height function h:

$$h(r_1) = 1,\ h(r_2) = 2,\ h(r_3) = 4,\ h(r_4) = 3. \tag{4.7}$$

[3] That is, higher ranked rules have higher values.

For admissible coloring $(\{r_1, r_2, r_3\}, \{r_4\})$, there is no D–height function because $r_3 < r_4$ and the blockage of r_4 by r_3 lead to a contradiction between Condition 1 and 3 in Definition 4.1.3. Hence, only $(\{r_1, r_2, r_4\}, \{r_3\})$ is $<^D$–preserving and $\{p, b, f'\}$ is the only $<^D$–preserving answer set.

An alternative D–height function for the admissible coloring $(\{r_1, r_2, r_4\}, \{r_3\})$, could be $h(r_1) = 42,\ h(r_2) = 98,\ h(r_3) = 101,\ h(r_4) = 99$. Because of the simplicity of program $(\Pi_{4.1}, <)$, all resulting D–height functions induce a total order on the rules of $\Pi_{4.1}$. A partial order is possible, if we replace r_2 in $\Pi_{4.1}$ by $b \leftarrow$.

For B– and W–preferences, we can define height functions in an analogous way.

Definition 4.1.4 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$, C be a total coloring of Ψ and let (V, E_0, E_1, E_2) be a subgraph of Ψ.*

We define a B–height function of (V, E_0, E_1, E_2) as a function $h : V \to \mathbb{N}$ such that for all $r \in V$ we have

1. *if $(r', r) \in E_2$, then $h(r') < h(r)$,*
3. *if $r \in C_\ominus \cap V$, then one of the following conditions is fulfilled:*
   (a) *there exists $r' \in C_\oplus$ and $(r', r) \in E_1$ such that $h(r') < h(r)$;*
   (b) *$rule(head(r)) \cap C_\oplus \neq \emptyset$.*

First of all, observe that Condition 2 from Definition 4.1.3 has been dropped. This is because a B–height function does not take into account 0-edges, since B–preference decouples supportedness from preference handling [34]. If $X = head(C_\oplus)$ is a set of atoms, then $rule(head(r)) \cap C_\oplus \neq \emptyset$ states that $head(r) \in X$ for some $r \in \Pi$. Hence, Condition 3b weakens the concept of order preservation given in Condition 3 of Definition 4.1.3, whenever the head of a blocked rule is derived by another applied rule. By this weakening, more admissible colorings are $<^B$–preserving than $<^D$–preserving.

The next concept of a height function addresses W–preferences.

Definition 4.1.5 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$, C be a total coloring of Ψ and let (V, E_0, E_1, E_2) be a subgraph of Ψ.*

We define a W–height function of (V, E_0, E_1, E_2) as a function $h : V \to \mathbb{N}$ such that for all $r \in V$, we have

1. *if $(r', r) \in E_2$, then $h(r') < h(r)$,*
2. *if $r \in C_\oplus$, then one of the following conditions is fulfilled:*
   (a) *if $(r', r) \in E_0$ and $r' \in C_\oplus$, then $h(r') < h(r)$;*
   (b) *there exists $r' \in rule(head(r)) \cap C_\oplus$ such that $h(r') < h(r)$,*
3. *if $r \in C_\ominus \cap V$, then one of the following conditions is fulfilled:*
   (a) *there exists $r' \in C_\oplus$ and $(r', r) \in E_1$ such that $h(r') < h(r)$;*
   (b) *there exists $r'' \in rule(head(r)) \cap C_\oplus$ such that $h(r'') < h(r)$.*

W–height functions combine supportedness and preference handling similar to D–height functions. In contrast to D–height functions, however, Definition 4.1.5 allows for supporting and blocking a rule r by lower ranked rules, if $head(r)$ is derived by some applied rule with a lower h-value than r. Hence, Condition 2b and 3b weaken the concept of a D–height function given in Definition 4.1.3, but they are not so generous as the conditions for a B–height function given in Definition 4.1.4. For this reason, the conditions for the existence of a D–height function are stronger that than for a W–height function, which are stronger conditions than for a B–height function.

For illustration, consider ordered program $(\Pi_{4.1}, <)$. For admissible coloring $(\{r_1, r_2, r_4\}, \{r_3\})$, the D–height function given in (4.7) is also a B– as well as a W–height function. Observe that $h(r_1) = 2,\ h(r_2) = 1,\ h(r_3) = 4,\ h(r_4) = 3$ provides an alternative B–height function, which is neither a W– nor a D–height function. No σ–height function is obtained for the second admissible coloring, corresponding to answer set $\{p, b, f\}$, for any $\sigma \in \{D, B, W\}$.

In analogy to Theorem 4.1.1, B– and W–height functions allow us to characterize $<^B$– and $<^W$–preserving admissible colorings.

Theorem 4.1.2 *Theorem 4.1.1 still holds, when replacing D by either B or W.*

Whenever no preferences are given, Theorem 4.1.1 and 4.1.2 fall back to characterizations of standard admissible colorings:

Corollary 4.1.3 *Let Ψ be the RDG of ordered logic program $(\Pi, \emptyset)$ and C be a total coloring of Ψ. Then, C is an admissible coloring iff C is a $<^\sigma$–preserving admissible coloring for $\sigma \in \{D, B, W\}$.*

4.1.3 Operational characterization

In this section, we exemplarily provide an operational characterization of $<^D$–preserving answer sets; for brevity, the corresponding characterizations for B– and W–preferences are omitted. As in Section 3.3, the idea is to start with the empty coloring and to successively apply operators that turn a partial coloring C into another one C' such that $C \sqsubseteq C'$. This is done until a total coloring is obtained that corresponds to a $<^D$–preserving answer set.

When extending a partial coloring C, we encounter below the case that rules have to be unsupported and thus colored with $\ominus$ in the final total coloring, although this cannot be ensured by C. [4] In contrast to the case of standard logic programs, we have additionally to introduce an intermediate version of $\ominus$, denoted by $\oslash$, to handle preferences. Whenever a rule is colored $\oslash$ in a partial coloring, it must be colored $\ominus$ in any (total) $<^D$–preserving admissible coloring containing C.[5] Accordingly, a partial coloring is now a partial mapping $C : \Pi \to \{\oplus, \ominus, \oslash\}$. Analogously, we define $C_\oslash = \{r \in \Pi \mid C(r) = \oslash\}$. C is a total coloring if $C_\oplus \cup C_\ominus = \Pi$, $C_\oplus \cap C_\ominus = \emptyset$, and $C_\oslash = \emptyset$. Furthermore, we define a conservative extension of $\sqsubseteq$ for partial colorings C and C' containing $\oslash$ as follows: $C \sqsubseteq C'$ if $C_\oplus \subseteq C'_\oplus$, $C_\ominus \subseteq C'_\ominus$ and $C_\oslash \subseteq (C'_\ominus \cap \overline{S}(\Psi, C')) \cup C'_\oslash$. [6] This reflects the idea that rules in $C_\oslash$ cannot be supported in C' and have to be unsupported in all total colorings compatible with C. We define $C \sqcup C'$ as $(C_\oplus \cup C'_\oplus, C_\ominus \cup C'_\ominus, (C_\oslash \cup C'_\oslash) \setminus (C_\ominus \cup C'_\ominus))$. Accordingly, Definition 4.1.2 and that of $AS^\sigma_{(\Pi,<)}(C)$ have to be extended by replacing $C_\ominus$ by $C_\ominus \cup C_\oslash$. Otherwise, the definition of $AC^D_{(\Pi,<)}(C)$ for a given partial coloring C and all concepts introduced in Section 4.1.2 directly carry over. We denote the set of all partial colorings of a *RDG* $\Psi_{(\Pi,<)}$ by $\mathbb{C}_{\Psi_{(\Pi,<)}}$. Whenever clear from the context, we simply write $\mathbb{C}$.

First, we concentrate on operations deterministically extending partial colorings. For unordered logic programs Π, the corresponding deterministic operator $\mathcal{P}_\Psi$ is defined in Definition 3.3.1 on page 28 through standard colors $C_\oplus$ and $C_\ominus$. For our purpose, we define this operator as follows:

Definition 4.1.6 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C be a partial coloring of Ψ.*

Then, define $\mathcal{P}_\Psi : \mathbb{C} \to \mathbb{C}$ as $\mathcal{P}_\Psi(C) = C'$ where

1. $C'_\oplus = C_\oplus \cup (S(\Psi, C) \cap \overline{B}(\Psi, C) \cap M(\Psi, C))$,
2. $C'_\ominus = C_\ominus \cup \overline{S}(\Psi, C) \cup (B(\Psi, C) \cap S(\Psi, C) \cap M(\Psi, C))$ *and*
3. $C'_\oslash = C_\oslash \setminus \overline{S}(\Psi, C)$.

[4] Note that blocked rules are handled separately from unsupported rules.

[5] The color $\oslash$ is comparable to DLV's "must_be_true" [92].

[6] Observe that for some admissible C', $C_\oslash \subseteq (C'_\ominus \cap \overline{S}(\Psi, C')) \cup C'_\oslash$ is equivalent to $C_\oslash \subseteq \overline{S}(\Psi, C')$.

This definition offers a modification of the standard operator in Definition 3.3.1 on page 28 by adding a third color, restricting propagation to maximal rules only and by separating unsupportness from blockage. In fact, except for unsupported rules belonging to $\overline{S}(\Psi, C)$, all propagatable rules must belong to $M(\Psi, C)$.[7] That is, all maximal, supported, and unblocked rules in (Ψ, C) are added to $C'_{\oplus}$; all maximal, supported, and blocked rules in (Ψ, C) are added to $C'_{\ominus}$. Unsupported rules in (Ψ, C) are added to $C'_{\ominus}$ and removed from $C_{\oslash}$. The underlying idea is to propagate along a D–height function by only adding maximal rules that are free to be considered for application. As shown in Theorem 4.1.1, a D–height function takes only supported rules into account. Hence, unsupported rules are not subject to the order-driven propagation enforced by $M(\Psi, C)$ and rather directly colored $\ominus$ by $\mathcal{P}_\Psi$. In contrast to the original operator, $\mathcal{P}_\Psi$ colors rules $\ominus$ because they are either unsupported or supported and blocked (and maximal) in (Ψ, C). Hence, we separate unsupportedness from blockage. That is, no rule is colored $\ominus$ just because it is blocked. This ensures that rules in $C_{\oslash}$ are colored $\ominus$ only if they are unsupported.

A partial coloring C is closed under $\mathcal{P}_\Psi$, if $C = \mathcal{P}_\Psi(C)$. Note that $C \sqsubseteq \mathcal{P}_\Psi(C)$. As expressed in Theorem 3.3.1 for answer sets, $\mathcal{P}_\Psi(C)$ is not guaranteed to be a partial coloring. To see this, observe that $\mathcal{P}_\Psi((\{a \leftarrow not\ a\}, \emptyset, \emptyset))$ would be $(\{a \leftarrow not\ a\}, \{a \leftarrow not\ a\}, \emptyset)$, which is no mapping and thus no partial coloring. Analogously to Theorem 3.3.1 on page 29, $\mathcal{P}_\Psi$ exists on colorings expressing preferred answer sets.

Theorem 4.1.4 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C a partial coloring of Ψ.*
If $AC^D_{(\Pi,<)}(C) \neq \emptyset$, then $\mathcal{P}_\Psi(C)$ exists.

Actually, the precondition can be weakened by only requiring $AC_{(\Pi,<)}(C) \neq \emptyset$.

Now, we can define our principal propagation operator in the following way.

Definition 4.1.7 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C a partial coloring of Ψ.*
Then, define $\mathcal{P}^*_\Psi : \mathbb{C} \to \mathbb{C}$ as the $\sqsubseteq$-smallest partial coloring closed under $\mathcal{P}_\Psi$ and containing C.

Although $\mathcal{P}^*_\Psi$ is not always defined, it is on colorings expressing preferred answer sets.

Theorem 4.1.5 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C a partial coloring of Ψ.*
If $AC^D_{(\Pi,<)}(C) \neq \emptyset$, then $\mathcal{P}^*_\Psi(C)$ exists.

Essentially, $\mathcal{P}^*_\Psi(C)$ amounts to computing deterministic "consequences" from a given partial coloring C. In fact, $\mathcal{P}^*_\Psi(C)$ is monotonic and preserves preferred answer sets in the following sense.

Theorem 4.1.6 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C be a partial coloring of Ψ.*

1. *If $AC^D_{(\Pi,<)}(C') \neq \emptyset$ and $C \sqsubseteq C'$ then $\mathcal{P}^*_\Psi(C) \sqsubseteq \mathcal{P}^*_\Psi(C')$;*
2. *$AC^D_{(\Pi,<)}(C) = AC^D_{(\Pi,<)}(\mathcal{P}^*_\Psi(C))$.*

All these properties are analogously to the standard case (cf. Section 3.3.1 on page 28).

The next operation draws upon the maximal support graph of colored *RDG*s.

Definition 4.1.8 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C be a partial coloring of Ψ. Furthermore, let (V, E) be a maximal support graph of (Ψ, C) for some $E \subseteq (\Pi \times \Pi)$.*
Then, define $\mathcal{U}_\Psi : \mathbb{C} \to \mathbb{C}$ as $\mathcal{U}_\Psi(C) = (C_{\oplus}, \Pi \setminus V, C_{\oslash} \setminus (\Pi \setminus V))$.

Observe that $C_{\ominus} \subseteq \Pi \setminus V$. A two-digit version of $\mathcal{U}_\Psi$ was proposed in Definition 3.3.3 on page 30 for standard programs. This operator allows for coloring rules with $\ominus$ whenever it is clear from the given partial coloring that they will remain unsupported. Those rules are deleted from $C_{\oslash}$ since they are known to be unsupported w.r.t. C. As with $\mathcal{P}^*_\Psi$, $\mathcal{U}_\Psi(C)$ gives an extension of C. Unlike $\mathcal{P}^*_\Psi$, however, $\mathcal{U}_\Psi$ allows for coloring nodes unconnected with the already colored part of the graph. Although $\mathcal{U}_\Psi$ is not defined in general, it is on colorings guaranteeing the existence of support graphs.

[7] See also Condition 3a in Definition 2.4.1 on page 13.

Theorem 4.1.7 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C be a partial coloring of Ψ.*
If (Ψ, C) has a support graph, then $\mathcal{U}_\Psi(C)$ exists.

Furthermore, (Ψ, C) has a support graph whenever $AC_{(\Pi,<)}(C) \neq \emptyset$. As with $\mathcal{P}^*_\Psi$, operator $\mathcal{U}_\Psi$ is reflexive, idempotent, monotonic, and preserves preferred answer sets (cf. on page 28, Section 3.3.1).

Definition 3.3.12 on page 31 states that a total coloring C is admissible iff $\mathcal{P}_\Psi(C) = C$ and $\mathcal{U}_\Psi(C) = C$. That is, both operators are sufficient for deciding answersetship. For selecting preferred answer sets among standard answer sets, we additionally rely on the concept of a height function.

The next operator provides a partial check for the existence of a D–height function.

Definition 4.1.9 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$.*
For a total coloring C of Ψ, define $\mathcal{H}_\Psi : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ as $\mathcal{H}_\Psi(C', C) = C''$ where

$$C''_\oplus = C'_\oplus \cup (C_\oplus \cap S(\Psi, C') \cap M(\Psi, C')),$$

$$C''_\ominus = C'_\ominus \cup (C_\ominus \cap S(\Psi, C) \cap B(\Psi, C') \cap M(\Psi, C')) \cup (C_\ominus \cap \overline{S}(\Psi, C) \cap M(\Psi, C')),$$

$$C''_\oslash = C'_\oslash.$$

Operator $\mathcal{H}_\Psi$ extends a partial coloring C' to a partial coloring C'' relative to a given total coloring C. That is, the status of applicability of all rules expressed in C guides the transition from C' to C''. By construction, we have $C'' \sqsubseteq C$, if $C' \sqsubseteq C$. Also, only "maximal" rules from C' belonging to C are added to C''.

Iterated application of $\mathcal{H}_\Psi$ guarantee the existence of a D–height function in (Γ, C), as given in Theorem 4.1.1. For this, define $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = \bigsqcup_{i<\omega} \mathcal{H}^i_\Psi((\emptyset,\emptyset,\emptyset), C)$ where $\mathcal{H}^0_\Psi((\emptyset,\emptyset,\emptyset), C) = (\emptyset,\emptyset,\emptyset)$ and $\mathcal{H}^{i+1}_\Psi((\emptyset,\emptyset,\emptyset), C) = \mathcal{H}_\Psi(\mathcal{H}^i_\Psi((\emptyset,\emptyset,\emptyset), C), C)$ for $i < \omega$. Then, we then have the following result.

Theorem 4.1.8 *Let $\Psi = (\Pi, E_0, E_1, E_2)$ be the RDG of ordered logic program $(\Pi, <)$. Let C be an admissible coloring of Ψ and let $(C_\oplus, E'_0)$ be a support graph of (Ψ, C) for some $E'_0 \subseteq E_0$.*
Then, we have $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = C$ if and only if there exists a D–height function of the graph $(S(\Psi, C), E'_0, E_1|_{S(\Psi,C)}, E_2|_{S(\Psi,C)})$.

This leads us to the following alternative characterization of $<^D$–preserving admissible colorings.

Theorem 4.1.9 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and let C be a total coloring of Ψ.*
Then, C is a $<^D$–preserving admissible coloring if and only if $\mathcal{P}_\Psi(C) = C$, $\mathcal{U}_\Psi(C) = C$, and $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = C$.

C being closed under $\mathcal{P}_\Psi$ and $\mathcal{U}_\Psi$ ensures that C is admissible. The operator $\mathcal{H}^*_\Psi$ guarantees the existence of a D–height function, and hence an enumeration of Π according to Definition 2.4.1 on page 13. In fact, D–preference stipulates that all generating rules are formed in a supported way, as witnessed by Condition 2 in Definition 2.4.1 on page 13. A similar result is obtained for the resulting operators $\mathcal{H}^*_\Psi$ and $\mathcal{U}_\Psi$.

Theorem 4.1.10 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$. Let C be an total coloring of Ψ.*
If $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = C$, then $\mathcal{U}_\Psi(C) = C$.

Closedness under $\mathcal{U}_\Psi$ is enforced by $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = C$. $\mathcal{H}^{i+1}_\Psi((\emptyset,\emptyset,\emptyset), C)$ takes only rules from $C_\oplus$ into account which are supported in $(\Psi, \mathcal{H}^i_\Psi((\emptyset,\emptyset,\emptyset), C))$. Since $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = C$, all rules in $C_\oplus$ are in a support graph in (Ψ, C) and hence, $\mathcal{U}_\Psi(C) = C$.

Accordingly, we obtain the following simplification of Theorem 4.1.9.

Corollary 4.1.11 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and let C be a total coloring of Ψ.*
Then, C is a $<^D$–preserving admissible coloring iff $\mathcal{P}_\Psi(C) = C$ and $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = C$.

Note that a similar result *cannot* be obtained for B-preference, since it discards supportedness.

Now we develop an elementary strategy for choice operations.

Definition 4.1.10 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C be a partial coloring of Ψ.*

For $r \in \Pi \setminus (C_\oplus \cup C_\ominus \cup C_\oslash)$, we define the following operator $\mathcal{C}^\circ_\Psi : \mathbb{C} \to \mathbb{C}$ for $\circ \in \{\oplus, \ominus\}$:

1. $\mathcal{C}^\oplus_\Psi(C) = (C_\oplus \cup \{r\}, C_\ominus, C_\oslash)$;
2. $\mathcal{C}^\ominus_\Psi(C) = (C_\oplus, C_\ominus \cup \{r\}, C_\oslash)$.

As done in Section 3.3.2, we use $\mathcal{C}^\circ_\Psi$ whenever the distinction between $\mathcal{C}^\oplus_\Psi(C)$ and $\mathcal{C}^\ominus_\Psi(C)$ is of no importance. Strictly speaking, $\mathcal{C}^\circ_\Psi$ is also parametrized with r; we leave this implicit to abstract from the actual choice. In fact, whenever both operators $\mathcal{C}^\oplus_\Psi(C)$ and $\mathcal{C}^\ominus_\Psi(C)$ are available, the choice of r is only a *"don't care"* choice, while that among $\oplus$ and $\ominus$ is the crucial *"don't know"* choice. Intuitively, this is because all rules must be colored either way; it is the attributed color that is of prime importance for the existence of a preferred answer set.

Combining the previous guess and check operators yields our first operational characterization of preferred admissible colorings (along with its underlying preferred answer sets).

Theorem 4.1.12 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and let C be a total coloring of Ψ.*

Then, C is a $<^D$–preserving admissible coloring of Ψ iff there exists a sequence $(C^i)_{0 \leq i \leq n}$ with the following properties:

1. $C^0 = (\emptyset, \emptyset, \emptyset)$;
2. $C^{i+1} = \mathcal{C}^\circ_\Psi(C^i)$ *for* $\circ \in \{\oplus, \ominus\}$ *and* $0 \leq i < n$;
3. $C^n = \mathcal{P}_\Psi(C^n)$;
4. $C^n = \mathcal{H}^*_\Psi(C^0, C^n)$;
5. $C^n = C$.

In what follows, we refer to such sequences also as *coloring sequences*. Note that all sequences satisfying conditions 1-5 of Theorem 4.1.12 are *successful* in the sense that their last element corresponds to an existing preferred answer set. If a program has no preferred answer set, then no such sequence exists.

Although this straightforward guess and check approach may not be of great implementational value, it supplies us with an initial skeleton for the coloring process that we refine in the sequel. In particular, this characterization stresses the basic fact that we possess complete freedom in forming a coloring sequence as long as we can guarantee that the resulting coloring is a fixed point of $\mathcal{P}_\Psi$ and $\mathcal{H}_\Psi$ (and so implicitly of $\mathcal{U}_\Psi$ as well). This strategy is analogously to the one in Section 3.3, where we have characterized several operational characterizations for the computation of answer sets. There, we ensure that the resulting coloring is a fixed point of the propagation and of the unfounded set operator for (standard) answer sets.

We observe the following properties.

Theorem 4.1.13 *Given the prerequisites in Theorem 4.1.12, let $(C^i)_{0 \leq i \leq n}$ be a sequence satisfying conditions 1-5 in Theorem 4.1.12.*

Then, we have the following properties for $0 \leq i \leq n$.

1. C^i *is a partial coloring;*
2. $C^i \sqsubseteq C^{i+1}$;
3. $AC^D_{(\Pi,<)}(C^i) \supseteq AC^D_{(\Pi,<)}(C^{i+1})$;

4. $AC^D_{(\Pi,<)}(C^i) \neq \emptyset$;

5. (Ψ, C^i) *has a (maximal) support graph.*

As expressed in Theorem 3.3.14 for answer sets, all these properties represent invariants of the consecutive colorings. While the first three properties are provided by operator $\mathcal{C}^\circ_\Psi$ in choosing among uncolored rules only, the last two properties are actually enforced by of the final coloring C^n, that is, the "check" expressed by conditions 3–5 in Theorem 4.1.12. In fact, sequences only enjoying conditions 1 and 2 in Theorem 4.1.12, fail to satisfy Property 4 and 5 in Theorem 4.1.13 in general. In practical terms, this means that computations of successful sequences may be led numerous times on the "garden path" before termination.

As it is well-known and demonstrated in Section 3.3.2, the number of choices can be significantly reduced by applying deterministic operators.

Theorem 4.1.14 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and let C be a total coloring of Ψ.*

Then, C is a $<^D$–preserving admissible coloring of Ψ iff there exists a sequence $(C^i)_{0\leq i\leq n}$ with the following properties:

1. $C^0 = \mathcal{P}^*_\Psi((\emptyset,\emptyset,\emptyset))$;

2. $C^{i+1} = \mathcal{P}^*_\Psi(\mathcal{C}^\circ_\Psi(C^i))$ *for* $\circ \in \{\oplus, \ominus\}$ *and* $0 \leq i < n$;

3. $C^n = \mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C^n)$;

4. $C^n = C$.

On the one hand, the continuous application of $\mathcal{P}^*_\Psi$ extends partial colorings after each choice. On the other hand, this proceeding guarantees that each partial coloring (including C^n) is closed under $\mathcal{P}_\Psi$.

Although the last characterization allows for deterministic propagation its choices are still arbitrary. In particular, preference information is not taken into account. We address this shortcoming by developing a strategy for choice operations based on maximality and support.

Definition 4.1.11 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C be a partial coloring of Ψ.*

For $r \in (\Pi \cap M(\Psi, C)) \setminus (C_\oplus \cup C_\ominus \cup C_\oslash)$, we define the following operators $\mathcal{D}^\circ_\Psi : \mathbb{C} \to \mathbb{C}$ for $\circ \in \{\oplus, \oslash\}$:

1. $\mathcal{D}^\oplus_\Psi(C) = (C_\oplus \cup \{r\}, C_\ominus, C_\oslash)$, *if* $r \in S(\Psi, C)$;

2. $\mathcal{D}^\oslash_\Psi(C) = (C_\oplus, C_\ominus, C_\oslash \cup \{r\})$, *if* $r \notin S(\Psi, C)$.

The $\mathcal{D}^\oplus_\Psi$ operator colors a maximal, supported rule r with $\oplus$, that is, r is taken to be applied. This avoids supporting higher ranked rules by lower ranked ones (w.r.t. $<$). $\mathcal{D}^\oslash_\Psi$ colors a maximal, not supported rule with $\oslash$. In fact, a rule r colored by $\mathcal{D}^\oslash_\Psi$ must be unsupported in the final total coloring. Observe that rules being blocked (and supported) in the final coloring are never colored by $\mathcal{D}^\oslash_\Psi$ in order to guarantee Condition 3 in Definition 4.1.3; they must be colored by propagation. Similarly, unsupported rules are taken care of by propagation. The above operator takes preference into account in two ways. First, it restricts the choice of r to uncolored rules belonging to $M(\Psi, C)$. Second, through coloring with $\oslash$ instead of $\ominus$ it delegates the coloration with $\ominus$ to the deterministic operator $\mathcal{P}_\Psi$ (see Definition 4.1.6). Similarly, unsupported rules are colored $\ominus$ by propagation.

Combining our deterministic operators with choice operator $\mathcal{D}^\circ_\Psi$ yields an operational characterization of order preserving admissible colorings that avoids a belated check for a D–height function. For this, for a partial coloring C we define $(\mathcal{PU})^*_\Psi(C)$ as the $\sqsubseteq$-smallest partial coloring containing C and being closed under $\mathcal{P}_\Psi$ and $\mathcal{U}_\Psi$.

Theorem 4.1.15 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and let C be a total coloring of Ψ.*

Then, C is a $<^D$–preserving admissible coloring of Ψ iff there exists a sequence $(C^i)_{0\leq i\leq n}$ with the following properties:

1. $C^0 = (\mathcal{PU})^*_\Psi((\emptyset,\emptyset,\emptyset))$;
2. $C^{i+1} = (\mathcal{PU})^*_\Psi(\mathcal{D}^\circ_\Psi(C^i))$ *for some* $\circ \in \{\oplus, \oslash\}$ *and* $0 \leq i < n$;
3. $C^n = C$.

Note that this characterization does not rely on operator $\mathcal{H}_\Psi$ and thus avoids the respective conditions in Theorem 4.1.12 and 4.1.14. In fact, one can show that $C^n = \mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C^n)$ for every successful coloring sequence. The formation of such sequences is driven by the coloration of maximal rules. Operators $\mathcal{P}_\Psi$, $\mathcal{U}_\Psi$, and $\mathcal{D}^\circ_\Psi$ color along a D–height function, where lower valued rules are colored first. That is, the sequence starts by coloring most preferred rules and ends with the lowest ones. All rules being colored $\oslash$ in some intermediate step i ($0 < i < n$, see Theorem 4.1.15) must be found unsupported by $\mathcal{P}_\Psi$ and $\mathcal{U}_\Psi$ in order to guarantee $C^n_\oslash = C_\oslash = \emptyset$.

For illustration, consider the coloring sequence in Figure 4.2, obtained for $<^D$–preserving answer set $\{b, p, f'\}$ of program $(\Pi_{4.1}, <)$. First, maximal rules r_1 and r_2 are colored by $(\mathcal{PU})^*_\Psi$. From the remain-

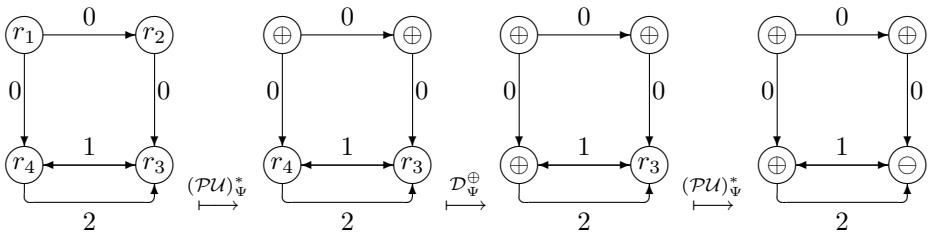

Figure 4.2: A coloring sequence obtained for $<^D$–preserving answer set $\{b, p, f'\}$ of $(\Pi_{4.1}, <)$.

ing uncolored rules, only r_4 is maximal and (since being supported) taken to be $\oplus$ by $\mathcal{D}^\oplus_\Psi$. This leads to coloring r_3 with $\ominus$ by $(\mathcal{PU})^*_\Psi$ since it is maximal, supported, and blocked. The resulting admissible coloring $(\{r_1, r_2, r_4\}, \{r_3\})$ is $<^D$–preserving and reflects the $<^D$–preserving answer set $\{p, b, f'\}$. Note that coloring a maximal, supported and blocked rule r in the absence of a blocker of r would lead to illegal situations. Unlike this, coloring r_4 with $\oslash$ by $\mathcal{D}^\oslash_\Psi$ is impossible since $r_4 \in S(\Psi, (\mathcal{PU})^*_\Psi((\emptyset,\emptyset,\emptyset)))$. In fact, there is no way to obtain the second admissible (not $<^D$–preferred) coloring. We can neither color r_4 with $\ominus$, since $r_4 \in S(\Psi, (\mathcal{PU})^*_\Psi((\emptyset,\emptyset,\emptyset)))$, nor r_3 with $\oplus$, since r_3 is not maximal in $(\Psi, (\mathcal{PU})^*_\Psi((\emptyset,\emptyset,\emptyset)))$.

To illustrate the usage of $\oslash$, consider the following program $(\Pi_{4.8}, <)$, where

$$\begin{array}{lllll} r_1: & a & \leftarrow & c & \\ r_2: & b & \leftarrow & not\ c & \qquad r_2 < r_1 \\ r_3: & c & \leftarrow & not\ b & \qquad r_3 < r_2 \end{array} \tag{4.8}$$

At first, operator $(\mathcal{PU})^*_\Psi$ cannot color any rule. Only r_1 is maximal and available for our choice operator. By $r_1 \notin S(\Psi_{(\Pi_{4.8},<)}, (\emptyset,\emptyset,\emptyset))$, we color r_1 by $\mathcal{D}^\oslash_\Psi$, which leads to partial coloring $(\emptyset,\emptyset,\{r_1\}) = (\mathcal{PU})^*_\Psi((\emptyset,\emptyset,\{r_1\}))$. Applying $\mathcal{D}^\oplus_\Psi$ to maximal, supported rule r_2 and applying $(\mathcal{PU})^*_\Psi$ lead to total coloring $(\{r_2\}, \{r_1, r_3\}, \emptyset)$, which is $<^D$–preserving and corresponds to the only existing $<^D$–preserving answer set $\{b\}$. The successful coloring sequence is given in Figure 4.3. [8]

In analogy to Theorem 4.1.13, we observe the following properties.

[8] Note that the *RDG* contains the 2-edge (r_1, r_3) since by definition $<$ is a strict partial order and thus transitive.

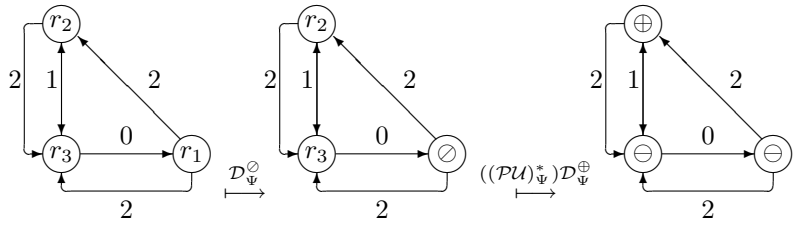

Figure 4.3: A (successful) coloring sequence.

Theorem 4.1.16 *Given the prerequisites in Theorem 4.1.15, let $(C^i)_{0\leq i\leq n}$ be a sequence satisfying conditions 1-3 in Theorem 4.1.15 for RDG $\Psi = (\Pi, E_0, E_1, E_2)$.*

Then, we have the following properties for $0 \leq i \leq n$.

1.–5. as given in Theorem 4.1.13;

6. $C^{i+1}_{\oplus} \supseteq S(\Psi, C^i) \cap \overline{B}(\Psi, C^i) \cap M(\Psi, C^i)$;

7. $C^{i+1}_{\ominus} \supseteq \overline{S}(\Psi, C^i) \cup (B(\Psi, C^i) \cap S(\Psi, C^i) \cap M(\Psi, C^i))$;

8. $C^{i+1}_{\oslash} \supseteq C^i_{\oslash} \setminus \overline{S}(\Psi, C^i)$

9. $(C^i_{\oplus}, E)$ *is a support graph of* (Ψ, C^i) *for some* $E \subseteq E_0$.

10. $h(r) < h(r')$ *for every* $r, r' \in S(\Psi, C^i)$ *where for some* $E'_0 \subseteq E_0$, h *is some D–height function of* $(S(\Psi, C^i), E'_0|_{S(\Psi,C^i)}, E_1|_{S(\Psi,C^i)}, E_2|_{S(\Psi,C^i)})$.

The five additional properties reflect the fact that the characterization in Theorem 4.1.15 yields much more structured coloring sequences than its predecessors in Theorem 4.1.12 and 4.1.14.

Although the application of the deterministic operator $\mathcal{U}_{\Psi}$ is beneficial insofar as every deterministically colored rule reduces the search space, it is not strictly necessarily. Rather unsupported rules can be guessed by $\mathcal{D}^{\oslash}_{\Psi}$.

Theorem 4.1.17 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and let C be a total coloring of Ψ.*

Then, C is a $<^D$–preserving admissible coloring of Ψ iff there exists a sequence $(C^i)_{0\leq i\leq n}$ with the following properties:

1. $C^0 = \mathcal{P}^{*}_{\Psi}((\emptyset, \emptyset, \emptyset))$;

2. $C^{i+1} = \mathcal{P}^{*}_{\Psi}(\mathcal{D}^{\circ}_{\Psi}(C^i))$ *for some* $\circ \in \{\oplus, \oslash\}$ *and* $0 \leq i < n$;

3. $C^n = C$.

In this way, the coloration of unsupported rules is initiated by $\mathcal{D}^{\oslash}_{\Psi}$ and accomplished by $\mathcal{P}_{\Psi}$. That is, $\mathcal{D}^{\oslash}_{\Psi}$ only marks a rule as unsupported by attributing color $\oslash$, while $\mathcal{P}_{\Psi}$ confirms this choice by turning $\oslash$ into $\ominus$. Although this avoids using (deterministic) operator $\mathcal{U}_{\Psi}$, it delegates the treatment of unsupported rules to a non-deterministic operator, which seems not advisable from a computational point of view.

4.1.4 Discussion, related work, and conclusions

Many approaches to adding preferences to answer set programming can be found in the literature [173, 27, 101, 198, 104, 34, 62, 177, 194] (cf. Section 4.4 for a detailed discussion). Among them, we have considered the three approaches, interpreting preferences as inducing a selection function among the answer sets of the underlying program [34, 62, 194]. Up to now, the latter approaches have either been implemented

by meta-interpretation [77] or by pre-compilation front-ends [62]. The advantage of both approaches is that one can harness existing answer sets solvers without any need for modification. On the other hand, it remains unclear whether the "selection of answer sets" cannot be realized more efficiently within a solver by restricting its search space. For instance, "weight-based" approaches, as pursed in the `DLV` [130] and `smodels` [180] systems, can be implemented rather efficiently through branch-and-bound techniques. Such a quantitative approach is unfortunately inapplicable in our setting.

For addressing this problem, we have put forward the usage of graphs and colorings as an appropriate computational model. Preferences are simply taken as a third type of edges in a graph, reflecting an additional dependency among rules. In particular, we have demonstrated that this approach allows us to capture all three "selection function" approaches to preferences in a uniform setting by means of the concept of a height function. To a turn, we have exemplarily developed an operational characterization for one of these strategies. For this purpose, we have extended a recently proposed operational framework for graph-based computation of answer sets (cf. Chapter 3). Apart from the extension of colorings by a third "transitory" coloring $\oslash$ (comparable to `DLV`'s "`must_be_true`"), we have extended the deterministic and non-deterministic operations by preference handling. This is done through the restriction of propagation and choice operations to those rules that are not dominated by any preferred rules whose application status is indeterminate (viz $M(\Psi, C)$). We have prototypically implemented different operational variants, using different operators; the resulting Prolog implementation is available at

`http://www.cs.uni-potsdam.de/~konczak/system/GCplp`.

The description of the C++ implementation follows in Section 4.3, as well as benchmark tests and a detailed comparison of the approach presented in this section with the compilation front-end from the `plp` system and the meta-interpreter from `DLV`.

A preliminary study of this approach has been done in [117]. There, we have considered programs, where all rules have at most one positive body element. In this work, we have extended this approach to ordered programs without any restriction of positive body elements of rules.

4.2 Experimental Evaluation

In this section, we present new benchmarks for ordered logic programs. These benchmarks will be used in Section 4.3 for the comparison of the integrative method for computing preferred answer sets described in Section 4.1 with a compilation method of preferences. We extend well-known graph problems by preferences, which filter out unwanted situations. First, we consider some graph types we are dealing with, and second, we consider well-known benchmark problems which are extended by preference information.

Graph types. *Complete graphs* with n vertices are defined as undirected graphs (V, E) where $V = \{v_1, \ldots, v_n\}$ and $E = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$. *Ladder graphs* with $2 * n$ vertices (cf. Figure 4.4) are undirected graphs (V, E) where $V = \{v_1, \ldots, v_{2*n}\}$ and

$$\begin{aligned} E \;=\; & \{(v_i, v_{i+1}) \mid i \in \{1, \ldots, n-1\}\} \cup \\ & \{(v_i, v_{i+1}) \mid i \in \{n+1, \ldots, 2*n-1\}\} \cup \\ & \{(v_i, v_j) \mid i \in \{1, \ldots, n\}, j = 2*n-i+1\} \end{aligned}$$

A *(directed) circle graph* with n vertices is a directed graph (V, E) where $V = \{v_1, \ldots, v_n\}$ and $E = \{(v_i, v_{i+1}) \mid i \in \{1, \ldots, n-1\}\} \cup \{(v_1, v_n)\}$. All these graphs can be represented as rules:

$$\begin{aligned} vtx(v_i) \;&\leftarrow\; \text{for each vertex } i \in \{1, \ldots n\} \\ edge(v_i, v_j) \;&\leftarrow\; \text{for every directed edge } (v_i, v_j) \in E \end{aligned}$$

Note that whenever (v_i, v_j) is an undirected edge in the graph, we have to add the rules $edge(v_i, v_j) \leftarrow$ as well as $edge(v_j, v_i) \leftarrow$ to the program.

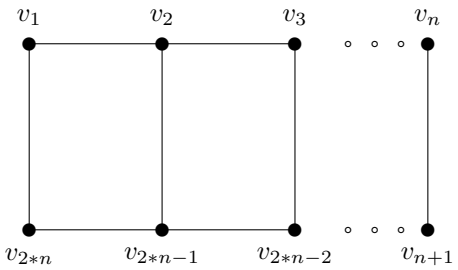

Figure 4.4: Ladder Graph with $2 * n$ vertices.

In the following, we consider well-known graph problems extended by rule preferences.

Hamiltonian Cycle problem. Hamiltonian cycles are cycles in a graph passing each vertex exactly once. Since there are usually many possibilities for such cycles, we use preferences for guiding the search within a graph. More precisely, with rule preferences we can express statements like "visit node v_{42} before node v_7". Such expressions are given by a set of preference relations at the form

$$\mathit{before}(v_i, v_j) \leftarrow$$

as rules in a program, denoting that a vertex v_i has to be visited before vertex v_j in the Hamiltonian cycle. The logic program for Hamiltonian cycles, where an ordering among the visited vertices is respected, is as follows:

$$\begin{array}{rrcl}
r_1: & hc(V,U) & \leftarrow & edge(V,U), not\ otherroute(V,U) \\
r_2: & otherroute(V,U) & \leftarrow & edge(V,U), edge(V,W), hc(V,W), U \neq W \\
r_3: & otherroute(V,U) & \leftarrow & edge(V,U), edge(W,U), hc(W,U), V \neq W \\
r_4: & reached(U) & \leftarrow & edge(V,U), hc(V,U), reached(V), not\ initialnode(V) \\
r_5: & reached(U) & \leftarrow & edge(V,U), hc(V,U), initialnode(V) \\
r_6: & & \leftarrow & vtx(V), not\ reached(V) \\
r_7: & hctrans(X,Y) & \leftarrow & hc(X,Y), not\ initialnode(Y), vtx(X), vtx(Y), X \neq Y \\
r_8: & hctrans(X,Y) & \leftarrow & hc(X,Z), not\ initialnode(Z), hctrans(Z,Y), \\
 & & & vtx(X), vtx(Y), vtx(Z), X \neq Y, Y \neq Z, X \neq Z \\
r_{g(X,Y)}: & goto(X,Y) & \leftarrow & name(g(X,Y)), not\ ngoto(X,Y), \\
 & & & hctrans(Y,X), vtx(X), vtx(Y), X \neq Y \\
r_{ng(Y,X)}: & ngoto(X,Y) & \leftarrow & name(ng(Y,X)), vtx(X), vtx(Y), X \neq Y
\end{array}$$

The preference relation $<$ is defined as

$$r_{ng(Y,X)} < r_{g(X,Y)} \text{ whenever } \mathit{before}(X,Y) \leftarrow \text{ is given.}$$

Rules r_1–r_6 are taken from [149] and encode the standard problem of finding Hamiltonian cycles in a graph. Rules r_7 and r_8 are for computing transitivity of edges being in a Hamiltonian cycle, where $hctrans(X,Y)$ denotes the existence of a path from X to Y in the Hamiltonian cycle. Rules $r_{g(X,Y)}$ and $r_{ng(Y,X)}$ together with the preference relation $<$ encode that whenever we prefer X over Y ($\mathit{before}(X,Y)$), a Hamiltonian way from Y to X ($hctrans(Y,X)$) is excluded as preferred solution.

Coloring Problem. In the coloring problem, we want to color a graph with k colors $c_1, \ldots, c_k$ such that no two adjacent vertices have the same color. This can be encoded by the following set of rules:

$$\begin{array}{rrcl}
r_{v_i,c_j}: & color(v_i,c_j) & \leftarrow & not\ color(v_i,c_1), \ldots, not\ color(v_i,c_{j-1}), \\
 & & & not\ color(v_i,c_{j+1}), \ldots, not\ color(c_k) \\
 & & \leftarrow & color(v_i,c_l), color(v_j,c_l) \text{ for all } 1 \leq l \leq k \text{ and all edges } (v_i,v_j)
\end{array}$$

The first rule ensures that one vertex has exactly one color and the second rule, the integrity constraint, ensures that no two adjacent vertices have the same color. We can integrate preference information for preferring a special color for a vertex. As an example, we have two colors, *red* and *green*, and we want to prefer that every odd vertex is colored with red. In this case we can add the preference relation

$$r_{v_i,green} < r_{v_i,red} \text{ whenever } i \text{ is odd.}$$

Kernel Problem. A *kernel* of a graph is a subset S of all vertices V such that between two vertices in S there are no edges and from every vertex outside S, there exists an edge to a vertex of S. For the kernel problem [45], we have the following encoding

$$\begin{array}{lrcl} r_{in(V)}: & in(V) & \leftarrow & vtx(V), not\ out(V) \\ r_{out(V)}: & out(V) & \leftarrow & vtx(V), not\ in(V) \\ & & \leftarrow & not\ out(X), not\ out(Y), edge(X,Y) \\ & & \leftarrow & out(X), \{out(y_i) \mid edge(x,y_i)\} \end{array}$$

where the set $\{out(y_i) \mid edge(x,y_i)\}$ is the shortening for all predicates $out(y_i)$, where y_i is a successor of x in the graph, e.g. if v_4 has the successors v_7 and v_9 then we add the rule $\leftarrow out(v_4), out(v_7), out(v_9)$. Rules $r_{in(V)}$ and $r_{out(V)}$ generate that vertex V is either in the kernel or not. The first integrity constraint ensures that no two adjacent vertices are both in the kernel. The second integrity constraint makes sure that every vertex, which is not in the kernel, has an edge to a vertex in the kernel.

We can include preferences for enforcing that special subsets have to be in a kernel. For example, expressing that we prefer every odd vertex to be in a kernel, we add the following rule preferences:

$$r_{out(v)} < r_{in(v)} \text{ for every odd vertex } v.$$

Independent Set Problem. An independent set [45] of a graph is a subset S of the vertices V such that between two vertices of S no edge exists. Here, we use again rule preferences to prefer certain subsets of the vertices being in an independent set or not. For example, we have an undirected circle graph with n vertices. The independent set problem is then defined as follows, for $i = 1, \ldots, n$:

$$r_{in(v_i)}: \quad in(v_i) \quad \leftarrow \quad not\ in(v_j), not\ in(v_k)$$

where v_j and v_k are the two adjacent vertices of v_i. Then, the preference relation

$$r_{in(v_j)} < r_{in(v_i)}$$

for i is even and v_j is adjacent to v_i, expresses that every even vertex is preferred to be in an independent set.

Artificial Examples. Beside known graph problems, we can construct artificial examples, as follows: Given a set of nodes $v_1, \ldots v_n$, we have the following program:

$$\begin{array}{lrcl} r_1: & a(v_1) & \leftarrow & not\ a(v_2) \\ r_i: & a(v_i) & \leftarrow & not\ a(v_{i-1}), not\ a(v_{i+1}) \text{ for } i \in \{2, \ldots, n-1\} \\ r_n: & a(v_n) & \leftarrow & not\ a(v_{n-1}) \end{array}$$

For the preferences we have several possibilities:

1. $r_i < r_{i+1}$ for $i = 1, \ldots, n-1$
2. $r_i < r_{i+1}$ for $i \in \{1, \ldots, n\}$ and i is odd.

The program without preferences has exponentially many answer sets. The preferences are used to select preferred ones. In both encodings, we get exactly one preferred answer set whenever n is even, since every $a(v_i)$ with an even i has to be in a preferred answer set to guarantee that rules are blocked in an order preserving way.

Within the given graph problems, preferences are used to exclude unwanted solutions. In Section 4.3 we use these benchmark instances to compare the integration of preference information into an ASP solver with compilation methods.

4.3 The nomore$^<$ System

The `nomore`$^<$ *(pronounced: nomorepref)* system is a C++ implementation of the approach presented in Section 4.1 and an improvement of the `GCplp` [98] system. The current version 1.0 can be downloaded from `http://www.cs.uni-potsdam.de/wv/nomorepref/`. Since the `nomore`$^<$ system has no own parser for handling preference relations, it uses `lparse` [146] as parser, where preference statements are encoded as:

$$\begin{array}{rcl} name(r) & \leftarrow & \text{for all rules } r \text{ involved in } < \\ preferred(r_1, r_2) & \leftarrow & \text{for every preference relation } r_2 < r_1, \end{array}$$

where the name predicates $name$ are needed for rule labeling and the $preferred$ predicates make the preferences explicit. The $name$ predicates appear in the positive body of the corresponding rules. For example, the ordered program $\{r_1 : a \leftarrow not\ b,\ r_2 : b \leftarrow not\ a,\ r_2 < r_1\}$ is represented as program

$$\begin{array}{l} a \leftarrow name(r_1), not\ b \\ b \leftarrow name(r_2), not\ a \\ name(r_1) \leftarrow \\ name(r_2) \leftarrow \\ preferred(r_1, r_2) \leftarrow \end{array}$$

which has standard answer sets $\{a\}$ and $\{b\}$, but only $\{a\}$ as preferred one.

For our experiments, we have considered the following examples:

- $indset\_\langle N\rangle$ encodes the independent set problem for an undirected circle graph with N vertices, where even numbered vertices are preferred to be in an independent set.
- $art2\_\langle N\rangle$ and $art\_\langle N\rangle$ are artificial ordered programs, where all N rules of the underlying logic program have only negative body atoms (except for $name$ predicates used for rule labeling). That is, we have the following program:

$$\begin{array}{lrcl} r_1: & a(v_1) & \leftarrow & not\ a(v_2) \\ r_i: & a(v_i) & \leftarrow & not\ a(v_{i-1}), not\ a(v_{i+1}) \text{ for } i \in \{2, \ldots, N-1\} \\ r_n: & a(v_N) & \leftarrow & not\ a(v_{N-1}) \end{array}$$

  where $art\_\langle N\rangle$ contains additionally the preference relations $r_i < r_{i+1}$ for $i = 1, \ldots, N-1$ and $art2\_\langle N\rangle$ the preferences $r_i < r_{i+1}$ for $i \in \{1, \ldots, N\}$ and i is odd.
- $kercomp\_\langle N\rangle$ encodes the kernel problem for a complete graph with N vertices, where one special vertex is preferred to be in a kernel, but no other ones.
- $collad\_\langle N\rangle$ encodes the coloring problem of a ladder graph with two colors, where a particular color is preferred for every odd vertex.
- $k\langle N\rangle\_test1$ encodes the Hamiltonian cycle problem with N vertices $v_0, v_1, \ldots, v_{N-1}$, where we have given the preference relations

$$before(v_i, v_j) \leftarrow \text{ for } i < j \text{ and } i = 1, ..., N\text{-}3, j = 2, ..., N\text{-}2$$

expressing that v_1 has to be visited before v_2, v_2 before v_3, ..., and v_{N-3} has to be visited before v_{N-2}, where v_0 is the initial node. There are exactly $N-1$ preferred answer sets, representing $N-1$ possibilities for Hamiltonian cycles with the given preference that v_1 is visited before v_2, v_2 is visited before v_3, ..., and v_{N-3} is visited before v_{N-2}. Note that for $N \leq 3$ no preferences are specified.

- $k\langle N\rangle\_test2$ encodes the Hamiltonian cycle problem with N vertices $v_0, v_1, \ldots, v_{N-1}$, where we have given the preference relations

$$before(v_1, v_{N-2}) \leftarrow$$

with v_0 as initial node. Thus, the preferred answer sets represent all Hamiltonian cycles, where node v_1 is visited before node v_{N-2}.

A detailed description of the used examples including downloads can be found at `http://www.cs.uni-potsdam.de/~konczak/benchmarks/BenchPref/`.

Table 4.1 and 4.2 show the time measurements and the number of choice points on an AMD processor with 2.2 Ghz and 2 GB memory. We have tested the above described problem classes for finding one preferred answer set (Table 4.1) and finding all preferred answer sets (Table 4.2). In both tables, DH denotes a coloring strategy, where the propagation operators are preference-based (i.e. we propagate from higher preferred rules to lower preferred rule) and where the choice operator is a conventional one. There, generated solutions have to be checked by the operator $\mathcal{H}$, which verifies the existence of a *height function* (see Definition 4.1.3 on page 54) ensuring that an answer set is preferred. $D^<$ denotes a coloring strategy, where the propagation operators and the choice operators are fully preference-based. More precisely, it denotes the coloring sequence $[(\mathcal{PU})^* \circ \mathcal{D}]^n \circ (\mathcal{PU})^*$ (see Theorem 4.1.15 on page 61). That is, propagating and choosing goes along the given preferences from higher preferred rules to lower preferred ones. In short, DH offers a partial integration of preference information into an ASP solver, where a check is still needed, whereas $D^<$ fully integrates preference information into an ASP solver. The symbol "-" in tables 4.1 and 4.2 means that the examples need more than 1 hour running time or the ASP solvers have memory problems because the examples become too large.

In contrast to the integration of preferences into an ASP solver, the `plp` system [161] compiles an ordered program into a logic program such that the preferred answer sets correspond to the standard answer set of the compiled program. We have used `plp` to compare our integrative approach with the compilation method provided by `plp`. In tables 4.1 and 4.2, $plp + n$ denotes the time for computing preferred answer sets via the `plp` compilation while using the `nomore`$^<$ system as a standard ASP solver. Additionally, we have run the compilation in connection with `smodels` [181] (see $plp + s$ in both tables) for showing differences in the current development status of the `nomore`$^<$ system, when computing standard answer sets. Additionally, we have inserted time measurements for the meta-interpreter [77, 74] (see also Section 2.4), where the test have been run under `smodels` (see $meta + s$ in both tables) and under `nomore`$^<$ as standard ASP solver (see $meta + n$).

We have considered several problem classes, as described in Section 4.2. The independent set and kernel problems as well as the artificial examples have exponentially many answer sets (w.r.t. N), where only one of them is a preferred answer set. The coloring problem has two answer sets, where only one is preferred, and the problem for finding Hamiltonian cycles has exponentially many answer sets, where several ones are preferred. We have chosen the problems, where there exists only one preferred answer set, to show how the integrative approach, the meta-interpreter, and the compilation method can handle the remaining search space w.r.t. non-preferred solutions. In contrast to that, we have additionally concentrated on the Hamilton problem to show, how these three approaches can handle complexer real problems.

Table 4.1 shows the results for finding one preferred answer set. The $D^<$ strategy solves almost all examples, except for the Hamilton problem, in increasing linear time and number of choice points w.r.t. N. In contrast to that, the DH strategy is able to solve the Hamilton examples, whereas the time and the number of choice points increase exponentially. Considering the compilation method, we have chosen `smodels` ($plp+s$ in both tables) and the `nomore`$^<$ system, with the computing standard answer set option ($plp+n$ in

	DH		D$^<$		plp+n		plp+s		meta+n		meta+s	
file	Time	CP	Time	CP	Time	CP	Time	CP	Time	CP	Time	CP
indset_5	0	10	0	2	0	2	0.002	0	0.03	0	0.02	0
indset_10	0	217	0	5	0.02	4	0.006	0	0.3	0	0.14	0
indset_15	0.22	9563	0	7	0.07	7	0.013	0	0.99	0	0.45	0
indset_20	6.907	218462	0.014	10	0.202	9	0.02	0	2.43	0	1.01	0
indset_25	409.019	9786720	0.04	12	0.752	12	0.035	0	5.02	0	1.84	0
indset_30	-	-	0.107	15	2.768	14	0.049	0	9.09	0	3.06	0
indset_35	-	-	0.222	17	6.071	17	0.068	0	15.21	0	5.48	0
indset_40	-	-	0.896	20	10.843	19	0.092	0	23.85	0	7.11	0
art2_10	0	29	0	5	0.002	4	0.004	0	0.25	0	0.14	0
art2_20	0.018	395	0	10	0.03	9	0.012	0	1.83	0	0.8	0
art2_30	0.448	6333	0	15	0.099	14	0.024	0	5.9	0	2.58	0
art2_40	10.388	105004	0	20	0.413	19	0.042	0	14.37	0	6.18	0
art2_50	-	-	0.367	25	01.01	24	0.062	0	27.06	0	11.19	0
art2_60	-	-	0.008	30	1.863	29	0.091	0	-	-	20.03	0
art2_70	-	-	0.01	35	3.037	34	0.126	0	-	-	32.68	0
art2_80	-	-	0.01	40	4.862	39	0.17	0	-	-	49.58	0
art2_90	-	-	0.014	45	6.855	44	0.216	0	-	-	-	-
art2_100	-	-	0.02	50	9.598	49	0.267	0	-	-	-	-
kercomp_10	0.01	55	0.001	10	0.04	10	0.015	0	-	-	33.77	0
kercomp_20	0.076	210	0.02	20	0.496	20	0.052	0	-	-	-	-
kercomp_30	0.314	465	0.063	30	2.089	30	0.126	0	-	-	-	-
kercomp_40	0.973	820	0.142	40	5.095	40	0.231	0	-	-	-	-
kercomp_50	4.03	1275	0.339	50	10.162	50	0.37	0	-	-	-	-
kercomp_60	5.631	1830	0.88	60	18.065	60	0.529	0	-	-	-	-
kercomp_70	19.87	2485	1.229	70	31.531	70	0.726	0	-	-	-	-
kercomp_80	73.329	3240	2.466	80	44.066	80	0.973	0	-	-	-	-
kercomp_90	89.192	4095	3.37	90	62.641	90	-	-	-	-	-	-
kercomp_100	136.808	5050	4.943	100	92.842	100	-	-	-	-	-	-
collad_2	0	4	0	6	0	8	0.001	0	0.25	0	0.13	0
collad_4	0	8	0	12	0.082	37	0.003	0	2.28	0	1.04	0
collad_8	0	16	0	24	16.298	599	0.01	0	22.49	0	8.73	0
collad_10	0.044	20	0.004	30	153.73	2392	0.013	0	-	-	17.06	0
collad_20	0.02	40	0.02	60	-	-	0.045	0	-	-	-	-
collad_30	0.05	60	0.05	90	-	-	0.106	0	-	-	-	-
collad_40	0.091	80	1.295	120	-	-	0.191	0	-	-	-	-
art_10	0.005	345	0	5	0.02	5	0.006	0	0.29	0	0.14	0
art_20	11.137	349534	0.02	10	0.231	10	0.021	0	2.42	0	0.93	0
art_30	-	-	0.134	15	3.072	15	0.049	0	9.16	0	2.94	0
art_40	-	-	1.813	20	11.669	20	0.091	0	24.32	0	6.86	0
art_50	-	-	6.629	25	28.741	25	0.148	0	-	-	14.45	0
art_60	-	-	5.975	30	63.395	30	0.228	0	-	-	25.28	0
art_70	-	-	9.224	35	120.936	35	0.327	0	-	-	38.83	0
art_80	-	-	22.058	40	215.709	40	0.446	0	-	-	63.02	0
art_90	-	-	33.051	45	346.068	45	0.59	0	-	-	-	-
art_100	-	-	56.004	50	521.591	50	0.757	0	-	-	-	-
k3_test_1	0	7	-	-	0.183	5	0.044	1	10.74	0	5.43	0
k4_test_1	0.04	74	-	-	4.062	11	0.367	1	-	-	-	-
k5_test_1	1.008	1068	-	-	37.573	33	1.796	1	-	-	-	-
k6_test_1	46.727	26594	-	-	-	-	6.387	4	-	-	-	-
k7_test_1	-	-	-	-	-	-	17.448	5	-	-	-	-
k8_test_1	-	-	-	-	-	-	41.507	6	-	-	-	-
k3_test_2	0	7	-	-	0.189	5	0.046	1	10.66	0	5.24	0
k4_test_2	0.04	69	-	-	4.051	11	0.362	1	-	-	-	-
k5_test_2	0.43	456	-	-	30.23	14	1.798	3	-	-	-	-
k6_test_2	5.588	3501	-	-	-	-	6.226	4	-	-	-	-
k7_test_2	82.01	30789	-	-	-	-	17.691	5	-	-	-	-
k8_test_2	1778.888	305762	-	-	-	-	42.251	6	-	-	-	-

Table 4.1: Measurements - one preferred answer set.

DH, D$^<$ = integrative approach
meta = metainterpretation
plp+n = compilation method with `nomore`$^<$
plp+s = compilation method with `smodels`

Time = Time in seconds
CP = number of choice points
— = indeterminable value

	DH		D$^<$		plp+n		plp+s		meta+n		meta+s	
file	Time	CP	Time	CP	Time	CP	Time	CP	Time	CP	Time	CP
indset_5	0	11	0	2	0	2	0.002	0	0.03	0	0.02	0
indset_10	0.004	383	0	5	0.02	4	0.006	0	0.3	0	0.14	0
indset_15	17.495	12287	0	7	0.083	7	0.013	0	1.03	0	0.43	0
indset_20	12.374	393215	0.013	10	0.23	9	0.02	0	2.48	0	0.94	0
indset_25	525.581	12582900	0.041	12	0.893	12	0.032	0	5.21	0	2.09	0
indset_30	-	-	0.108	15	3.45	14	0.049	0	9.39	0	3.16	0
indset_35	-	-	0.22	17	7.388	17	0.069	0	15.85	0	5.47	0
indset_40	-	-	0.425	20	12.804	19	0.098	0	24.48	0	7.65	0
art2_10	0	37	0	31	0.02	14	0.004	0	0.25	0	0.13	0
art2_20	0.03	662	0.04	1023	0.254	54	0.012	0	1.82	0	0.8	0
art2_30	0.794	11065	2.025	32767	1.64	119	0.022	0	5.99	0	2.6	0
art2_40	18.322	184203	88.871	1048580	14.992	209	0.041	0	14.53	0	6.36	0
art2_50	-	-	-	-	57.102	324	0.062	0	27.57	0	12.23	0
art2_60	-	-	-	-	150.419	464	0.097	0	-	-	21.18	0
art2_70	-	-	-	-	352.189	629	0.127	0	-	-	33.86	0
art2_80	-	-	-	-	698.799	819	0.169	0	-	-	44.83	0
art2_90	-	-	-	-	1158.669	1034	0.222	0	-	-	-	-
art2_100	-	-	-	-	1957.975	1274	0.263	0	-	-	-	-
kercomp_10	0.01	55	0.08	1023	0.04	10	0.014	0	-	-	32.75	0
kercomp_20	0.077	210	262.254	1048580	0.507	20	0.054	0	-	-	-	-
kercomp_30	0.31	465	-	-	2.128	30	0.126	0	-	-	-	-
kercomp_40	0.979	820	-	-	5.435	40	0.229	0	-	-	-	-
kercomp_50	3.737	1275	-	-	10.055	50	0.359	0	-	-	-	-
kercomp_60	6.145	1830	-	-	17.965	60	0.541	0	-	-	-	-
kercomp_70	20.039	2485	-	-	29.303	70	0.73	0	-	-	-	-
kercomp_80	71.2	3240	-	-	43.596	80	0.942	0	-	-	-	-
kercomp_90	93.259	4095	-	-	64.528	90	-	-	-	-	-	-
kercomp_100	145.146	5050	-	-	84.731	100	-	-	-	-	-	-
collad_2	0	7	0	28	0.001	8	0.001	0	0.25	0	0.12	0
collad_4	0	15	0.01	568	0.088	37	0.003	0	2.33	0	1.07	0
collad_8	0	31	8.373	200736	16.101	599	0.009	0	22.04	0	9.65	0
collad_10	0.008	39	194.735	3753640	157.658	2392	0.013	0	-	-	19.16	0
collad_20	0.385	79	-	-	-	-	0.048	0	-	-	-	-
collad_30	0.063	119	-	-	-	-	0.105	0	-	-	-	-
collad_40	0.119	159	-	-	-	-	0.189	0	-	-	-	-
art_10	0.009	511	0	5	0.02	5	0.006	0	0.3	0	0.13	0
art_20	16.665	524287	0.02	10	0.271	10	0.021	0	2.48	0	0.99	0
art_30	-	-	0.133	15	3.715	15	0.05	0	9.56	0	3.54	0
art_40	-	-	0.554	20	15.07	20	0.093	0	26.5	0	7.9	0
art_50	-	-	1.679	25	35.823	25	0.155	0	-	-	14.25	0
art_60	-	-	4.157	30	78.776	30	0.224	0	-	-	26.62	0
art_70	-	-	9.193	35	149.36	35	0.334	0	-	-	38.58	0
art_80	-	-	18.421	40	265.117	40	0.436	0	-	-	61.22	0
art_90	-	-	33.09	45	414.595	45	0.594	0	-	-	-	-
art_100	-	-	55.994	50	642.926	50	0.756	0	-	-	-	-
k3_test_1	0	7	-	-	0.222	10	0.044	1	11.04	0	5.14	0
k4_test_1	0.04	74	-	-	8.4	55	0.372	2	-	-	-	-
k5_test_1	1.009	1068	-	-	163.59	359	1.783	3	-	-	-	-
k6_test_1	45.997	26594	-	-	-	-	7.471	119	-	-	-	-
k7_test_1	-	-	-	-	-	-	31.961	719	-	-	-	-
k8_test_1	-	-	-	-	-	-	227.254	5050	-	-	-	-
k3_test_2	0	7	-	-	0.221	10	0.044	1	11.06	0	4.94	0
k4_test_2	0.04	69	-	-	8.205	55	0.371	2	-	-	-	-
k5_test_2	0.43	456	-	-	161.446	351	1.865	11	-	-	-	-
k6_test_2	5.589	3501	-	-	-	-	7.49	119	-	-	-	-
k7_test_2	83.004	30789	-	-	-	-	32.201	719	-	-	-	-
k8_test_2	1781.329	305762	-	-	-	-	230.804	5050	-	-	-	-

Table 4.2: Measurements - all preferred answer set.

DH, D$^<$ = integrative approach
meta = metainterpretation
plp+n = compilation method with `nomore`$^<$
plp+s = compilation method with `smodels`

Time = Time in seconds
CP = number of choice points
— = indeterminable value

both tables). $plp+n$ needs more choice points and much more time to solve the problem than $plp+s$ does. The reason for that is that `smodels` is a highly optimized system with heuristics, lookahead, as well as forward and backward propagation, whereas the `nomore`$^<$ system has only forward propagation. Hence, `nomore`$^<$ has a lower base speed than optimized systems. For this reason, we compare the DH and $D^<$ strategy only with $plp + n$, since the measurements rely on the same system. Furthermore, the number of choice points could not be compared directly, since the problem encodings for $plp + n$ have another structure and they use additional symbols. Surprisingly, the DH strategy behaves better than $plp + n$. For example, for Hamiltonian problems the integrative approach provided by the DH strategy is better than the compilation method. Also, the $D^<$ strategy performs in the other examples better than the compilation method. Moreover, the meta-interpretation method is worse than the compilation method and worse than any integrative method provided by the DH and $D^<$ strategy.

Table 4.2 shows the time measurements and number of choice points for finding all preferred answer sets. Comparing the DH and $D^<$ strategy, we obtain that DH performs in most examples and in the Hamilton examples better than $D^<$. $D^<$ is only better for the independent set and for the artificial examples. If we compare the results for both strategies with the results for finding one preferred answer set, then we obtain that $D^<$ has much more difficulties with discovering the search space than DH. Hence, DH performs better while checking whether the answer sets are not preferred. It remains to be further work, to study why $D^<$ has problems with discovering non-preferred solutions. Concerning the compilation method, $plp+s$ is again much better than $plp+n$ since `nomore`$^<$ has a low base speed due to missing optimizations of the system. $plp+n$ behaves better than the integrative approach for the $art2$ examples and for the kernel problems. The DH strategy performs better for the coloring and for the Hamiltonian problems and the $D^<$ strategy performs better than $plp + n$ for the independent set and for the art examples. Whereas the compilation method and the integrative approach need more time and number of choice points for finding all preferred answer sets instead of finding one, the meta-interpreter shows no differences at that point. Again, the meta-interpretation method is worse than the compilation method and worse than the DH or $D^<$ strategy.

To summarize the results, for finding one preferred answer set, the integrative approach provided by the DH and $D^<$ strategy performs better than the compilation method ($plp + n$) or the meta-interpretation ($meta + n$). Furthermore, the $D^<$ strategy performs on most problem classes better than the DH strategy. Concerning finding all preferred answer sets, the integrative approach behaves better than meta-interpretation and the compilation method, whereas DH and $D^<$ seem to be both adequate.

The integrative approach for the computation of preferred answer sets concentrates on one semantics for preference handling. Further research issues should concentrate on other preference semantics and their integration into an ASP solver.

4.4 Related Work

In this section, we give a brief overview of preference semantics within answer set programming and their application areas.

In this chapter, we have considered the D-, W-, and $B-$ preferences, which are rule-based preferences. Another rule-based semantics is due to Zhang and Foo, which is discussed in Section 4.4.1. Beside rule-based preferences, we discuss in Section 4.4.2 weak constraints, which allow to assign weights to rules. In Section 4.4.3 we discuss the literal-based semantics due to Sakama and Inoue and in Section 4.4.4 the more flexible literal-based approach of ordered disjunction. We continue in Section 4.4.5 with CR-Prolog, where preferences are used to restore consistency of answer sets, and in Section 4.4.6, we discuss answer set optimization, which is a combination of answer set programming and qualitative optimization techniques. At last, we take a look at ceteris paribus preferences in Section 4.4.7.

4.4.1 Prioritized Logic Programming (Zhang and Foo)

Another semantics for logic preferences with preferences among rules (cf. on page 13, Section 2.4) is the approach of Zhang and Foo [199, 200, 198].

The idea of a preference $r' < r$ (stating that r is more preferred than r') is that "r is preferred to apply first and then defeat rule r' after applying rule r" [199]. The computation of preferred answer sets is based on an iterative reduction of a logic program with preferences to a logic program without preferences. There, preference relations are iteratively used to remove rules as follows: If $r' < r$ and $\Pi \setminus \{r'\}$ "defeats" r', then rule r' is eliminated from Π if no less preferred rule can be eliminated. The procedure is continued until a fixed point is reached. The answer sets of the resulting program are the preferred answer sets of the original program.

The presented semantics is most related to the B-semantics [34] (cf. on page 13, Section 2.4), but it may lead to unintuitive results. For example, consider the following program with rule preferences [198]:

$$\Pi: \quad \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow not\ c \\ r_3 : c \leftarrow not\ b \\ r_2 < r_1 \end{array}$$

Π has two standard answer sets, namely $\{a, b\}$ and $\{a, c\}$, which are both $<^\sigma$- preferred, for $\sigma \in D, W, B$ (see on page 13, Section 2.4 for more details). But in the semantics of Zhang and Foo, only $\{a, c\}$ is preferred. The reason for this lies in the concept of their "defeatness". Rule r_2 has a lower priority than r_1 to be taken into account in the evaluation of the whole program, while other rules should be retained in the evaluation process if no preference is specified between r_2 and them. Although there exists no conflict between r_1 and r_2 and no preference is specified between r_2 and r_3, in the above given example, r_2 indeed has a lower priority than r_1 to be applied in the evaluation of the reduct of Π, which causes r_2 to be defeated. Hence, r_2 will be eliminated from Π. Thus, only $\{a, c\}$ is the preferred answer set.

An advantage of the semantics from Zhang and Foo is that there always exists a preferred answer set if and only if there exists a standard answer sets of the underlying logic program. In contrast to this, the semantics given on page 13 in Section 2.4 are sometimes so restrictive such that no preferred answer set of an ordered program exists. Furthermore, Zang and Foo's preference semantics fulfills Brewka's and Eiter's Principles I and II [34] (see Section 2.4 on page 13), it is defined for extended logic programs, and it can handle static and dynamic preferences, where the dynamic case is reduced to the static one. An application for their semantics lies in updating logic programs, as presented in [197].

4.4.2 Weak Constraints in DLV

Priorities within the DLV system are modeled by weak constraints [130, 41, 40]. A *weak constraint* is an expression of the form

$$:\sim b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m\ [w : l]$$

where each b_i is a literal, while l (the priority $level$ or layer) and w (the $weight$ among the level) are positive integer constants or variables. The preferred answer sets are those answer sets which minimise the sum of weights of the violated constraints in the greatest priority layer, and among them those which minimise the sum of weights of the violated constraints in the previous layer, etc.

Example 11 *Let us consider the following example [91]*

$$\Pi = \left\{ \begin{array}{l} a \vee b \leftarrow \\ c \leftarrow \\ :\sim a, c\ [2:1] \\ :\sim b[1:10] \end{array} \right\}$$

Clearly, $\{a, c\}$ and $\{b, c\}$ are all answer sets. But only $\{b, c\}$ is a preferred one, since $\{a, c\}$ violates the more important weak constraint $:\sim a, c\ [2:1]$.

Weak constraints have been successfully applied in time tabling problems [91]. Regarding complexity, we obtain the following results for an disjunction logic program Π with weak constraints:

- Deciding whether an atom a is true in some answer set of Π is Δ_3^P-complete.
- Deciding whether an atom a is true in all answer sets of Π is Δ_3^P-complete.
- Deciding whether a set of literals X is an answer set of Π is Π_P^2-complete.

Hence, introducing weak constraints increase the complexity of the underlying decision problems.

4.4.3 Literal-based Preferences (Sakama and Inoue)

Instead of preferences among rules one can define preferences among literals, as presented in [172, 173, 191, 192]. There, the authors consider extended logic programs with a disjunction of literals and default negated literals in the head of a rule, so-called *general extended disjunctive programs* (GEDP). A *prioritized logic program* is then a pair (Π, Φ), where Π is a general extended logic program and Φ is a set of priorities. Priorities are expression of the form $e_1 \leq e_2$ stating that literal e_2 has higher priority than e_1. $e_1 < e_2$ holds whenever $e_1 \leq e_2$ and $e_2 \not\leq e_1$. These priorities form a reflexive and transitive order relation over the set of all literals Lit and default negated literals, i.e. over $Lit \cup \{not\ L \mid L \in Lit\}$. The semantics of (Π, Φ) is given by *preferred* answer sets defined as follows. Let S_1 and S_2 be answer sets of Π. Then, S_2 is preferred over S_1 if for some $e_2 \in S_2 \setminus S_1$ we have

(i) there is an $e_1 \in S_1 \setminus S_2$ such that $e_1 \leq e_2$ holds in Φ; and

(ii) there is no $e_3 \in S_1 \setminus S_2$ such that $e_2 < e_3$ holds in Φ.

Thus, S is a preferred answer set of (Π, Φ) if for any other answer set S' we have: if S' is preferred over S then S is preferred over S'. Intuitively, the preferred answer sets are the answer sets including elements with the highest priority wrt Φ.

Example 12 *Let us consider the following program [173].*

$$\Pi = \left\{ \begin{array}{l} p; q \leftarrow \\ q; r \leftarrow \end{array} \right\} \qquad \textit{and } \Phi : p \leq q, q \leq r$$

Program Π has the answer sets $\{p, r\}$ and $\{q\}$. We have that $\{p, r\}$ is preferred over $\{q\}$, since $q \leq r$ and there is no higher preferred literal than r. But, $\{q\}$ is not preferred over $\{p, r\}$ since $q \leq r$. Hence, $\{p, r\}$ is the preferred answer set.

The advantage of this approach is that there always exists a preferred answer set if there exists at least a standard answer set. If a program has a unique standard answer set, then it is the unique preferred one. Unfortunately, increasing priorities in a prioritized logic program does not always decrease the number of preferred answer sets. This is a disadvantage compared to the semantics of ordered logic programs presented in Section 2.4 on page 13. We also want to point out that priorities over literals can be modeled with rule-based approaches. More precisely, whenever we prefer a literal a over a literal b, the preference relation includes that all rules with a as head are higher preferred than all rules with b as head. Of course, one has to find an adequate semantics for handling such rule-based preferences modeling priorities over literals.

To present an application, priorities over literals can be used for finding minimal explanations within an abduction problem [173] (see also on page 12 in Section 2.2) by using expressions of the form $l; not\ l \leftarrow$

and priorities $l \leq not\ l$ for literals l from the hypothesis of the abduction problem. A similar approach to abduction of programs with priorities is presented in Section 4.4.4 on page 73, where ordered disjunction offers a similar way for computing explanations of abduction problems within answer set programming. Also, these priorities were used for finding preference information to derive desired conclusions in non-monotonic reasoning [109]. There, the notion of *preference abduction* was introduced, in which priorities were abducted to explain given observations within an abduction problem.

Regarding complexity, we have the following results for prioritized logic programs [173] (Π, Φ), where Π is an extended logic program: (i) Deciding the existence of a preferred answer set of (Π, Φ) is NP-complete. (ii) Deciding whether a literal is true in some preferred answer set of (Π, Φ) is Σ_2^P-complete. (iii) Deciding whether a literal is true in all preferred answer sets of (Π, Φ) is Π_P^2-complete. Hence, for the problems (ii) and (iii), introducing priorities to a program causes an increase in complexity by one level of the polynomial hierarchy. More complexity results for general extended disjunctive programs are given in [173].

4.4.4 Ordered Disjunction

Logic programs with ordered disjunction combine ideas underlying qualitative choice logic [33] and answer set programming.

Qualitative choice logic is a propositional logic for representing alternative, ranked options for problem solutions. The logic adds to classical propositional logic a new connective called ordered disjunction. An *ordered disjunction* is, e.g., $A \times B$, which means intuitively that if possible A, but if A is not possible then at least B. This logic is combined with answer set programming in [28, 29, 35, 36].

Logic programs with ordered disjunction (LPODs) are an extension of extended logic programs. The new connective $\times$, representing ordered disjunction, is allowed to appear in the head of rules only and induces a preference relation among answer sets. A (propositional) LPOD thus consists of rules of the form

$$C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m, not\ B_1, \ldots, not\ B_k \tag{4.9}$$

where $n \geq 0, m \geq 0, k \geq 0$ and each C_i, A_j and B_l is a ground literal. The intuitive reading of the rule head is as follows: if possible C_1, if C_1 is not possible then C_2, ..., if all of $C_1, \ldots, C_{n-1}$ are not possible then C_n.

The set of answer sets of an LPOD Π is defined via the answer sets of the split programs of Π, which represent every option of the ordered disjunction. For each rule $r : C_1 \times \ldots C_n \leftarrow body$, the k-th option of r is defined as

$$r^k : C_k \leftarrow body, not\ C_1, \ldots not\ C_{k-1}$$

Then, a split program of an LPOD is obtained by replacing each rule containing ordered disjunction in the head, by one of its options.

Example 13 *Let be* Π *the following LPOD:*

$$\begin{array}{lll} r_1: & A \times B \leftarrow & not\ C \\ r_2: & B \times C \leftarrow & not\ D \end{array}$$

Then, we obtain 4 split programs:

$$\Pi_1 = \left\{ \begin{array}{lll} A \leftarrow & not\ C \\ B \leftarrow & not\ D \end{array} \right\} \qquad \Pi_2 = \left\{ \begin{array}{lll} A \leftarrow & not\ C \\ C \leftarrow & not\ D, not\ B \end{array} \right\}$$

$$\Pi_3 = \left\{ \begin{array}{lll} B \leftarrow & not\ C, not\ A \\ B \leftarrow & not\ D \end{array} \right\} \quad \Pi_4 = \left\{ \begin{array}{lll} B \leftarrow & not\ C, not\ A \\ C \leftarrow & not\ D, not\ B \end{array} \right\}$$

These programs have three answer sets, $\{A, B\}, \{C\}$, and $\{B\}$. The answer set $\{A, B\}$ gives us the best option for both rules from Π, whereas $\{C\}$ gives us only the second best option for rule r_2 and $\{B\}$ the second best option for rule r_1. Intuitively, $\{A, B\}$ should be the preferred answer set, since it gives the best option for every rule containing ordered disjunction.

Following the intuition from Example 13, degrees for satisfaction of a rule where defined in [28], which are used to determine preferred answer sets. Given an answer set S, whenever the body of a preference rule r is satisfied by S, the satisfaction degree $deg_S(r)$ is the smallest index i such that $C_i \in S$. If the body is not satisfied w.r.t. answer set S, then the rule is irrelevant and gets the satisfaction degree 1. Based on this satisfaction degree of single rules, a global preference ordering on answer sets is defined, which can be done through a number of different combination strategies. For this, we define $S^i(\Pi)$ as the set of rules of Π, which are satisfied by S to the degree i.

We have the following conditions for defining that one answer set S_1 is strictly preferred to another one S_2 [36, 30]:

1. *Pareto*: there is a rule $r \in \Pi$ such that $deg_{S_1}(r) < deg_{S_2}(r)$ and for no $r' \in \Pi$ we have $deg_{S_1}(r') > deg_{S_2}(r')$;

2. *Inclusion*: at the smallest degree j where $S_1^j(\Pi) \neq S_2^j(\Pi)$, we have $S_1^j(\Pi) \supset S_2^j(\Pi)$;

3. *Cardinality*: at the smallest degree j, where $|S_1^j(\Pi)| \neq |S_2^j(\Pi)|$, we have $|S_1^j(\Pi)| > |S_2^j(\Pi)|$;

4. *Penalty sum*: the sum of the satisfaction degrees of all rules is smaller in S_1 than in S_2.

S_1 is inclusion preferred to S_2 whenever it is Pareto preferred to S_2, and S_1 is cardinality preferred to S_2 whenever it is inclusion preferred to S_2.

We define an answer set S of LPOD Π as preferred if there exists no other answer set S' of Π such that S' is preferred to S (w.r.t. to the underlying strategy).

Example 14 *Reconsidering the LPOD from Example 13,*

$$\begin{aligned} A \times B \leftarrow &\quad not\ C \\ B \times C \leftarrow &\quad not\ D, \end{aligned}$$

we get $\{A, B\}$ as single preferred answer set w.r.t. Pareto, Inclusion, Cardinality, and Penalty sum.

A nice property of this approach is that there always exists a preferred answer set as long as there exists an answer set [36]. Hence, the semantics of ordered disjunction guarantees that at least one option is accepted, which is useful in many applications. E.g. ordered disjunctions can be used in design and configuration, since they serve as a basis for qualitative decision making [36]. Regarding configuration, ordered disjunction allows to model user's preferences for different versions of a product in the Linux configuration domain [182], e.g. whenever we install emacs, we can express preferences among different versions as follows:

$$emacs\text{-}21.2 \times emacs\text{-}20.7.2 \times emacs\text{-}19.34 \leftarrow emacs$$

Example 15 *Ordered disjunction offers a very natural way for the configuration of a menu [29]. For example, you are in an Asian restaurant and you want to order a menu consisting of a* starter*, a* main *course, and a* dessert*. For a starter you prefer* Jiaozi *(Chinese dumplings) over miso* soup*. For the main course, you can choose between* sushi*, Teriyaki* salmon*, and tandoori* chicken*, where your preferences are in this order. For a better composition of the menu, you can only combine sushi with miso soup and Jiaozi*

*with chicken or salmon as main course. Whenever you take sushi as main course, you prefer to have green-tea-*icecream *over deep-fried* bananas*, whereas and in all other cases you prefer to have bananas over ice-cream. This menu can be modeled with ordered disjunction as follows:*

$$
\begin{array}{lll}
r_1: & starter & \leftarrow \\
r_2: & main & \leftarrow \\
r_3: & dessert & \leftarrow \\
r_4: & jiaozi \times soup & \leftarrow \ starter \\
r_5: & sushi \times salmon \times chicken & \leftarrow \ main \\
r_6: & & \leftarrow \ soup, chicken \\
r_7: & & \leftarrow \ soup, salmon \\
r_8: & & \leftarrow \ jiaozi, sushi \\
r_9: & icecream \times bananas & \leftarrow \ sushi \\
r_{10}: & bananas \times icecream & \leftarrow \ not\ sushi, main
\end{array}
$$

We get the following answer sets of Π*:*

$$
\begin{array}{l}
S_1 = \{starter, main, dessert, jiaozi, salmon, icecream\} \\
S_2 = \{starter, main, dessert, jiaozi, salmon, bananas\} \\
S_3 = \{starter, main, dessert, jiaozi, chicken, icecream\} \\
S_4 = \{starter, main, dessert, jiaozi, chicken, bananas\} \\
S_5 = \{starter, main, dessert, soup, sushi, icecream\} \\
S_6 = \{starter, main, dessert, soup, sushi, bananas\}
\end{array}
$$

Furthermore, we get the following set of rules $S_i^k(\Pi)$ *for answer set* S_i *which are satisfied by* S_i *to the degree* k*. Note that rules* r_1, r_2, r_3 *as well as* r_6, r_7, r_8 *are all satisfied to the degree* 1 *and are not given in the table for better clarity.:*

$AnswerSet$	$degree\ 1$	$degree\ 2$	$degree\ 3$
S_1	$\{r_4\}$	$\{r_5, r_{10}\}$	$\emptyset$
S_2	$\{r_4, r_{10}\}$	$\{r_5, \}$	$\emptyset$
S_3	$\{r_4\}$	$\{r_{10}\}$	$\{r_5\}$
S_4	$\{r_4, r_{10}\}$	$\emptyset$	$\{r_5\}$
S_5	$\{r_5, r_9\}$	$\{r_4\}$	$\emptyset$
S_6	$\{r_5\}$	$\{r_4, r_9\}$	$\emptyset$

For Inclusion preferences, we get $S_5 > S_6$ *and* $S_2 > S_4 > S_1 > S_3$*. Analogously, we get for Pareto, Cardinality, and Penalty sum that* S_2 *and* S_5 *are preferred over the other answer sets. Thus, our preferred menus are (i) Jiaozi, Teriyaki salmon, and deep-fried bananas, as well as (ii) miso soup, sushi, and green-tea icecream.*

Ordered disjunction can also be used for abduction and diagnoses [30] (see also on page 12, Section 2.2). Given an abduction problem $\langle \Pi, H, O \rangle$, where Π is a logic program, H is a set of hypotheses, and O is a set of observations, we can formulate the abduction problem as program with ordered disjunction:

$$
\begin{array}{rcl}
\Pi_{abd} & = & \Pi \cup \{\leftarrow not\ o \mid o \in O\} \cup \\
 & & \{\neg ass(h) \times ass(h) \mid h \in H\} \cup \{h \leftarrow ass(h) \mid h \in H\}, l
\end{array}
$$

where $ass(h)$ means that h is assumed. Then, $\Delta \in H$ is an explanation for O iff there exists a consistent answer set S of Π_{abd} such that $\Delta = \{h \in H \mid ass(h) \in S\}$.

Another important feature of LOPD's is that a restricted version of `DLV`'s weak constraints (for more details see on page 71, Section 4.4.2) can be formulated in terms of an ordered disjunction. More precisely,

weak constraints of the form $\leftarrow body$ with weight n can be represented as $\bot \times ... \times \bot \times \top \leftarrow body$, where the ordered disjunction has n disjuncts.

Regarding complexity results, we have for an LPOD Π the following [36]: Deciding whether Π has a preferred answer set is NP-complete. Let S be an answer set of Π. Then, deciding whether S is preferred is co-NP-complete. Let l be a literal appearing in Π. Then, deciding whether there exists an preferred answer set S such that $l \in S$ is Σ_2^P-complete.

4.4.5 Consistency-Restoring Rules

In the following, we want to consider an extension of answer set programming by consistency restoring rules with given preference relations [8, 9, 55].

A *consistency-restoring* rule (CR-rule) is a statement of the form:

$$r : h_1 \text{ or } \ldots \text{ or } h_k \stackrel{+}{\leftarrow} l_1, \ldots, l_m, \text{ not } l_{m+1}, \ldots, \text{ not } l_n \tag{4.10}$$

where r is the name of the rule, and each h_i and l_j is a literal. This rules express that if $l_1, \ldots, l_m$ belong to a set of agent's belief and none of the $l_{m+1}, \ldots, l_n$ belong to it, then the agent "may possibly" believe one of the $h_1, \ldots, h_k$. This possibility is used only if the agent has no way to obtain a consistent set of beliefs using regular rules only. For example, the program $\Pi = \{a \leftarrow not\ b\}$ has answer set $\{a\}$, but adding $\neg a \leftarrow$ to Π leads to an inconsistence of $\Pi' = \Pi \cup \{\neg a \leftarrow\}$. Consistency of Π' is restored by adding the CR-rule $r_1 : b \stackrel{+}{\leftarrow}$ to Π', which allow the reasoner to believe in b, leading to the answer set $\{\neg a, b\}$.

The agent's selection of CR-rules to restore consistency can be guided by the preference relation

$$prefer(r_1, r_2)$$

which says that sets of beliefs obtained by applying CR-rule r_1 are preferred over those obtained by applying CR-rule r_2.

A *CR-Prolog program* consists of CR-rules of the form (4.10) and of rules of the form

$$r : h_1 \text{ or } \ldots \text{ or } h_k \leftarrow l_1, \ldots, l_m, \text{ not } l_{m+1}, \ldots, \text{ not } l_n \tag{4.11}$$

Furthermore, preferences between CR-rules are expressed by atoms of the form $prefer(r_1, r_2)$ (including the transitive closure of this preference relation).

Programs of CR-Prolog are closely related to abductive logic programs [116, 114] (see also on page 12, Section 2.2). The semantics of CR-Prolog is based on a transformation into abductive programs, where the set of abducibles is a set of atoms, such that each atom refers to the application of one CR-rule. Hence, minimal explanations indicate which CR-rules have to be considered to restore consistency for obtaining *candidate answer sets*. The use of minimal explanations is founded in the fact that only a minimal number of CR-rules have to be applied to restore consistency.

Among the candidate answer sets, the preference statements select the preferred ones. Verbally spoken, a candidate answer set X is a preferred one if there exists no other candidate answer set Y such that there exists an applied CR-rule w.r.t. X and an applied rule r_2 w.r.t. Y such that r_2 is preferred over r_1.

Example 16 *You want to buy a car. Either you can buy a* Smart *or a* Porsche*. Of course, the Smart doesn't look good if it is a* cabriolet *and the Porsche doesn't look good if it has a* sunroof *or if it is a* hardtop *without a sunroof. This can be modeled by the following program:*

$$\Pi = \left\{ \begin{array}{lll} r_1 : & smart & \leftarrow\ not\ cabriolet \\ r_2 : & porsche & \leftarrow\ not\ sunroof, not\ hardtop \\ r_3 : & & \leftarrow\ smart, porsche \end{array} \right\}$$

Clearly, this program has no consistent answer sets. To restore consistency, we can use the following CR-rules:

$$\begin{array}{lll} r_4: & cabriolet & \stackrel{+}{\leftarrow} \\ r_5: & sunroof & \stackrel{+}{\leftarrow} \\ r_6: & hardtop & \stackrel{+}{\leftarrow} \end{array}$$

Furthermore, we want to prefer a cabriolet over a sunroof and a hardtop, which can be modeled by the following preference statements:

$$\begin{array}{lll} r_7: & prefer(r_4, r_5) & \leftarrow \\ r_8: & prefer(r_4, r_6) & \leftarrow \end{array}$$

To carry over the program into an abduction problem (for more details see on page 12, Section 2.2), we use the set of abducibles $\{appl(r_4), appl(r_5), appl(r_6)\}$, where $appl(r_i)$ denotes that CR-rule r_i is applied. Furthermore, we have to replace rules r_4, r_5, r_6 by

$$\begin{array}{lllll} r'_4: & cabriolet & \leftarrow & appl(r_4) \\ r'_5: & sunroof & \leftarrow & appl(r_5) \\ r'_6: & hardtop & \leftarrow & appl(r_6) \end{array}$$

Hence, the minimal explanations of our abduction problem are $\{appl(r_4)\}$, $\{appl(r_5)\}$, and $\{appl(r_6)\}$. as a result, we get the following candidate answer sets:

$$\begin{array}{l} C_1 = \{prefer(r_4, r_5), prefer(r_4, r_6), appl(r_4), cabriolet, porsche\} \\ C_2 = \{prefer(r_4, r_5), prefer(r_4, r_6), appl(r_5), sunroof, smart\} \\ C_3 = \{prefer(r_4, r_5), prefer(r_4, r_6), appl(r_6), hardtop, smart\} \end{array}$$

Since we want to prefer rule r_4 over r_5 and r_6, we get the preferred answer set:

$$\hat{C}_1 = \{prefer(r_4, r_5), prefer(r_4, r_6), cabriolet, porsche\}$$

CR-rules can be used to encode types of common-sense knowledge, where there is no natural formalization within answer set programming. E.g. CR-Prolog is applied in diagnostic reasoning [8] and within planning problems for the decision support system of the Space Shuttle [7, 151], where it is used by flight controllers to find plans for the operation of the Reactive Control System, as well as checking correctness of existing plans.

CR-Prolog has been extended with ordered disjunction in [9]. (For further information about ordered disjunction see also on page 73, Section 4.4.4). There, ordered disjunction is allowed to appear in heads of rules, representing preferences among literals guaranteeing consistent solutions. Furthermore, their approach yields intuitive conclusions in cases for which the semantics of ordered disjunction seems to yield unintuitive results.

Example 17 *[9] " A television show conducts a game where the first winner is offered a prize of* \$200000 *and the second winner is offered a prize of* \$100000*. John wants to play, if possible. Otherwise he will give up. If he plays he wants to gain* \$200000 *if possible; otherwise,* \$100000*. He is told that he cannot win the first prize." This is modeled as the following logic program with ordered disjunction:*

$$\begin{array}{ll} r_1: & play \times give\_up \leftarrow \\ r_2: & gain(200000) \times gain(100000) \leftarrow play \\ r_3: & \leftarrow gain(200000) \end{array}$$

Intuitively, John should play and gain \$100000*. Under the LPOD semantics, this program has the two answer sets $\{play, gain(100000)\}$ and $\{give\_up\}$ with the following satisfaction degrees for rules r_1, r_2 and r_3:*

Answer Set	*degree 1*	*degree 2*
$\{play, gain(100000)\}$	r_1, r_3	r_2
$\{give\_up\}$	r_2, r_3	r_1

We get that both answer sets are preferred since neither answer set $\{play, gain(100000)\}$ *is strictly better than answer set* $\{give\_up\}$ *nor* $\{give\_up\}$ *is strictly better than* $\{play, gain(100000)\}$ *w.r.t. Pareto, Inclusion, Cardinality, and Penality sum. This is caused by the fact that the satisfaction degree* 1 *is assigned to rules whose body is not satisfied.*

The same program under the CR-Prolog semantics gives only $\{play, gain(100000)\}$*, the intuitive one, as preferred answer set, since the preference relations are kept during the process of computing preferred answer sets. (For more details, see [9].)*

4.4.6 Answer Set Optimization

In [37], the combination of answer set programming and qualitative optimization techniques have been investigated. Such a combination, called an *answer set optimization (ASO) program* consists of two parts. First, a generating program Π_{gen}, which produces answer sets representing possible (acceptable) solutions, and, second, a preference program Π_{pref} in which user's preferences for optimization methods are formulated. These preferences are used to compare answer sets of Π_{gen} such that a form of preference ordering under these acceptable solutions can be made.

In the following we will assume that we have generated with Π_{gen} answer sets representing possible, acceptable solutions. Next, we will concentrate on the preference program, which selects the optimal answer sets. In ASO programs, we can define preferences among boolean expressions, e.g. $a > b$ means that we prefer a over b and $(fish \vee vegetarian) > meat$ expresses that we prefer fish or a vegetarian meal over meat. A *boolean combination* over a set of atoms A is a formula built of atoms in A by means of disjunction, conjunction, classical ($\neg$) and default negation (not), with the restriction that classical negation is allowed to appear only in front of atoms and default negation only in front of literals.

Then, a *preference program* Π_{pref} over a set of atoms A is a finite set of rules of the form

$$C_1 > \ldots > C_k \leftarrow a_1, \ldots, a_n, not\, b_1, \ldots, not\, b_m$$

where each a_i and b_j are literals, and each C_i is a boolean combination over A.

Given an answer set S of Π_{gen}, we say that a rule r in the preference program Π_{pref} is *irrelevant* if either the body of r is not satisfied in S, or the body of r is satisfied in S, but none of the C_i's in the head are satisfied in S.

Next, we can define the satisfaction degree $v_S(r)$ of a rule r in an answer set S:

$$v_S(r) = \begin{cases} I & \text{if } r \text{ is irrelevant in } S \\ min\{i : S \models C_i\} & \text{otherwise} \end{cases}$$

where the values 1 and I are regarded as equally good ($1 \geq I$ and $I \geq 1$) and better than all others. Given two answer sets S_1 and S_2, we say that S_1 is better than S_2, written as $S_1 \geq S_2$, if for every $r \in \Pi_{pref}$ we have that $v_{S_1}(r) \geq v_{S_2}(r)$. Additionally, S_1 is strictly better than S_2, written as $S_1 > S_2$, if $S_1 \geq S_2$ and for at least one $r \in \Pi_{pref}$ we have $v_{S_1}(r) > v_{S_2}(r)$. Thus, an answer set S is an *optimal model* of an ASO program (Π_{gen}, Π_{pref}) if S is an answer set of Π_{gen} and there is no answer set S' of Π_{gen} such that $S' > S$

Example 18 *Assume you are moving into a new flat and you have to furnish your living room. On the floor you can put either a* carpet *or modern* laminate*. You have to buy a table, where you can choose between a simple* wood table *or a modern* glass table*. Regarding seating, you can choose between a standard* sofa

or a leather sofa*. To put your stuff somewhere, you can either buy a* wall_unit*, a* commode*, or a* shelf*. All these possibilities and combinations can be generated by the following program* Π_{gen}*:*

$$\begin{array}{ll} 1\,\{glassTable, woodTable\}\,1 & \leftarrow \\ 1\,\{carpet, laminate\}\,1 & \leftarrow \\ 1\,\{leatherSofa, sofa\}\,1 & \leftarrow \\ 1\,\{wall\_unit, shelf, commode\}\,1 & \leftarrow \end{array}$$

We have 24 *answer sets, that is,* 24 *possible combinations of furniture for our living room. Of course, everybody has his preferences. If you choose to have a modern laminate floor, you prefer the leather sofa, the glass table, and the shelf. Otherwise, if you want a carpet, you prefer a normal sofa, the wood table, and not the modern shelf. These preferences can be expressed by the following preference program* Π_{pref}*:*

$$\begin{array}{lll} leatherSofa > sofa & \leftarrow & laminate \\ shelf > commode > wall\_unit & \leftarrow & laminate \\ glassTable > woodTable & \leftarrow & laminate \\ sofa > leatherSofa & \leftarrow & carpet \\ woodTable > glassTable & \leftarrow & carpet \\ wall\_unit \vee commode > shelf & \leftarrow & carpet \end{array}$$

As preferred answer sets we get:

$$\begin{array}{l} S_1 = \{laminate, leatherSofa, shelf, glassTable\} \\ S_2 = \{carpet, sofa, woodTable, wall\_unit\} \\ S_3 = \{carpet, sofa, woodTable, commode\} \end{array}$$

An extension of answer set optimization is presented in [31], where each C_i in the head of a preference rule gets a special "weight", so called *penalty*. That is, the preference program consists of rules of the form:

$$C_1 : p_1 > C_2 : p_2 > \ldots > C_k : p_k \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$$

where each a_i and each b_j are literals, each C_i is a boolean expression, and each p_i is an integer such that $p_i < p_j$ for $i < j$. The preference rules $C_1 > C_2 > \ldots > C_k \leftarrow body$ are a shorthand for $C_1 : 0 > C_2 : 1 > \ldots > C_k : k\text{-}1 \leftarrow body$. With these penalties we can find a preorder (a reflexive and transitive relation) on answer sets depending on different criteria. For example, one can find a Pareto ordering or a lexicographic ordering on the set of answer sets.

An advantage of ASO programs is that answer set generation and answer set comparison are decoupled. Hence, preferences can be defined independently from the type of the generating program. That is, the generating program may contain classical negation, weight constraints, or other constructs as presented in Example 18. Furthermore, every user can describe independently from the generating program his preferences, which is an advantage in the sense of usability.

Answer set optimization can be used to schedule meetings. For this, let Π be a program describing meetings, required participants, time restrictions of participants, etc., and S be the set describing the scheduled meetings. When now new time restrictions for a participant arise then S may no longer be a valid solution of the meeting scheduling problem. To ensure that the new solution is as close as possible to the original one, we can use answer set optimization techniques as given in [30]. Also, answer set optimization can be used for inconsistency handling [30].

The complexity of AS0 programs depends on the class of generating programs we are considering. In the following we will assume that we consider only generating programs where deciding existence of an answer set is NP-complete [31].

- Let S be an answer set of Π_{gen}. Then, deciding whether S is optimal is co-NP-complete.

- Given an ASO program Π and a literal l, deciding whether there is an optimal answer set S such that $l \in S$ is Σ_2^P-complete.

Hence, allowing preferences raises the complexity.

Ordered disjunction $\times$ (for more details see Section 4.4.4 on page 73) and preference statements based on $>$ as in optimization programs have their merits. The latter can express preferences based on formulas satisfied in answer sets, even if corresponding generating programs are not available. The former are highly convenient whenever preferences are expressed among options which have to be generated anyway, as it is the case in abduction and diagnosis.

ASO programs have also been combined with CP-nets in [38], where techniques from CP-nets are used to compute most preferred outcomes, hence preferred answer sets. (For more details about CP-nets see on page 80, Section 4.4.7.) Furthermore, the ranked based description language introduced in [32] shares some motivations with answer set optimization, where it allows to express nested combinations of ranked knowledge bases together with preference strategies using various connectives.

4.4.7 Ceteris Paribus Preferences

In [105] and later in [22, 23, 21], preference relations where introduced, where preferences are expresses under the assumption that *everything else is being equal*. For example, when buying a car you express something like "I prefer the silver car over the yellow car". But this does not mean that you prefer the silver small car from Fiat over the yellow Audi A4. Rather, you mean that you prefer the silver Audi A4 over the yellow Audi A4. Such kinds of preferences are called *ceteris paribus* preference.

First, we will present some formal background and then, we will consider a graphical representation of ceteris paribus preferences.

We assume a set of *variables* $V = \{X_1, \ldots, X_n\}$ over which the decision maker has preferences, e.g. variables are the color of a car and the name of the car manufacturer. Each variable X_i is associated with a domain $Dom(X_i) = \{x_i^1, \ldots, x_i^{n_i}\}$ of *values* it can take, e.g. "yellow" and "silver" are the values of the variable color. An assignment x of values $X \subseteq V$ is a function that maps each variable in X to an element of its domain. We denote the set of all assignments to $X \subseteq V$ by $Asst(X)$. An outcome is a state where all variables are instantiated by one of its values. The set of all outcomes is denoted by $\mathcal{O}$. Then, a *preference relation* is a total preorder $\succeq$ over the set of outcomes $\mathcal{O}$. $o_1 \succeq o_2$ means that outcome o_1 is equally or more preferred than outcome o_2 and $o_1 \succ o_2$ denotes than o_1 is *strictly* preferred over o_2. Furthermore, $o_1 \sim o_2$ denotes indifference of o_1 and o_2, i.e. $o_1 \succeq o_2$ and $o_2 \succeq o_1$ hold. The aim of the decision maker is to get the most preferred outcome.

A set of variables X is *preferentially independent* [23] of its complement $Y = V \setminus X$ if for all $x_1, x_2 \in Asst(X)$ and $y_1, y_2 \in Asst(Y)$ we have

$$x_1 y_1 \succeq x_2 y_1 \text{ iff } x_1 y_2 \succeq x_2 y_2$$

In other words, the structure of the preference relation over assignments to X, when all other variables are held fixed, is the same no matter what values these other variables take. If the relation above holds, we say x_1 is preferred to x_2 ceteris paribus. Thus, one can assess the relative preferences over assignments to X once, knowing these preferences do not change as other attributes vary.

Let X, Y, Z be nonempty sets that partition V. X is *conditionally preferentially independent* of Y given an assignment z to Z if for all $x_1, x_2 \in Asst(X)$ and $y_1, y_2 \in Asst(Y)$ we have

$$x_1 y_1 z \succeq x_2 y_1 z \text{ iff } x_1 y_2 z \succeq x_2 y_2 z$$

In other words, X is preferentially independent of Y when Z is assigned z.

Now, we consider the graphical representation of ceteris paribus preferences presented in [22, 23, 21] which reflects conditional dependence and independence of such kinds of preference statements.

Given a variable X_i, we identify a set of parent variables $Pa(X_i)$ that can affect the users preferences over various values of X_i. The user should be able to determine a preference order for the values of X_i, all other things being equal. The node X_i is annotated with a *conditional preference table* (CPT) describing the user's preferences over the values of X_i given every combination of parent values. A *CP-net* over variables $V = \{X_1, \ldots X_n\}$ is a directed graph G over $X_1, \ldots X_n$ whose nodes are annotated with conditional preference tables $CPT(X_i)$ for each $X_i \in V$. Each $CPT(X_i)$ associates a total order $\succ_u^i$ with each instantiation u of X_i's parents $Pa(X_i) = U$

Example 19 *Imagine it is Saturday evening and you have to buy something for dinner on Sunday, when your mother-in-law will be visiting you. Since it is Saturday evening, the supermarket does not have much offers. For the main course M, you can buy fish M_f or pork cutlets M_p, where your mother-in-law prefers to have fish. As side dish V, you can only buy carrots V_c or leek V_l. Your choice of vegetables depends on whether you want to cook fish or meat. Whenever you cook fish, you prefer leek over carrots and whenever you cook meat, you prefer carrots over leek. For a dessert D you can choose between yogurt D_y and mousse au chocolate D_c, where you prefer to have mousse au chocolate, which is independently from your choice of the prepared meal.*

In Figure 4.5 we have given the corresponding CP-net. As we can see, the vegetables depend conditionally on the main course and the dessert is preferentially independent from the main course and from the choosen vegetables.

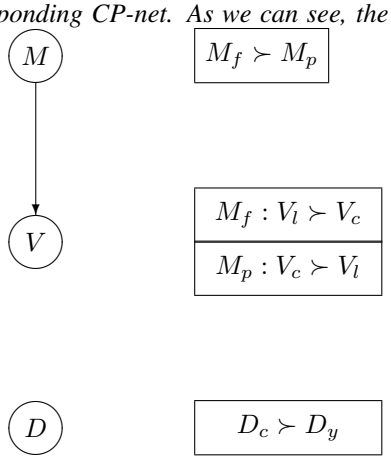

Figure 4.5: CP-net for the supermarket

Another graphical representation of a CP-net is the *induced preference graph* of a CP-net, which is a directed graph, where the set of all nodes is the set of all outcomes and 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 tables. The top elements of the preference graph are the *worst* outcomes, and the bottom elements are the *best* outcomes.

Example 20 *Let us reconsider Example 19. The corresponding preference graph is given in Figure 4.6. As we can see, buying fish, leek, and mousse au chocolate is the most preferred outcome.*

A CP-net N is satisfied by $\succ$ iff $\succ$ satisfies each of the conditional preferences expresses in the CPT's of N under the ceteris paribus interpretation. Every acyclic CP-net is satisfiable. N *entails* $o \succ o'$ (outcome o is preferred over outcome o'), written $N \models o \succ o'$, iff $o \succ o'$ holds in *every* preference ordering that satisfies N. Preferential entailment is transitive. That is, if $N \models o \succ o'$ and $N \models o' \succ o''$ then $N \models o \succ o''$.

Example 21 *Coming back to the CP-net N of our Example 19, we get $N \models M_f \wedge V_l \wedge D_c \succ M_f \wedge V_c \wedge D_c$, $N \models M_f \wedge V_l \wedge D_c \succ M_f \wedge V_c \wedge D_y$, etc. However, e.g., we can neither entail that outcome $M_p \wedge V_c \wedge D_c$ is preferred of $M_f \wedge V_c \wedge D_y$, or vice versa.*

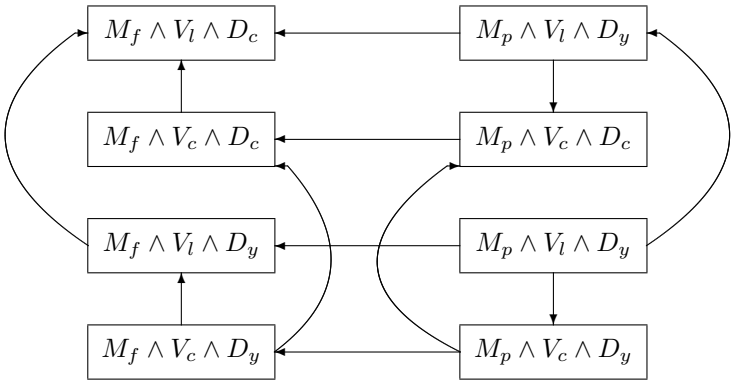

Figure 4.6: Induced Preference Graph for the CP-net of the supermarket

Please not that in general, most preferred outcomes are searched by so called *flipping sequences*. There, a sequence of increasingly preferred outcomes is constructed, which correspond to paths in the induced preference graph. A "flip" from one outcome to another is possible, whenever there is a ceteris paribus preference among both outcomes. That is, these both outcomes differ only in one value of a variable for which a preference is expressed in the conditional preference table. For more details see [22, 23].

4.4.8 Summary

In this section, we have presented several approaches for handling preferences. In general, preferences are used to resolve indeterminate solutions. Current existing approaches offer a variety for modeling preferences. We have considered rule-based and literal-based approaches and several semantics for handling such kinds of preferences. Rule-based approaches are, for example, ordered programs along with the D-, W-, and B-semantics (cf. Section 2.4) and the semantics from Zhang and Foo (cf. Section 4.4.1). On the one hand, the D-, W-, and B-semantics do not guarantee that there always exists a preferred answer set, whereas the semantics from Zhang and Foo guarantees the existence of preferred answer sets as long as answer sets of the underlying logic program exist. On the other hand, given the set of all answer sets, preferred answer sets according to the D-, W-, and B-semantics are computable in polynomial time, whereas it is not polynomial for the semantics due to Zhang and Foo [197, 65]. One approach for literal-based preferences is described in Section 4.4.3. An advantage of this approach is the guarantee of the existence of a preferred answer set as long as there exists an answer set of the underlying logic program. But unfortunately, the complexity is increased by one level of the polynomial hierarchy. We also want to point out that priorities over literals can be modeled with rule-based approaches. More precisely, whenever we prefer a literal a over a literal b, the preference relation includes that all rules with a as head are higher preferred than all rules with b as head. Of course, one has to find an adequate semantics for handling such rule-based preferences modeling priorities over literals.

In contrast to rule-based or literal-based preferences, we have studied other strategies for modeling preferences. Weak constraints assign values to integrity constraints, which eliminate solutions. Preferred answer sets are those answer sets, which minimise the weights of the violated constraints. Ordered disjunction is in the broadest sense a literal-based semantics, where preferences among literals appear in the head

of rules. Those kind of preferences are adequate for configuration problems, where you have to choose one item from a set of possibilities. Furthermore, within this semantics there always exists a preferred answer set of the program as long as there exists an answer set. Answer set optimization is in the broadest sense an extension of ordered disjunction. There, it is possible to express preferences about Boolean formulas, which is very useful for configuration problems. Moreover, we have presented Ceteris Paribus preferences for the sake of completeness. In Section 6.1 we refer to it as related work. Ceteris Paribus preferences are more or less a property, which preference relations can have. Last, but not least, we have considered consistency restoring rules. They are designed to restore consistency of answer sets, e.g. if a program has no answer set, consistency restoring rules are used to "repair" a problem such that there exists an answer set. They are very helpful in configuration problems to guarantee that a configuration exists.

Also, preferences have been used for updating logic programs [1, 197], for revision [160], within planning [63, 183], and preference semantics have been considered under the well-founded semantics, e.g. [27, 176]. Other approaches of preference handling within logic programming are, for example, [104, 101, 71, 162]. A brief overview about preference semantics is given in [64].

4.5 Summary

In this chapter, we have concentrated on different methods for computing preferred answer sets. In Section 4.1, we have presented a graph-theoretical characterization for computing preferred answer sets, concerning the D-, W-, and B-strategy (cf. Section 2.4), respectively. Preferences are simply taken as an additional type of edges in the rule dependency graph, reflecting an additional dependency among rules. In particular, we have demonstrated that this approach allows us to capture all three approaches for preference handling in a uniform setting by means of the concept of a height function. To a turn, we have exemplarily developed an operational characterization for one of these strategies. For this purpose, we have extended the proposed operational framework for graph-based computation of answer sets from Chapter 3. This operational framework allows us to integrate preference information fully or partially into a solver.

This integrative approach for computing preferred answer sets has been implemented in the `nomore`$^<$ system, presented in Section 4.3. Additionally, we have compared our integrative approach with the compilation method of preference handling provided by the `plp` system and the meta-interpretation method. For the experiments we have defined new benchmarks for logic programs with preference handling in Section 4.2. It turned out that the integrative approach performs better on the considered problem classes than the compilation or meta-interpretation method.

In Section 4.4 we have presented related work on preference handling and their application.

Chapter 5

Notions of Equivalence for Logic Programs with Preferences

The availability of efficient solvers has stimulated the use of Answer Set Programming (ASP) in practical applications in recent years. This development had quite some implications on ASP research. For example, increasingly large applications require features for modular programming. Another requirement is the fact that in applications, ASP code is often generated automatically, calling for optimization methods which remove redundancies, as also found in database query optimizers. For these purposes the rather recently suggested notion of strong equivalence for ASP [133, 188] can be used. Indeed, if two ASP programs are strongly equivalent, they can be used interchangeably in any context. This gives a handle on showing the equivalence of ASP modules. Moreover, if a program is strongly equivalent to a subprogram of itself, then one can always use the subprogram instead of the original program, a technique which serves as an effective optimization method.

In this chapter, we tackle this issue and generalize the notion of strong equivalence to ASP with preferences. Since several formalisms and semantics has been introduced for extending ASP by preferences, we have to limit ourselves in this chapter to ordered logic programs (cf. Section 2.4 on page 13) with the underlying D-, W-, and B-semantics [62, 194, 34]. The reason for this choice is that these semantics seem to be widely accepted and their properties and interrelationships are fairly well-understood.

In Section 5.1 we define the novel notion of strong order equivalence for logic programs with preferences (ordered logic programs). In analogy to standard strong equivalence, we consider for strong order equivalence extensions by ordered programs. Based on this definition, we present for the first time, for three semantics for preference handling, necessary and sufficient conditions for programs to be strongly order equivalent. These results allow us also to associate a novel structure, so-called SOE structure, to each ordered logic program, such that two ordered logic programs are strongly order equivalent if and only if their SOE structures coincide. We also present the relationships among the studied semantics with respect to strong order equivalence, which differs considerably from their relationships with respect to preferred answer sets. Furthermore, we study the computational complexity of several reasoning tasks associated to strong order equivalence. Finally, based on the obtained results, we present – for the first time – simplification methods for ordered logic programs.

Instead of considering program extensions by *ordered* programs, we concentrate in Section 5.2 on extensions by *normal* programs. Whereas strong order equivalence makes great demands on ordered logic programs, this weakened notion, called *n-strong order equivalence* [1], enables additional program transformations, particularly simplifications of preference relations. Also, we study the relationship among the

[1] We have called this notion weak order equivalence in [119], but refer to it now as n-strong equivalence according to the extensions by *n*ormal programs.

considered preference semantics and computational complexity issues. [2]

We conclude with Section 5.3, where we compare both notions of order equivalence, give related work, and future research issues.

5.1 Strong Order Equivalence

In this section, we define the notion of strong order equivalence, a generalization of strong equivalence to ordered logic programs. We declare that two ordered programs are strongly order equivalent, if they have the same preferred answer sets no matter which ordered program we add to them. It turns out that the definition for standard ASP cannot be straightforwardly extended for two main reasons: (1) the "union" of ordered programs needs to be defined properly, and (2) not any "union" of ordered programs yields a valid ordered program. To overcome these problems, we define the notion of *admissible extension*, which provides a general method for defining strong equivalence for arbitrary structures. We show several properties of strong order equivalence, providing necessary and sufficient conditions for programs to be strongly order equivalent. We also explore the relationship of strong order equivalence between three semantics for preference handling, and the relationship to strong equivalence of standard logic programs. These results also allow for the novel definition of a SOE-structures, which are analogous to SE-models for strong equivalence. Furthermore, we study the computational complexity of strong order equivalence and associated reasoning tasks. It turns out that testing strong order equivalence is precisely as difficult as testing strong equivalence for logic programs without preferences. Based on these results, we study the applicability of program simplification rules, known from standard ASP, in the presence of preferences. We obtain that many of these simplifications can be applied, provided that the simplified rules do not occur in the preference relation. Other important consequences of this work are that the relationship between the three preference semantics is different under strong equivalence than under answer sets and that preferences cannot be "compiled away" under strong equivalence.

This section is organized as follows: In Section 5.1.1 we define strong order equivalence, and we present properties and characterizations of it, along with the notion of SOE-structures in Section 5.1.2. The complexity of testing strong equivalence and related tasks is analyzed in Section 5.1.3. After assessing the applicability of program simplifications in Section 5.1.4, we draw conclusions in Section 5.1.5.

5.1.1 Strong Equivalence for Preferred Answer Sets

In this section, we define variants of equivalences for ordered logic programs. Defining "standard" equivalence (two programs permit precisely the same answer sets) is straightforward by considering the respective preferred answer sets.

However, concerning the notion of strong equivalence, there are some caveats. For standard programs, strong equivalence for Π_1 and Π_2 holds if for any program Π, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets. If we now take preferences into account, for any ordered program $(\Pi, <)$, $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ would be strongly equivalent if $(\Pi_1 \cup \Pi, <_1 \cup <)$ and $(\Pi_2 \cup \Pi, <_2 \cup <)$ have the same preferred answer sets. The problem is that $(\Pi_1 \cup \Pi, <_1 \cup <)$ is not necessarily an ordered logic program, because of two issues: 1. Rule label clashes, and [3] 2. partial order clashes.

Rule label clashes, i.e. two different rules have the same name or one rule has two different names, can be resolved by a *renaming*.

Definition 5.1.1 *Let $(\Pi, \mathcal{N}, n, <)$ be an ordered program.*

We say that $(\Pi, \mathcal{N}^*, n^*, <^*)$ is a renaming *of $(\Pi, \mathcal{N}, n, <)$ if the following holds:*

[2] Complexity results for n-strong order equivalence have not been presented in [119]. They are presented in this chapter for the first time.

[3] Note that rule label clashes are caused by the definition of ordered logic programs presented in [62], where there exists a bijective function between rules and names for rules.x

1. *$\mathcal{N}^*$ is a new set of names of rules,*
2. *$n^*(.)$ is a bijective function assigning each rule of Π an unique name of $\mathcal{N}^*$, and*
3. *$n^*(r) <^* n^*(r')$ holds iff $n(r) < n(r')$, for all $r, r' \in \Pi$.*

As an example, let us consider the following ordered programs:

$$(\Pi_1, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow \end{array} \right\} \quad \text{and} \quad (\Pi_2, <) = \left\{ \begin{array}{l} r_2 : a \leftarrow \\ r_1 : b \leftarrow a \\ r_1 < r_2 \end{array} \right\} \tag{5.1}$$

For the union of $(\Pi_1, \emptyset)$ and $(\Pi_2, <)$ we have different rule label clashes. First, the rule $a \leftarrow$ has two different names. Second, the name "r_1" is assigned to different rules. A renaming of program $(\Pi_2, <)$ would resolve these clashes. Our new set of names is $\mathcal{N}^* = \{r_3, r_4\}$, rule $a \leftarrow$ gets the name r_3 and rule $b \leftarrow a$ gets the name r_4. Furthermore, we get $r_4 <^* r_3$ as renamed preference relation. Hence, we get the resulting program

$$(\Pi_2, <^*) = \left\{ \begin{array}{l} r_3 : a \leftarrow \\ r_4 : b \leftarrow a \\ r_4 <^* r_3 \end{array} \right\}$$

Now, we can define the union of two ordered logic programs.

Definition 5.1.2 *Let $(\Pi, \mathcal{N}, n, <)$ and $(\Pi', \mathcal{N}', n', <')$ be ordered logic programs where $\mathcal{N}, \mathcal{N}'$ are disjoint sets of rule names.*

We define the union of $(\Pi, \mathcal{N}, n, <)$ and $(\Pi', \mathcal{N}', n', <')$ as $(\Pi^*, \mathcal{N}^*, n^*, <^*)$, where

- $\Pi^* = \Pi \cup \Pi'$
- *$\mathcal{N}^*$ is a new set of names of rules,*
- *$n^*(r)$ for $r \in \Pi \cup \Pi'$ is a bijective function assigning each rule of $\Pi \cup \Pi'$ an unique name of $\mathcal{N}^*$, and*
- *$<^* = < \cup <'$, where for $r, r' \in \Pi \cup \Pi'$, $n^*(r) <^* n^*(r')$ holds whenever $n(r) < n(r')$ or $n'(r) <' n'(r')$ holds.*

We usually leave $\mathcal{N}^*$ and n^* implicit. This is a reasonable assumption, as one can assume that all rules are labeled by themselves (assuming a standardized ordering of the rule body). Sometimes we write $(\Pi \cup \Pi', < \cup <')$ for the union of $(\Pi, <)$ and $(\Pi', <')$.

Coming back to our example programs given in (5.1), we obtain

$$(\Pi_1, \emptyset) \cup (\Pi_2, <^*) = \left\{ \begin{array}{l} r_5 : a \leftarrow \\ r_6 : b \leftarrow \\ r_7 : b \leftarrow a \\ r_7 <' r_5 \end{array} \right\}$$

as union of $(\Pi_1, \emptyset)$ and $(\Pi_2, <^*)$.

We next observe that the union of ordered logic programs is not necessarily an ordered logic program. For this, consider

$$(\Pi_1, <_1) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow a \\ r_2 <_1 r_1 \end{array} \right\} \quad \text{and} \quad (\Pi_2, <_2) = \left\{ \begin{array}{l} r_3 : a \leftarrow \\ r_4 : b \leftarrow a \\ r_3 <_2 r_4 \end{array} \right\}$$

Then, we have

$$(\Pi^*, <^*) = \left\{ \begin{array}{l} r_5 : a \leftarrow \\ r_6 : b \leftarrow a \\ r_5 <^* r_6 \\ r_6 <^* r_5 \end{array} \right\}$$

where $<^*$ is not a strict partial order. Hence, $(\Pi^*, <^*)$ is not an ordered logic program. To avoid this, we define the notion of admissible extensions of an ordered logic program.

Definition 5.1.3 *Let $(\Pi, <)$ and $(\Pi', <')$ be ordered logic programs.*

Then, $(\Pi', <')$ is an admissible extension of $(\Pi, <)$ if $(\Pi \cup \Pi', < \cup <')$ is an ordered logic program.

We remark that the notion of compatibility for orderings, as defined in [34], is similar in spirit to admissible extensions, but concerns only the order relation and not the rules in the program.

Analogously to (normal) logic programs, we can define the notion of equivalence of ordered logic programs.

Definition 5.1.4 *(Order equivalence) Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered logic programs and $\sigma \in \{D, B, W\}$. Then, $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ are* $<^\sigma$-equivalent *iff $AS^\sigma((\Pi_1, <_1)) = AS^\sigma((\Pi_2, <_2))$.*

As an abbreviation, in the sequel we write $\equiv^\sigma$ for $<^\sigma$- equivalence.

Since the definition of strong equivalence for logic programs can be interpreted as having the same answer sets in any extension, we employ the notion of admissible extensions for defining the analogue for ordered logic programs.

Definition 5.1.5 *(Strong order equivalence) Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered logic programs and $\sigma \in \{D, B, W\}$.*

Then, $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ are strongly $<^\sigma$-equivalent *iff for all admissible extensions $(\Pi, <)$ of $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ it holds that $(\Pi_1 \cup \Pi, <_1 \cup <)$ and $(\Pi_2 \cup \Pi, <_2 \cup <)$ are $<^\sigma$- equivalent.*

As an abbreviation, we write $\equiv_s^\sigma$ whenever two ordered logic programs are strongly $<^\sigma$-equivalent.

5.1.2 Properties and Relationships

In this section, we will give characterizations of strong order equivalence. In particular, we present several necessary conditions for strong order equivalence, which, taken together, can also be shown to be sufficient. These results also allow for establishing the relationship of strong order equivalence under different semantics.

5.1.2.1 Preference Relation

For finding necessary conditions for strong order equivalence, a promising candidate is comparing the preference relations of the two ordered logic programs. To get a better idea of the impact of the preference relation, let us first look at an example.

Example 22 *Consider the logic program*

$$\Pi = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow a \end{array} \right\}$$

and the preference relation $<$ in which only $r_2 < r_1$ holds. Let us now study the relationships between $(\Pi, \emptyset)$ and $(\Pi, <)$.

We first observe that $(\Pi, \emptyset) \equiv^\sigma (\Pi, <)$ holds for all $\sigma \in \{D, W, B\}$. Indeed, for $(\Pi, \emptyset)$, rule r_1 must be ordered before rule r_2 in D- and W-preference, since the derivation of the positive body of r_2 depends on

r_1. Thus, the preference relation $r_2 < r_1$ is compatible with this ordering and cannot inhibit any $<^D$- and $<^W$- preferred answer sets. Furthermore, $r_2 < r_1$ has no effect on $<^B$- preferred answer sets, since the B-semantics fully decouples preference handling from the order induced by consecutive rule application.

But, perhaps surprisingly, these programs are not strongly order equivalent. Consider the following ordered logic program:

$$(\Pi', <') = \left\{ \begin{array}{ll} r_1 : a \leftarrow & \\ r_2 : b \leftarrow a & \\ r_3 : z \leftarrow & r_3 <' r_2 \\ r_4 : y \leftarrow not\ z & r_1 <' r_4 \\ r_5 : a \leftarrow x & \\ r_6 : x \leftarrow & \end{array} \right\}$$

Note that $\Pi \subset \Pi'$. The only answer set of Π' is $\{a, b, z, x\}$.

In B-semantics, we may order the rules of Π' in any way, provided that r_3 is ordered before r_4 and that $<'$ is respected. This is obviously feasible, one ordering would be $\langle r_2, r_3, r_4, r_1, r_5, r_6\rangle$. Note that r_5 and r_6 can be ordered arbitrarily for this semantics. Also note that r_1 does not have to be ordered before r_2 in the B–semantics. Hence, $AS^B((\Pi', <')) = \{\{a, b, z, x\}\}$.

Considering W-semantics, any ordering of the rules must respect $<'$ and put r_3 before r_4, put either r_1 or r_5 before r_2, and put r_1 or r_6 before r_5. It turns out that r_1 cannot be put before r_2, as r_2 must precede r_3 due to $<'$ and r_3 must occur before r_4, so r_1 would be listed before r_4, which is incompatible with $<'$. Therefore r_5 must be ordered before r_2, while r_1 must be ordered behind r_2, and hence also r_5. So r_6 must be put before r_5 and so the only admissible ordering is $\langle r_6, r_5, r_2, r_3, r_4, r_1\rangle$. Using similar reasoning, we obtain that the same ordering is the only admissible ordering for the D-semantics. Therefore $AS^D((\Pi', <')) = AS^W((\Pi', <')) = \{\{a, b, z, x\}\}$. Note that rules r_5 and r_6 are essential to guarantee this preferred answer set for the W- and D-semantics.

Now we observe that $\Pi \cup \Pi' = \Pi'$, so $(\Pi, \emptyset) \cup (\Pi', <') = (\Pi', <')$, and hence $\{a, b, z, x\}$ is the preferred answer set for $(\Pi, \emptyset) \cup (\Pi', <')$ in all three semantics.

For $(\Pi, <) \cup (\Pi', <') = (\Pi', <^*)$ the situation is different. We first note that $<^*$ contains $r_3 <^* r_2$, $r_2 <^* r_1$, $r_1 <^* r_4$, and therefore also $r_3 <^* r_1$, $r_2 <^* r_4$, and, importantly, $r_3 <^* r_4$. So, in order to be compatible with $<^*$, each ordering must put r_4 before r_3. But all of the orderings discussed above require also that r_3 is put before r_4. Therefore no compatible ordering exists for any of the three semantics, and hence $AS^\sigma((\Pi', <^*)) = \emptyset$ for $\sigma \in \{D, W, B\}$.

As a consequence, $(\Pi, \emptyset) \not\equiv_s^\sigma (\Pi, <)$ for $\sigma \in \{D, W, B\}$.

Note that it is possible to construct an ordered logic program like $(\Pi', <')$ in this example for any pair of ordered logic programs for which the preference relations do not coincide. Even if the ordered logic programs do not admit preferred answer sets, it is possible to add suitable rules such that one of the composed programs has a preferred answer set while the other has a transitively induced preference which is in conflict with a pair of rules like r_3 and r_4 in Example 22.

Theorem 5.1.1 *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered programs and $\sigma \in \{D, W, B\}$.*
If $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ then $<_1 = <_2$.

This theorem also implies that strongly order equivalent programs must coincide on their rules involved in the preference relation. We say that a rule r is involved in a preference relation $<$ if there exists an r' such that $r < r'$ or $r' < r$ holds.

Definition 5.1.6 *Let $(\Pi, <)$ be an ordered logic program.*
We define $PR((\Pi, <)) = \{r, r' \in \Pi \mid r < r'\}$.

We obtain the following corollary.

Corollary 5.1.2 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered programs and* $\sigma \in \{D, W, B\}$.
If $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ *then* $PR((\Pi_1, <_1)) = PR((\Pi_2, <_2))$.

As a special case of Theorem 5.1.1, we obtain that $(\Pi, \emptyset) \not\equiv_s^\sigma (\Pi, <)$ holds for any program Π, any non-empty preference relation $<$ on Π, and any $\sigma \in \{D, W, B\}$. Moreover, for every ordered logic program $(\Pi, <), < \neq \emptyset$, there exists no strongly order equivalent program $(\Pi', \emptyset)$.

Corollary 5.1.3 *Let* $(\Pi, <)$ *be an ordered logic program such that* $< \neq \emptyset$ *and* $\sigma \in \{D, W, B\}$.
Then, there exists no logic program Π' *such that* $(\Pi, <) \equiv_s^\sigma (\Pi', \emptyset)$.

Interestingly, this tells us that no transformation from an ordered logic program with a non-empty preference relation to a logic program exists such that the transformed program is strongly order equivalent to the original one. Note that such a transformation may not introduce any new symbols, as then the (preferred) answer sets are in general no longer comparable. As discussed in Section 5.1.5, the `plp` system does introduce new symbols, and in this way the preferences can be "compiled away." Hence, the compilation method from the `plp` system is not feasible under strong order equivalence. Furthermore, it is not feasible under order equivalence, since the transformation introduces new symbols, which appear in the resulting answer sets.

5.1.2.2 Traditional Strong Equivalence

Another good candidate for a necessary condition for strong order equivalence would be strong equivalence of the logic programs (without considering the preference relation). This would be very natural since the definition of strong order equivalence "covers" strong equivalence of the underlying logic programs.

However, investigating a bit more, it is not immediately evident that this property holds. Even if the preferred answer sets of the two programs coincide in every extension, one could believe that it could happen that the preference relation of one program inhibits an answer set, which is not an answer set of the other program.

But, as suggested in the following examples, in such a situation, one can always find another extension, for which the preferred answer sets do not coincide.

Example 23 *Consider the ordered logic programs*

$$(\Pi_1, <) = \left\{ \begin{array}{l} r_1 : a \leftarrow not\ b \\ r_2 : b \leftarrow not\ a \\ r_2 < r_1 \end{array} \right\} \quad \textit{and} \quad (\Pi_2, <) = \left\{ \begin{array}{l} r_1 : a \leftarrow not\ b \\ r_2 : b \leftarrow not\ a \\ r_3 : a \leftarrow not\ a \\ r_2 < r_1 \end{array} \right\}$$

They coincide on their preference relations, but the underlying logic programs are not equivalent, hence also not strongly equivalent.

We observe that $AS(\Pi_1) = \{\{a\}, \{b\}\}$, *while* $AS(\Pi_2) = \{\{a\}\}$. *However, it holds that* $AS^\sigma((\Pi_1, <)) = AS^\sigma((\Pi_2, <)) = \{\{a\}\}$ *for* $\sigma \in \{D, W, B\}$, *as for the answer set* $\{b\}$ *of* $(\Pi_1, <)$, *any enumeration would have to order* r_1 *before* r_2 *because of the preference relation, while it would also have to order* r_2 *before* r_1 *in all three semantics to avoid blockedness by lower ranked rules. Hence, the preference relation* $<$ *inhibits* $\{b\}$ *as preferred answer set in* $(\Pi_1, <_1)$, *whereas* $(\Pi_2, <_2)$ *misses* $\{b\}$ *as answer set.*

However, we can construct program $\Pi' = \{b \leftarrow\}$, *such that* $AS(\Pi_1 \cup \Pi') = AS(\Pi_1)$ *and* $AS(\Pi_2 \cup \Pi') = \emptyset$. *We now obtain* $AS^\sigma((\Pi_1 \cup \{b \leftarrow\}, <)) = \{\{b\}\}$ *but* $AS^\sigma((\Pi_2 \cup \{b \leftarrow\}, <)) = \emptyset$ *for* $\sigma \in \{D, W, B\}$, *providing a counterexample to* $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$.

The next example shows a similar situation, but this time we try to formulate the counterexample in a somewhat more systematic way.

Example 24 *For instance, consider the ordered logic programs:*

$$(\Pi_1, \emptyset) = \left\{ \; r_1 : a \leftarrow \; \right\} \quad \textit{and} \quad (\Pi_2, <_2) = \left\{ \begin{array}{l} r_2 : a \leftarrow not\ b \\ r_3 : b \leftarrow not\ a \\ r_3 <_2 r_2 \end{array} \right\}$$

We get that $AS(\Pi_1) = \{\{a\}\}$ *and* $AS(\Pi_2) = \{\{a\}, \{b\}\}$*, hence* $\Pi_1 \not\equiv_s \Pi_2$*. But* $AS^\sigma((\Pi_1, \emptyset)) = \{a\}$ *and also* $AS^\sigma((\Pi_2, <_2)) = \{a\}$*. The answer set* $\{b\}$ *of* Π_2 *is inhibited by the preference relation, as* r_2 *must be enumerated before* r_3 *because of the preference, yet all semantics also require that* r_3 *should be enumerated before* r_2 *in order to block* r_2*.*

Taking

$$(\Pi', \emptyset) = \left\{ \begin{array}{l} r_4 : x \leftarrow \\ r_5 : a \leftarrow x, not\ b \\ r_6 : b \leftarrow x, not\ a \end{array} \right\}$$

yields $AS(\Pi_1 \cup \Pi') = \{\{a, x\}\}$ *and* $AS(\Pi_2 \cup \Pi') = \{\{a, x\}, \{b, x\}\}$*. The only preferred answer set of* $(\Pi_1 \cup \Pi', \emptyset)$ *is again* $\{a, x\}$*, but both* $\{a, x\}$ *and* $\{b, x\}$ *are preferred answer sets of* $(\Pi_2 \cup \Pi', <_2)$*. Indeed* $\langle r_4, r_6, r_5, r_2, r_3\rangle$ *is a valid enumeration in all semantics. The compensating program* $(\Pi', \emptyset)$ *"repairs" the non-preferred answer set* $\{b\}$ *in* $(\Pi_2, <_2)$*, whereas it does not introduce* $\{b\}$ *as answer set of* $\Pi_1 \cup \Pi'$*.*

It turns out that one can always construct a program as in the previous example for any answer set which is inhibited by the preference relation. We therefore arrive at the following Theorem.

Theorem 5.1.4 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered programs and* $\sigma \in \{D, W, B\}$*.*
If $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ *then* $\Pi_1 \equiv_s \Pi_2$*.*

We now already have found two rather strict necessary conditions. As we will see next, there is yet a third one.

5.1.2.3 Generating and Contributing Rules

Unfortunately, the conditions of Theorems 5.1.1 and 5.1.4 are not yet sufficient to characterize strong order equivalence. There are programs, which are not strongly order equivalent, while they coincide on the preference relation and their logic programs are strongly equivalent, as the following example shows.

Example 25 *Consider the following ordered logic programs:*

$$(\Pi_1, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_3 : b \leftarrow \end{array} \right\} \quad \textit{and} \quad (\Pi_2, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow a \end{array} \right\}$$

We observe that $\Pi_1 \equiv_s \Pi_2$ *holds and that both ordered programs coincide on their (empty) preference relation.*

But, taking the ordered logic program

$$(\Pi^*, <^*) = \left\{ \begin{array}{l} r_2 : b \leftarrow a \\ r_4 : y \leftarrow not\ b \\ r_2 <^* r_4 \end{array} \right\}$$

as admissible extension of $(\Pi_1, \emptyset)$ *and* $(\Pi_2, \emptyset)$ *yields* $AS^\sigma((\Pi_1 \cup \Pi^*, <^*)) = \{\{a, b\}\}$ *but* $AS^\sigma((\Pi_2 \cup \Pi^*, <^*)) = \emptyset$ *for* $\sigma \in \{D, W, B\}$*.*

The problem is that r_3 *does not occur in* Π_2*, and while* r_2 *can take the "role" of* r_3 *for standard answer sets, it cannot do so for preferred answer sets, when the preference relation is chosen in a conflicting way.*

A more general observation is that the programs are not strongly order equivalent because they do not coincide on their set of generating rules. $(\Pi_1, \emptyset)$ *contains* r_3 *and* $(\Pi_2, \emptyset)$ *rule* r_2 *to derive the atom* b*. Adding* r_4 *and the preference that* r_4 *is preferred to* r_2 *entails that* $(\Pi_2 \cup \Pi^*, <^*)$ *has no preferred answer set since* r_4 *is blocked by the lower preferred rule* r_2*, while* $(\Pi_1 \cup \Pi^*, <^*)$ *has a preferred answer set since* r_3 *allows to block* r_4 *in an alternative way.*

Since the considered preference semantics are rule-based, this allows us to suppose that strong order equivalence enforces that the ordered logic programs must coincide on all rules which can become applicable.

Accordingly, we say that a rule $r \in \Pi$ *contributes* to an answer set X, if there exists an extension Π^* and an answer set $X \in AS(\Pi \cup \Pi^*)$ such that $r \in R_{\Pi \cup \Pi^*}(X)$.

Definition 5.1.7 *Let* Π *be a logic program. We define*

$$Cont(\Pi) = \{r \in \Pi \mid \textit{ there exists an } \Pi' \textit{ and an } X \in AS(\Pi \cup \Pi') \textit{ s.t. } r \in R_{\Pi \cup \Pi'}(X)\}$$

Rules which are not contributing to answer sets, are never applicable, e.g. where $body^+(r) \cap body^-(r) \neq \emptyset$ or $head(r) \in body^-(r)$. We want to stress that with $Cont(\Pi)$ we denote all rules which can become a generating rule *in some extension* of Π.

Example 26 *For the program*

$$\Pi = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow a \end{array} \right\}$$

we obtain $r_1, r_2 \in Cont(\Pi)$ *since* $r_1, r_2 \in R_\Pi(X)$ *for answer set* $X = \{a, b\}$ *of* Π.

For the program

$$\Pi = \{r_2 : b \leftarrow a\}$$

we have $r_2 \notin R_\Pi(X)$ *for answer set* $X = \emptyset$ *of* Π*. But, for the extension* $\Pi' = \{r_1 : a \leftarrow\}$ *of* Π*, we obtain that* r_2 *becomes a generating rule for the program* $\Pi \cup \Pi'$ *and answer set* $X = \{a, b\}$*. Hence, we have* $r_2 \in Cont(\Pi)$.

For B-preferences, we can show that two strongly order equivalent programs must coincide on their rules contributing to answer sets.

Theorem 5.1.5 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered logic programs.*
If $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$*, then* $Cont(\Pi_1) = Cont(\Pi_2)$.

Interestingly, for D– and W–preferences, these conditions have to be weakened.

Theorem 5.1.6 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered logic programs and* $\sigma \in \{D, W\}$.
If $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$*, then* $Cont(\Pi_1) \setminus \{r \in \Pi_1 \mid head(r) \in body^+(r)\} = Cont(\Pi_2) \setminus \{r \in \Pi_2 \mid head(r) \in body^+(r)\}$

Since D- and W- preferences couple rule application with preference handling, answer sets can be inhibited by preferences because applied rules are not enumerable in an order preserving way (see Condition 2 in Definition 2.4.1 and 2.4.2). But one can always find an extension such that those answer sets are regenerated. For applied "loop" rules, i.e. $head(r) \in body^+(r)$, another rule must always exist which derives the head within an answer set in an order preserving way. Hence, such loop rules are redundant w.r.t. D- and W-preferences as long as they are not involved in the preference relations (see Section 5.1.4). In contrast, B- preferences decouple preference handling from rule application and hence, one can always find an extension, where those loop rules regenerate a non-preferred answer set to a preferred one in one program but not in the other one.

Example 27 *The ordered logic programs*

$$(\Pi_1, \emptyset) = \left\{ \; r_1 : a \leftarrow \; \right\} \quad \textit{and} \quad (\Pi_2, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : a \leftarrow a \end{array} \right\}$$

are strongly equivalent on their logic programs, coincide on their preference relation, but they are not strongly $<^B$- equivalent. Taking

$$(\Pi, <) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_3 : y \leftarrow not\ a \\ r_1 < r_3 \end{array} \right\}$$

yields $AS^B((\Pi_1 \cup \Pi, <)) = \emptyset$ but $AS^B((\Pi_2 \cup \Pi, <)) = \{a\}$. Here, rule r_2 can be used to block r_3. For D- and W-preferences, however, r_2 has to be ordered after r_1, hence it cannot be used to block r_3.

Indeed, for any admissible extension $(\Pi', <')$, for which $AS^\sigma((\Pi_1 \cup \Pi', <')) = \emptyset$, even if $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi') \neq \emptyset$, any enumeration of $\Pi_2 \cup \Pi'$ must put r_2 behind r_1 or another generating rule $r \in \Pi'$ with $head(r) = a$, so r_2 cannot "fix" an answer set inhibited in $(\Pi_1 \cup \Pi', <')$. If instead $AS^\sigma((\Pi_2 \cup \Pi', <')) = \emptyset$ holds, it is clear that also $AS^\sigma((\Pi_1 \cup \Pi', <')) = \emptyset$ must hold. Moreover any $<^\sigma$-preferred answer set of $(\Pi_1 \cup \Pi', <')$ will also be a $<^\sigma$-preferred answer set of $(\Pi_2 \cup \Pi', <')$ and vice versa. Therefore $(\Pi_1, \emptyset) \equiv_s^\sigma (\Pi_2, \emptyset)$ holds, this reasoning being valid for $\sigma = \{D, W\}$.

Beside "loop" rules, one could wonder whether also non-trivial loops are redundant in the preference semantics. Indeed, they are not, as shown by the following example.

Example 28 *The ordered logic programs*

$$(\Pi_1, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow a \end{array} \right\} \quad \textit{and} \quad (\Pi_2, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow a \\ r_3 : a \leftarrow b \end{array} \right\}$$

are not strongly $<^B$- equivalent. Taking

$$(\Pi, <) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_4 : y \leftarrow not\ a \\ r_1 < r_4 \end{array} \right\}$$

yields $AS^B((\Pi_1 \cup \Pi, <)) = \emptyset$ but $AS^B((\Pi_2 \cup \Pi, <)) = \{a, b\}$. Here, rule r_3 can be used to block r_4, a possible enumeration being $\langle r_3, r_2, r_4, r_1 \rangle$.

Different from Example 27, $(\Pi_1, \emptyset)$ and $(\Pi_2, \emptyset)$ are neither strongly $<^D$, nor strongly $<^W$- equivalent. For this, consider

$$(\Pi', <) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_4 : y \leftarrow not\ a \\ r_5 : b \leftarrow \\ r_1 < r_4 \end{array} \right\}$$

as $AS^\sigma((\Pi_1 \cup \Pi', <)) = \emptyset$ (since no enumeration can list rule r_1 before r_4, and so no earlier rule can block r_4), but $AS^\sigma((\Pi_2 \cup \Pi', <)) = \{a, b\}$ for $\sigma = \{D, W\}$, as r_3 can be ordered before r_4, thus blocking r_4, provided that r_5 is enumerated before r_3, which is feasible. An admissible enumeration would be $\langle r_5, r_3, r_2, r_4, r_1 \rangle$.

We note that for two logic programs Π_1 and Π_2, $\Pi_1 \equiv_s \Pi_2$ does not imply $Cont(\Pi_1) = Cont(\Pi_2)$ (as seen in Example 25), and also $Cont(\Pi_1) = Cont(\Pi_2)$ does not imply $\Pi_1 \equiv_s \Pi_2$, as the following example shows.

Example 29 *Considering the logic programs*

$$\Pi_1 = \{\ r_1 : a \leftarrow\ \} \quad \textit{and} \quad \Pi_2 = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow not\ b \end{array} \right\}$$

we observe that $Cont(\Pi_1) = \{r_1\}$*, but also* $Cont(\Pi_2) = \{r_1\}$*, as* r_2 *is not a generating rule in any extension* $\Pi_2 \cup \Pi$*: If there is an answer set* $X \in AS(\Pi_2 \cup \Pi)$ *and if* $b \in X$*, then* $r_2 \notin R_{\Pi_2 \cup \Pi}(X)$*, while if* $b \notin X$*, then* X *would not be an answer set, as each model of* $(\Pi_2 \cup \Pi)^X$ *must contain* b*, otherwise* $r_2 \in (\Pi_2 \cup \Pi)^X$ *is not satisfied. So* $Cont(\Pi_1) = Cont(\Pi_2)$ *holds.*

Since $AS(\Pi_1) = \{\{a\}\}$ *and* $AS(\Pi_2) = \emptyset$*,* $\Pi_1 \equiv_s \Pi_2$ *obviously does not hold.*

Theorems 5.1.5 and 5.1.6 do therefore not subsume Theorem 5.1.4, nor vice versa.

5.1.2.4 Characterization of Strong Order Equivalence

It turns out that the conditions in Theorems 5.1.1, 5.1.5, 5.1.6, and 5.1.4 are, taken together, sufficient for strong order equivalence.

Example 30 *For example, for*

$$(\Pi_1, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow not\ a \end{array} \right\} \quad \textit{and} \quad (\Pi_2, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_3 : c \leftarrow not\ a \end{array} \right\}$$

the preference relations are equal, Π_1 *and* Π_2 *are strongly equivalent, and* $Cont(\Pi_1) = Cont(\Pi_2) = \{r_1\}$*.*

We can now observe that for any admissible extension $(\Pi, <)$*, if* $(\Pi_1 \cup \Pi, <)$ *admits a* $<^\sigma$*-preferred answer set, then it is also a* $<^\sigma$*-preferred answer set of* $(\Pi_2 \cup \Pi, <)$ *and vice versa. The reason is that the answer sets of the underlying programs must be equal, as they are strongly equivalent, and that any compatible enumeration of* $(\Pi_1 \cup \Pi, <)$ *needs to put a generating rule* r *with* $head(r) = a$ *before* r_2*. This rule is either* r_1 *or a rule in* Π*, and is therefore also present and generating in* $\Pi_2 \cup \Pi$*. Indeed, we can just swap rule* r_2 *with rule* r_3 *and arrive at a valid enumeration for* $(\Pi_2 \cup \Pi, <)$*, since the preference relation is the same. Using a symmetric argument, any admissible enumeration of* $(\Pi_2 \cup \Pi, <)$ *can be modified to yield an admissible enumeration for* $(\Pi_1 \cup \Pi, <)$ *in any of the three semantics. The two ordered logic programs are therefore strongly order equivalent in all three semantics.*

The only difference among the semantics concerning the necessary conditions was for generating rules. Generating "loop rules" need not be equal for strong $<^D$- and $<^W$- order equivalence.

Example 31 *For the ordered logic programs*

$$(\Pi_1, \emptyset) = \{r_1 : a \leftarrow\} \quad \textit{and} \quad (\Pi_2, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : a \leftarrow a \end{array} \right\}$$

the preference relations are equal, Π_1 *and* Π_2 *are strongly equivalent, and* $Cont(\Pi_1) = Cont(\Pi_2) = \{r_1, r_2\}$*.*

We can now observe that for any admissible extension $(\Pi, <)$*, if* $(\Pi_1 \cup \Pi, <)$ *admits a* $<^\sigma$*-preferred answer set for* $\sigma \in \{D, W\}$*, we can modify the enumeration of rules of* $\Pi_1 \cup \Pi$ *to an enumeration of the rules of* $\Pi_2 \cup \Pi$ *by inserting* r_2 *at the very end. On the other hand, if* $(\Pi_2 \cup \Pi, <)$ *admits a* $<^\sigma$*-preferred answer set, then any enumeration must list a generating rule* r *with* $head(r) = a$ *before* r_2*. So for any rule* r' *occurring after* r_2*, which needs a preceding rule* r'' *with* $head(r'') = a$*, both* r_2 *and* r *can serve this purpose. We can therefore remove* r_2 *from the enumeration safely, arriving at an admissible enumeration for* $(\Pi_1 \cup \Pi, <)$*. These programs are therefore strongly* $<^D$*- and* $<^W$*- equivalent.*

The picture is different for $<^B$ *preferred answer sets. While any* $<^B$ *preferred answer set for* $(\Pi_1 \cup \Pi, <)$ *is also a* $<^B$ *preferred answer set for* $(\Pi_2 \cup \Pi, <)$ *(by inserting* r_2 *at an arbitrary position of any*

enumeration for $\Pi_1 \cup \Pi$), a $<^B$ preferred answer set for $(\Pi_2 \cup \Pi, <)$ is not necessarily one of $(\Pi_1 \cup \Pi, <)$: Different from above, we cannot guarantee that a rule r with $head(r) = a$ precedes r_2 in the enumeration of $\Pi_2 \cup \Pi$ (since Definition 2.4.3 lacks an equivalent for Condition 2 of Definitions 2.4.1 and 2.4.2). So some rule r' after r_2 might need a preceding generating rule r'' with $head(r'') = a$. However, all generating rules apart from r_2 might be constrained to occur after r'', and so it might not be feasible to delete r_2 from the enumeration. An example for this situation can be found in Example 27. The programs are therefore not strongly $<^B$ - equivalent.

The techniques sketched in this example can indeed be generalized, yielding proofs to the following results on sufficiency of the three conditions.

Theorem 5.1.7 *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered logic programs.*
If $\Pi_1 \equiv_s \Pi_2$, $<_1 = <_2$, and $Cont(\Pi_1) = Cont(\Pi_2)$, then $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$.

Theorem 5.1.8 *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered logic programs and $\sigma \in \{D, W\}$.*
If $\Pi_1 \equiv_s \Pi_2$, $<_1 = <_2$, and $Cont(\Pi_1) \setminus \{r \in \Pi_1 \mid head(r) \in body^+(r) = Cont(\Pi_2) \setminus \{r \in \Pi_2 \mid head(r) \in body^+(r)\}$, then $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$.

Taking together the necessary and sufficient conditions, we can formalize the following characterization of strong order equivalence.

Corollary 5.1.9 *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered logic programs.*
Then, $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$ iff

- $\Pi_1 \equiv_s \Pi_2$,
- $<_1 = <_2$, *and*
- $Cont(\Pi_1) = Cont(\Pi_2)$.

Corollary 5.1.10 *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered programs and $\sigma \in \{D, W\}$.*
Then, $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$, iff

- $\Pi_1 \equiv_s \Pi_2$,
- $<_1 = <_2$, *and*
- $Cont(\Pi_1) \setminus \{r \in \Pi_1 \mid head(r) \in body^+(r)\} = Cont(\Pi_2) \setminus \{r \in \Pi_2 \mid head(r) \in body^+(r)\}$.

5.1.2.5 Strong Order Equivalence Structures

In analogy to SE-models, these results allow us to associate to each ordered logic program a structure comprised of its SE-models, the preference relation, and the set of contributing rules (without loop rules for D- and W-semantics). If for a pair of programs these structures are equal, they are strongly equivalent under the respective semantics. We refrain from calling these structures models, as they contain non-standard elements like preferences and rules.

Definition 5.1.8 *An SOE^B-structure of an ordered logic program $(\Pi, <)$ is the triple*

$$\langle SE(\Pi), <, Cont(\Pi) \rangle,$$

denoted by $SOE^B((\Pi, <))$.
An SOE^σ-structure (for $\sigma \in \{D, W\}$) of an ordered logic program $(\Pi, <)$ is the triple

$$\langle SE(\Pi), <, Cont(\Pi) \setminus \{r \in \Pi \mid head(r) \in body^+(r)\} \rangle,$$

denoted by $SOE^\sigma((\Pi, <))$.

The following result follows from Corollaries 5.1.9 and 5.1.10.

Theorem 5.1.11 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered programs and* $\sigma \in \{D, W, B\}$. *Then,* $SOE^\sigma((\Pi_1, <_1)) = SOE^\sigma((\Pi_2, <_2))$ *holds iff* $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$.

We have thus fully characterized strong order equivalence under the three preference semantics, and we have also given a structure associated to each ordered logic program, such that two programs are strongly order equivalent if and only if their structures coincide for the respective semantics.

5.1.2.6 Strong Order Equivalence under Different Semantics

Regarding preferred answer sets, recall that the different strategies for preference handling yield an increasing number of preferred answer sets, in particular

$$AS^D((\Pi, <)) \subseteq AS^W((\Pi, <)) \subseteq AS^B((\Pi, <)) \subseteq AS(\Pi)$$

as shown in [177].

Interestingly, the above theorems show that there is no difference in the behavior of D– and W–preferences when considering strong order equivalence. In contrast, fewer programs are strongly order equivalent in the B–semantics, since preference handling is fully decoupled from the order induced by consecutive rule application.

Theorem 5.1.12 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered logic programs. Then*

(a) *if* $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$ *then* $(\Pi_1, <_1) \equiv_s^W (\Pi_2, <_2)$,

(b) $(\Pi_1, <_1) \equiv_s^W (\Pi_2, <_2)$ *iff* $(\Pi_1, <_1) \equiv_s^D (\Pi_2, <_2)$.

Considering answer sets, each $<^D$-preferred answer set is $<^W$-preferred and each $<^W$-preferred is $<^B$-preferred. Interestingly, regarding strong order equivalence we obtain that $\equiv_s^D$ iff $\equiv_s^W$, so the differences between the D- and W- semantics can be compensated by extending the program in a suitable way. Moreover, since $\equiv_s^B$ implies $\equiv_s^W$ and $\equiv_s^B$ implies $\equiv_s^D$, the B-semantics imposes a strictly stronger criterion for programs to be strongly order equivalent. The reason for this is that B-preference fully decouples preference handling from the order induced by consecutive rule application, and in addition, these differences cannot be compensated by adding suitable extensions.

5.1.3 Complexity Results

In this section, we discuss the computational complexity of deciding strong order equivalence and related problems. We first start by examining the complexity of standard equivalence under $<^D$-, $<^W$-, and $<^B$-preferred answer sets. It turns out that this problem is not more difficult than deciding whether two logic programs have the same answer sets.

The important point is that checking whether an answer set is preferred (and therefore also whether it is not preferred) can be done in polynomial time. This can be achieved by a modified topological sorting, as described in [34].

Lemma 5.1.13 *Given an ordered logic program* $(\Pi, <)$ *and an interpretation* I*, checking whether* $I \in AS^\sigma((\Pi, <))$ *is feasible in polynomial time for all* $\sigma \in \{D, W, B\}$.

It is therefore possible to guess an interpretation, and check whether it is an answer set and preferred for one program and not an answer set or not preferred for the other or vice versa, all in polynomial time.

Theorem 5.1.14 *Given two ordered programs* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$*, deciding whether* $(\Pi_1, <_1) \equiv^\sigma (\Pi_2, <_2)$ *is* co-NP*-complete for* $\sigma \in \{D, W, B\}$.

Let us now turn to strong order equivalence. A key notion for deciding strong order equivalence is the set of contributing rules, following from Corollaries 5.1.9 and 5.1.10. We first note that in order to check whether a rule of a program is contributing to an answer set of an extension, it is not needed to check all arbitrary program extensions, but just particular ones:

Lemma 5.1.15 *Let Π be a logic program and $r \in \Pi$.*
If there exists a program Π' and an $X \in AS(\Pi \cup \Pi')$ such that $r \in R_{\Pi \cup \Pi'}(X)$ then there exists a program $\Pi^* \in facts(Atm(\Pi))$ [4] and an $X^* \in AS(\Pi \cup \Pi^*)$ such that $r \in R_{\Pi \cup \Pi^*}(X^*)$.

Leveraging this lemma, we can show that checking whether a rule is contributing is NP-complete.

Theorem 5.1.16 *Let $(\Pi, <)$ be an ordered program.*
Given a rule r, deciding whether $r \in Cont(\Pi)$ is NP*-complete.*

Next, we show that two strongly equivalent programs differ in their contributing rules if there is a rule, which occurs in exactly one of the two programs, which is contributing.

Lemma 5.1.17 *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered programs and $\Pi_1 \equiv_s \Pi_2$.*
If $Cont(\Pi_1) \neq Cont(\Pi_2)$, then there exists an r such that either $r \in \Pi_1$, $r \notin \Pi_2$ and a Π' exists such that $X \in AS(\Pi_1 \cup \Pi')$ and $r \in R_{\Pi_1 \cup \Pi'}(X)$, or $r \in \Pi_2$, $r \notin \Pi_1$ and a Π' exists such that $X \in AS(\Pi_2 \cup \Pi')$ and $r \in R_{\Pi_2 \cup \Pi'}(X)$.

We can use this result for showing that deciding whether two strongly equivalent programs differ on their contributing rules is co-NP-complete.

Theorem 5.1.18 *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered programs and $\Pi_1 \equiv_s \Pi_2$.*
Deciding whether $Cont(\Pi_1) = Cont(\Pi_2)$ is co-NP*-complete.*

Using a result from [134], Corollaries 5.1.9 and 5.1.10, and Theorem 5.1.18, we can show that checking whether two ordered logic programs are strongly order equivalent is co-NP-complete.

Theorem 5.1.19 *Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered programs and $\sigma \in \{D, W, B\}$.*
Then, deciding whether $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ is co-NP*-complete.*

We have thus shown that strong order equivalence testing is not more complex than strong equivalence testing.

5.1.4 Program Simplifications

Corollaries 5.1.10 and 5.1.9 show that two ordered logic programs which are strongly order equivalent, differ only on rules which are not involved in the preference relation and which are not applicable in any extension. Additionally, for D- and W- semantics, strongly order equivalent logic programs may differ in loop rules, in which the head is contained in the positive body. These results allow us to formalize the following results concerning transformations on ordered logic programs.

Corollary 5.1.20 *Let $(\Pi, <)$ be an ordered logic program and $r \in \Pi$ such that $r \notin Cont(\Pi), r \notin PR((\Pi, <))$ and $\Pi \equiv_s \Pi \setminus \{r\}$. Then, $(\Pi, <) \equiv_s^\sigma (\Pi \setminus \{r\}, <)$ holds for $\sigma \in \{D, W, B\}$.*

This corollary expresses that a rule, which is never applicable and not involved in $<$, is redundant regarding strong order equivalence as long as the simplification is feasible under strong equivalence. The condition $\Pi_1 \equiv_s \Pi_2$ is necessary. To see this, consider the following example.

[4] For a set A, we denote with $facts(A)$ the set of rules $facts(A) = \{a \leftarrow | a \in A\}$.

Example 32 *Let us examine the program*

$$(\Pi_1, \emptyset) = \{c \leftarrow not\ c\}.$$

Clearly, we have $\Pi_1 \not\equiv_s \emptyset$. *Rule* $c \leftarrow not\ c$ *also never contributes to an answer set. Waiving the condition for strong equivalence of the underling logic programs in Corollary 5.1.20 would lead to wrong simplifications in general. For example,*

$$(\Pi_2, \emptyset) = \left\{ \begin{array}{l} a \leftarrow \\ c \leftarrow not\ c \end{array} \right\} \quad \textit{and} \quad (\Pi_3, \emptyset) = \left\{ \ a \leftarrow \ \right\}$$

are not even equivalent.

The next corollary focuses on loop rules.

Corollary 5.1.21 *Let* $(\Pi, <)$ *be an ordered logic program. Furthermore, let* $r \in \Pi$ *s.t.* $head(r) \in body^+(r)$ *and* $r \notin PR((\Pi, <))$. *Then,* $(\Pi, <) \equiv_s^\sigma (\Pi \setminus \{r\}, <)$ *holds for* $\sigma \in \{D, W\}$.

In [80, 154], several transformations on logic programs from the literature ([25, 193]) have been examined whether they can be used for simplifying a program in a modular way (cf. on page 16, Section 2.5). For those modular transformations a program must be strongly equivalent to the transformed one. Considering ordered logic programs, we observe that some of them carry over to strong order equivalence unconditionally (provided that the rule to be deleted is not involved in the preference relation), while others need additional restrictions in order to be applicable. We now briefly discuss the respective simplification rules and provide an overview in Table 5.1.

Transformation *TAUT* expresses that for all logic programs Π and all $r \in \Pi$ where $head(r) \in body^+(r)$ we have $\Pi \equiv_s \Pi \setminus \{r\}$. Corollary 5.1.21 implies that the analogue for ordered logic programs only holds for the D- and W- semantics as long as r is not involved in $<$.

Transformation *RED*$^-$ expresses that for a logic program Π and for rules r_1, r_2, where $head(r_2) \in body^-(r_1)$ and $body(r_2) = \emptyset$, we have $\Pi \equiv_s \Pi \setminus \{r_1\}$. Since r_2 is always a generating rule, regardless which rules we add to Π, r_1 can never contribute to an answer set. Hence, *RED*$^-$ is applicable for ordered logic programs unless r_1 is involved in $<$.

Transformation *CONTRA* states that for $r \in \Pi$, where $body^+(r) \cap body^-(r) \neq \emptyset$ we have that $\Pi \equiv_s \Pi \setminus \{r\}$. Since such a rule is never applicable, this result can be carried over to ordered logic programs whenever r is not involved in $<$.

Transformation *NONMIN* states that for $r_1, r_2 \in \Pi$, where $head(r_2) = head(r_1)$ and $body(r_2) \subseteq body(r_1)$, we have that $\Pi \equiv_s \Pi \setminus \{r_1\}$ holds. Since there is no information whether r_1 can become applicable or not, this transformation can only be made on ordered logic programs whenever r_1 is not involved in $<$ and never becomes applicable.

Example 33 *For example, consider the program:*

$$(\Pi_1, \emptyset) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow \\ r_3 : c \leftarrow a \\ r_4 : c \leftarrow a, b \end{array} \right\}$$

Here it holds that $head(r_3) = head(r_4)$ *and* $body(r_3) \subseteq body(r_4)$, *hence,* $\Pi_1 \equiv_s \Pi_1 \setminus \{r_4\}$. *Since* r_4 *is applicable, the simplified program is, however, not strongly order equivalent to the original one. As a counterexample, consider*

$$(\Pi^*, <^*) = \left\{ \begin{array}{l} r_3 : c \leftarrow a \\ r_5 : y \leftarrow not\ c \\ r_3 <^* r_5 \end{array} \right\}$$

This program yields $\{a, b, c\} \in AS^\sigma((\Pi_1 \cup \Pi^*, <^*))$ *but* $AS^\sigma(((\Pi_1 \setminus \{r_4\}) \cup \Pi^*, <^*)) = \emptyset$, *for* $\sigma \in \{D, W, B\}$. *Thus,* $(\Pi_1, \emptyset) \not\equiv_s^\sigma (\Pi_1 \setminus \{r_4\}, \emptyset)$.

	TAUT	*RED*$^-$	*CONTRA*	*NONMIN*	*S-IMP**	*WGPPE*
$\equiv_s^D$	yes	yes	yes	no	yes	no
$\equiv_s^W$	yes	yes	yes	no	yes	no
$\equiv_s^B$	no	yes	yes	no	yes	no

Table 5.1: Applicability of simplifications if the simplified rules do not occur in $<$.

Similar considerations apply for the transformation *S-IMP*, where for all $r, r' \in \Pi$ such that there exists an $A \subseteq body^-(r')$ such that $head(r) \in \{head(r')\} \cup A$, $body^-(r) \subseteq body^-(r') \setminus A$ and $body^+(r) \subseteq body^+(r')$ we have $\Pi \equiv_s \Pi \setminus \{r'\}$. This transformation has originally been developed for disjunctive logic programs, and boils down to two cases for programs without disjunctions, as in our setting. Whenever $head(r) = head(r')$, we get $body^-(r) \subseteq body^-(r') \setminus A$, hence $body^-(r) \subseteq body^-(r')$, and $body^+(r) \subseteq body^+(r')$, so in total $body(r) \subseteq body(r')$. So this case coincides with the transformation *NONMIN*, where r' is only redundant if it never becomes applicable and is not involved in the preference relation, as discussed above.

The other case, in which $head(r) \in A$, is more interesting. We will refer to it as *S-IMP**. We obtain that r' never contributes to an answer set, which can be seen as follows: Assume, there exists an extension Π^* of Π such that $X \in AS(\Pi \cup \Pi^*)$. If $r \in R_{\Pi\cup\Pi^*}(X)$, then $r' \notin R_{\Pi\cup\Pi^*}(X)$ by $head(r) \in A \subseteq body^-(r')$ and $head(r) \in X$. If $r \notin R_{\Pi\cup\Pi^*}(X)$, then we obtain two cases: (i) $body^+(r) \not\subseteq X$ or (ii) $body^-(r) \cap X \neq \emptyset$. In case (i) we observe $body^+(r') \not\subseteq X$ by $body^+(r) \subseteq body^+(r')$. In case (ii) we have $(body^-(r') \setminus A) \cap X \neq \emptyset$ by $body^-(r) \subseteq body^-(r') \setminus A$. Hence, $body^-(r') \cap X \neq \emptyset$. In both cases we obtain $r' \notin R_{\Pi\cup\Pi^*}(X)$. Hence, r' never contributes to an answer set and we can carry over this transformation to strong order equivalence as long as r' is not involved in the preference relation.

Example 34 *For example, taking*

$$\Pi = \left\{ \begin{array}{ll} r': & a \leftarrow b, not\ c \\ r: & c \leftarrow b \end{array} \right\}$$

and $A = \{c\}$ yields $\Pi \equiv_s \{r\}$. Whenever r is applicable in any extension, r' will be blocked. Whenever r is not applicable, neither is r'.

Transformation *WGPPE* states for a rule $r_1 \in \Pi$, where $a \in body^+(r_1)$, $G_a = \{r_2 \in \Pi \mid head(r_2) = a\}$ and $G_a \neq \emptyset$ that $\Pi \equiv_s \Pi \cup G'_a$ holds where $G'_a = \{head(r_1) \leftarrow (body^+(r_1) \setminus \{a\}) \cup not\ body^-(r_1) \cup body(r_2) \mid r_2 \in G_a\}$. Again, this transformation is transferable to strong order equivalence if the rules from G_a never become applicable and are not involved in the preference relation.

Example 35 *Considering*

$$\Pi = \left\{ \begin{array}{l} r_1 : b \leftarrow a, \\ r_2 : a \leftarrow \end{array} \right\}$$

we obtain that $\Pi \equiv_s \Pi \cup \{b \leftarrow\}$ holds. By taking

$$(\Pi^*, <^*) = \left\{ \begin{array}{l} r_1 : b \leftarrow a \\ r^y : y \leftarrow not\ b \\ r_1 <^* r^y \end{array} \right\}$$

we can see that $(\Pi, \emptyset) \not\equiv_s^\sigma (\Pi \cup \{b \leftarrow\}, \emptyset)$ since $(\Pi \cup \Pi^*, <^*)$ has no $<^\sigma$- preferred answer set while $(\Pi \cup \{b \leftarrow\} \cup \Pi^*, <^*)$ admits the $<^\sigma$- preferred answer set $\{a, b\}$, for $\sigma \in \{D, W, B\}$.

5.1.5 Conclusions

We have presented the notion of strong order equivalence for ordered logic programs with the underlying D-, W-, and B- preference semantics. We have provided an extensive analysis of this novel notion, and studied the relationship between the three preference semantics for it. We have also analyzed the conditions for certain simplification methods to guarantee strong order equivalence (hence applicability to modules).

We could show in Corollaries 5.1.9 and 5.1.10 that two programs are strongly W- and D-equivalent, if and only if the preference relation is equal, their standard ASP programs are strongly equivalent, and their "non-looping" generating rules are identical for all answer sets of any extension of them. For B-preference, also the "looping" generating rules must be identical in any answer set.

Based on these results, we are able to define SOE structures, which can be thought of as a generalization of SE models. Each ordered logic program has an associated set of SOE structures. If these SOE structures are equal for two programs, then these programs are strongly order equivalent. Different from SE models, these structures contain also information on the preference relation and on the set of contributing rules.

From Theorem 5.1.12 we obtain that exactly the same pairs of programs are strongly $<^D$- and $<^W$-equivalent, while not all of them are strongly $<^B$- equivalent. That is, while preferred answer sets in general differ between D- and W- preference, there is no difference between the two semantics when considering strong order equivalence. On the contrary, for B- semantics the differences to D- and W- preferences are strengthened under strong order equivalence, since it decouples preference handling from the order induced by consecutive rule application.

We have studied the computational complexity of the main decision problems, in particular deciding whether two programs are order equivalent and whether two programs are strongly order equivalent. It turned out that both tasks are co-NP-complete, and thus are neither harder nor simpler than equivalence and strong equivalence.

We have furthermore studied possibilities to simplify ordered programs. In Corollaries 5.1.20 and 5.1.21 we have given abstract conditions for simplifications, and have assessed under which circumstances simplification rules from standard ASP can be applied on ASP modules with preferences.

Corollary 5.1.3 shows that no transformation from an ordered program $(\Pi, <)$ into a logic program Π' exists such that $(\Pi, <)$ is strongly order equivalent to Π'. Also, Corollaries 5.1.9 and 5.1.10 describe rigorous conditions on ordered logic programs being strongly order equivalent under the considered preference semantics. Hence, we consider in Section 5.2 a weakened notion of strong order equivalence, where preference simplifications are possible.

5.2 n-Strong Order Equivalence

In Section 5.1, we have found out that strong order equivalent programs must be (i) strongly equivalent in the standard sense, (ii) have to coincide on their preference relations, and (iii) have to coincide on their set of rules contributing to answer sets. Since these conditions are very strict, we examine in this section a weaker notion of strong order equivalence. Whereas for strongly order equivalent programs we require that they have the same preferred answer sets no matter which *ordered* program we add, the weaker notion only stipulates that they have the same preferred answer sets no matter which *normal* logic program we add. Based on this idea, we define *n-strong order equivalence* and show several properties according to the characterizations for strong order equivalence. Furthermore, we present for the first time new program simplifications, particularly simplifications for preference relations, which cannot be applied under strong order equivalence. Additionally, we study computational complexity of n-strong order equivalence. It turns out that testing n-strong order equivalence is precisely as difficult as testing strong order equivalence or strong equivalence for logic programs without preferences.

This section is organized as follows: In Section 5.2.1 we define and characterize n-strong order equivalence and give some relations to strong order equivalence. In Section 5.2.2 we present new program

transformations under n-strong order equivalence, which simplify preference relations and the underlying logic program. In Section 5.2.3 we study computational complexity and draw conclusions in Section 5.2.4.

5.2.1 n-strong order equivalence

Strong order equivalence considers all admissible extensions of ordered programs by other ordered programs. For considering a weaker notion of order equivalence, we define n-strong order equivalence, where we consider extensions of ordered logic programs by normal logic programs. Since the union of an ordered program and a normal logic program always represents an ordered program, we do not have to restrict the set of program extensions to admissible ones as for strong order equivalence.

Definition 5.2.1 *(n-strong order equivalence) Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered logic programs and* $\sigma \in \{D, B, W\}$.

Then, $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *are* n-strongly $<^\sigma$-equivalent *iff for all normal programs* Π *it holds that* $(\Pi_1 \cup \Pi, <_1)$ *and* $(\Pi_2 \cup \Pi, <_2)$ *are* $<^\sigma$*- equivalent.*

As an abbreviation, we write $\equiv_n^\sigma$ whenever two ordered programs are n-strongly $<^\sigma$-equivalent.

In the following, we show some simple relationships between strong and n-strong order equivalence. Whenever the preference relation is empty, n-strong order equivalence corresponds exactly to strong equivalence of the underlying logic programs.

Lemma 5.2.1 *Let* Π_1 *and* Π_2 *be logic programs and* $\sigma \in \{D, W, B\}$.

Then, $(\Pi_1, \emptyset)$ *and* $(\Pi_2, \emptyset)$ *are n-strongly* $<^\sigma$*-equivalent iff* Π_1 *and* Π_2 *are strongly equivalent.*

Also, strong order equivalence implies n-strong order equivalence.

Lemma 5.2.2 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered logic programs and* $\sigma \in \{D, W, B\}$.

If $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *are strongly* $<^\sigma$*-equivalent, then* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *are n-strongly* $<^\sigma$*-equivalent.*

As with strong order equivalence, n-strong order equivalence requires strong equivalence of the underlying logic programs (cf. Theorem 5.1.4).

Theorem 5.2.3 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered logic programs.*

If $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *are n-strongly* $<^\sigma$*-equivalent, for* $\sigma \in \{D, W, B\}$*, then* $\Pi_1 \equiv_s \Pi_2$.

In Section 5.1.2.1 we have shown that strong order equivalence requires that the programs must coincide on their preference relations (cf. Theorem 5.1.9 and 5.1.10). Interestingly, this is not required for n-strong order equivalence. That is, n-strong order equivalent programs can differ on their preference relations. To see this, consider the following example:

$$(\Pi, <) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow a \\ r_2 < r_1 \end{array} \right\}$$

We obtain $(\Pi, <) \equiv_n^\sigma (\Pi, \emptyset)$ for $\sigma \in \{D, W, B\}$, since the preference $r_2 < r_1$ never selects answer sets as non-preferred, no matter which extension by normal programs is considered. Regarding strong order equivalence, we take

$$(\Pi', <') = \left\{ \begin{array}{ll} r_3 : y \leftarrow not\ x & \\ r_1 : a \leftarrow & r_1 <' r_3 \\ r_2 : b \leftarrow a & r_4 <' r_2 \\ r_4 : x \leftarrow & \\ r_5 : a \leftarrow z & \\ r_6 : z \leftarrow & \end{array} \right\}$$

and obtain $AS^\sigma((\Pi \cup \Pi', <')) = \{\{a, b, x, z\}\}$ but $AS^\sigma((\Pi \cup \Pi', < \cup <')) = \emptyset$ since the additional preference $r_2 < r_1$ discards the answer set $\{a, b, x, z\}$ as non-preferred. Hence, under n-strong order equivalence preference relations can be removed, which is not allowed under strong order equivalence. In Section 5.2.2 we will have a closer look on program transformations simplifying preference relations.

In contrast to strong order equivalence, n-strong order equivalent programs can also differ on their sets of generating rules and hence, on their sets of rules contributing to answer sets. For example, consider the following ordered programs:

$$(\Pi_1, <_1) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow \\ r_2 <_1 r_1 \end{array} \right\} \text{ and } (\Pi_2, <_2) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow \\ r_3 : a \leftarrow b \\ r_2 <_2 r_1 \end{array} \right\}$$

We obtain $(\Pi_1, <_1) \equiv_n^\sigma (\Pi_2, <_2)$ for $\sigma \in \{D, W, B\}$. By taking

$$(\Pi', <') = \left\{ \begin{array}{l} r_4 : y \leftarrow not\ a \\ r_1 : a \leftarrow \\ r_5 : b \leftarrow x \\ r_6 : x \leftarrow \\ r_1 < r_4 \end{array} \right\}$$

we obtain $AS^\sigma((\Pi_1 \cup \Pi', <_1 \cup <')) = \emptyset$ but $AS^\sigma((\Pi_2 \cup \Pi', <_2 \cup <')) = \{\{a, b, x\}\}$ since r_3 is used to block rule r_4 in an order preserving way. Hence, $(\Pi_1, <_1) \not\equiv_s^\sigma (\Pi_2, <_2)$ for all $\sigma \in \{D, W, B\}$. In Section 5.2.2, we will reconsider program simplifications known from (standard) strong equivalence and analyze them under n-strong order equivalence.

Whenever two ordered programs are n-strongly order equivalent, we have to impose further conditions for achieving strong order equivalence.

Lemma 5.2.4 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered programs such that* $(\Pi_1, <_1) \equiv_n^B (\Pi_2, <_2)$.
If $<_1 = <_2$ *and* $Cont(\Pi_1) = Cont(\Pi_2)$, *then* $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$.

Lemma 5.2.5 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered programs such that* $(\Pi_1, <_1) \equiv_n^\sigma (\Pi_2, <_2)$ *and* $\sigma \in \{D, W\}$.
If $<_1 = <_2$ *and* $Cont(\Pi_1) \setminus \{r \in \Pi_1 \mid head(r) \in body^+(r)\} = Cont(\Pi_2) \setminus \{r \in \Pi_2 \mid head(r) \in body^+(r)\}$, *then* $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$.

The preference semantics yield an increasing number of preferred answer sets, i.e. $AS^D((\Pi, <)) \subseteq AS^W((\Pi, <)) \subseteq AS^B((\Pi, <)) \subseteq AS(\Pi)$ for any ordered program $(\Pi, <)$. In Theorem 5.1.12 we have shown that strong $<^B$-equivalence of two ordered programs implies strong $<^W$-equivalence and that strong $<^W$-equivalence holds if and only if strong $<^D$-equivalence holds. Hence, the differences between the D- and W-semantics disappear under strong order equivalence. Furthermore, the differences between the B-semantics and the D- and W-semantics are strengthens under strong order equivalence, since the B-semantics decouples preference handling from rule application. Under n-strong order equivalence, we observe that any relationship between these three semantics disappears.

Theorem 5.2.6 *For any* $\sigma, \sigma' \in \{D, W, B\}$ *with* $\sigma \neq \sigma'$, *there exist ordered logic programs* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *such that* $(\Pi_1, <_1) \equiv_n^\sigma (\Pi_2, <_2)$ *but* $(\Pi_1, <_1) \not\equiv_n^{\sigma'} (\Pi_2, <_2)$.

This can be seen by considering the following examples:

$$(\Pi_{5.2}, <) \quad = \quad \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : a \leftarrow a \\ r_1 < r_2 \end{array} \right\} \tag{5.2}$$

We have $(\Pi_{5.2}, <) \equiv_n^B (\Pi_{5.2}, \emptyset)$ but $(\Pi_{5.2}, <) \not\equiv_n^\sigma (\Pi_{5.2}, \emptyset)$ for $\sigma \in \{D, W\}$, since $AS^\sigma((\Pi_{5.2}, <)) = \emptyset$ and $\{a\} \in AS^\sigma((\Pi_{5.2}, \emptyset))$. Hence, n-strong $<^B$- equivalence does not imply n-strong $<^\sigma$ equivalence for $\sigma \in \{D, W\}$. For the ordered program

$$(\Pi_{5.3}, <) \quad = \quad \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow a \\ r_3 : b \leftarrow \\ r_1 < r_2 \end{array} \right\} \tag{5.3}$$

we obtain $(\Pi_{5.3}, <) \equiv_n^W (\Pi_{5.3}, \emptyset)$ but $(\Pi_{5.3}, <) \not\equiv_n^D (\Pi_{5.3}, \emptyset)$, since $AS^D((\Pi_{5.3}, \emptyset)) = \{\{a, b\}\}$ and $(\Pi_{5.3}, <)$ has no $<^D$-preferred answer set. Hence, n-strong $<^W$- equivalence does not imply n-strong $<^W$- equivalence.

For showing that n-strong $<^W$- equivalence does not imply n-strong $<^B$- equivalence, let us consider

$$(\Pi_{5.4a}, <) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : a \leftarrow a \\ r_3 : y \leftarrow not\ a \\ r_1 < r_3 \end{array} \right\} \quad \text{and} \quad (\Pi_{5.4b}, <) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_3 : y \leftarrow not\ a \\ r_1 < r_3 \end{array} \right\} \tag{5.4}$$

We have $(\Pi_{5.4a}, <) \equiv_n^W (\Pi_{5.4b}, <)$, but $(\Pi_{5.4a}, <) \not\equiv_n^B (\Pi_{5.4b}, <)$ since $AS^B((\Pi_{5.4b}, <)) = \emptyset$ and $\{a\} \in AS^B((\Pi_{5.4a}, <))$. To see that n-strong $<^D$- equivalence does not imply n-strong $<^\sigma$- equivalence for $\sigma \in \{W, B\}$, let us consider

$$(\Pi_{5.5a}, <) = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow \\ r_3 : a \leftarrow not\ b \\ r_2 < r_3 \end{array} \right\} \quad \text{and} \quad (\Pi_{5.5b}, <') = \left\{ \begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow \\ r_4 : c \leftarrow not\ b \\ r_2 <' r_4 \end{array} \right\} \tag{5.5}$$

We observe $(\Pi_{5.5a}, <) \equiv_n^D (\Pi_{5.5b}, <')$, but $AS^\sigma((\Pi_{5.5b}, <')) = \emptyset$ and $AS^\sigma((\Pi_{5.5a}, <)) = \{\{a, b\}\}$ for $\sigma \in \{W, B\}$.

5.2.2 Transformations

In this section we provide some simplifications of ordered programs under n-strong order equivalence. Program simplifications that can be applied under strong order equivalence are also allowed under n-strong order equivalence. That is, $(\Pi, <) \equiv_n^\sigma (\Pi \setminus \{r\}, <)$ holds for rules r, where (i) $r \notin PR((\Pi, <))$, $head(r) \in body^+(r)$ and $\sigma \in \{D, W\}$, or (ii) $r \notin PR((\Pi, <))$, $r \notin Cont(\Pi)$, and $\sigma \in \{D, W, B\}$ (cf. Corollaries 5.1.20 and 5.1.21). Note that preference relations cannot be simplified under strong order equivalence (cf. Theorems 5.1.9 and 5.1.10).

5.2.2.1 Simplifications of preference relations

In contrast to strong order equivalence, we now consider program transformations under n-strong order equivalence simplifying preference relations. Whenever we remove a preference relation $r < r'$, all other preference relationships are maintained, e.g. we have $r^* < r < r'$ and remove $r < r'$, we keep the relations $r^* < r$ and $r^* < r'$.

Preference relations reflecting the order of rule application in enumerations (cf. Definition 2.4.1) are redundant under n-strong order equivalence and can be removed.

Theorem 5.2.7 *Let $(\Pi, <)$ be an ordered program and $r_1, r_2 \in \Pi$ such that $body(r_1) = \emptyset$, $head(r_1) \in body(r_2)$ and $r_2 < r_1$.*

Then, $(\Pi, <) \equiv_n^\sigma (\Pi, <')$ for $<' = < \setminus \{r_2 < r_1\}$ and $\sigma \in \{D, W, B\}$.

Since B- preferences decouple preference handling from rule application, we can delete all preferences between rules r_1 and r_2 that are applicable w.r.t. any answer set and any extension by a logic program and where $head(r_1) \in body^+(r_2)$. For this, we define similarly to the T_Π operator [144] the following:

$$\begin{array}{lcl} A^0(\Pi) & = & \{head(r) \mid r \in \Pi, body(r) = \emptyset\} \\ A^{i+1}(\Pi) & = & \{head(r) \mid r \in \Pi, body^-(r) = \emptyset, body^+(r) \subseteq A^i(\Pi)\} \\ A(\Pi) & = & \bigcup_{0 \leq i} A^i(\Pi) \end{array}$$

The set $A(\Pi)$ covers all monotonic conclusions, i.e. atoms that are true in every answer set. We say that $r \in Appl(\Pi)$ whenever $body^-(r) = \emptyset$ and $body^+(r) \subseteq A(\Pi)$. That is, r is a generating rule in all program extensions Π' of Π and for all answer sets X of $\Pi \cup \Pi'$.

The following theorem states that we can remove a preference relation between rules in $Appl(\Pi)$ under the B-semantics.

Theorem 5.2.8 *Let $(\Pi, <)$ be an ordered logic program and $r_1, r_2 \in Appl(\Pi)$ such that $head(r_1) \in body^+(r_2)$.*
Then, $(\Pi, <) \equiv_n^B (\Pi, <')$ for $<' = < \setminus \{r_2 < r_1, r_1 < r_2\}$.

The B-semantics allows to block a rule r by a lower ranked one r' as long as $head(r)$ is in the answer set (cf. Definition 2.4.3 on page 15). By stipulating $head(r) \in A(\Pi)$, we make sure that $head(r)$ is in any resulting answer set and hence, we can remove the corresponding preference relation.

Theorem 5.2.9 *Let $(\Pi, <)$ be an ordered program and $r_1, r_2 \in \Pi$ such that $r_1 \in Appl(\Pi)$, $head(r_1) \in body^-(r_2)$ and $head(r_2) \in A(\Pi)$.*
Then, $(\Pi, <) \equiv_n^B (\Pi, <')$ for $<' = < \setminus \{r_1 < r_2\}$.

E.g., for the program $(\Pi, <) = \{r_1 : a \leftarrow,\ r_2 : b \leftarrow not\ a,\ r_3 : b \leftarrow,\ r_1 < r_2\}$ we obtain $(\Pi, <) \equiv_n^B (\Pi, \emptyset)$. For the W-semantics, this simplifications is not directly transferable. There, rules can be applied and blocked by lower ranked ones, whenever the head of such a rule is derived earlier in an order preserving enumeration. But this can only be guaranteed by considering possible enumerations of rules from Π.

Within all three semantics, the preference relation $<$ is redundant whenever all standard answer sets are also preferred ones and all rules involved in $<$ are either in $Appl(\Pi)$ or blocked w.r.t. $A(\Pi)$.

Theorem 5.2.10 *Let $(\Pi, <)$ be an ordered logic program such that $PR(\Pi) \subseteq \{r \in Appl(\Pi)\} \cup \{r \in \Pi \mid body^-(r) \cap A(\Pi) \neq \emptyset\}$, and $AS^\sigma((\Pi, <)) = AS(\Pi)$ for some $\sigma \in \{D, W, B\}$.*
Then, $(\Pi, <) \equiv_n^\sigma (\Pi, \emptyset)$.

5.2.2.2 Transformations from standard strong equivalence

In [25, 193, 80, 154], transformations on logic programs are reported, which can be used for simplifying programs (cf. Section 2.5.2 on page 17). For those modular transformations, programs are strongly equivalent to the transformed one. Such program simplifications were considered for strong order equivalence in Section 5.1.4. Under strong order equivalence, we can remove rules $r \notin PR((\Pi, <))$ and where either $head(r) \in body^+(r)$ or $r \notin Cont(\Pi)$ holds.

In what follows, we reconsider program simplification from [25, 193, 80, 154] under the notion of n-strong order equivalence. Since we have presented simplifications for preference relations in the previous section, we now assume that removable, hence redundant, rules are not involved in preference relations. That is, we separate preference simplifications from simplifications of the underlying logic programs.

The transformation *TAUT*, stating that $\Pi \equiv_s \Pi \setminus \{r\}$ for all $r \in \Pi$ where $head(r) \in body^+(r)$, is allowed under strong order equivalence for the D- and W-semantics, but not for the B-semantics. The

same applies for n-strong order equivalence. Regarding B-semantics, let us consider

$$(\Pi, <) = \left\{ \begin{array}{l} r_1 : a \leftarrow a \\ r_2 : a \leftarrow \\ r_3 : y \leftarrow not\ a \\ r_2 < r_3 \end{array} \right\}$$

We observe $AS^B((\Pi, <)) = \{a\}$, but $AS^B((\{r_2, r_3\}, <)) = \emptyset$. Hence, rules where $head(r) \in body^+(r)$ cannot removed under n-strong $<^B$-equivalence.

The simplifications *S-IMP** (cf. Section 5.1.4), *RED*$^-$, and *CONTRA* are allowed under strong order equivalence as long as the redundant rules are not involved in the preference relation. Hence, we can apply them analogously under n-strong order equivalence.

Let be $r_1, r_2 \in \Pi$, $head(r_1) = head(r_2)$, and $body(r_2) \subseteq body(r_1)$, then the transformation *NONMIN* states $\Pi \equiv_s \Pi \setminus \{r_1\}$. Let us consider the following example

$$(\Pi, <) = \left\{ \begin{array}{l} r_2 : a \leftarrow \\ r_1 : a \leftarrow b \\ r_3 : b \leftarrow \\ r_4 : y \leftarrow not\ a \\ r_2 < r_4 \end{array} \right\} \quad \not\equiv_n^\sigma \quad (\Pi \setminus \{r_1\}, <).$$

Here, we cannot remove r_1 since r_2 is involved in $<$ and r_1 is used to derive $head(r_1) = head(r_2)$ in an alternative way. The following Lemma states that this transformation can be applied under n-strong order equivalence whenever $r_1, r_2 \notin PR((\Pi, <))$.

Lemma 5.2.11 *(NONMIN*) Let $(\Pi, <)$ be an ordered logic program and $\sigma \in \{D, W, B\}$. Furthermore, let be $r_1, r_2 \in \Pi$ such that $head(r_1) = head(r_2)$, $body(r_2) \subseteq body(r_1)$, and $r_1, r_2 \notin PR((\Pi, <))$.*

Then, $(\Pi, <) \equiv_n^\sigma (\Pi \setminus \{r_1\}, <)$.

The following transformation creates new rules to make other transformations applicable. Let be $r_1 \in \Pi$, where $a \in body^+(r_1)$, $G_a = \{r_2 \in \Pi \mid head(r_2) = a\}$, and $G_a \neq \emptyset$. Then, transformation *WGPPE* states that $\Pi \equiv_s \Pi \cup G'_a$ holds where $G'_a = \{head(r_1) \leftarrow (body^+(r_1) \setminus \{a\}) \cup not\ body^-(r_1) \cup body(r_2) \mid r_2 \in G_a\}$. Let us consider the following example:

$$(\Pi, <) = \left\{ \begin{array}{l} r_2 : a \leftarrow \\ r_1 : b \leftarrow a \\ r_3 : y \leftarrow not\ b \\ r_1 < r_3 \end{array} \right\} \quad \text{and } (\Pi \cup \{r^{G_a}\}, <) = \left\{ \begin{array}{l} r_2 : a \leftarrow \\ r_1 : b \leftarrow a \\ r^{G_a} : b \leftarrow \\ r_3 : y \leftarrow not\ b \\ r_1 < r_3 \end{array} \right\}$$

for $\sigma \in \{D, W, B\}$. We observe $(\Pi, <) \not\equiv_n^\sigma (\Pi \cup \{r^{G_a}\}, <)$. Analogously to Lemma 5.2.11, we observe that r_1 cannot be involved in $<$ if this transformation should be possible under $\equiv_n^\sigma$, since rules from G'_a are alternatives to derive rules in an order preserving way.

Lemma 5.2.12 *(WGPPE*) Let $(\Pi, <)$ be an ordered logic program and $\sigma \in \{D, W, B\}$. Furthermore, let be $r_1 \in \Pi$, $r_1 \notin PR((\Pi, <))$ $a \in body^+(r_1)$, $G_a = \{r_2 \in \Pi \mid head(r_2) = a\}$ and $G_a \neq \emptyset$.*

Then, we have $(\Pi, <) \equiv_s^\sigma (\Pi \cup G'_a, <)$ for

$$G'_a = \{head(r_1) \leftarrow (body^+(r_1) \setminus \{a\}) \cup not\ body^-(r_1) \cup body(r_2) \mid r_2 \in G_a\}.$$ [5]

An overview of the applicable simplifications in shows in table 5.2

[5] For a set B of atoms, we denote with $not\ B$ the set $\{not\ b \mid b \in B\}$.

	TAUT	*RED*$^-$	*CONTRA*	*NONMIN**	*S-IMP**	*WGPPE**
$\equiv_n^D$	yes	yes	yes	yes	yes	yes
$\equiv_n^W$	yes	yes	yes	yes	yes	yes
$\equiv_n^B$	no	yes	yes	yes	yes	yes

Table 5.2: Applicability of simplifications under $\equiv_n^\sigma$.

5.2.3 Complexity Results

In this section, we discuss the computational complexity of deciding n-strong order equivalence.

First, we show that whenever two programs are not n-strongly order equivalent, there exists a program extension Π^* such that $AS^\sigma((\Pi_1 \cup \Pi^*, <_1)) \neq AS^\sigma((\Pi_2 \cup \Pi^*, <_2))$, where Π^* has the form

$$\Pi^* \quad = \quad \Pi' \cup \{x \leftarrow\} \tag{5.6}$$

where x is a new atom not appearing in $\Pi_1 \cup \Pi_2$ and $\Pi' \subseteq \Pi_1 \cup \Pi_2 \cup \{a \leftarrow x \mid a \in Atm(\Pi_1 \cup \Pi_2)\}$. That is, whenever two ordered programs are not n-strongly order equivalent, there exists an extension which contains only rules from Π_1, Π_2, the rule $x \leftarrow$, and rules of the form $a \leftarrow x$, where a is an atom appearing in Π_1 or Π_2.

Lemma 5.2.13 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered logic programs and* $\sigma \in \{D, W, B\}$.

If $(\Pi_1, <_1) \not\equiv_n^\sigma (\Pi_2, <_2)$ *then there exists a normal program* Π *of the form (5.6) such that* $AS^\sigma((\Pi_1 \cup \Pi, <_1)) \neq AS^\sigma((\Pi_2 \cup \Pi, <_2))$.

By that lemma, we can show that deciding whether two ordered programs are n-strongly order equivalent is co-NP-complete.

Theorem 5.2.14 *Let* $(\Pi_1, <_1)$ *and* $(\Pi_2, <_2)$ *be ordered logic programs and* $\sigma \in \{D, W, B\}$.

Then, deciding whether $(\Pi_1, <_1) \equiv_n^\sigma (\Pi_2, <_2)$ *is* co-NP-*complete.*

We have thus shown that n-strong order complexity testing is not more complex than strong equivalence testing or strong order equivalent testing.

5.2.4 Conclusions

Since strong order equivalence imposes rigorous conditions on ordered programs being strongly order equivalent (cf. Theorem 5.1.9 and 5.1.10), e.g. programs have to coincide on their preference relations and on rules contributing to answer sets, we have considered in this section a weaker version of order equivalence. We have presented the notion of n-strong order equivalence for ordered programs. We have considered three semantics for handling preferences and have characterized n-strong order equivalence under these semantics. We have pointed out that this weakened notion imposes softer conditions on ordered programs being n-strongly order equivalent. More precisely, only strong equivalence of the underlying logic program is required, whereas the identity of preference relations and rules contributing to answer sets is not supposed. This property allows the simplification of preference relations under n-strong order equivalence. Hence, we have provided in Section 5.2.2 several program transformations for simplifying preference relations and the underlying logic programs. We have concentrated on two types of program transformations: (1) simplifications of preference relations and (2) simplifications of the underlying logic programs. Theorem 5.2.7- 5.2.9 describe simplifications of preference relations. Theorem 5.2.10 shows one condition when the given preference relation is totally redundant. Regarding simplifications of the underlying logic programs, we have reconsidered the transformations in [25, 193, 80, 154] w.r.t. strong equivalence (cf. Section 2.5.2 on page 17). All are considered under the aspect that removable rules are not involved in

preference relations. The transformation *TAUT* is possible only for the D- and W-semantics. Reductions *RED*$^-$, *CONTRA*, and *S-IMP** are possible w.r.t. all preference semantics. Transformation *NONMIN* and *WGPPE* are not possible under strong order equivalence, but they become applicable under n-strong order equivalence as long as one makes further restrictions to the preference relations.

The considered preference semantics yield an increasing number of preferred answer sets that is for any ordered program $(\Pi, <)$ we have $AS^D((\Pi, <)) \subseteq AS^W((\Pi, <)) \subseteq AS^B((\Pi, <)) \subseteq AS(\Pi)$. Theorem 5.1.12 shows that under strong equivalence this relationship is changed. Interestingly, Theorem 5.2.6 gives us that in terms of n-strong order equivalence no relationship between the semantics exists.

Additionally, we have investigated computational complexity. It turns out that n-strong order complexity testing is not more complex than strong equivalence testing or strong order equivalent testing.

5.3 Summary

Strong equivalence for logic programs under the answer set semantics has been studied intensively in the recent years [80, 133, 134, 154, 188]. Since strong equivalence is rather strict (in the sense that not many programs are strongly equivalent), one can consider the more liberal notion of *uniform equivalence* [78, 159]. Instead of allowing for arbitrary program extensions, uniform equivalence restricts extensions to sets of facts. As discussed in [159], uniform and strong equivalence are essentially the only concepts of equivalence obtained by varying the logical form of the program extension. Latest research concentrates on relativized notions of strong and uniform equivalence. Instead of considering *all* possible program extensions, one can consider only programs built from a restricted alphabet or, orthogonally, compare answer sets only on a subset of the alphabet [196, 82], which seems to be very natural from practical point of view. Other work focuses on the extension of notions of equivalence for the non-ground case [81, 134].

Besides this variety of program equivalences, a notion for strong equivalence under updating logic programs has been defined in [110]. Two programs Π_1 and Π_2 are strongly update equivalent if $AS((\Pi_1 \setminus Q) \cup R) = AS((\Pi_2 \setminus Q) \cup R)$ holds for all programs Q and R [110]. Interestingly, two strong update equivalent programs can differ only in rules r of the form:

(i) $head(r) \in body^+(r)$ and

(ii) $body^+(r) \cap body^-(r) \neq \emptyset$

(restricted to the case of normal logic programs). Strong update equivalence allows only a few program simplifications since the conditions for strong update equivalence are very strict, actually so strict that checking whether two programs are strong update equivalent can be done in polynomial time. This is based on the fact that two strongly update equivalent programs differ only in rules described above.

Another line of considering equivalences was presented in [111], namely equivalence of abductive logic. Considering equivalences within abduction leads to different observations. First, *explainable equivalence* requires that two abductive programs have the same explainability for any observation. Second, *explanatory equivalence* guarantees that any observation has exactly the same explanations in each abductive framework.

In this chapter, we have concentrated on notions of equivalence for logic programs with preferences, which have to the best of our knowledge never been studied before. In analogy to strong equivalence, which concentrates on extensions of logic programs by logic programs, we have defined strong order equivalence and n-strong order equivalence. Strong order equivalence refers to ordered programs extended by ordered programs and n-strong equivalence to ordered programs extended by normal programs. For both notions of equivalence we have studied characterizations concerning three semantics for preference handling, namely the D-, W-, and B- semantics. We could show in Corollaries 5.1.9 and 5.1.10 that two programs are strongly $<^W$- and $<^D$-equivalent, if and only if their preference relations are equal, their standard ASP programs are strongly equivalent, and their "non-looping" generating rules are identical for all answer sets of any extension of them. For B-preference, also the "looping" generating rules must be identical in any

answer set. By Theorem 5.2.3 we could also show that n-strongly equivalent programs must be strongly equivalent in the standard sense. But, in contrast to strong order equivalence, n-strongly order equivalent programs can differ on their preference relations and on rules contributing to answer sets. This fact allows for simplifications of preference relations (cf. theorems 5.2.7- 5.2.10), whereas under strong order equivalence, no simplifications of preference relations are possible. Theorem 5.2.10 gives a condition were a preference relation can totally be removed under n-strong order equivalence, whereas Corollary 5.1.3 shows that under strong order equivalence no ordered program with a non-empty preference relation can be replaced by a normal logic program without preferences. Additionally, we have analyzed program simplifications known from normal logic programs (see Section 2.5.2 on page 17). We have considered all transformations under the aspect that the simplified rules do not occur in the preference relation. This is necessary since under strong order equivalence we cannot simplify preference relations and under n-strong order equivalence we have separated transformations simplifying preference relations from simplifications acting on the underlying logic program. The transformations *TAUT* is possible under n-strong and strong order equivalence only for the D- and W- semantics. Transformations *RED*$^-$ and *CONTRA* are possible under both notions of order equivalence and under all three preference semantics. The transformations *NONMIN* and *WGPPE* are not possible under strong order equivalence, but with a small restriction under n-strong order equivalence (Lemma 5.2.11 and 5.2.12). The transformation *S-IMP* was originally developed for disjunctive logic programs. Restricted to normal logic programs, *S-IMP* falls in one case back to *NONMIN* and in the other case the simplified rule become never applicable. Hence, the restricted transformation *S-IMP*, called *S-IMP**, is possible under both strong and n-strong order equivalence as long as the simplified rules are not involved in the preference relation.

Program transformations under standard strong equivalence for disjunctive logic programs have also been studied in [135]. There, the authors have defined k-m-n problems, where m rules are replaced by simpler n rules in the presence of k rules under strong equivalence. The 0-1-0 problem deletes one rule and falls back to *TAUT* and *CONTRA* for the case of normal logic programs. The 1-1-0 problem deletes one rule in the presence of another one and falls back to *NONMIN* in one case. In the other case, we have for two rules r_1 and r_2, where $body(r_1) \subseteq body(r_2)$ and $head(r_1) \in body^-(r_2)$ that r_2 can be removed, i.e. $\{r_1, r_2\} \equiv_s \{r_1\}$. Whenever r_1 is applicable, w.r.t. any extension and any answer set of it, we have that r_2 is inapplicable and whenever r_1 is not applicable, then $body(r_2)$ is also not derivable and hence, r_2 becomes not applicable. Thus, 1-1-0 falls back to Corollary 5.1.20 and can be applied under n-strong and strong order equivalence as long as the redundant rule is not involved in the preference relation. The problem 0-1-1 restricted to normal logic programs, where one rule r_1 is replaced by another one r_2, falls also back to Corollary 5.1.20, since both rules become never applicable.

We have studied the computational complexity of the main decision problems, in particular deciding whether two programs are order equivalent, whether two programs are strongly order equivalent, and whether two programs are n-strongly order equivalent. It turned out that all three problems are co-NP-complete (Theorem 5.1.14, 5.1.19, and 5.2.14), and thus are neither harder nor simpler than equivalence and strong equivalence.

The considered preference semantics yield an increasing number of preferred answer sets w.r.t. set inclusion. Theorem 5.1.12 shows us that under strong equivalence this relationship is changed. More precisely, if two ordered programs are strongly $<^B$-equivalent, then they are strongly $<^W$-equivalent, and they are strongly $<^W$-equivalent if and only if they are strongly $<^D$-equivalent. Interestingly, Theorem 5.2.6 shows us that under n-strong order equivalence no relationship between the semantics exists. Hence, the relationship between the three preference semantics is totally changing under different notions of order equivalence.

In Section 5.2.2, we have given some transformations under n-strong order equivalence. Future work would include simplifications on ordered programs under n-strong order equivalence, where also the difference between the preference semantics are appearing. In Section 5.2, we have made a first step into this direction with Theorems 5.2.8 and 5.2.9 for B-semantics. Also, we have only studied transformations, where one rule or preference relations are removed separately. Another possibility is to analyze program

transformations, which (i) remove several rules in one step and (ii) remove rules and preference relations together. Corollaries 5.1.20 and 5.1.21 indicate a direction, which kind of program simplifications are possible under strong order equivalence. Further research issues investigate in conditions describing exactly all possible program simplifications under strong order equivalence as it was done for strong equivalence in [135].

We also want to point out that integrity constraints are not allowed in ordered programs. For integrity constraints one uses normally the transformation $f \leftarrow body, not\ f$ for the integrity constraint $\leftarrow body$ and a new atom f. But it can be shown that $\{a \leftarrow not\ a\} \equiv_s \{\leftarrow not\ a\}$ but $\{a \leftarrow not\ a\} \not\equiv_s \{f \leftarrow not\ a, not\ f\}$ [135, 82]. However, such a rewriting is not sensitive under strong equivalence. Since n-strong and strong order equivalence require strong equivalence of the underlying logic programs, such a rewriting would also not be sensitive under order equivalence if one allows integrity constraints in the language.

Based on the characterizations for strong order equivalence, we are able to define SOE structures, which can be thought of as a generalization of SE models (see also on page 16 in Section 2.5). Each ordered logic program has an associated set of SOE structures. If these SOE structures are equal for two programs, then these programs are strongly order equivalent. Different from SE models, these structures contain also information on the preference relation and on the set of contributing rules. Further research studies include to define similar structures to characterize n-strong order equivalence.

Another way of analyzing strong order equivalence could be offered by a compilation of the programs, as done e.g. in the `plp`-System [62, 161]. There, ordered programs are compiled into logic programs such that their answer sets directly correspond to the preferred answer sets of the original ordered program. Since the compilation is a logic program, one could suppose that two ordered programs are strongly order equivalent if the compilation of them possesses the same SE-models. However, the compilation introduces additional symbols. Hence, the answer sets of two compiled programs, which represent the same preferred answer set, are in general incomparable because of these additional symbols. One would therefore have to consider strong equivalence under a projection of the answer sets. This could be handled by the methods presented in [82].

But in addition one would also have to take into account that compiled programs have a particular, rather than arbitrary, structure. Therefore, the extensions to the programs to be considered are not arbitrary but only those that could have resulted from a `plp` compilation. Notably, this affects the structure, and not only the alphabet of the extensions. Such a notion of strong equivalence has (to the best of our knowledge) not been studied so far. Therefore, we leave this kind of analysis for future work.

Furthermore, we have only considered three of the many semantics for logic programs with preferences. Further work would include investigations on how our results can generalized to other rule-based approaches, such as prioritized logic programming [199] or CR-Prolog [8]. Instead of expressing preferences among rules, several semantics for preferences among atoms have been studied, e.g. PLP [172], ordered disjunction [28], answer set optimization [37]. There, it may be possible to define SOE-models, without having to include any rule or preference data explicitly.

For strong equivalence, it has been shown that two programs are strongly equivalent if and only if they are equivalent in the Logic of Here and There [133]. We are not aware of any similar logic which would allow for preferences or related structures, so we do not believe that a similar characterization in a non-classical logic can be found for strong order equivalence.

Chapter 6

Applications of Preferences

In this chapter, we present new applications of preferences within answer set programming.

First, we consider new ideas in group decision making. Initially, group decision making often requires total orders among the set of candidates. But sometimes the voters want to express only partial orders instead of total orderings. In Section 6.1, we thus focus on the application of a voting procedure when the voters preferences are incomplete. For this, we define possible (resp. necessary) winners for a given partial preference profile $\mathcal{R}$ with respect to a given voting procedure as the candidates being the winners in some (resp. all) of the complete extensions of $\mathcal{R}$. We show that, although the computation of possible and necessary winners may be hard in the general case, it is polynomial for the family of positional scoring procedures, e.g. Borda, Plurality. We show that the possible and necessary Condorcet winners for a partial preference profile can be computed in polynomial time as well. Additionally, we point out connections to vote manipulation and elicitation. In Section 6.2 we present for the first time an encoding of these voting procedures with partial preference profiles within answer set programming by using aggregate functions from DLV (cf. Section 2.1.3.4 on page 11). In Section 6.3, we present an application of these new voting procedures within a scheduling problem. We consider the problem of scheduling a meeting for a research group, where we describe first the basic problem, then we include diagnostic reasoning whenever no meeting is schedulable and at last, we use the techniques defined in Section 6.1 to determine preferred meetings, whenever several meetings are schedulable.

As a second new application, we associate optimality theory with abduction and preference handling for the first time in Section 6.4. We present linguistic problems that appear in the study of dialects as new application of abduction and preference handling. We consider differences in German dialects, which originate from different rankings of linguistic constraints which determine the well-formedness of expressions within a language. We introduce a framework for analyzing differences in German dialects by abduction of preferences. More precisely, we will take the perspective of a linguist and reconstruct dialectal variation as an abduction problem: Given an observation that a sentence is found as grammatically correct, abduct the underlying constraint ranking. For this, we give a new definition for the determination of optimal candidates for total orders with indifferences. Additionally, we give an encoding for the diagnosis front-end of the DLV system.

In Section 6.5, we briefly summarize the results and we give further application areas for preferences within answer set programming.

6.1 Voting procedures with incomplete preferences

Automated group decision making is an important issue in Artificial Intelligence: autonomous agents often have to agree on a common decision, and may for this reason apply *voting procedures*, which is one of the

most common ways of making a collective choice. Voting procedures, studied extensively by social choice theorists from the normative point of view, have been recently studied from the computational point of view: (i) While winner determination is easy with most usual voting procedures (at least when the number of candidates is small), a few of them are hard. Their complexity and their practical computation have been investigated in [169, 106, 58]. (ii) Some work focuses on sets of candidates with a combinatorial structure and investigate compact representation issues [129, 168]. (iii) Even when computing the outcome of a voting procedure is easy, it might be the case that determining whether there is a successful manipulation for a coalition of voters is hard [49, 48, 51]. (iv) Elicitation issues and partial winner determination for partial preference profiles have been studied in [50].

In this section, we focus on the last of these issues, which raises the question of the application of a voting procedure when the voters preferences are incomplete. Let $X = \{x_1, \ldots, x_m\}$ be a finite set of candidates and $I = \{1, \ldots, n\}$ be a finite set of *voters* ; a collective preference profile is a collection of partial preference relations $\mathcal{R} = \langle R_1, \ldots, R_n \rangle$ on X (formal details follow in Section 6.1.2). Winner determination under incomplete preference consists in applying (in some sense that we make precise later on) a voting procedure over such a collective preference profile. This is particularly relevant in the following situations:

- some voters have expressed their preference profile and some others have not yet done it; in that case, the collective preference profile is a collection consisting of n_1 complete preference relations and $n - n_1$ empty preference relations.
- all voters have expressed their preferences on a given subset of candidates, and now new candidates are introduced, about which the voters' preferences are unknown.
- voters are allowed to express their preferences in an incremental way: they left some comparisons between candidates unspecified, because either they *don't know* or they *don't want* to compare some candidates (we comment further on the various possible interpretations of incomplete preferences).
- preferences have been only partially elicited and/or are expressed in a language for compact preference representation such as CP-nets [21] (cf. Section 4.4.7 on page 80), which induce partial preference relations in the general case.

In all these cases it may be worth having an idea of the possible outcomes of the vote without waiting for the preferences to be complete (which sometimes never happens, as in the "refuse to compare" case mentioned above). In some cases, we may conclude that the preferences known so far, although incomplete, are informative enough so that the outcome of the vote can be determined; if this is not the case, we may compute a set of candidates that may win the vote after the preferences have become complete, thus giving the voters an opportunity to focus on these candidates and forget about the others. Lastly, similarly as in [50], we may determine from these incomplete preferences which preferences should be elicited from whom so as to be able to compute the winner.

In this section, we extend the application of a voting procedure (usually defined on complete preference relations over candidates) when the voters' preferences consist of partial orders. For this, we give in Section 6.1.1 some basic background on voting procedures. In Section 6.1.2, after briefly discussing three different ways of applying voting procedures (tailored for complete preferences) to incomplete preferences, we focus on one of these ways, that is more suited to the case where incompleteness corresponds to an incomplete knowledge of the voters' preferences. We then introduce the natural notions of necessary and possible winners for a given partial preference profile and a voting procedure. In Section 6.1.3, we show that in the case of positional scoring procedures, possible and necessary winners can be computed in polynomial time by a very simple algorithm. In Section 6.1.4, we investigate the notion of possible and necessary Condorcet winners and show that they can be computed as well in polynomial time. Section 6.1.5 considers related issues such as vote manipulation and elicitation. An application of these new voting procedures within scheduling problems follows in Section 6.3.

6.1.1 Voting Procedures

Let $X = \{x_1, \ldots, x_m\}$ be a finite set of *candidates* and $I = \{1, \ldots, n\}$ be a finite set of *voters*. A *complete preference profile* is a collection $\mathcal{T} = \langle T_1, \ldots, T_n \rangle$ of linear orders on X, where T_i represents the preferences of voter i. A *voting procedure* F maps every complete preference profile $\mathcal{T}$ to a nonempty subset of X, where $F(\mathcal{T})$ denotes the set of winners of $\mathcal{T}$ w.r.t. F. Note, importantly, that the outcome $F(\mathcal{T})$ of a voting procedure is always nonempty, i.e., the outcome of the procedure is defined for all preference profiles.

A *positional scoring procedure* is defined from a *scoring vector* which is a vector $\vec{s} = (s_1, \ldots, s_m)$ of integers such that $s_1 \geq s_2 \geq \ldots \geq s_m$ and $s_1 > s_m$. Some well-known examples of positional scoring procedures are:

- the *Borda* procedure, which is the defined from the scoring vector $s_k = m - k$ for all $k = 1, \ldots, m$;
- the *plurality* procedure, which is defined from the scoring vector $s_1 = 1$, and $s_k = 0$ for all $k > 1$.

Let $\mathcal{T} = \langle T_1, \ldots, T_n \rangle$ be a complete preference profile. For every $x \in X$ and every $i \in I$, let $r(T_i, x) = |\{y \mid y >_{T_i} x\}| + 1$ be the rank of x in the complete order T_i. Then, we define the *scoring function* as

$$S(x, \mathcal{T}) = \sum_{i=1}^{n} s_{r(T_i,x)}.$$

The positional scoring rule $F_{\vec{s}}$ associated with a scoring vector $\vec{s}$ is defined by its set

$$F_{\vec{s}}(\mathcal{T}) = \{x \mid S(x, \mathcal{T}) \text{ is maximal } \}.$$

That is, the set of winning candidates for $\mathcal{T}$ with respect to $F_{\vec{s}}$ is the set of candidates in X maximizing the scoring function $S(., \mathcal{T})$.

Example 36 *We have three voters $I = \{v_1, v_2, v_3\}$ and three candidates $X = \{x_1, x_2, x_3\}$ (hence, $m = 3$). Voter v_1 prefers x_1 over x_2 and x_2 over x_3. Hence T_1 is the preference relation $x_1 > x_2 > x_3$. For voter v_2 we have T_2 as $x_2 > x_3 > x_1$ and for voter v_3 we have T_3 as $x_2 > x_1 > x_3$. The rank for each candidate in each complete order T_i is given in the following table:*

$r(T_i, x)$	T_1	T_2	T_3
x_1	*1*	*3*	*2*
x_2	*2*	*1*	*1*
x_3	*3*	*2*	*3*

For the Borda procedure, we get the following values $s_{Borda}(T_i, x)$ for the scoring vector, where $s_{Borda}(T_i, x) = m - r(T_i, x)$:

$s_{Borda}(T_i, x)$	T_1	T_2	T_3
x_1	*2*	*0*	*1*
x_2	*1*	*2*	*2*
x_3	*0*	*1*	*0*

Hence, we get $S_{Borda}(x_1, \mathcal{T}) = 3$, $S_{Borda}(x_2, \mathcal{T}) = 4$, and $S_{Borda}(x_3, \mathcal{T}) = 1$. Thus, x_2 is the winning candidate w.r.t. Borda, since it maximizes the Borda scoring function.

For the Plurality procedure, we get the following scoring vectors:

$s_{Plural}(T_i, x)$	T_1	T_2	T_3
x_1	*1*	*0*	*0*
x_2	*0*	*1*	*1*
x_3	*0*	*0*	*0*

Hence, we get $S_{Plural}(x_1, \mathcal{T}) = 1$, $S_{Plural}(x_2, \mathcal{T}) = 2$, and $S_{Plural}(x_3, \mathcal{T}) = 0$. Thus, x_2 is the winning candidate w.r.t. Plurality, since it maximizes the Plurality scoring function.

Among the many voting procedures that exist in the literature (for an extensive presentation see for instance [24]), some require preference profiles to be linear orders and some allow more generally preferences to be weak orders (where antisymmetry is not required, which implies that a voter can express an indifference between two candidates). However, for the sake of simplicity, we assume that all preference relations considered are antisymmetric. This is not a real loss of generality, as most of our definitions and results would extend to the case where indifference is allowed (see Section 6.1.6).

Even if some voting procedures work on linear orders and some on weak orders, a common point of all procedures is that they apply to *complete* preference relations: in other words, they are not tailored for dealing with *incomparability*. The question now is how F should be extended when we have only a partial knowledge of the preferences of the voters – in other terms, how should F be defined when the input is a collection of *orders* rather than a collection of *linear orders*. This issue is investigated in the next section.

6.1.2 Voting procedures with incomplete preferences: definitions

6.1.2.1 Extending voting procedures to incomplete preferences

A *voting problem under incomplete preferences* is composed of a finite set $X = \{x_1, \ldots, x_m\}$ of *candidates*, a finite set of *voters* $I = \{1, \ldots, n\}$, and for each i, an order R_i on X denoting the *individual preference profile* of voter i. The collection of orders $\mathcal{R} = \langle R_1, \ldots, R_n \rangle$ is called a *(collective) preference profile*. $\mathcal{R}$ is said to be complete iff R_i is complete for each $0 \leq i \leq n$. In the following, R_i is often denoted as $\succeq_{R_i}$ or as $\succeq_i$: thus, we write indifferently $R_i(x, y)$, $x \succeq_{R_i} y$, or $x \succeq_i y$.

The notion of complete extension is generalized from individual to collective preference profiles in a natural way (cf. on page 13 in Section 2.3 for extensions of partial preference relations):

$$Ext(\mathcal{R}) = Ext(R_1) \times \ldots \times Ext(R_n)$$

There are at least two interpretations for incomplete preferences: *intrinsic* incompleteness, where the voter refuses to compare some alternatives, or *epistemic* incompleteness, where the voter has a complete preference but it is only partially known at the time the voting procedure has to be applied. These different interpretations lead to different ways of extending voting procedures to partial preferences. Here are three possible ways that can be followed, where F is a given voting procedure defined for complete preference relations:

1. apply F to all complete extensions of the preference relations and gather the results;

2. select a subset of those complete extensions (ideally a singleton) using some completion process, apply F to these and gather the results; [1]

3. rewrite directly the definition of F so that it applies more generally to partial preference relations (obviously, this extension of F must coincide with F on complete preference profiles).

In the following, we explore only the first of these three ways, which looks the most natural of all three ways; furthermore it seems to be more suited to epistemic incompleteness of preference (see Section 6.1.5).

[1] This completion process may consist in letting candidates gravitate towards preference such as in [20] or towards indifference such as in [187].

6.1.2.2 Possible and necessary winners

For applying a voting procedure to all complete extensions of a partial preference profile, we define upper and lower bounds for winners.

Definition 6.1.1 *Let F be a voting procedure on X and $\mathcal{R}$ a (possibly incomplete) preference profile.*

- *$x \in X$ is a* necessary winner *for $\mathcal{R}$ (w.r.t. F) iff for all $\mathcal{T} \in Ext(\mathcal{R})$ we have $x \in F(\mathcal{T})$.*
- *$x \in X$ is a* possible winner *for $\mathcal{R}$ (w.r.t. F) iff there exists a $\mathcal{T} \in Ext(\mathcal{R})$ such that $x \in F(\mathcal{T})$.*

A necessary winner for $\mathcal{R}$ is thus a candidate which wins in all complete extensions of $\mathcal{R}$ and a possible winner wins in at least one complete extension of $\mathcal{R}$. Hence, necessary winners constitute an upper bound and possible winners a lower bound for winners of a partial preference profile. We denote by $NW_F(X)$ (respectively, $PW_F(X)$) the set of necessary (respectively, possible) winners for $\mathcal{R}$ w.r.t. F. By Definition 6.1.1, the following properties hold, for any voting procedure F:

- for all $\mathcal{R}$, $NW_F(\mathcal{R}) \subseteq PW_F(\mathcal{R})$;
- for all $\mathcal{R}$, $\mathcal{R}'$ such that $\mathcal{R} \subseteq \mathcal{R}'$, $PW_F(\mathcal{R}') \subseteq PW_F(\mathcal{R})$ and $NW_F(\mathcal{R}') \subseteq NW_F(\mathcal{R})$.

Note also that $NW_F(X)$ can be empty, but not $PW_F(X)$. Whenever $\mathcal{R}$ is a complete preference profile, possible and necessary winners coincide.

The next sections try to evaluate the difficulty of applying some well-known voting procedures to partial preference relations, by assessing the computational complexity of the problems and by giving explicit algorithms for computing possible and necessary winners.

Since there are, in the general case, exponentially many extensions of a partial preference profile, nothing guarantees that computing possible and necessary winners can be done in polynomial time, even if the voting procedure F is polynomially computable. All we can say is that, by Definition 6.1.1, provided that F is polynomially computable:

- Determining whether $x \in PW_F(\mathcal{R})$ is in NP.
- Determining whether $x \in NW_F(\mathcal{R})$ is in co-NP.

The question is now: are there any voting procedures such that necessary and possible winners can still be determined in polynomial time? We answer this question positively in the next two sections.

6.1.3 Positional scoring procedures

The question now is, how hard is it to determine whether x is a necessary or a possible winner for $\mathcal{R}$ w.r.t. a scoring procedure F?

For this, let us define the *minimal* (respectively, *maximal*) *rank* of a candidate x for a (partial) order R as the lowest[2] (respectively, highest) possible rank of x obtained when considering all complete extensions of R that is,

$$rank_R^{min}(x) = \min_{T \in Ext(R)} r(T, x)$$
$$rank_R^{max}(x) = \max_{T \in Ext(R)} r(T, x),$$

where $r(T, x)$ is the rank of x in the complete order T, which is a complete extension of R.

These bounds are actually much easier to compute than what their definition suggests. For some voter, the minimal rank of x is determined by the number of candidates which are higher ranked in the order and the maximal rank of x is determined by the number of lower ranked candidates.

[2]Recall that the lower its rank, the more preferred a candidate.

Theorem 6.1.1 *Let R be a (partial) strict order. Then,*

$$rank_R^{min}(x) = |\{y \mid y >_R x\}| + 1 \text{ and}$$
$$rank_R^{max}(x) = m - |\{y \mid x >_R y\}|.$$

Next, necessary and possible winners can be computed by considering the best and the worst case for values of scoring functions.

Theorem 6.1.2 *Let $\mathcal{R}$ be a preference profile, where each R_i is a (partial) order, F_s be a positional voting procedure, and*

$$S_{\mathcal{R}}^{min}(x) = \sum_{i=1}^{n} s_{rank_{R_i}^{max}(x)}$$
$$S_{\mathcal{R}}^{max}(x) = \sum_{i=1}^{n} s_{rank_{R_i}^{min}(x)}.$$

Then,

1. *x is a necessary winner for $\mathcal{R}$ w.r.t. F_s iff $S_{\mathcal{R}}^{min}(x) \geq S_{\mathcal{R}}^{max}(y)$ holds for all $y \neq x$;*
2. *x is a possible winner for $\mathcal{R}$ w.r.t. F_s iff $S_{\mathcal{R}}^{max}(x) \geq S_{\mathcal{R}}^{min}(y)$ holds for all $y \neq x$.*

$S_{\mathcal{R}}^{min}(x)$ considers the worst case and $S_{\mathcal{R}}^{max}(x)$ the best case for a scoring value for x. Hence, x is a necessary winner whenever the worst value is higher than the best value and x is a possible winner whenever the best value is higher than the worst value for x. Furthermore, we get the following result:

Corollary 6.1.3 *Possible and necessary winners for positional scoring procedures can be computed in polynomial time.*

Example 37 *Let us consider the following example, where we have candidates $X = \{x_1, x_2, x_3, x_4\}$ and $p + q$ voters. The first group of p voters have the preferences $R_i = \{x_1 > x_2 > x_4, x_1 > x_3 > x_4\}$, for $0 \leq i \leq p$ and the other q voters have the preferences $R_i = \{x_3 > x_2 > x_1\}$, for $p + 1 \leq i \leq p + q$. For the second group of voters, nothing is known about the position of x_4 with respect to other candidates (it is fully incomparable to them all). For the Borda procedure we get:*

	$S_{\mathcal{R}}^{min}$	$S_{\mathcal{R}}^{max}$
x_1	$3p$	$3p + q$
x_2	$p + q$	$2(p + q)$
x_3	$p + 2q$	$2p + 3q$
x_4	0	$3q$

Candidate x_1 is a possible winner whenever $2p \geq q$, x_2 is possible if $2q \geq p$, x_3 is possible if $3q \geq p$, and x_4 is a possible winner if $q \geq p$. Furthermore, candidate x_1 is necessary if $p \geq 3q$; and x_2, x_3 as well as x_4 become never a necessary winner.

For the plurality voting procedure we get

	$S_{\mathcal{R}}^{min}$	$S_{\mathcal{R}}^{max}$
x_1	p	p
x_2	0	0
x_3	0	q
x_4	0	q

Hence, candidate x_1 is always possible, x_2 is never a possible winner, and x_3 as well as x_4 are possible winners if $q \geq p$. x_1 is a necessary winner if $p \geq q$, whereas x_2, x_3 and x_4 become never necessary winners.

6.1.4 Condorcet winners

A candidate x is a *Condorcet winner* for a complete profile $\mathcal{T} = \langle \succeq_1, \ldots, \succeq_n \rangle$ iff for all $y \neq x$, $|\{i \mid x \succ_i y\}| > \frac{n}{2}$, or equivalently, $|\{i \mid x \succ_i y\}| > |\{i \mid y \succ_i x\}|$. In the more general case when indifferences are allowed, this equivalence no longer holds and the latter expression is chosen as the usual definition of a Condorcet winner.

Analogously to positional scoring procedures, we define upper and lower bounds for sets of Condorcet winners in case of partial preference profiles.

Definition 6.1.2 *Let $\mathcal{R}$ be an (incomplete) preference profile. Then,*

- *$x \in X$ is a* necessary Condorcet winner *for $\mathcal{R}$ iff for all $\mathcal{T} \in Ext(\mathcal{R})$, x is a Condorcet winner for $\mathcal{T}$.*
- *$x \in X$ is a* possible Condorcet winner *for $\mathcal{R}$ iff there exists a $\mathcal{T} \in Ext(\mathcal{R})$ such that x is a Condorcet winner for $\mathcal{T}$.*

Again, let us first focus on the worst and the best cases, this time by defining, for a *pair* of candidates (x, y), the number of voters for which x is preferred to y in the worst and in the best cases when considering all complete extensions of $\mathcal{T}$. If $\mathcal{T}$ is a collection of linear orders, let us first define

$$N_{\mathcal{T}}(x, y) = |\{i \mid x >_i y\}| - |\{i \mid y >_i x\}|.$$

Then , we define

$$N^{min}_{\mathcal{R}}(x, y) = \min_{\mathcal{T} \in Ext(\mathcal{R})} N_{\mathcal{T}}(x, y) \text{ and}$$
$$N^{max}_{\mathcal{R}}(x, y) = \max_{\mathcal{T} \in Ext(\mathcal{R})} N_{\mathcal{T}}(x, y).$$

$N^{min}_{\mathcal{R}}(x, y)$ (respectively, $N^{max}_{\mathcal{R}}(x, y)$) corresponds to the worst (respectively, best) case for x among extensions of $\mathcal{R}$. That is, as many as possible candidates are ranked higher than x. On the other hand, $N^{max}_{\mathcal{R}}(x, y)$ considers the best case that is, as many as possible candidates are ranked lower than x. Again, these bounds can be computed in polynomial time as follows:

Theorem 6.1.4 *Let $\mathcal{R}$ be a (partial) preference profile and x, y two distinct candidates from X. We define*

$$N^{max}_{R_i}(x, y) = \begin{cases} +1 & \text{if not } (y \geq_i x); \\ -1 & \text{if } y >_i x \end{cases} \quad \text{and} \quad N^{min}_{R_i}(x, y) = \begin{cases} +1 & \text{if } x >_i y; \\ -1 & \text{if not } (x \geq_i y) \end{cases}$$

Then, we have the following

1. *$N^{min}_{\mathcal{R}}(x, y) = \sum_{i=1}^{n} N^{min}_{R_i}(x, y)$ and $N^{max}_{\mathcal{R}}(x, y) = \sum_{i=1}^{n} N^{max}_{R_i}(x, y)$;*
2. *x is a necessary Condorcet winner for $\mathcal{R}$ iff $\forall y \neq x,\ N^{min}_{\mathcal{R}}(x, y) > 0$;*
3. *x is a possible Condorcet winner for $\mathcal{R}$ iff $\forall y \neq x,\ N^{max}_{\mathcal{R}}(x, y) > 0$.*

If x is strictly preferred to y, then $N^{min}_{\mathcal{R}}(x, y)$ and $N^{max}_{\mathcal{R}}(x, y)$ assign the value 1 as in the case for complete preferences. Furthermore, if candidates x and y are incomparable, the function $N^{max}_{R_i}(x, y)$ assigns the value 1 and $N^{min}_{R_i}(x, y)$ the value -1. This follows the intuition that $N^{max}_{\mathcal{R}}$ covers the "best" case and $N^{min}_{\mathcal{R}}$ the "worst" case for candidates x and y. Hence, if $N^{min}_{\mathcal{R}}(x, y) > 0$ for all $y \neq x$, then in the worst case, strictly more voters prefer x strictly over y than y over x. In this case, x is a necessary Condorcet winner. Whenever $N^{max}_{\mathcal{R}}(x, y) > 0$, there exists a complete extension $\mathcal{T}$ of $\mathcal{R}$ such that x is a Condorcet winner and hence, x is a possible Condorcet winner.

Example 38 *Let us reconsider Example 37.*

We get that x_1 is a necessary Condorcet winner if $p > q$. All other candidates become never a necessary Condorcet winner. Furthermore, x_1 is a possible Condorcet winner if $p > q$ and x_3 and x_4 are possible Condorcet winners if $q > p$. Candidate x_2 is not a possible Condorcet winner.

As stated in Corollary 6.1.3, we can compute necessary and possible Condorcet winners in polynomial time.

Corollary 6.1.5 *Possible and necessary Condorcet winners can be computed in polynomial time.*

One may wonder whether this way of determining possible and necessary winners just by computing lower and upper bounds of scores, which works for scoring procedures and Condorcet winners, extends to Condorcet-consistent voting procedures such as the Simpson or the Copeland procedures [24]. Unfortunately, this is not so simple, as the method consisting in computing lower and upper bounds does not suffice. Consider for instance the Simpson (or *maximin*) procedure, consisting of choosing the candidates maximizing the Simpson score $S_{\mathcal{T}}(x) = \min_{y \neq x} N_{\mathcal{T}}(x, y)$. Then, given a partial preference profile $\mathcal{R}$, we may compute in polynomial time a lower bound $S^{min}_{\mathcal{R}}(x) = \min_{\mathcal{T} \in Ext(\mathcal{R})} S_{\mathcal{T}}(x)$ and an upper bound $S^{max}_{\mathcal{R}}(x) = \max_{\mathcal{T} \in Ext(\mathcal{R})} S_{\mathcal{T}}(x)$. However, even if, for instance, $S^{min}_{\mathcal{R}}(x) > S^{min}_{\mathcal{R}}(y)$ for all y implies that x is a necessary winner, the converse implication is not guaranteed to hold, because it may be the case that no extension of $\mathcal{R}$ simultaneously gives a minimal score to x and a maximal score to y. Furthermore, computing possible and necessary winners for such procedures might be NP-hard and co-NP-hard. This issue is left for further research.

6.1.5 Manipulation and Elicitation

We now investigate the links between possible and necessary winners and some issues such as vote elicitation and manipulation.

6.1.5.1 Manipulation

The Gibbard-Satterthwaite theorem [102, 174] states that any vote procedure can be manipulated, or in other terms, that voters sometimes have an interest to report unsincere preferences. The notion of manipulation was recently revisited from the computational point of view in [49, 48, 51]. Let $J \subseteq I$ be a coalition of voters, $x \in X$ be a candidate, and $\mathcal{R}_{I \setminus J} = \langle R_j \rangle_{j \in I \setminus J}$ be individual profiles of the voters in $I \setminus J$.

- A *constructive manipulation* for x by J given $\mathcal{R}_{I \setminus J}$ (with respect to a given vote procedure F) is a way for the voters in J to cast their votes such that x is guaranteed to win the election, that is, a set of individual profiles $\mathcal{R}_J$ such that $F(\langle \mathcal{R}_{J \setminus I}, \mathcal{R}_J \rangle) = \{x\}$.
- A *destructive manipulation* for x by J given $\mathcal{R}_{I \setminus J}$ (with respect to a given vote procedure F) is a way for the voters in J to cast their votes such that x is guaranteed not to win the election, that is, a set of individual profiles $\mathcal{R}_J$ such that $x \notin F(\langle \mathcal{R}_{J \setminus I}, \mathcal{R}_J \rangle)$.

We then have the following easy results, where $R_\emptyset = \{(x, x) \mid x \in X\}$.

Theorem 6.1.6 *Let F be a voting procedure, $J \subseteq I$ be a coalition of voters, $x \in X$ and $\mathcal{R}_{I \setminus J} = \langle R_j \rangle_{j \in I \setminus J}$. We let $\mathcal{R}^* = \langle R^*_i \rangle_{i \in I}$ where $R^*_i = R_i$ if $i \in I \setminus J$ and $R^*_i = R_\emptyset$ if $i \in J$.*

1. *there is a constructive manipulation for x by J given $\mathcal{R}_{I \setminus J}$ iff $PW_F(\mathcal{R}^*) = \{x\}$;*
2. *there is a destructive manipulation for x by J given $\mathcal{R}_{I \setminus J}$ iff $x \notin NW_F(\mathcal{R}^*)$.*

Thus, deciding whether there is a constructive or a destructive manipulation for a given candidate is a sub-problem of voting with partial preference relations. As an obvious corollary, whenever computing necessary and possible winners is polynomial, then deciding whether there is a (constructive/destructive) manipulation is polynomial as well[3] .

6.1.5.2 Elicitation

Given a set of individual profiles $\mathcal{R}_J = \langle R_j \rangle_{j \in J}$ corresponding to a subset of voters $J \subseteq I$ who have already expressed their votes. *Vote elicitation* [50] consists in determining, whether (a) the outcome of the vote can be determined without needing any further information and (b) which information must be asked to which voter. We generalize these notions to the more general situation where the initial knowledge about the votes is any partial preference profile: given a partial preference profile $\mathcal{R}$, the elicitation task is over iff it is useless to learn more about the voter's preferences that is, the outcome of the vote will be the same in any complete extension of $\mathcal{R}$: for any $\mathcal{T}, \mathcal{T}' \in Ext(\mathcal{R})$, $F(\mathcal{T}) = F(\mathcal{T}')$. This condition is easily shown to be equivalent to the fact that possible and necessary winners coincide:

Theorem 6.1.7 *Given a voting procedure F and a partial preference profile $\mathcal{R}$, the elicitation process is over iff $PW_F(\mathcal{R}) = NW_F(\mathcal{R})$.*

6.1.6 Discussion and Further work

In this section, we made first steps towards computing the outcome of voting procedures when the voters' preferences are incomplete, and we pointed connections to vote manipulation and elicitation.

For the sake of simplicity, we required the voters' preferences to be antisymmetric. However, definitions of possible and necessary winners carry over to the more general case where voters' incomplete preferences are weak orders (allowing for indifferences), provided that the voting procedure F allows for indifferences as well. Especially, possible and necessary Condorcet winners can still be defined, and computed in polynomial time.

Further work obviously includes the investigation of other voting rules, as briefly evoked at the end of Section 6.1.4. Another interesting issue would consist in defining a middle way between possible and necessary winners by counting the number of extensions in which a candidate is a winner. This probabilistic criterion will probably be much harder to compute than the extremely optimistic and pessimistic criteria underlying the notions of possible and necessary winners.

Incompleteness here refers only to *preferences*. Another place where incompleteness may be relevant is *in the voting procedure itself*: this is the way followed by [51], who introduce some uncertainty in the way the voting procedure will be applied so as to make manipulation more difficult. Although both issues are significantly different, it is worth considering whether studying both in a unifying framework would be relevant.

6.2 Voting Theory in Answer Set Programming

In this section, we want to give an encoding of the voting procedures described in Section 6.1. For this, we use aggregate functions as provided by the `DLV` system (cf. Section 2.1.3 on page 9).

Let $X = \{x_1, \ldots, x_m\}$ be a set of candidates and $V = \{v_1, \ldots, v_n\}$ be a set of voters, where each voter has a partial preference profile over the set of candidates. For representing the number of candidates,

[3]Note that the NP-hardness results of [49] do not apply here, since they apply to *weighted* votes.

voters and preference relations, we use the following rules:

(6.1) $c(x_i) \leftarrow$ for all candidates $i = 1, \ldots, m$

(6.2) $v(v_i) \leftarrow$ for all voters $i = 1, \ldots, n$

(6.3) $pref(V, X, Y) \leftarrow$ for all voters V preferring X over Y

Additionally, we need the following rules defining some basic concepts.

(6.4) $\#maxint = 50 \leftarrow$

(6.5) $nbc(Nb) \leftarrow Nb = \#count\{M : c(M)\}$

(6.6) $nbv(Nb) \leftarrow Nb = \#count\{V : v(V)\}$

(6.7) $vtp(V) \leftarrow$ for $V \in \{borda, plurality, condorcet\}$

(6.8) $sp(S) \leftarrow$ for $S \in \{borda, plurality\}$

Rule (6.4) is `DLV` specific. The number of all candidates and voters is computed by rules (6.5) and (6.6). Furthermore, rules (6.7)-(6.8) initialize the voting, respectively scoring, procedures. Since the computation of Borda, plurality, and Condorcet winners are done independently from each other, one can initialize $vtp(.)$ and $sp(.)$ for all voting procedures.

According, to Theorem 6.1.2 on page 116 and 6.1.4 on page 117, we show that candidate x is a possible, respectively necessary, winner if there exists no candidate who "beats" x. In the case of scoring procedures, we have to prove whether there exists no candidate such that $S^{max}(y) > S^{min}(x)$. For the Condorcet procedure, it is enough to compare the candidates y related x with the number of all voters.

(6.9) $possible(VP, X) \leftarrow c(X), vtp(VP), not\ no\_possible(VP, X)$

(6.10) $no\_possible(VP, X) \leftarrow c(X), c(Y), X \neq Y, vtp(VP), sp(VP),$
$s\_max(VP, X, XS), s\_min(VP, Y, YS), XS < YS$

(6.11) $no\_possible(condorcet, X) \leftarrow c(X), c(Y), Y \neq X, nbv(Nbv),$
$Z = \#count\{V : pref(V, Y, X), v(V)\}, Z1 = 2 * Z, Nbv \leq Z1$

(6.12) $necessary(VP, X) \leftarrow c(X), vtp(VP), not\ no\_necessary(VP, X)$

(6.13) $no\_necessary(VP, X) \leftarrow c(X), c(Y), X \neq Y, vtp(VP), sp(VP),$
$s\_min(VP, X, XS), s\_max(VP, Y, YS), XS < YS$

(6.14) $no\_necessary(condorcet, X) \leftarrow c(X), c(Y), Y \neq X, nbv(Nbv),$
$Z = \#count\{V : pref(V, X, Y), v(V)\}, Z1 = 2 * Z, Z1 \leq Nbv$

For computing the Borda score, we use rules (6.15)- (6.18).

(6.15) $borda\_smax(V, X, S) \leftarrow v(V), c(X), S{=}\#count\{Y : pref(V, X, Y), c(Y)\}$

(6.16) $borda\_smin(V, X, S) \leftarrow v(V), c(X), nbc(M),$
$Rk = \#count\{Y : pref(V, Y, X), c(Y)\}, M = Rk + Rk1, Rk1 = S + 1$

(6.17) $s\_min(borda, X, S) \leftarrow c(X), S = \#sum\{Sc, V : borda\_smax(V, X, Sc)\}$

(6.18) $s\_max(borda, X, S) \leftarrow c(X), S = \#sum\{Sc, V : borda\_smin(V, X, Sc)\}$

For the Plurality score, we use rules (6.19)- (6.26).

(6.19) $rmin(V, X, R) \leftarrow v(V), c(X), R1 = \#count\{Y : pref(V, Y, X), c(Y)\}, R = R1 + 1$

(6.20) $rmax(V, X, R) \leftarrow v(V), c(X), nbc(M), R1{=}\#count\{Y : pref(V, X, Y)\}, M = R1 + R$

(6.21) $plural\_smin(V, X, 1) \leftarrow v(V), c(X), rmin(V, X, 1)$

(6.22) $plural\_smin(V, X, 0) \leftarrow v(V), c(X), rmin(V, X, R), R \neq 1$

(6.23) $plural\_smax(V, X, 1) \leftarrow v(V), c(X), rmax(V, X, 1)$

(6.24) $plural\_smax(V, X, 0) \leftarrow v(V), c(X), rmax(V, X, R), R \neq 1$

(6.25) $s\_min(plurality, X, S) \leftarrow c(X), S{=}\#count\{V : plural\_smax(V, X, 1)\}$

(6.26) $s\_max(plurality, X, S) \leftarrow c(X), S{=}\#count\{V : plural\_smin(V, X, 1)\}$

Thus, rules (6.1)- (6.26) represent the encoding of our voting problem within answer set programming. The logic program Π consisting of the rules (6.1)- (6.26) has exactly one answer set, which gives us the set of all possible and necessary winners.

Theorem 6.2.1 *Let X be a set of candidates, V be a set of voters, for each voter let be given partial preference profiles over the set of candidates, and let $VP \in \{Borda, plurality, Condorcet\}$ be a voting procedure.*

Then, the logic program Π, consisting of the rules (6.1)- (6.26), has exactly one answer set Y, where

1. *the set $\{X : possible(VP, X) \in Y\}$ is the set of all possible winners w.r.t. voting procedure VP, and*
2. *the set $\{X : necessary(VP, X) \in Y\}$ is the set of all necessary winner w.r.t. voting procedure VP.*

6.3 Scheduling a meeting

In this section, we consider an application of the voting procedures defined in Section 6.1 and implemented in Section 6.2 within ASP. We consider the problem of scheduling a meeting for a group, where we describe the basic problem in Section 6.3.1. Then, in Section 6.3.2, we include diagnostic reasoning, where a diagnostic model gives us the reasons, whenever no meeting was schedulable. Lastly in Section 6.3.3, we show how efficient and intuitively the voting procedures from Section 6.1 and 6.2 can be used in scheduling problems for the determination of preferred meetings.

6.3.1 Schedule a meeting for a group

We want to describe a solution for the basic problem of scheduling a meeting for a group. We have given m possible times $d_1, \ldots, d_m$ for scheduling a meeting. The group consists of n subgroups $X_1, \ldots, X_n$. Each subgroup X_i has k_i members, where each member may have unavailabilities for certain possible dates. We want to schedule one meeting such that from every group at least k persons are available for that meeting.

Definition 6.3.1 *Let D be a set of dates, $X = X_1, \ldots, X_n$ be a set of groups, where X_i has k_i members, and let NA be a set of unavailabilities $na(p, d)$ expressing that person $p \in X_i$ is unavailable at time $d \in D$. Furthermore, let k be the number of required persons from every subgroup, which at least should attend a meeting.*

Then, we call $\mathcal{M} = \langle D, X, NA, k\rangle$ a meeting scheduling problem*.*

Furthermore, $m \in D$ is called meeting *whenever for all X_i, $1 \leq i \leq n$, we have $|\{p \in X_i : na(p, m) \notin NA\}| \geq k$.*

Let $\mathcal{M} = \langle T, X, NA, k\rangle$ be a meeting scheduling problem. Then, this problem is encoded within answer set programming by the following sets of rules, where k in rule (6.31) presents the number of required persons from each group.

(6.27) $g(X_i) \leftarrow$ for all subgroups X_i of group X

(6.28) $p(P, X_i) \leftarrow$ for all members P of subgroup X_i

(6.29) $d(T) \leftarrow$ for all possible dates T

(6.30) $na(P, T) \leftarrow$ for all P and T, where P is unavailable for time T

(6.31) $rnb(k) \leftarrow$

Furthermore, we include the logic program in Figure 6.1. Rule (6.32) expresses the status of availability

(6.32) $a(P, G, D) \leftarrow not\ na(P, D), p(P, G), g(G), d(D)$

(6.33) $present\_group(G, D) \leftarrow rnb(R), N = \#count\{P : a(P, G, D)\}, N \geq R, g(G), d(D)$

(6.34) $absentgroup(D) \leftarrow d(D), g(G), not\ present\_group(G, D)$

(6.35) $meeting(M) \leftarrow d(M), not\ absentgroup(M)$

(6.36) $meet \leftarrow meeting(M), d(M)$

(6.37) $\leftarrow not\ meet$

Figure 6.1: Meeting Scheduler

of a person P from subgroup G for date D. Rules (6.33) and (6.34) express the status of subgroups. A subgroup G is $present$ for date D whenever the number of available persons is higher than the number of required persons. Rule (6.35) generates possible meetings, where rules (6.36)–(6.37) ensure that at least one meeting is generated. Let $\Pi_{\mathcal{M}}$ be the logic program consisting of rules (6.27)–(6.37). $\Pi_{\mathcal{M}}$ has exactly one answer set if a meeting is schedulable, and $\Pi_{\mathcal{M}}$ has no answer set if no meeting is schedulable. If there exists an answer set X of $\Pi_{\mathcal{M}}$, then every date in the set $\{M : meeting(M) \in X\}$ represents a solution of our meeting scheduling problem.

Theorem 6.3.1 *Let $\mathcal{M} = \langle D, X, NA, k\rangle$ be a meeting scheduling problem and $\Pi_{\mathcal{M}}$ as described in Rules (6.27)-(6.37).*

Then, one of the following holds:

1. *$\Pi_{\mathcal{M}}$ has no answer set iff there exists no $m \in D$ such that for all X_i we have $|\{P \in X_i : P$ is available for $m\}| \geq k$; or*
2. *$\Pi_{\mathcal{M}}$ has exactly one answer set X, where $\{m : meeting(m) \in X\}$ presents all schedulable meetings for the scheduling problem $\mathcal{M}$.*

Example 39 *At a university, we have a group consisting of three subgroups: g_1, g_2 and g_3. Group g_1 has three members (p_1^1, p_2^1, p_3^1), group g_2 has three members (p_1^2, p_2^2, p_3^2), and group g_3 has only two members (p_1^3, p_2^3). They want to meet either on Monday or Tuesday, where a meeting in the morning, in the afternoon, or in the evening is possible. Hence, we have 6 times:*

d_1	*(Monday morning)*	d_4	*(Tuesday morning)*
d_2	*(Monday afternoon)*	d_5	*(Tuesday afternoon)*
d_3	*(Monday evening)*	d_6	*(Tuesday evening)*

Furthermore, we have the following unavailabilities:

Person	*Unavailabilities*	*Date times*
p_1^1	*Monday*	d_1, d_2, d_3
p_2^3	*Tuesday*	d_4, d_4, d_6
p_3^2	*in the mornings*	d_1, d_4
p_2^2	*in the evenings*	d_3, d_6

We want to schedule a meeting, where at least two persons from every subgroup can attend to it. With the unavailabilities at hand, we can schedule a meeting at time d_1, d_2, d_3 that is on Monday. If additionally person p_1^3 becomes unavailable on Monday, no meeting is schedulable.

6.3.2 Including diagnostic reasoning

With a huge number of subgroups, it often happens that no meeting is schedulable since there are excessively different unavailabilities of persons from different subgroups. For this case, we develop a diagnostic model such that the reasons why no meeting is schedulable is determined by the diagnostic model. Furthermore, whenever a meeting is schedulable, the diagnostic model should determine all possible meetings. This idea of including diagnostic reasoning is closely related to the diagnostic model for the configuration of the Debian GNU/Linux system [186, 185]. There, the configuration problem gives suitable combinations of software packages which have to be installed in a Linux system. Software packages may interact in different ways, e.g. they are conflicting with each other or are requiring other software packages. In the case where no suitable configuration of software packages exists, the diagnostic model in [186, 185] determines an error set, a problem set, and an explanation set for analyzing why no configuration of the software packages is possible. The *error set* expresses why no configuration has been found, e.g. required software packages are missing or selected packages are conflicting with each other. The *problem set* contains all software packages which are involved in a conflict. The *explanation set* points out why software packages causing errors are chosen to be in a configuration, e.g. the user has selected them or a package is required by another one.

In the following, we want to apply the diagnostic model for configuration problems to our problem of scheduling a meeting. Let $\mathcal{M} = \langle D, X, NA, k \rangle$ be a meeting scheduling problem. We have the following reasons, why no meeting is schedulable:

(R1) There are no dates for a meeting available.

(R2) For each date there exists at least one subgroup such that one of the following conditions holds:

- **(R2a)** the number of persons from that group is smaller than the number of required persons (without consideration of unavailabilities), or
- **(R2b)** there are too many persons unavailable for that date such that not enough persons can attend that meeting.

Whenever no meeting is schedulable, we want to get the reason. This is made precise in the logic program given in Figure 6.2. Rules (6.38)–(6.42) are the same as in Figure 6.1 except for Rule (6.37), which is replaced by the following rules for the diagnostic output. Rule (6.43) and (6.44) handle reason (R1), whenever no date time is available. Rule (6.45) and (6.46) handle reason (R2a), whenever one subgroup is smaller than the required number of persons. Rule (6.47)–(6.50) handle reason (R2b) where no meeting is schedulable since for every date at least one subgroup is not present due to unavailabilities of group members.

The logic program $\Pi_{\mathcal{M}}^{\mathcal{D}}$ consisting of rules (6.27)–(6.31) and (6.38)–(6.50) is called *diagnostic model* for the problem $\mathcal{M}$ of scheduling a meeting. In contrast to [186], we define only the error set and the explanation set, since the problem set is needed in [186] to detect transitive relationships among conflicting candidates, which have no longer any effect here.

(6.38) $meet \leftarrow d(M), meeting(M)$

(6.39) $meeting(M) \leftarrow d(M), not\ absentgroup(M)$

(6.40) $absentgroup(D) \leftarrow d(D), g(G), not\ present\_group(G, D)$

(6.41) $a(P, G, D) \leftarrow not\ na(P, D), p(P, G), g(G), d(D)$

(6.42) $present\_group(G, D) \leftarrow rnb(R), N = \#count\{P : a(P, G, D)\}, N \geq R, g(G), d(D)$

(6.43) $exists\_date \leftarrow d(D)$

(6.44) $nodate \leftarrow not\ exists\_date$

(6.45) $smallgroup(G) \leftarrow g(G), rnb(R), nbgroup(N, G), N < R$

(6.46) $nbgroup(N, G) \leftarrow N = \#count\{P : p(P, G)\}, g(G)$

(6.47) $absent(G, D) \leftarrow g(G), d(D), not\ present\_group(G, D), not\ meet$

(6.48) $person\_unavailable(P, G, D) \leftarrow absent(G, D), na(P, D), p(P, G), g(G), d(D), not\ meet$

(6.49) $incomplete \leftarrow d(M), not\ h(M), not\ meet$

(6.50) $h(M) \leftarrow d(M), not\ absentgroup(M)$

Figure 6.2: Diagnostic Model of meeting scheduler.

Definition 6.3.2 *Let $\mathcal{M} = \langle D, X, NA, k\rangle$ be a meeting scheduling problem and $\Pi^{\mathcal{D}}_{\mathcal{M}}$ be the corresponding diagnostic model.*

Then, a diagnosis is a triple $D = (X, E_X, R_X)$, where

1. *X is an answer set of Π,*
2. *E_X is the error set $E_X = \{nodate \in X\} \cup \{smallgroup(G) \in X\} \cup \{incomplete \in X\}$*
3. *R_X is the explanation set $R_X = \{person\_unavailable(P, G, D) \in X\} \cup \{absent(G, D) \in X\}$*

The error set gives the reason why no meeting is schedulable. More precisely, it distinguishes the cases where no date time exists ($nodate \in E_X$), one subgroup is smaller than the required number of persons ($smallgroup(G) \in E_X$), or that one subgroup is underrepresented ($incomplete \in E_X$). The explanation set contains information why no meeting is schedulable. Whenever $nodate \in E_X$ or $smallgroup(G) \in E_X$ we need no further explanations since these errors are self-explanatory. Whenever $incomplete \in E_X$, all subgroups G, which are not present for each date time D, are added to the explanation set, and corresponding to that all persons, which are unavailable from these subgroups, are added to the explanation set. Whenever a meeting is schedulable, there exists an answer set X, where the error set E_X and the explanation set R_X are empty. Otherwise, $\Pi^{\mathcal{D}}_{\mathcal{M}}$ has exactly one answer set, where the error set and the explanation sets are non-empty and are explaining why no meeting is schedulable

Theorem 6.3.2 *Let $\mathcal{M} = \langle D, X, NA, k\rangle$ be a meeting scheduling problem and $\Pi^{\mathcal{D}}_{\mathcal{M}}$ be the corresponding diagnostic model.*

Then,

1. *$m \in D$ is a meeting iff $\Pi^{\mathcal{D}}_{\mathcal{M}}$ has an answer set X, where $E_X = \emptyset, R_X = \emptyset$ and $meeting(m) \in X$,*
2. *there exists no meeting iff $\Pi^{\mathcal{D}}_{\mathcal{M}}$ has an answer set X such that $E_X \neq \emptyset$.*

Example 40 *Let us reconsider Example 39. In the last case, where person p^3_1 became unavailable on Monday, no meeting is schedulable.*

In this case, our diagnostic model has one answer set X with a nonempty error set E_X and a nonempty explanation set R_X. Our error set E_X contains the atom incomplete*, denoting the case that at least one subgroup cannot attend the meeting due to unavailabilities. Our explanation set R_X contains the atoms* absent(g3,D) *for all possible times $D \in \{d_1, d_2, d_3, d_4, d_5, d_6\}$. That is, no meeting is schedulable since group $g3$ is not present for each possible date time. Furthermore, the explanation set R_X contains* person_unavailable(p_1^3,g3,D) *for $D = d_1, d_2, d_3$ and* person_unavailable(p_2^3,g3,D) *for $D = d_4, d_5, d_6$. That is, group $g3$ cannot attend to a meeting since person p_1^3 is unavailable on Monday and person p_2^3 is unavailable on Tuesday.*

6.3.3 Selecting preferred meetings

In the case where more than one meeting is schedulable, we have to choose one meeting out of the set of possible meeting times. In order to do so, every person expresses preferences for meeting times. Often, one can express his preferences only as a partial order, e.g. one prefers a meeting in the morning over a meeting in the afternoon, instead of a total order. E.g. one prefers a meeting on Monday morning over Tuesday morning over Monday afternoon. For this reason, we use the voting procedures defined in Section 6.1 to determine upper and lower bounds for preferred meetings w.r.t. different voting strategies

Each person of our group provides his preferred dates. That is, each person (voter) p_k^i has a partial order $<_k^i$ among the set of dates. Since in general not all dates are schedulable as meetings, we have to restrict the partial order $<_k^i$ to the set of schedulable meetings. Furthermore, we assume that each person expresses only preference relations on dates, where he is available.

Let $\mathcal{M} = \langle D, X, NA, k\rangle$ be a meeting scheduling problem. For each person $p_j^i, 1 \leq j \leq k_i$ from subgroup X_i $(i = 1, \ldots, n)$, we have a partial preference relation $<_j^i \subseteq D \times D$ for the set of all dates $D = d_1, \ldots, d_m$. Note that each voter expresses only preference relations on the set of dates available by him. That is, $<_j^i \subseteq D_a \times D_a$ where $D_a = \{d \in D \mid na(p_j^i, d) \notin NA\}$. We call $\mathcal{M}^< = \langle D, X, NA, k, <\rangle$ an *ordered meeting scheduling problem*, where $<$ describes the above given preference relations of each person.

Notice that we have to restrict the partial orders to the set of all schedulable meetings. For this, let $M \subseteq D$ be the set of all schedulable meetings corresponding to the requirements given in Section 6.3.1. Then, we let $\prec_j^i \subseteq M \times M$ such that $d \prec_j^i d'$ holds if $d <_j^i d'$ holds.

We combine the logic programs from the basic meeting scheduling problem in Figure 6.1 and encoding of voting procedures within ASP (Rules (6.1)- (6.26)) from Section 6.2 as follows: We define $\Pi_{\mathcal{M}}^<$ as the union of the rules (6.27)-(6.37), rules (6.4)-(6.26), and the following rules, which replace rules (6.1)-(6.3):

$$c(M) \leftarrow meeting(M), d(M) \tag{6.51}$$

$$v(P) \leftarrow p(P, G), not\ na(P, M), meeting(M), g(G) \tag{6.52}$$

$$pref(P_j^i, X, Y) \leftarrow \text{where } Y <_j^i X \text{ holds for } X, Y \text{ being meetings} \tag{6.53}$$

With this logic program, we can compute possible meetings and necessary meetings. In the case, where meetings are schedulable, possible meetings give a lower bound and necessary meetings give an upper bound for meetings.

Theorem 6.3.3 *Let $\mathcal{M}^< = \langle D, X, NA, k, <\rangle$ be an ordered meeting scheduling problem, $\Pi_{\mathcal{M}}^<$ the corresponding logic program, and let $VP \in \{Borda, plurality, Condorcet\}$ be a voting procedure.*

Then, one of the following holds

1. *$\Pi_{\mathcal{M}}^<$ has either no answer set, expressing that no meetings are schedulable, or*
2. *$\Pi_{\mathcal{M}}^<$ has exactly one answer set Y, where*
   (a) *the set $\{X : possible(VP, X) \in Y\}$ is the set of all possible meetings w.r.t. voting procedure VP, and*

(b) the set $\{X : necessary(VP, X) \in Y\}$ is the set of all necessary meetings w.r.t. voting procedure VP.

Example 41 *Let us reconsider Example 39. Assume that no person of the group has unavailabilities. That is, all dates are schedulable as meetings. Consider the following preference relations: group g_1 prefers Monday morning over all other dates ($d_1 > d_i, i = 2, ..., 6$); g_2 prefers Monday over Tuesday and the afternoon and evening over the morning ($d_2 > d_1 > d_5 > d_4, d_3 > d_1 > d_6 > d_4$); and g_3 prefers Monday morning over Monday afternoon and Monday evening and Monday over Tuesday ($d_1 > d_2, d_1 > d_3$ and $d_i > d_j$ for $i = 2, 3$ and $j = 4, 5, 6$).*

For the Borda voting procedure, we get d_1, d_2, d_3 as possible winners and no necessary winner. For the plurality and for the Condorcet procedure, we get d_1 as possible and as necessary winners. Hence, a meeting should be scheduled on Monday morning.

6.3.4 Conclusions and Further Work

We have linked voting theory to answer set programming for the first time. We have considered the voting procedures for incomplete preference profiles defined in Section 6.1 and implemented in Section 6.2. Furthermore, we have presented the meeting scheduling problem of a group, where we have integrated these voting procedures for computing preferred meetings. First, we have defined the basic problem (Section 6.3.1), then we have included diagnostic reasoning in Section 6.3.2. Whenever no meetings are schedulable, the diagnostic model determines why no solution to a problem exists. In Section 6.3.3, we have integrated voting procedures into our meeting scheduling problem. Voters, members of the group, can express their preferences among a set of dates, and the voting procedures provide possible and necessary preferred meetings. Hence, we have shown the usefulness of voting procedures for incomplete preference profiles within an application. The example presented in Section 6.3.1 is solvable in polynomial time. Although ASP is able to handle complexer problems (e.g. NP-complete ones), we have taken a polynomial problem to demonstrate how efficient voting procedures can be integrated into logic programming. In further work, we want to integrate voting procedures into more complex, NP-complete, timetabling problems. The voting procedure for incomplete preference relations are used as a "filter" for determining preferred solutions. Since they are computable in polynomial time, the complexity of the underlying problem does not increase.

6.4 Abduction and Preferences in Linguistics

In this section, we associate Optimality Theory (OT) [113] with abduction and preference handling within ASP. We want to find preference structures in a linguistic framework as explanations of an abduction problem. A linguistic grammar is a model of the implicit knowledge that guides linguistic behavior. This knowledge is usually conceived as a system of rules and/or well-formedness constraints which determine for a given language which expressions are well-formed and which are not. Language particular knowledge (grammars of individual languages) thereby has to be distinguished from knowledge about languages in general (universal grammar). In the grammar theoretical framework of Optimality Theory, the set of well-formedness constraints is universal, while their importance varies from language to language.

OT constraints are violable and ranked relative to each other. The effect of a constraint violation depends on the rank of the constraint. OT models grammar as a competition of candidate expressions on the constraint hierarchy – relative to a given input which determines what is to be expressed. The candidate that performs best on the constraint hierarchy is the grammatical expression, all others are losers and therefore impossible as expressions for the input. Hence, learning a language can be understood as inferring the underlying constraint ranking from observations.

The linguistic example that we deal with is dialectal variation in the word order possibilities of German 3-verb clusters, as discussed in [178] within the OT framework. An example of a 3-verb cluster is the group of verbs underlined in the German sentence below:

Maria glaubt, dass sie das Lied singen müssen wird.
Maria thinks that she the song sing must will.

Standard German and Swiss German variants differ in their default orders for verb clusters of this type (as well as further non-default ordering possibilities which will briefly be discussed in Section 6.4.3):

	Default verb cluster orders
Standard German:	*singen müssen wird*
Swiss German:	*wird müssen singen*

In the following, we take the perspective of a linguist, and reconstruct the dialectal variation in German as abduction problem: Given the observation of a particular verb order, abduce the underlying constraint ranking.

In Section 6.4.1, we recall the background for the considered linguistic problems. In Section 6.4.2, we reconsider optimal candidates and elaborate on the above example, for which we then give an implementation within ASP in Section 6.4.3. Furthermore, we develop in Section 6.4.3 a new definition for optimal candidates for orders with indifferences. After presenting the results on our example, we draw conclusions and discuss further research issues in Section 6.4.4.

6.4.1 Linguistic Problems

Linguists use observations of two kinds to find out the set of well-formed expressions of a given language. One method is corpus-based, that is, structures which appear more than rarely can be assumed to be well-formed. The other method is explicit elicitation, speakers of a language are asked to give a well-formedness judgment on particular constructed examples.

The task of the linguist is similar to the one of the language learner: figure out the underlying system of constraints for a language, based on observations.

Our example is the verb order in 3-verb clusters of German dialects with the following different ordering strategies for the verbs:

	Default verb cluster orders
Standard German:	*singen müssen wird*
Swiss German:	*wird müssen singen*

The relative order of object noun phrases and their governing verbs does not differ, however: the object precedes the verb in all German dialects, contrary to, e.g., English:

	Default object-verb orders
German:	*ein Lied singen*
English:	*to sing a song*

Syntactic structures are composed recursively by *complementation*. The object, here: "*ein Lied*" is the complement of its governing head, here: the predicative verb "*singen*". This verb phrase, "*ein Lied singen*", is the complement of the modal verb "*müssen*", and this modal verb phrase, in Standard German: "*ein Lied singen müssen*", is the complement of the temporal auxiliary verb, "*werden*". The differences between the languages and variants can now be described in terms of complement-head order:

	Default complement-head orders
Standard German:	All complements precede their heads: "*ein Lied singen müssen wird*"
Swiss German:	Noun complements precede their heads, verbal complements follow them: "*wird müssen ein Lied singen*"
English	All complements follow their heads: "*will have to sing a song*"

The differences between the three languages can be reconstructed within OT using the following three constraints:

H-Comp	A complement follows its head.
Comp-H	A complement precedes its head.
H-VComp	A verbal complement follows its head.

Constraint rankings are indicated with "$\gg$", meaning "has higher priority than". The three rankings that conform to the observations are the following:

Standard German:	$Comp\text{-}H \gg H\text{-}Comp$ ($H\text{-}VComp$)
Swiss German:	$H\text{-}VComp \gg Comp\text{-}H \gg H\text{-}Comp$
English:	$H\text{-}Comp \gg Comp\text{-}H$ ($H\text{-}VComp$)

The exact rank of $H\text{-}VComp$ can only be determined in Swiss German. While its effects are completely subsumed by the high rank of $H\text{-}Comp$ in English, in Standard German all that is necessary is that $Comp\text{-}H$ has highest priority, while the relative order of $H\text{-}Comp$ and $H\text{-}VComp$ is irrelevant because of the low rank of these two constraints. Grammars are usually, but not necessarily, strict total orders of constraints. The rankings given here are only the crucial ones. For those which are left open, any order is compatible with the observations.

6.4.2 Optimal candidates

A linguistic grammar predicts the well-formedness of expressions. OT grammars do so by establishing a competition between different candidate expressions which are evaluated on a hierarchy of well-formedness constraints like $H\text{-}Comp$, $Comp\text{-}H$, and $H\text{-}VComp$. An OT grammar is an input-output mapping. The input defines what is to be expressed. We then have a set of candidate output expressions. The candidates incur different constraint violations. Each candidate is evaluated on the basis of the constraint hierarchy, and the candidate that performs best in this evaluation is the winner, the optimal, hence, grammatical expression.

In the following, we consider the determination of optimal candidates w.r.t. well-formedness of expressions. For this, let $\mathcal{X}$ be a set of candidates (sentences), $\mathcal{C}$ be a set of constraints, $\delta : \mathcal{X} \times \mathcal{C} \to \mathbb{N}$ be a violation function, where $\delta(x, c)$ denotes the degree of violation of $x \in \mathcal{X}$ w.r.t. $c \in \mathcal{C}$, and $\ll$ be a strict total order on $\mathcal{C}$. Then, we call $\mathcal{L} = (\mathcal{X}, \mathcal{C}, \delta, \ll)$ a *linguistic framework*.

In Section 6.4.1, we have taken an example for the dialectal variation in German 3-verb clusters. In the following we will elaborate this example. For the 3-verb cluster with the verbs {*wird, müssen, singen*} ("will, must, sing"), we have the following possible word orders:[4]

Maria glaubt, dass sie das Lied ...

(321) singen müssen wird.	**(231)** müssen singen wird.
(123) wird müssen singen.	**(132)** wird singen müssen.
(312) singen wird müssen.	**(213)** müssen wird singen.

[4]The numbers signal the hierarchical position of the verb. Verb 1 is the temporal auxiliary (*werden*), verb 2 the modal verb (*müssen*), and verb 3 the predicative verb (*singen*).

Our set of candidates $\mathcal{X} = \{321, 231, 123, 132, 312, 213\}$ is constituted by these possible word orders. [5] $\mathcal{C} = \{H\text{-}VComp, Comp\text{-}H, H\text{-}Comp\}$ is our set of constraints. The degree of violation denotes how well a sentence fulfills a constraint. E.g. for constraint $H\text{-}VComp$ the best order is where the auxiliary verb ("wird") precedes the modal verb ("müssen"), which precedes the predicative verb ("singen"). Candidate 123 has this order. Candidate 132 violates this constraint once, since "müssen" comes after "singen", and candidate 321 violates $H\text{-}VComp$ three times, since the verbs are in the reverse order. The violation degrees for all constraints and candidates are given in the Figure 6.3, where the degree of violation is represented by the number of asterisks $*$.

	$H\text{-}VComp$	$Comp\text{-}H$	$H\text{-}Comp$
321	$***$		$**$
231	$**$	$*$	$*$
123		$**$	
132	$*$	$*$	$*$
312	$**$	$*$	$**$
213	$*$	$**$	$*$

Figure 6.3: Constraint Violations

Next, we want to clarify when a sentence is a best candidate w.r.t. a given constraint ranking [13].

Definition 6.4.1 *Let $\mathcal{L} = (\mathcal{X}, \mathcal{C}, \delta, \ll)$ be a linguistic framework, where $\ll$ is a strict total order on $\mathcal{C}$.*

Then, candidate $x \in \mathcal{X}$ is a winner *if there does not exist a candidate $y \in \mathcal{X}, x \neq y$ such that there exists a constraint $c \in \mathcal{C}$ where*

1. *for all $c' \in \mathcal{C}$ where $c \ll c'$ we have that $\delta(x, c') \geq \delta(y, c')$, [6] and*
2. *$\delta(x, c) > \delta(y, c)$.*

Candidate x is a winner, also called *optimal* candidate, if there does not exist another candidate y who is better than x. That is the case when there exists a constraint, where y has a lower degree of violation than x (Condition 2) and when y behaves better for all higher constraints (Condition 1).

In our example, for the constraint order $H\text{-}VComp \gg Comp\text{-}H \gg H\text{-}Comp$ (Swiss German), we get that 123 is a winner. There, for the highest constraint $H\text{-}VComp$, all other candidates have a higher degree of violation and are hence worse than 123. For constraint ranking $Comp\text{-}H \gg H\text{-}VComp \gg H\text{-}Comp$ (observed in Standard German), candidate 321 is a winner, since for the highest constraint $Comp\text{-}H$ all other candidates are worse than 321. Hence, we get the verb cluster *singen müssen wird* as winner for the Standard German dialect and *wird müssen singen* as winner for the Swiss German dialect.

6.4.3 Abduction of constraint rankings

In Optimality theoretic terms, linguists observe that a candidate is determined by a speaker as a *winner*, expressing that the sentence is grammatically correct. The problem is that the observer does not know which underlying constraint ranking the speaker has. Here, abduction comes into play.

Given a linguistic framework $\mathcal{L} = (\mathcal{X}, \mathcal{C}, \delta)$ with an unknown constraint ranking $\ll$ and an observation that candidate $x \in \mathcal{X}$ is a winner, we want to abduce the constraint ranking $\ll$ which explains x. For this,

[5] Due to OT we have to consider all possibilities.

[6] For a better understanding we use the condition $\delta(x, c') \geq \delta(y, c')$ instead of $\delta(x, c') = \delta(y, c')$. Requiring $\delta(x, c') = \delta(y, c')$ instead of $\delta(x, c') \geq \delta(y, c')$ would select the crucial constraint c, which is not requested here, and is also sufficient for characterizing optimal candidates.

our set of hypotheses is the set of all possible (pairwise) constraint rankings. Then, the explanations give us possible strict total orders such that x is optimal. In the following, we give an encoding for this.

The set of candidates is given by rules

(6.54) $cd(x) \leftarrow$ for each $x \in \mathcal{X}$

(6.55) $cst(c) \leftarrow$ for each constraint $c \in \mathcal{C}$

(6.56) $viol(x, c, \delta) \leftarrow$ where δ is the degree of violation of x w.r.t c.

According to Definition 6.4.1, a winner, can be determined by the rules given in Figure 6.4.

(6.57) $winner(X) \leftarrow cd(X), not\ defeated(X)$

(6.58) $defeated(X) \leftarrow cd(X), cd(Y), Y \neq X, better(Y, X)$

(6.59) $better(Y, X) \leftarrow cd(X), cd(Y), Y \neq X, cst(C), wins(Y, X, C), hp(X, Y, C)$

(6.60) $hp(X, Y, C) \leftarrow cd(X), cd(Y), Y \neq X, cst(C), pref(C_1, C), wins(X, Y, C_1)$

(6.61) $wins(X, Y, C) \leftarrow cd(X), cd(Y), cst(C), viol(X, C, NX), viol(Y, C, NY), NX < NY$

(6.62) $pref(X, Z) \leftarrow pref(X, Y), pref(Y, Z)$

(6.63) $\leftarrow pref(C, C), cst(C)$

(6.64) $\leftarrow cst(C_1), cst(C_2), unrkd(C_1, C_2), C_1 \neq C_2$

(6.65) $unrkd(C_1, C_2) \leftarrow not\ pref(C_1, C_2), not\ pref(C_2, C_1), cst(C_1), cst(C_2)$

Figure 6.4: Winner determination

Rules (6.57)–(6.61) encode Definition 6.4.1, where $winner(x)$ is derived whenever x is not defeated, i.e. there exists no candidate $y \neq x$ which is better than x. The relation that y is better than x (Condition 1 and 2 in Definition 6.4.1) is encoded by Rules (6.59)-(6.60). Rule (6.61) only encodes the expression $\delta(x, c) < \delta(y, c)$. Rule (6.62) is inserted for encoding the transitivity relation [7] of preferences and Rules (6.63)–(6.65) ensure that the preference ordering is strict and total. Our logic program Π consists of the rules (6.54)–(6.65). An observation is that exactly one candidate is observed as a winner, but the other candidates are not. Hence, for $x \in \mathcal{X}$ we have

$$O(x) = \left\{ \begin{array}{ll} winner(x) \leftarrow & \text{for } x \in \mathcal{X} \\ defeated(y) \leftarrow & \text{for all } y \in \mathcal{X}, y \neq x \end{array} \right\}$$

This is due by the fact that we want to observe unique optimal winners [13]. Our hypotheses are the set of all possible pairwise preference relations:

$$H = \{pref(c, c') \leftarrow | \ c, c' \in \mathcal{C}, c \neq c'\}.$$

Then, an explanation $\Delta \subseteq H$ for the abduction problem $\langle \Pi, H, O \rangle$ gives us a possible strict total order among the constraints such that $\Pi \cup \Delta$ explains O. More precisely, $\Delta \subseteq H$ is an explanation if $O(x) \subseteq S$ for some $S \in AS(\Pi \cup \Delta)$.

In our linguistic example, we observe that a sentence is a winner and want to abduce possible rankings among the set of constraints. Additionally, we have the background knowledge that constraint $Comp$-H is strictly higher preferred than H-$Comp$. This is due to the observation that object noun phrases precede their governing verbs in all German dialects. Hence, we have additionally in Π the fact

$$pref(comph, hcomp) \leftarrow .$$

[7] Note that this is useful as simplification for formulating the hypotheses.

Then, our hypotheses are

$$H = \left\{ \begin{array}{ll} pref(comph, hvcomp) \leftarrow & pref(hvcomp, hcomp) \leftarrow \\ pref(hvcomp, comph) \leftarrow & pref(hcomp, hvcomp) \leftarrow \end{array} \right\}$$

which gives us together with $pref(comph, hcomp) \leftarrow$ all possible constraint rankings. For computing explanations, we use the `DLV` system [73, 76] with the command `-FD` for abductive diagnosis.

x	Explanations
321	$\Delta_1 = \{Comp\text{-}H \gg H\text{-}Comp \gg H\text{-}VComp\}$ $\Delta_2 = \{Comp\text{-}H \gg H\text{-}VComp \gg H\text{-}Comp\}$
231	no explanations
123	$\Delta = \{H\text{-}VComp \gg Comp\text{-}H \gg H\text{-}Comp\}$
132	no explanations
312	no explanations
213	no explanations

Table 6.1: Explanations for $Comp\text{-}H \gg H\text{-}Comp$ under strict total orders

Table 6.1 lists all possible explanations (strict total orders of the set of constraints) for observation $O(x)$. Whenever a candidate has no explanation, this means that the constraints are not sufficient for explaining these candidates as a winner. In [178], further, more special cases were considered, e.g. where one of the verbs is focused, i.e., stressed. This opens additional ordering possibilities. Experiments for focus-dependent orders follow later in this section. In [178], observations lead to the conclusion that for Standard German $Comp\text{-}H$ is ranked strictly higher than $H\text{-}VComp$. By additional analysis, the authors identified candidate 321 as the winner in Standard German with the underlying constraint ranking $Comp\text{-}H \gg H\text{-}VComp \gg H\text{-}Comp$. This is also found out by the method of abduction, i.e. by explanation Δ_2 for observation $O(321)$. Analogously, candidate 123 is the winner in Swiss German with the underlying constraint ranking $H\text{-}VComp \gg Comp\text{-}H \gg H\text{-}Comp$ as stated by explanation Δ for $O(123)$. Interestingly, $O(321)$ is also explained by another constraint ranking, namely $\Delta_1 = \{Comp\text{-}H \gg H\text{-}Comp \gg H\text{-}VComp\}$. This confirms the fact that the relative ordering of $H\text{-}Comp$ and $H\text{-}VComp$ is irrelevant.

In the example, we have found out that observing candidate 321 yields two explanations, $H\text{-}Comp \gg H\text{-}VComp$ and $H\text{-}VComp \gg H\text{-}Comp$. This supposes that $H\text{-}Comp$ and $H\text{-}VComp$ can be ranked equally. For this reason, we want to abduct total orders (not necessarily strict). Since Definition 6.4.1 is only valid for strict total orders, we have to extend it for total orders.

Definition 6.4.2 *Let $\mathcal{L} = (\mathcal{X}, \mathcal{C}, \delta, \preceq)$ be a linguistic framework, where $\preceq$ is a total order on $\mathcal{C}$.*
Then, candidate $x \in \mathcal{X}$ is a winner *if there exists no $y \neq x$ such that there exists a $c \in \mathcal{C}$ such that*

1. *for all $c' \neq c$ such that $c' \approx c$ or $c' \succ c$ we have $\delta(c', x) \geq \delta(c', y)$, and*
2. *$\delta(c, y) < \delta(c, x)$.*

This definition coincides for strict total orders with Definition 6.4.1. Condition 1 has been extended to handle equally ranked constraints. Now, a candidate y is better than x if there exists a constraint c such that for all equally ranked and for all higher ranked constraints, y behaves at least as good as x (Condition 1) and y is w.r.t. c strictly better than x (Condition 2). Hence, candidate x is a winner if there exists no other candidate which is better than x. One major motivation for allowing equally ranked constraints in Optimality Theory is the observation of non-unique winners. Often, more than one expression is possible for a given input. That two candidates do not differ at a single constraint, is very unlikely. So under strict

total constraint ranking, one should win over the other. Non-unique winners occur if two candidates differ with respect to two constraints where one constraint favors one, and the other constraint the other candidate. These two constraints must be ranked equally.

For example, let x, y, z be candidates, c_1, c_2 be constraints where $c_1 \approx c_2$ and let be given the following constellations for violation degrees:

Case 1:

	c_1	c_2
x	$*$	$**$
y	$**$	$*$

Case 2:

	c_1	c_2
x	$*$	$**$
y	$*$	$*$

Case 3:

	c_1	c_2
x	$**$	
y	$*$	$*$
z	$**$	

In case 1 we get x and y as winners. In case 2 we get only y as a winner, and in case 3 we get x, y, and z as winners.

In Figure 6.5, we give an encoding of Definition 6.4.2 in ASP.

(6.66) $winner(X) \leftarrow cd(X), not\ defeated(X)$

(6.67) $defeated(X) \leftarrow cd(X), cd(Y), Y \neq X, better(Y, X)$

(6.68) $better(Y, X) \leftarrow cd(X), cd(Y), Y \neq X, cst(C), wins(Y, X, C), not\ hp(X, Y, C)$

(6.69) $hp(X, Y, C) \leftarrow cd(X), cd(Y), Y \neq X, cst(C), C \neq C_1, prefeq(C_1, C), wins(X, Y, C_1)$

(6.70) $hp(X, Y, C) \leftarrow cd(X), cd(Y), Y \neq X, cst(C), pref(C_1, C), wins(X, Y, C_1)$

(6.71) $wins(X, Y, C) \leftarrow cd(X), cd(Y), cst(C), viol(X, C, NX), viol(Y, C, NY), NX < NY$

(6.72) $pref(X, Z) \leftarrow pref(X, Y), pref(Y, Z)$

(6.73) $pref(X, Z) \leftarrow pref(X, Y), prefeq(Y, Z)$

(6.74) $pref(X, Z) \leftarrow prefeq(X, Y), pref(Y, Z)$

(6.75) $prefeq(X, Z) \leftarrow prefeq(X, Y), prefeq(Y, Z)$

(6.76) $prefeq(X, Y) \leftarrow prefeq(Y, X)$

(6.77) $\leftarrow pref(C, C), cst(C)$

(6.78) $\leftarrow cst(C_1), cst(C_2), unrkd(C_1, C_2), C_1 \neq C_2$

(6.79) $unrkd(C_1, C_2) \leftarrow cst(C_1), cst(C_2), not\ pref(C_1, C_2), not\ pref(C_2, C_1), not\ prefeq(C_1, C_2)$

Figure 6.5: Winner determination (total pre-order)

Rule (6.66)–(6.71) encode Definition 6.4.2. Rule (6.69) has been added compared to the encoding for Definition 6.4.1, which addresses the handling of equally ranked constraints. Rules (6.72)–(6.79) make, analogously to Rules (6.62)–(6.65), the preference relation, which is transitive and total explicit, where $prefeq(X, Y)$ denotes that the constraints X and Y are ranked equally and $pref(X, Y)$ expresses that x is ranked strictly higher than y. Hence, our logic program Π for our abduction problem now consists of the rules (6.54)–(6.56) and (6.66)–(6.79). Our hypotheses are now the set of all pairwise strict preference relations and the set of all possible indifferences.

$$H = \{pref(c, c') \leftarrow | \ c, c' \in \mathcal{C}, c \neq c'\} \cup \{prefeq(c, c') \leftarrow | \ c, c' \in \mathcal{C}, c \neq c'\}$$

From this set of hypotheses, all possible total orders [8] are constructable. Coming back to our example, Table 6.2 shows us all explanations for observation $O(x)$ when abducing total orders (not necessarily strict) with the given background knowledge $pref(comph, hcomp) \leftarrow$. For candidate 321, we additionally get the supposed explanation that $H\text{-}VComp \approx H\text{-}Comp$.

[8] Note that for total orders indifferences are allowed, whereas for strict total orders no indifferences are allowed.

x	Explanations
321	$\Delta_1 = \{Comp\text{-}H \gg H\text{-}Comp \gg H\text{-}VComp\}$ $\Delta_2 = \{Comp\text{-}H \gg H\text{-}VComp \gg H\text{-}Comp\}$ $\Delta_3 = \{Comp\text{-}H \gg H\text{-}VComp \approx H\text{-}Comp\}$
231	no explanations
123	$\Delta = \{H\text{-}VComp \gg Comp\text{-}H \gg H\text{-}Comp\}$
132	no explanations
312	no explanations
213	no explanations

Table 6.2: Explanations for $Comp\text{-}H \gg H\text{-}Comp$ under total orders

Next, we want to extend our example to stress dependent non-default orders. While 123 is the default order in the St. Gallen (Swiss German) dialect, further orders are possible with particular stress patterns. In particular, order 213 (*müssen wird singen*, "must will sing") is possible with main stress on verb 2, "*müssen*", and order 312 is possible with stress on verb 3, "*singen*". Along with these stress patterns comes a meaning change, contrastive focus. These expressions are only usable for particular purposes, e.g., where one wants to emphasize that she HAD TO sing the song, though she didn't want to (order 213 with stress on verb 2), or that she had to SING the song, though she only wanted to sum it (order 312 with stress on verb 3).

In the following, we concentrate exemplarily on the St. Gallen (Swiss German) dialect. For our 3 verb cluster, the focus can either be on the modal, on the auxiliary, or on the predicative verb. The word order effect of focus is that the focused verb may be leftmost in the verb cluster. This is required by the focus constraints that we use here. Table 6.3 shows the violation degrees for the focus constraints [178], where $Foc(V)$ denotes stress on the predicative verb, $Foc(Mod)$ stress on the modal verb, and $Foc(Aux)$ stress on the auxiliary verb. For focus considerations the constraint $H\text{-}Comp$ is irrelevant. Hence, we only have to

	Foc(V)	Foc(Mod)	Foc(Aux)
321		$*$	$*$
231	$*$		$*$
123	$*$	$*$	
132	$*$	$*$	
312		$*$	$*$
213	$*$		$*$

Table 6.3: Violations of narrow focus for St. Gallen Swiss German

consider the constraints $H\text{-}VComp$, $Comp\text{-}H$, and a constraint $Foc(.)$ which indicates whether the focus is leftmost. Additionally, we have the knowledge that $H\text{-}VComp \gg Comp\text{-}H$ holds, since we are in the Swiss German dialect.

Applying our abduction framework to this case yields the following results: 321, 231, and 132 have no explanations. They cannot be obtained as winners for any focus considered here. 123 has $H\text{-}VComp \gg Foc(i) \approx Comp\text{-}H$ as explanation for all three foci, where $i \in \{V, Mod, Aux\}$. Additionally, 123 has the explanation $Foc(Aux) \approx H\text{-}VComp \gg Comp\text{-}H$ for focus on the auxiliary verb. Candidate 312 has an explanation only for focus on the predicative verb, i.e. $Foc(V) \gg H\text{-}VComp \gg Comp\text{-}H$. Candidate 213 has an explanation only for focus on the modal verb. That is, $Foc(Mod) \gg H\text{-}VComp \gg Comp\text{-}H$.

Instead of abducing preference structures, one can also abduce, which dialect and underlying focus the speaker has. More precisely, we have knowledge about the constraint ranking for the different foci within Standard and Swiss German. Then, we observe a sentence as a winner. The question is then, which

focus and which underlying dialect the native speaker has. The abduction problem $\langle \Pi, H, O \rangle$ is defined as follows: Our hypotheses are the set of matters

$$H = \left\{ \begin{array}{ll} matter(focVSG) \leftarrow & matter(focVStG) \leftarrow \\ matter(focModSG) \leftarrow & matter(focModStG) \leftarrow \\ matter(focAuxSG) \leftarrow & matter(focAuxStG) \leftarrow \end{array} \right\}$$

where $matter(focVSG)$ denotes that the focus is on the verb in Standard German, $matter(focModSG)$ denotes focus on modal verb in Standard German, ..., and $matter(focAuxStG)$ denotes the focus on the auxiliary verb in St. Gallen Swiss German. Π contains information about preference orderings, which depend on stress and the dialect. We have taken for St. Gallen Swiss German the ranking $Foc(i) \gg H\text{-}VComp \gg Comp\text{-}H$, where $i \in \{V, Mod, Aux\}$. Analogously, for Standard German, we have taken the ranking $IF(i) \gg Comp\text{-}H \gg H\text{-}VComp$, where IF ("Ideal Focus") denotes the focus placement constraint that is at work in Standard German, cf. [178] – roughly, it requires focused verbs to be "isolated" from the next higher verb at an edge of the cluster, e.g., 312 is ideal for focus on verb 3, and 132 is ideal for focus on verb 1 or 2. Furthermore, Π contains the slightly modified rules (1)–(9), where the preference relations and the constraints depend on the abduct matter. Again, we have used `DLV` as diagnosis system. Since we are interested in abducing single explanations [9] for this example, we have used the `DLV` option `-FDsingle`.

As an explanation for $O(321)$, we get that the focus is on the auxiliary verb in Standard German, for $O(123)$ focus is on auxiliary in St. Gallen dialect, for $O(132)$ focus on modal in Standard German, for $O(312)$ we get two explanations, one where the focus is on the predicative verb in Standard German, and the other where it is on the predicative verb in St. Gallen German. $O(213)$ is explained by focus on the modal in St. Gallen German. $O(231)$ has no explanations. These results are in line with the empirical findings in [178].

6.4.4 Discussion and Further Work

In this section, we have associated OT with abduction and preference handling. Abduction and preference handling were studied, e.g., in [109] to derive intended conclusions. But, as far as we know, abduction and preferences were not yet linked to optimality theory before.

We have shown that abduction within ASP is a useful knowledge reasoning tool for linguistic problems, here: dialectic studies, where the abduced explanations match the empirical results found out by linguists. ASP has also successfully been used for research in historical linguistics [86].

We have taken the perspective of a linguist and have reconstructed dialectal variation as abduction problems: Given an observation that a sentence is found as grammatically correct (well-formed), abduce the underlying constraint ranking of the dialect. Furthermore, we have provided an encoding (within ASP) for the diagnosis front-end of the `DLV` system.

Regarding linguistic studies, there is an ongoing debate within linguistics about how unique the rule systems of language are in human cognition, as well as in biology in a very broad sense. The reconstruction of grammatical regularities with abduction and preference handling has consequences for this debate: if grammars can be modeled this way, then they share core properties with other non-linguistic rule systems. This supports a position that does not make special assumptions about the nature of linguistic rule systems.

Regarding well-formed expressions, optimal candidates were defined only for strict total orders of the underlying constraints before [13]. In this work, we have extended the definition for optimal candidates to total orders, where it is allowed that constraints can be ranked equally. We want to note that there are several possibilities to model the determination of winners in case of equally ranked constraints. Another possibility would be to "count" all constraint violations of candidates which are ranked equally and then apply Definition 6.4.1 where equally ranked constraints are understand as one constraint. But this may lead

[9] An explanation Δ is *single*, if $|\Delta| = 1$ holds.

to unintuitive results. For example, let x, y, z be candidates, c_1, c_2 be constraints where $c_1 \approx c_2$ and let be given the following violation degrees:

	c_1	c_2
x	$***$	
y	$*$	$*$
z	$***$	

Definition 6.4.2 yields x, y and z as optimal. Counting the violation degrees of all equally ranked constraints would lead to y as a winner. Then, x and z are not optimal candidates which is not desired.

We have abduced constraint orderings for dialects in German. Furthermore, we have exemplarily studied abduction for focus dependent orders. The results confirm the empirical findings. OT can thus successfully be modeled with abduction and preference handling. For the practicing linguist, this tool can be quite helpful. When considering a linguistic phenomenon, one usually starts with exploring it within a few languages. A constraint set is hypothesized that is held responsible for the observed patterns. However, as it is one premise of OT that *any* constraint ranking is a possible grammar, it is not enough to show that a set of constraints can be used to explain a particular phenomenon in one language. All possible constraint rankings and all possible candidates have to be considered. Every OT analysis leads to a proposal about *possible* languages. The calculation of these predictions can easily go beyond what one can handle without computational tools. The abductive framework introduced here can therefore aid the evaluation of linguistic theories.

An interesting further research topic is to study the outcome of observing non-unique optimal candidates. That is, we have observations like $winner(x)$, where it is possible that other candidates $y \neq x$ can also be obtained optimal. First experiments have shown that this leads to different results for the focus contexts. There, it happens that for special rankings among the constraints more than one candidate is observed as a winner. Furthermore, candidate 132 can be explained in dialect considerations, cf. Table 6.2, when we consider non-unique optimal candidates. We will leave it to further linguistic studies to interpret these behaviors. Another line would be to observe candidates as non-optimal. That is, the explanations give possible constraint rankings such that a candidate could not be a winner. Also, one can study the abduction of partial preference relations, which has not been considered in this paper. Although optimality theory excludes partial orders, we can ask here whether partial orders (not total) may lead to "optimal" candidates. More precisely, under which conditions and for which examples, is a partial order among constraints an explanation, but no total extension of that partial ordering leads to an explanation for observing a candidate as winner. For example, candidate 132 violates each of our word order constraints once and represents a kind of "compromise candidate", which could be obtained by partial orders. Interestingly, this candidate can be observed in German dialects very frequently, it even appears to be a second default order sometimes. In [178], this is captured by assuming additional constraints. Introducing new constraints is a delicate issue, as these require independent substantial justification which is not always easy to find. An explanation of the occurrence of the 132 order as default with partial constraint ranking would have the advantage of avoiding this consequence.

6.5 Summary

In this chapter, we have considered two new applications of preference handling within answer set programming.

First, we have linked voting theory to ASP. Referring to this, we have made in Section 6.1 first steps towards computing the outcome of voting procedures when the voter's preferences are incomplete. We have introduced natural notions of possible and necessary winners for partial preference profiles w.r.t. different voting procedures. Furthermore, we have shown that for positional scoring procedures and for the

Condorcet procedure, possible and necessary winners can be computed in polynomial time by very simple algorithms. Additionally, we pointed connections to vote manipulation and elicitation.

Afterwards in Section 6.2, we have given an encoding of these new voting procedures within answer set programming. This encoding was integrated in Section 6.3 into the problem of scheduling a meeting, whereas scheduling problems are well-known applications for answer set programming. For example, in [91] an encoding of school timetabling within answer set programming has been presented, where preferences are presented as weak constraints which induce a priority level among the set of possible timetables. In [179], an agent-based meeting scheduling system was developed that can automate the task of scheduling meetings between groups of users by interacting with the users. They have used techniques from voting theory to arrive at consensus choices for meetings while balancing different preferences, which are presented as weights. A more complex overview of timetabling algorithms is given in [175].

After we have described the basic problem of scheduling a group meeting in Section 6.3, we have included for the first time diagnostic reasoning into the scheduling problem. The introduced diagnostic model gives us explanations, whenever no meeting was schedulable, e.g. if not enough attendees are available due to conflicting unavailabilities. Finally, we have integrated the voting procedures to compute preferred meetings, whenever several meetings are schedulable. For this, the voter, attendees of the meeting, can express partial preferences among the set of schedulable dates, which seems to be very natural, e.g. one prefers a meeting on Monday over Tuesday without given any time restrictions. Hence, the voting procedures defined in Section 6.1 seem to be predestinated for this application.

As a second new application of answer set programming, we have associated in Section 6.4 Optimality Theory with abduction and preference handling. We have shown that abduction within ASP is a useful knowledge reasoning tool for linguistic problems, namely dialect studies. We have taken the perspective of a linguist and have reconstructed dialectal variation as abduction problem: Given an observation that a sentence is found as grammatically correct (well-formed), abduce the underlying constraint ranking of the dialect. Furthermore, we have provided an encoding (within ASP) for the diagnosis front-end of the `DLV` system. Before [13], optimal candidates were defined regarding well-formed expressions only for strict total orders of the underlying constraints. In this work, we have, additionally, extended the definition for optimal candidates to total orders with indifferences. Regarding linguistic studies, there is an ongoing debate within linguistics about how unique the rule systems of language are in human cognition, as well as in biology in a very broad sense. The reconstruction of grammatical regularities with abduction and preference handling has consequences for this debate: if grammars can be modeled this way, then they share core properties with other non-linguistic rule systems. This supports a position that does not make special assumptions about the nature of linguistic rule systems.

There is a huge number of further applications of preferences within answer set programming, e.g. for the decision support system of the Space Shuttle (USA-Advisor) [7], in information cite selection [79], auctioning, configuration, revision programming [160], preference grammars used by US Postal Service to standardize postal addresses [56].

Chapter 7

Concluding remarks

In this thesis, we have concentrated on preferences within answer set programming. At the beginning we had the following questions:

- Whether and how can preferences be integrated into an existing ASP solver?
- Which method for the computation of preferred answer sets is better?
- How should logic programs with preferences be handled by optimization methods?

We have answered these questions as follows: First, we have shown that preference information can be integrated into an answer set solver. Among several formalisms and semantics for preference handling within ASP, we have chosen ordered logic programs with the underlying D-, W-, and B- semantics. To provide a base for the integration of preference information , we have developed a graph-based framework for the computation of answer sets of logic programs. Then, we have extended this framework by preferences, such that preferred answer sets are directly computed by this graph-based approach. Second, we have compared our new method for computing preferred answer sets with existing approaches. It turned out that the integrative method performs better on most considered problem classes than the other existing approaches. Third, we have defined and studied novel notions of equivalences for ordered logic programs, where we have presented several program transformations for ordered logic programs. Additionally, we have presented two new applications for preference handling within answer set programming. The detailed description of our contributions is as follows:

In Chapter 3, we have elaborated upon rule dependency graphs (*RDG*s) and their colorings for characterizing and computing answer sets of logic programs. The general idea is to start from an uncolored *RDG* and to employ specific operators that turn a partially colored graph gradually in a totally colored one, finally representing an answer set. To this end, we have developed a variety of deterministic and nondeterministic operators. Different coloring sequences are obtained by selecting different combinations of operators. In particular, we have identified the basic strategies employed by the `noMoRe` system. The goal of this framework is to offer an intermediate stage between declarative characterizations of answer sets and corresponding algorithmic specifications. We believe that this greatly facilitates the formal elaboration of computational approaches. In fact, our operational framework can be seen as a "theoretical toolbox" that allows for assembling specific strategies for answer set formation. This operational framework establishes the basics for the `nomore++` system [152, 3, 2], the successor of the `noMoRe` system.

In Chapter 4, we have incorporated preferences as a third type of edges of rule dependency graphs. Among several approaches for adding preference information to answer set programming, we have considered three semantics, interpreting preferences as inducing a selection function among the answer sets of the underlying program [34, 62, 194]. We have shown that these selection functions can be characterized by graph-oriented methods in a uniform way and how this can be realized by means of an operational

semantics. Furthermore, we have extended the deterministic and non-deterministic operations from the operational framework for graph-based computation of answer sets (cf. Chapter 3) by preferences for one preference semantics. This is done through the restriction of propagation and choice operations to those rules that are not dominated by any preferred rules whose application status is indeterminate. Hence, we have answered the first question from the introduction (see on page 2):

"How can preferences be integrated into an existing ASP solver?"

We have shown that it is possible to integrate preference information into an ASP solver and we have introduced a framework for this integrative approach. Furthermore, we have presented a C++ implementation of this framework including experimental evaluations for which we have defined novel benchmarks for logic programs with preferences. Up to now, the considered preference approaches have either been implemented by meta-interpretation [77] or by pre-compilation front-ends [62]. In contrast to that, our operational framework characterizes the integration of preference information into an ASP solver. First experimental evaluations have show that the integrative method for computing preferred answer sets seems to be better than meta-interpretation or a pre-compilation front-end. Thus, we have answered the second question from the introduction (see on page 2):

"Which method for the computation of preferred answer sets of ordered logic programs is better; the integration of preferences into an ASP solver or a compilation or meta-interpretation method for preferences?"

Our first studies have shown that the integrative approach for preference handling seems to be good method for computing preferred answer sets.

In Chapter 5 we have concentrated on notions of equivalence for logic programs with preferences, which have never been studied before. In analogy to strong equivalence for (normal) logic programs, we have defined strong order and n-strong order equivalence for ordered logic programs. While strong order equivalence considers all admissible extensions by ordered programs, the more liberal notion of n-strong order equivalence considers only extensions of ordered programs by normal logic programs. For both notions of equivalence we have studied characterizations concerning three semantics for preference handling, namely the D-, W-, and B- semantics. We have shown that two ordered programs are strongly $<^W$- and $<^D$-equivalent, if and only if their preference relations are equal, their standard ASP programs are strongly equivalent, and their "non-looping" generating rules are identical for all answer sets of any extension of them. For B-preference, also the "looping" generating rules must be identical in any answer set. In contrast to strong order equivalence, whenever two ordered programs are n-strongly order equivalent their underlying logic programs have to be strong equivalent, but they can differ on their preference relations and on their generating rules, i.e. rules contributing to answer sets. Hence, we are able to answer the third question from the introduction (see on page 2):

"How should ordered logic programs be handled by optimization methods? Under which conditions can we simplify ordered logic programs and their underlying preferences?".

For strong order equivalence no simplification of preference relations is possible, but under n-strong order equivalence we can simplify preference relation. Referring to this, we have defined several transformations under n-strong order equivalence, which remove redundant preference relations. Additionally, we have analyzed program simplifications known from strong equivalence for normal logic programs. Furthermore, we have studied the computational complexity of the main decision problems, in particular deciding whether two programs are order equivalent, whether two programs are strongly order equivalent, and whether two programs are n-strongly order equivalent. It turned out that all three issues are co-NP-complete and thus they are neither harder nor simpler than equivalence and strong equivalence for normal logic programs. Finally, we have studied the relationship among the considered preference semantics under strong order and n-strong order equivalence. It turned out that the relationship between the three preference semantics

is totally changing under the considered notions of order equivalence. Further work will focus on other notions of order equivalence. Instead of considering *all* possible extension of ordered programs by ordered ones, we will consider extensions by ordered programs, which do not interfere into the preference relation of the original program.

In Chapter 6, we have considered two new applications of preference handling within answer set programming. First, we have linked voting theory with ASP, which has to the best of our knowledge never been done before. We have defined new voting procedure to handle partial preference relations within automated group decision making processes. We have integrated these voting procedures into the problem of scheduling a group meeting, where we have also included diagnostic reasoning into the problem. As a second new application of answer set programming, we have associated optimality theory with abduction and preference handling. We have shown that abduction within ASP is a useful knowledge reasoning tool for linguistic problems, namely dialect studies. We have taken the perspective of a linguist and have reconstructed dialectal variation as abduction problem: Given an observation that a sentence is found as grammatically correct (well-formed), abduct the underlying constraint ranking of the dialect. Regarding linguistic studies, there is an ongoing debate within linguistics about how unique the rule systems of language are in human cognition, as well as in biology in a very broad sense. The reconstruction of grammatical regularities with abduction and preference handling has consequences for this debate: if grammars can be modeled this way, then they share core properties with other non-linguistic rule systems. This supports a position that does not make special assumptions about the nature of linguistic rule systems.

Appendix A

Chapter 3

A.1 Auxiliary results

In this section we want to provide theorems which are needed in the proofs. The first theorem gives an characterization of answer sets in terms of generating rules which corresponds to $C_{\oplus}$ in our approach.

Theorem A.1.1 *Let Γ be the RDG of logic program Π, let X be a set of atoms, and let C be a partial coloring of Γ. Then, $X \in AS_\Pi(C)$ iff $(R_\Pi(X), \Pi \setminus R_\Pi(X)) \in AC_\Pi(C)$.*

Proof A.1.1 This follows directly from the definition of $AS_\Pi(C)$ and $AC_\Pi(C)$. ■

Theorem A.1.2 *Let Π be a logic program and X be a set of atoms. Then, X is an answer set of Π iff $X = Cn((R_\Pi(X))^\emptyset)$.*

Proof A.1.2 Let X be an answer set and let Π be a logic program. The definition of answer sets states

$$X \text{ is an answer set of } \Pi \text{ iff } Cn(\Pi^X) = X. \tag{A.1}$$

We know that for positive logic programs Π^X and $(R_\Pi(X))^\emptyset$ we have:

$$\bigcup_{i\geq 0} T^i_{\Pi^X}(\emptyset) = Cn(\Pi^X) \text{ and} \tag{A.2}$$

$$\bigcup_{i\geq 0} T^i_{(R_\Pi(X))^\emptyset}(\emptyset) = Cn((R_\Pi(X))^\emptyset). \tag{A.3}$$

"⇒" Now let X be an answer set. Then we have

$$T^i_{\Pi^X}(\emptyset) \subseteq X. \tag{A.4}$$

for all $i \geq 0$ because of (A.1) and (A.2). With (A.1) it is sufficient to show $Cn((R_\Pi(X))^\emptyset) = Cn(\Pi^X)$.

We want to prove the equation

$$\bigcup_{i\geq 0} T^i_{\Pi^X}(\emptyset) = \bigcup_{i\geq 0} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$$

by induction over i. With this equation and using (A.2) and (A.3) the statement $X = Cn((R_\Pi(X))^\emptyset)$ is proven. For $i = 0$ we have: $T^0_{\Pi^X}(\emptyset) = \emptyset = T^0_{(R_\Pi(X))^\emptyset}(\emptyset)$. Now let $i = 1$. Then we have:

$$\begin{array}{rcll} T^1_{\Pi^X}(\emptyset) & = & T_{\Pi^X}(\emptyset) & \\ & = & \{head(r) : r \in \Pi^X, body(r) \subseteq \emptyset\} & \text{(by (2.2))} \\ & = & \{head(r) : r \in \Pi, & \\ & & body^-(r) \cap X = \emptyset, body^+(r) \subseteq \emptyset\} & \text{(by (2.1))} \\ & = & \{head(r) : r \in (R_\Pi(X))^\emptyset, body(r) = \emptyset\} & \text{(by } (body^+(r) = \emptyset)) \\ & = & T^1_{(R_\Pi(X))^\emptyset}(\emptyset) & \text{(by (2.2))} \end{array}$$

Assume that we have the induction hypothesis (IH) $T^i_{\Pi^X}(\emptyset) = T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$. For $i + 1$ we have:

$$\begin{array}{rcll} T^{i+1}_{\Pi^X}(\emptyset) & = & T_{\Pi^X}(T^i_{\Pi^X}(\emptyset)) & \\ & = & \{head(r) : r \in \Pi^X, body(r) \subseteq T^i_{\Pi^X}(\emptyset)\} & \text{(by (2.2))} \\ & = & \{head(r) : r \in \Pi, & \\ & & body^-(r) \cap X = \emptyset, body^+(r) \subseteq T^i_{\Pi^X}(\emptyset)\} & \text{(by (2.1))} \\ T^{i+1}_{(R_\Pi(X))^\emptyset}(\emptyset) & = & T_{(R_\Pi(X))^\emptyset}(T^i_{(R_\Pi(X))^\emptyset}(\emptyset)) & \\ & = & \{head(r) : r \in (R_\Pi(X))^\emptyset, body(r) \subseteq T^i_{(R_\Pi(X))^\emptyset}(\emptyset)\} & \text{(by (2.2))} \\ & = & \{head(r) : r \in \Pi, body^-(r) \cap X = \emptyset, & \\ & & body^+(r) \subseteq X, body^+(r) \subseteq T^i_{(R_\Pi(X))^\emptyset}(\emptyset)\} & \\ & = & \{head(r) : r \in \Pi, body^-(r) \cap X = \emptyset, & \\ & & body^+(r) \subseteq X, body^+(r) \subseteq T^i_{\Pi^X}(\emptyset)\} & \text{(by (IH))} \\ & = & \{head(r) : r \in \Pi, body^-(r) \cap X = \emptyset, & \\ & & body^+(r) \subseteq T^i_{\Pi^X}(\emptyset)\} & \text{(by (A.4))} \end{array}$$

Thus we have proven $X = Cn((R_\Pi(X))^\emptyset)$.

"$\Leftarrow$" Now let $X = Cn((R_\Pi(X))^\emptyset)$. We have to show that X is an answer set. Because of (A.3) we have $T^i_{(R_\Pi(X))^\emptyset}(\emptyset) \subseteq X$ for all $i \geq 0$. Using this equation we can show analogously that $T^i_{\Pi^X}(\emptyset) = T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$ holds for all $i \geq 0$. Therefore, we have $Cn(\Pi^X) = Cn((R_\Pi(X))^\emptyset)$ because of (A.3) and (A.2). Finally, (A.1) gives us that X is an answer set. ■

If we have an answer set, the set of generating rules possesses an enumeration which will provide the support graph of a (colored) *RDG*.

Theorem A.1.3 *Let Π be a logic program and X an answer set of Π. Then, there exists an enumeration $\langle r_i\rangle_{i\in I}$ of $R_\Pi(X)$ such that for all $i \in I$ we have $body^+(r_i) \subseteq head(\{r_j \mid j < i\})$.*

Proof A.1.3 Let X be an answer set of logic program Π. We have to show, that there exists an enumeration $\langle r_i\rangle_{i\in I}$ of $R_\Pi(X)$, such that for all $i \in I$ we have $body^+(r_i) \subseteq head(\{r_j \mid j < i\})$. Since X is an answer set, we know from Theorem A.1.2: $X = Cn((R_\Pi(X))^\emptyset)$. Furthermore we have $Cn((R_\Pi(X))^\emptyset) = \bigcup_{i\geq 0} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$. Thus, we have that $X = \bigcup_{i\geq 0} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$. For this reason, we find an enumeration $\langle x_j\rangle_{j\in J}$ of X such that $x_i \in T^k_{(R_\Pi(X))^\emptyset}(\emptyset)$ and $x_j \in T^l_{(R_\Pi(X))^\emptyset}(\emptyset)$ hold for $i < j$ and some minimal k and l such that $k < l$. This enumeration $\langle x_j\rangle_{j\in J}$ of X gives us an enumeration $\langle r_i\rangle_{i\in I}$ of $R_\Pi(X)$ such that for all $i \in I$ we have $body^+(r_i) \subseteq head(\{r_j \mid j < i\})$. ■

Given an answer set $X \in AS_\Pi(C)$ of a partial coloring we observe that $head(C_\oplus) \subseteq X$. Furthermore, if C is total, the heads of all rules in $C_\oplus$ generates the answer set X.

Theorem A.1.4 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ. Furthermore let $X \in AS_\Pi(C)$. Then, $head(C_\oplus) \subseteq X$. If C is admissible, then $head(C_\oplus) = X$.*

Proof A.1.4 Let Γ be the *RDG* of logic program Π, C be a partial coloring of Γ and let $X \in AS_\Pi(C)$ be an answer set of Π. By definition of X we have $C_\oplus \subseteq R_\Pi(X)$, $C_\ominus \cap R_\Pi(X) = \emptyset$ and $X = Cn((R_\Pi(X))^\emptyset) = \bigcup_{i<\omega} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$. By induction we show that for all $r \in R_\Pi(X)$ we have $head(r) \in \bigcup_{i<\omega} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$. Then, $head(C_\oplus) \subseteq X = \bigcup_{i<\omega} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$, by $head(C_\oplus) \subseteq head(R_\Pi(X))$. By Theorem A.1.3 we have an enumeration $\langle r_i \rangle_{i\in I}$ of $R_\Pi(X)$ such that for all $i \in I$ we have $body^+(r_i) \subseteq head(\{r_j \mid j < i\})$. Let be $I = \{0, \ldots, m\}$ for some $m < \omega$. Clearly, we have $body^+(r_0) = \emptyset \subseteq \bigcup_{i<\omega} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$. Let be $r_k \in R_\Pi(X)$ for $k < m$ and $head(\{r_0, \ldots, r_{k-1}\}) \subseteq \bigcup_{0\leq i\leq l} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$ for some $l < \omega$. We have to show, that $head(r_k) \subseteq \bigcup_{0\leq i\leq l+1} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$ holds. Because we have an enumeration of $R_\Pi(X)$ satisfying Theorem A.1.3 we have $body^+(r_k) \subseteq head(\{r_0, \ldots, r_{k-1}\})$. By Equation 2.2 we have $head(r_k) \subseteq \bigcup_{0\leq i\leq l+1} T^i_{(R_\Pi(X))^\emptyset}(\emptyset)$. Thus, we have $head(C_\oplus) \subseteq head(R_\Pi(X)) \subseteq \bigcup_{i<\omega} T^i_{(R_\Pi(X))^\emptyset}(\emptyset) = X$. Assume C is admissible, then $C_\oplus = R_\Pi(X)$. It remains to show that $X \subseteq head(C_\oplus)$. Let $p \in X$ be some atom. By $X = Cn((R_\Pi(X))^\emptyset)$ there must exists an $r \in R_\Pi(X)$ such that $p = head(r)$. By $C_\oplus = R_\Pi(X)$ we have $r \in C_\oplus$ and thus we have $p \in head(C_\oplus)$. ■

An analog is observed with the set $C_\ominus$. If for some atom q all rules with q as head are in the set $C_\ominus$ then q is not in the answer set $X \in AS_\Pi(C)$.

Theorem A.1.5 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ. Furthermore, let $X \in AS_\Pi(C)$ and $p \in Atm$. If $\{r \in \Pi \mid head(r) = p\} \subseteq C_\ominus$ then $p \notin X$.*

Proof A.1.5 Let Γ be the *RDG* of logic program Π and C be a partial coloring of Γ. Furthermore let $X \in AS_\Pi(C)$. Let C' be a total coloring such that $C \sqsubseteq C'$, $C'_\oplus = R_\Pi(X)$, and $C'_\ominus \cap R_\Pi(X) = \emptyset$ hold. Clearly, such C' must exists because we have an answer set X such that $\{X\} = AS_\Pi(C')$. By Theorem A.1.4 we have $head(C'_\oplus) = X$. Let be $p \in Atm$ such that $\{r \in \Pi \mid head(r) = p\} \subseteq C_\ominus \subseteq C'_\ominus$. Then, $\{r \in \Pi \mid head(r) = p\} \cap C'_\oplus = \emptyset$ and thus we have $head(r) = p \notin X$ for all such $r \in \Pi$. ■

Given an answer set $X \in AS_\Pi(C)$ for a partial coloring where $C_\oplus = R_\Pi(X)$ and $C_\ominus = \emptyset$ then all rules which are blocked are obtained by the operator $\mathcal{P}_\Gamma$.

Theorem A.1.6 *Let Γ be the RDG of logic program Π, X be an answer set of Π and C be a partial coloring of Γ such that $C_\oplus = R_\Pi(X)$ and $C_\ominus = \emptyset$. Furthermore, let C^X be a total coloring of Γ such that $\{X\} = AS_\Pi(C^X)$. Then, $\mathcal{P}_\Gamma(C) = (C_\oplus, C'_\ominus)$ where $B(\Gamma, C^X) \subseteq C'_\ominus$.*

Proof A.1.6 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of logic program Π, X be an answer set of Π and C be a partial coloring of Γ such that $C_\oplus = R_\Pi(X)$ and $C_\ominus = \emptyset$. Furthermore, let C^X be a total coloring of Γ such that $\{X\} = AS_\Pi(C^X)$. We have to show that $\mathcal{P}_\Gamma(C) = (C_\oplus, C'_\ominus)$ holds where $B(\Gamma, C^X) \subseteq C'_\ominus$. We have $\mathcal{P}_\Gamma(C) = (C_\oplus \cup (S(\Gamma, C) \cap \overline{B}(\Gamma, C)), C_\ominus \cup \overline{S}(\Gamma, C) \cup B(\Gamma, C))$. Thus we have to show:

1. $C_\oplus = C_\oplus \cup (S(\Gamma, C) \cap \overline{B}(\Gamma, C))$ and
2. $B(\Gamma, C^X) \subseteq C_\ominus \cup \overline{S}(\Gamma, C) \cup B(\Gamma, C)$.

1: By Theorem 3.1.3 we have $S(\Gamma, C) \cap \overline{B}(\Gamma, C) \subseteq R_\Pi(X) = C_\oplus$.
2: It remains to show that $B(\Gamma, C^X) \subseteq B(\Gamma, C)$. Let be $r \in B(\Gamma, C^X)$. Then there exists an $r' \in C^X_\oplus$ such that $(r', r) \in E_1$. By $C^X_\oplus = R_\Pi(X) = C_\oplus$ we have $r \in B(\Gamma, C)$. ■

The next Theorem about monotonicity of 3-valued interpretations is used in the proofs of Section 3.4.

Theorem A.1.7 *Let Γ be the RDG of logic program Π and C, C' be partial colorings of Γ.*
If $C \sqsubseteq C'$ then $(X_C, Y_C) \subseteq (X_{C'}, Y_{C'})$.

Proof A.1.7 Let Γ be the *RDG* of logic program Π and C, C' be a partial colorings of Γ such that $C \sqsubseteq C'$. Let be $a \in X_C$, then there exists an $r \in C_\oplus$ such that $head(r) = a$. By $C_\oplus \subseteq C'_\oplus$ we have $a \in X_{C'}$. Let be $a \in Y_C$, then for all $r \in \Pi$ such that $head(r) = a$ we have $r \in C_\ominus$. By $C_\ominus \subseteq C'_\ominus$ we have $a \in Y_{C'}$. ■

A.2 Inductive definitions

In this section we want to give the inductive definitions of our operators given in Section 3.3. We write $i < \omega$ for i being a finite natural number greater or equal than 0.

According to $\mathcal{P}^*_\Gamma$, we define $P(C)$ as $P(C) = \bigsqcup_{i<\omega} P^i(C)$ where $P^0(C) = C$ and $P^{i+1}(C) = \mathcal{P}_\Gamma(P^i(C))$ for $i < \omega$. Clearly, $P^i(C) \sqsubseteq P^{i+1}(C)$ for all $i < \omega$. By using $\mathcal{P}_\Gamma$ in every iteration step, we have that $P^i(C)$ is always a partial coloring for $i < \omega$. Note that $P(C)$ not always exists. To see this, observe that $P^1(\{a \leftarrow not\ a\}, \emptyset)$ would be $(\{a \leftarrow not\ a\}, \{a \leftarrow not\ a\})$ which is not a partial coloring.

Theorem A.2.1 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ. Then,*

1. *if $AC_\Pi(C) \neq \emptyset$ then $P(C)$ exists,*
2. *$P(C)$ is a partial coloring,*
3. *$AC_\Pi(C) = AC_\Pi(P(C))$ and for all $i < \omega$ we have $X \in AS_\Pi(C)$ iff $X \in AS_\Pi(P^i(C))$ iff $X \in AS_\Pi(P(C))$,*
4. *$C \sqsubseteq P(C)$,*
5. *$P(C)$ closed under $\mathcal{P}_\Gamma$,*
6. *$P(C)$ is the $\sqsubseteq$-smallest partial coloring closed under $\mathcal{P}_\Gamma$, and*
7. *$P(C) = \mathcal{P}^*_\Gamma(C)$.*

Proof A.2.1 Let Γ be the *RDG* of logic program Π and C be a partial coloring of Γ.
1: The existence of $P(C) = \bigsqcup_{i<\omega} P^i(C)$ follows from Theorem 3.3.1 and from property 3 in this theorem by induction over i.
2: By definition of $P(C)$ and Definition 3.3.1, $P(C)$ is a partial coloring, because $P(C)$ only operates on partial colorings.
3: By Theorem A.1.1 it remains to show that $AS_\Pi(C) = AS_\Pi(P(C))$. By $C \sqsubseteq P^i(C)$ we have, if $X \in AS_\Pi(P^i(C))$ then $X \in AS_\Pi(C)$ for all $i < \omega$. Furthermore, if $X \in AS_\Pi(P(C))$ then $X \in AS_\Pi(P^i(C))$ by $P^i(C) \sqsubseteq P(C)$ for all $i < w$. Let $X \in AS_\Pi(C)$. We prove by induction over i that $X \in AS_\Pi(P^i(C))$ for all $i < \omega$ and thus $X \in AS_\Pi(P(C))$. For $i = 0$ we have $P^0(C) = C$ and thus by definition of C, $X \in AS_\Pi(P^0(C))$. Assume, $X \in AS_\Pi(P^k(C))$ for all $0 \leq k \leq i$ for some $i < \omega$. We have to show that $X \in AS_\Pi(P^{i+1}(C))$. Abbreviatory we write C' instead of $P^i(C)$. We have $C'_\oplus \subseteq R_\Pi(X)$ and $C'_\ominus \cap R_\Pi(X) = \emptyset$. By

$$P^{i+1}(C) = \mathcal{P}_\Gamma(C') = (C'_\oplus \cup (S(\Gamma, C') \cap \overline{B}(\Gamma, C')), C'_\ominus \cup \overline{S}(\Gamma, C') \cup B(\Gamma, C')),$$

it remains to show, that $S(\Gamma, C') \cap \overline{B}(\Gamma, C') \subseteq R_\Pi(X)$ and $(\overline{S}(\Gamma, C') \cup B(\Gamma, C')) \cap R_\Pi(X) = \emptyset$. But this holds by Theorem 3.1.3. Thus, $X \in AS_\Pi(P^{i+1}(C))$.
4: $C = P^0(C) \sqsubseteq P(C)$ holds by definition of $P^0(C)$ and $P^{i+1}(C)$ for all $i < \omega$.
5: We have to show, that $\mathcal{P}_\Gamma(P(C)) = P(C)$. By finiteness there exists an $n < \omega$ such that $P(C) = P^n(C)$ and $P^n(C) = P^{n+1}(C)$. Thus, we have to show $\mathcal{P}_\Gamma(P^n(C)) = P(C)$. By definition of $P(C)$ we have $\mathcal{P}_\Gamma(P^n(C)) = P^{n+1}(C) \sqsubseteq P(C)$. It remains to show, that $P(C) = P^n(C) \sqsubseteq \mathcal{P}_\Gamma(P^n(C))$. But this holds by definition of $\mathcal{P}_\Gamma$. Thus, $\mathcal{P}_\Gamma(P(C)) = P(C)$.
6: We have to show that $P(C)$ is the $\sqsubseteq$-smallest partial coloring closed under $\mathcal{P}_\Gamma$. Assume there exists a $Q(C) \neq P(C)$ such that $Q(C) \sqsubseteq P(C)$ and $Q(C)$ is a partial coloring closed under $\mathcal{P}_\Gamma$. Given $Q^0(C) = C = P^0(C)$, there must exists a (minimal) $i < \omega$ such that $Q^{i+1}(C) \neq P^{i+1}(C)$ and $Q^j(C) = P^j(C)$ for all $j \leq i$. But then $Q^{i+1}(C) = \mathcal{P}_\Gamma(Q^i(C)) \neq \mathcal{P}_\Gamma(P^i(C)) = P^{i+1}(C)$. By $Q^i(C) = P^i(C)$, we have $\mathcal{P}_\Gamma(Q^i(C)) = \mathcal{P}_\Gamma(P^i(C))$ and thus $Q^{i+1}(C) = P^{i+1}(C)$ by definition of $\mathcal{P}_\Gamma$. This is a contradiction. For

this reason, $P(C)$ is the $\sqsubseteq$-smallest partial coloring closed under $\mathcal{P}_\Gamma$.
7: This follows directly from conditions 4–6 and by definition of $\mathcal{P}^*_\Gamma(C)$. ■

According to $\mathcal{T}^*_\Gamma$, we define $T(C)$ as $T(C) = \bigsqcup_{i<\omega} T^i(C)$ where $T^0(C) = C$ and $T^{i+1}(C) = \mathcal{T}_\Gamma(T^i(C))$ for $i < \omega$. Clearly, $T^i(C) \sqsubseteq T^{i+1}(C)$ for all $i < \omega$.

Theorem A.2.2 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ. Then,*

1. *$T(C)$ is a partial coloring,*
2. *$C \sqsubseteq T(C)$,*
3. *$T(C)$ closed under $\mathcal{T}_\Gamma$,*
4. *$T(C)$ is the $\sqsubseteq$-smallest partial coloring closed under $\mathcal{T}_\Gamma$, and*
5. *$T(C) = \mathcal{T}^*_\Gamma(C)$.*

Proof A.2.2 Let Γ be the *RDG* of logic program Π and C be a partial coloring of Γ.
1: We prove by induction over i that $T^i(C)$ is a partial coloring for all $i < \omega$. Then, $T(C)$ is a partial coloring. For $i = 0$ we have $T^0(C) = C$ is a partial coloring. Assume that $T^k(C)$ is a partial coloring for all $0 \leq k \leq i$ for some $i < \omega$. We have to show that $T^{i+1}(C)$ is a partial coloring. We have

$$T^{i+1}(C) = \mathcal{T}_\Gamma(T^i(C)) = (T^i(C)_\oplus \cup (S(\Gamma, T^i(C)) \setminus T^i(C)_\ominus), T^i(C)_\ominus).$$

But this is clearly a partial coloring because $T^i(C)$ is a partial coloring.
2-5: Hold analogous to Theorem A.2.1. ■

According to $(\mathcal{PU})^*_\Gamma$, we define $PU(C) = \bigcup_{i<\omega} PU^i(C)$ where $PU^0(C) = C$ and $PU^{i+1}(C) = \mathcal{U}_\Gamma(\mathcal{P}_\Gamma(PU^i(C)))$ for $i < \omega$. Clearly, $PU^i(C) \sqsubseteq PU^{i+1}(C)$ for all $i < \omega$.

Theorem A.2.3 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ. Then,*

1. *If $AC_\Pi(C) \neq \emptyset$ then $PU(C)$ exists,*
2. *$PU(C)$ is a partial coloring,*
3. *$AC_\Pi(C) = AC_\Pi(PU(C))$ and for all $i < \omega$ we have $X \in AS_\Pi(C)$ iff $X \in AS_\Pi(PU^i(C))$ iff $X \in AS_\Pi(PU(C))$,*
4. *$C \sqsubseteq PU(C)$,*
5. *$PU(C)$ closed under $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$,*
6. *$PU(C)$ is the $\sqsubseteq$-smallest partial coloring closed under $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$, and*
7. *$PU(C) = (\mathcal{PU})^*_\Gamma(C)$.*

Proof A.2.3 Let Γ be the *RDG* of logic program Π and C be a partial coloring of Γ.
1: The existence of $PU(C) = \bigcup_{i<\omega} PU^i(C)$ follows inductively by theorems 3.3.1 and 3.2.2 (existence of maximal support graphs) and by property 3 in this theorem.
2: This could be proven analogous to Theorem A.2.1 by the existence of $\mathcal{U}_\Gamma$ in each inductive step for defining $PU(C)$.
3: By Theorem A.1.1 it remains to show that $AS_\Pi(C) = AS_\Pi(PU(C))$. By $PU(C) \sqsupseteq PU^i(C) \sqsupseteq C$ for all $i < \omega$ we have if $X \in AS_\Pi(PU(C))$ then $X \in AS_\Pi(PU^i(C))$ and then $X \in AS_\Pi(C)$ for all $i < \omega$. Let $X \in AS_\Pi(C)$. We prove by induction over i that $X \in AS_\Pi(PU^i(C))$ for all $i < \omega$. Then, $X \in AS_\Pi(PU(C))$. For $i = 0$ we have $PU^0(C) = C$ and thus $X \in AS_\Pi(PU^0(C))$. Assume, $X \in AS_\Pi(PU^k(C))$ for all $0 \leq k \leq i$ for some $i < \omega$. Then, $R_\Pi(X) \supseteq PU^i(C)_\oplus$ and $R_\Pi(X) \cap PU^i(C)_\ominus = \emptyset$. Now, we prove that $X \in AS_\Pi(PU^{i+1}(C))$. We have to show that

(a) $R_\Pi(X) \subseteq PU^{i+1}(C)_\oplus$ and

(b) $R_\Pi(X) \cap PU^{i+1}(C)_\ominus = \emptyset$.

(a): We have $PU^{i+1}(C)_\oplus = \mathcal{U}_\Gamma(\mathcal{P}_\Gamma(PU^i(C)))_\oplus$. By Thm. A.2.1 we have $R_\Pi(X) \subseteq \mathcal{P}_\Gamma(PU^i(C))_\oplus = \mathcal{U}_\Gamma(\mathcal{P}_\Gamma(PU^i(C)))_\oplus$.
(b): We have $PU^{i+1}(C)_\ominus = \mathcal{U}_\Gamma(\mathcal{P}_\Gamma(PU^i(C)))_\ominus$. Analogous to Theorem A.2.1 we have $\mathcal{P}_\Gamma(PU^i(C))_\ominus \cap R_\Pi(X) = \emptyset$. Let (V, E) be a maximal support graph of $(\Gamma, \mathcal{P}_\Gamma(PU^i(C)))$ for some $E \subseteq (\Pi \times \Pi)$. To prove $\mathcal{U}_\Gamma(\mathcal{P}_\Gamma(PU^i(C)))_\ominus \cap R_\Pi(X) = \emptyset$, it is enough to show that $(\Pi \setminus V) \cap R_\Pi(X) = \emptyset$. But this holds by $\Pi \setminus V \subseteq \{r \mid \mathit{body}^+(r) \not\subseteq X\} \subseteq \{r \notin R_\Pi(X)\}$.
4: $C = PU^0(C) \sqsubseteq PU(C)$ holds by definition of $PU(C)$.
5: We have to show $\mathcal{P}_\Gamma(PU(C)) = PU(C)$ and $\mathcal{U}_\Gamma(PU(C)) = PU(C)$. By finiteness, there exists an $n < \omega$ s.t. $PU(C) = PU^n(C)$ and $PU^n(C) = PU^{n+1}(C)$. $PU(C) \sqsubseteq \mathcal{P}_\Gamma(PU(C))$ holds by definition of $\mathcal{P}_\Gamma$. And, we have $PU^{n+1}(C) = \mathcal{U}_\Gamma(\mathcal{P}_\Gamma(PU^n(C))) = PU^n(C)$. Thus, we have $\mathcal{P}_\Gamma(PU(C)) = PU(C)$. By use of Theorem 3.3.9 we have

$$PU^n(C) = \mathcal{U}_\Gamma(\mathcal{P}_\Gamma(PU^{n-1}(C))) = \mathcal{U}_\Gamma(\mathcal{U}_\Gamma(\mathcal{P}_\Gamma(PU^{n-1}(C)))) = \mathcal{U}_\Gamma(PU^n(C)) = \mathcal{U}_\Gamma(PU(C)).$$

6: Assume there exists a $Q(C) \neq PU(C)$ such that $Q(C) \sqsubseteq PU(C)$ and $Q(C)$ is a partial coloring closed under $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$. Given $Q^0(C) = C = PU^0(C)$, there must exists a (minimal) $i < \omega$ s.t. $Q^{i+1}(C) \neq PU^{i+1}(C)$ and $Q^j(C) = PU^j(C)$ for all $j \leq i$. But then

$$\begin{aligned} Q^{i+1}(C) &= \mathcal{U}_\Gamma Q^i(C) \sqcup \mathcal{P}_\Gamma Q^i(C) \sqcup Q^i(C) \\ &\neq \mathcal{U}_\Gamma PU^i(C) \sqcup \mathcal{P}_\Gamma PU^i(C) \sqcup PU^i(C) = PU^{i+1}(C). \end{aligned}$$

But this is a contradiction since $PU^i(C) = Q^i(C)$. For this reason, $PU(C)$ is the $\sqsubseteq$-smallest partial coloring closed under $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$.
7: This follows directly from condition 5 and by definition of $(\mathcal{PU})^*_\Gamma$. ■

We define $PV(C)$ as $PV(C) = \bigcup_{i<\omega} PV^i(C)$ $PV^0(C) = C$ and $PV^{i+1}(C) = \mathcal{V}_\Gamma(\mathcal{P}_\Gamma(PV^i(C)))$ for $i < \omega$. Clearly, $PV^i(C) \sqsubseteq PV^{i+1}(C)$ for all $i < \omega$.

Theorem A.2.4 *Let Γ be the RDG of logic program Π and C be a partial coloring of Γ. Then,*

1. *If $AC_\Pi(C) \neq \emptyset$ then $PV(C)$ exists,*
2. *$PV(C)$ is a partial coloring,*
3. *$AC_\Pi(C) = AC_\Pi(PV(C))$ and for all $i < \omega$ we have $X \in AS_\Pi(C)$ iff $X \in AS_\Pi(PV^i(C))$ iff $X \in AS_\Pi(PV(C))$,*
4. *$C \sqsubseteq PV(C)$,*
5. *$PV(C)$ closed under $\mathcal{P}_\Gamma$ and $\mathcal{V}_\Gamma$,*
6. *$PV(C)$ is the $\sqsubseteq$-smallest partial coloring closed under $\mathcal{P}_\Gamma$ and $\mathcal{V}_\Gamma$, and*
7. *$PV(C) = (\mathcal{PV})^*_\Gamma(C)$.*

Proof A.2.4 Holds analogous to Theorem A.2.3. ■

A.3 Proofs

A.3.1 Section 3.1

Proof 3.1.1 Let Γ be the *RDG* of logic program Π, C be a partial coloring of Γ and $X \in AS_\Pi(C)$. By Equation (3.4), we have $C_\oplus \subseteq R_\Pi(X)$ and by Theorem A.1.4 $head(C_\oplus) \subseteq X$.
1: Let $r \in S(\Gamma, C)$. By definition, for all $p \in body^+(r)$ there exists an $r' \in \Pi$ such that $(r', r) \in E_0$, $p = head(r')$ and $r' \in C_\oplus$. From $head(C_\oplus) \subseteq X$, we can reconclude that for each $p \in body^+(r)$, we have $p \in X$, and thus $body^+(r) \subseteq X$.
2-4 follow analogous to Condition 1 by Theorem A.1.4 and A.1.5. ■

Proof 3.1.2 Let Γ be the *RDG* of logic program Π, C be an admissible coloring of Γ, and $\{X\} = AS_\Pi(C)$. By Equation (3.4) we have $C_\oplus = R_\Pi(X)$ and $head(C_\oplus) = X$ (Theorem A.1.4). By Theorem 3.1.1 we only have to show "⇒".
1 "⇒": Let be $body^+(r) \subseteq X = head(C_\oplus)$ for $r \in \Pi$. Then, for each $p \in body^+(r)$ there exists an $r' \in \Pi$ such that $r' \in C_\oplus$ and $p = head(r')$. From the definition of the *RDG* we can conclude that $r \in S(\Gamma, C)$.
2-4 follow analogous. ■

Proof 3.1.3 This theorem follows by Theorem 3.1.1, Corollary 3.1.2, and Equation (2.3). ■

A.3.2 Section 3.2

Proof 3.2.1 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of logic program Π. Let (V'', E) and (V', E') be support graphs of Γ for some $E, E' \subseteq (\Pi \times \Pi)$ such that $V'' \neq V'$. We show that there exists a support graph (V, E) of Γ for some $E \subseteq (\Pi \times \Pi)$ such that $V'' \subseteq V$ and $V' \subseteq V$. Since Π is finite and by finiteness of the number of support graphs, we then conclude that there exists a maximal support graph of Γ. Trivially, according to Definition 3.2.1, there exists an enumeration $\langle r_i \rangle_{0 \leq i \leq n}$ of V'' such that $body^+(r_i) \subseteq \{head(r_j) \mid j < i\}$ and there exists an enumeration $\langle r'_i \rangle_{0 \leq i \leq m}$ of V' such that $body^+(r'_i) \subseteq \{head(r'_j) \mid j < i\}$. The idea of the construction of (V, E) is as follows: we merge the vertices of V'' and V' to V and additionally include all vertices into V whose positive body is derivable by the heads of the rules belonging to V. More precisely, V should be closed under all rules whose positive body is derivable by the heads of rules in V. Next, we define an enumeration $\langle s_i \rangle_{0 \leq i \leq k}$ of rules in Π inductively as follows. Let $s_0 = r \in \Pi$ where $body^+(r) = \emptyset$ and $s_i = r \in \Pi$ such that $body^+(r) \subseteq \{head(s_l) \mid l < i\}$ for $0 \leq i \leq k$ and for some maximal k such that there exists no $r' \in \Pi \setminus \{s_0, \ldots, s_k\}$ where $body^+(r') \subseteq \{head(s_l) \mid l \leq k\}$. Note that then the enumeration $\langle s_i \rangle_{0 \leq i \leq k}$ is maximal w.r.t. vertices. Furthermore, we have $n \leq k, m \leq k$ and $n + m - |V'' \cap V'| \leq k$.

Next, we show that for $V = \{s_i \mid 0 \leq i \leq k\}$ the following conditions are fulfilled:
1: $V'' \subseteq V$,
2: $V' \subseteq V$,
3: for all $r \in \Pi$ if $body^+(r) \subseteq \{head(r') \mid r' \in V\}$ then $r \in V$, and
4: (V, E) is a support graph of Γ for some $E \subseteq (\Pi \times \Pi)$.
Condition 3 states that all rules whose positive body can be derived by the heads of the rules in V are included in V. Observe that if there is no $r \in \Pi$ such that $body^+(r) = \emptyset$ we have that then $V' = \emptyset$ and $V'' = \emptyset$ and hence, $V'' = V'$ which is a contradiction to the assumption $V'' \neq V'$.
1+2: Note that for non-empty V'' and non-empty V' we have $body^+(r_0) = \emptyset$ and $body^+(r'_0) = \emptyset$ by construction of the enumerations of V'' and V'. Hence, there exist $0 \leq l, l' \leq k$ such that $r_0 = s_l$ and $r'_0 = s_{l'}$. Thus, we conclude by induction that $V'' \subseteq \{s_0, \ldots, s_k\}$ (Condition 1) and $V' \subseteq \{s_0, \ldots, s_k\}$ (Condition 2) by construction of V.
3: Condition 3 is fulfilled, since in we include as many rules as possible into $\langle s_i \rangle_{0 \leq i \leq k}$ (k being maximal).
4: For $0 < i \leq k$, we define

$$E^i = \bigcup \{(r', s_i) \mid r' \in \{s_0, \ldots, s_{i-1}\}\} \cap E_0$$

and $E = \bigcup_{0 \leq i \leq k} E^i$. Clearly, (V, E) is 0-subgraph of Γ by construction. Furthermore, (V, E) is acyclic since there are only edges (s_j, s_i) where $j < i$ for $j, i \in \{0, \ldots, k\}$. Also, we obtain $r \in V$ whenever $body^+(r) \subseteq \{head(r') \mid (r', r) \in E\}$. Thus, (V, E) is a support graph of Γ. Hence, there exists a support graph (V, E) of Γ such that $V' \subseteq V$ for all support graphs (V', E') of Γ. ■

Proof 3.2.2 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of logic program Π and C be a partial coloring of Γ. We abbreviate $\Gamma|_{C_\oplus \cup C_\ominus}$ with $\Gamma|_C$.
1: If $AC_\Pi(C) \neq \emptyset$ then we have $AS_\Pi(C) \neq \emptyset$ by Theorem A.1.1. Let be $X \in AS_\Pi(C)$. Then, we have by definition that $C_\oplus \subseteq R_\Pi(X)$ and $C_\ominus \cap R_\Pi(X) = \emptyset$ hold. We want to construct a support graph $(R_\Pi(X), E)$ of (Γ, C) for some $E \subseteq (\Pi \times \Pi)$. Then, there exists a maximal support graph of (Γ, C). By Theorem A.1.3 we have an enumeration $\langle r_i \rangle_{i \in I}$ of $R_\Pi(X)$ such that for all $i \in I$ we have $body^+(r_i) \subseteq head(\{r_j \mid j < i\})$. Define $V = \bigcup_{i \in I} \{r_i\}$ and $E = \{(r_j, r_i) \mid j < i\} \cap E_0$. Clearly, $V = R_\Pi(X)$ by construction and thus we have $C_\oplus \subseteq V$, $C_\ominus \cap V = \emptyset$, and (V, E) is a support graph of Γ. Furthermore, (V, E) is a support graph of (Γ, C). The existence of a maximal one follows by Theorem 3.2.1.
2: Let $(C_\oplus, E')$ be a support graph of $\Gamma|_C$ for some $E' \subseteq (\Pi \times \Pi)$. Then, $(C_\oplus, E')$ is a support graph of (Γ, C). By Theorem 3.2.1, there exists a (maximal) support graph of (Γ, C). ■

Proof 3.2.3 Let Γ be the *RDG* of logic program Π and C be an admissible coloring. By $\{C\} = AC_\Pi(C) \neq \emptyset$ and Theorem 3.2.2, there exists a support graph of (Γ, C). According to Definition 3.2.2, this must be $(C_\oplus, E)$ for some $E \subseteq \Pi \times \Pi$, since C is a total coloring. Furthermore, $(C_\oplus, E)$ is a maximal support graph of (Γ, C). ■

Proof 3.2.4 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ.
"$1 \Rightarrow 2$": Since C is an admissible coloring, there exists an answer set X of Π such that $\{X\} = AS_\Pi(C)$. By Theorem 3.2.2 and $\{C\} = AC_\Pi(C)$ there exists a support graph of (Γ, C). $C_\oplus = S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ holds by Theorem 3.1.3 and $C_\oplus = R_\Pi(X)$.
"$2 \Rightarrow 1$": We have to show that C is an admissible coloring that is $C_\oplus = R_\Pi(X)$ holds, where X is an answer set of Π. Let X be the set of atoms such that $X = head(C_\oplus)$, then

$$\begin{aligned} R_\Pi(X) &= \{r \mid body^+(r) \subseteq X, body^-(r) \cap X = \emptyset\} \\ &= \{r \mid body^+(r) \subseteq head(C_\oplus), body^-(r) \cap head(C_\oplus) = \emptyset\} \\ &= \{r \mid r \in S(\Gamma, C), r \in \overline{B}(\Gamma, C))\} \\ &= C_\oplus. \end{aligned}$$

By Theorem 3.2.6 and $(C_\oplus, E)$ is a (maximal) support graph of (Γ, C) for some $E \subseteq \Pi \times \Pi$ (Corollary 3.2.3), we have $head(C_\oplus) = Cn((\Pi \setminus C_\ominus)^\emptyset) = Cn(C_\oplus^\emptyset) = Cn((R_\Pi(X))^\emptyset)$. By $X = head(C_\oplus) = Cn((R_\Pi(X))^\emptyset)$ we have by Theorem A.1.2 that X is an answer set. Hence, C is an admissible coloring.
"$2 \Leftrightarrow 3$": This holds by C is a total coloring and by Theorem 3.1.3. ■

Proof 3.2.5 Let Γ be the *RDG* of logic program Π, C be a total coloring of Γ and X be a set of atoms such that $X = head(C_\oplus)$. We obtain for $r \in \Pi$, $r \in \overline{B}(\Gamma, C)$ iff $body^-(r) \cap head(C_\oplus) = \emptyset$ iff $body^-(r) \cap X = \emptyset$ iff $head(r) \leftarrow body^+(r) \in \Pi^X$. ■

Proof 3.2.6 Let Γ be the *RDG* of logic program Π, C be a partial coloring and (V, E) be a maximal support graph of (Γ, C). We have to show that $head(V) = Cn((\Pi \setminus C_\ominus)^\emptyset)$ holds. More precisely, we have to show that $head(V)$ is the smallest set of atoms which is closed under $(\Pi \setminus)^\emptyset$. First, we show that $head(V)$ is closed under $(\Pi \setminus)^\emptyset$. That is, for any $r \in (\Pi \setminus C_\ominus)^\emptyset$ we have $head(r) \in head(V)$ whenever $body^+(r) \subseteq head(V)$. Let be $r \in (\Pi \setminus C_\ominus)^\emptyset$. If we have $body^+(r) \subseteq head(V)$ then we have $head(r) \in head(V)$ since the support graph (V, E) is maximal. Thus, $head(V)$ is closed under $(\Pi \setminus C_\ominus)^\emptyset$. Second, we show that $head(V)$ is the smallest set of atoms which is closed under $(\Pi \setminus C_\ominus)^\emptyset$. Assume that $head(V)$ is not the smallest set of atoms which is closed under $(\Pi \setminus C_\ominus)^\emptyset$. Then, there exists a $q \in head(V)$ such that $head(V) \setminus q$ is closed under $(\Pi \setminus C_\ominus)^\emptyset$. We have $head(V) \setminus q = head(V \setminus Q)$ where

$Q = \{r \mid head(r) = q, r \in V\}$. Moreover, we have for all $r^+ \in (\Pi \setminus C_\ominus)^\emptyset$ that $head(r^+) \in head(V \setminus Q)$ whenever $body^+(r^+) \subseteq head(V \setminus Q)$. Let be $r' \in Q$. By $r' \in V$ we have that r is a vertex in the maximal support graph of (Γ, C). Hence, by Definition 3.2.2 we have that $body^+(r') \subseteq head(V)$. Moreover, we have $body^+(r') \subseteq head(V \setminus Q)$ since $Q = \{r \mid head(r) = q, r \in V\}$ and (V, E) is acyclic. Thus, we have that $head(r') \in head(V \setminus Q)$ by $head(V) \setminus q$ being closed under $(\Pi\setminus)^\emptyset$. But then we have $q = head(r') \in head(r) \setminus q$ which is a contradiction. Hence, $head(V)$ is the smallest set of atoms which is closed under $(\Pi \setminus C_\ominus)^\emptyset$. ■

Proof 3.2.7 Let Γ be the *RDG* for logic program Π and C be a total coloring of Γ.
"⇒": Let C be an admissible coloring of Γ. By Theorem 3.2.4 we have that $C_\oplus = S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ and there exists a support graph of (Γ, C). By C is total we have that $(C_\oplus, E)$ is a support graph of (Γ, C) for some $E \subseteq \Pi \times \Pi$. Hence, $(C_\oplus, E)$ is a support graph of Γ. Since $C_\oplus \subseteq \overline{B}(\Gamma, C)$ we have that $(C_\oplus, E)$ is a support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$. Next, we want to show the maximality of $(C_\oplus, E)$. Assume, $(C_\oplus, E)$ is not a maximal support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$. Then, there exists an $r \in C_\ominus$ such that $(C_\oplus \cup \{r\}, E')$ is a support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$ for some $E' \subseteq \Pi \times \Pi$. But then, $r \in \overline{B}(\Gamma, C)$ and $r \in S(\Gamma, C)$ by Definition 3.2.1 of a support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$. Hence, $r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C) = C_\oplus$ But this is a contradiction to $r \in C_\ominus$. Thus, $(C_\oplus, E)$ is a maximal support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$.
"⇐": Let $(C_\oplus, E)$ be a maximal support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$ for some $E \subseteq (\Pi \times \Pi)$. By Theorem 3.2.4 we have to show that $C_\oplus = S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ and there exists a support graph of (Γ, C). Since $(C_\oplus, E)$ is a support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$ we have that $(C_\oplus, E)$ is a also support graph of Γ. Furthermore, $(C_\oplus, E)$ is a support graph of (Γ, C) since C is a total coloring. It remains to show $C_\oplus = S(\Gamma, C) \cap \overline{B}(\Gamma, C)$.
"⊆": Assume that there exists an $r \in C_\oplus$ such that $r \notin S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. That is, $r \in \overline{S}(\Gamma, C) \cup B(\Gamma, C)$ holds. Since $(C_\oplus, E)$ is a support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$, $r \in B(\Gamma, C) \cap C_\oplus$ is not possible. Assume that $r \in \overline{S}(\Gamma, C) \cap C_\oplus$. But by Definition 3.2.1 and $r \in C_\oplus$ which is in a support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$, we have that $r \in S(\Gamma, C)$. But this is a contradiction and hence, we have $C_\oplus \subseteq S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. **"⊇":** Assume there exists an $r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ such that $r \notin C_\oplus$ that is $r \in C_\ominus$ holds. If $r \in \overline{B}(\Gamma, C)$ then r is a vertex of the graph $\Gamma|_{\overline{B}(\Gamma,C)}$. If furthermore $r \in S(\Gamma, C)$ then r is a vertex of the maximal support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$. Since $(C_\oplus, E)$ is a maximal support graph of $\Gamma|_{\overline{B}(\Gamma,C)}$, we have $r \in C_\oplus$. ■

Proof 3.2.8 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of a logic program Π and let C be a total coloring of Γ.
"⇒": Let C be an admissible coloring of Γ, then by Theorem 3.2.4 there exists a support graph of (Γ, C). Hence, it remains to show, that $(S, E_1|_S)$ is a blockage graph of (Γ, C) where $S = S(\Gamma, C)$. For all $r \in C_\oplus$ we have $r \in \overline{B}(\Gamma, C)$ by Theorem 3.2.4. Thus, Condition 1 for a blockage graph in Definition 3.2.3 holds. Let be $r \in S \cap C_\ominus$. By $C_\ominus = \overline{S}(\Gamma, C) \cup B(\Gamma, C)$ we have that $r \in B(\Gamma, C)$. Hence, Condition 2 from Definition 3.2.3 holds. Thus, $(S, E_1|_S)$ is a blockage graph of (Γ, C).
"⇐": By Theorem 3.2.4 and the existence of a support graph of (Γ, C) it remains to show that $C_\oplus = S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. Since C is total, the support graph of (Γ, C) must be $(C_\oplus, E)$ for some $E \subseteq \Pi \times \Pi$.
"⊆": Let be $r \in C_\oplus$. Then, we have $r \in S(\Gamma, C)$ by $(C_\oplus, E)$ is a support graph of (Γ, C) for some $E \subseteq (\Pi \times \Pi)$ and by Definition 3.2.2. Assume, we have $r \in B(\Gamma, C)$ then there exists an $r' \in C_\oplus$ such that $(r', r) \in E_1$. But this is a contradiction to Condition 1 in Definition 3.2.3 of a blockage graph. Thus, we have $r \in \overline{B}(\Gamma, C)$ and hence, we have $C_\oplus \subseteq S(\Gamma, C) \cap \overline{B}(\Gamma, C)$.
"⊇": Let be $r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. We have to show that $r \in C_\oplus$. Assume that we have $r \in C_\ominus$. Then, by Condition 2 in Definition 3.2.3 there exists an $r' \in C_\oplus$ such that r is blocked by r'. That's a contradiction and thus we have $r \in C_\oplus$. ■

Proof 3.2.9 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of a program Π and let C be a total coloring of Γ. To show this corollary we show the equivalence to the conditions given in Theorem 3.2.8. Clearly, a support graph of (Γ, C) is $(C_\oplus, E)$ for some $E \subseteq \Pi \times \Pi$ by Definition 3.2.2 since C is total. Furthermore, conditions 2 and 3 of this corollary are equivalent to conditions 1 and 2 in Definition 3.2.3 of a blockage graph. ■

A.3.3 Section 3.3

Proof 3.3.1 Let Γ be the *RDG* of logic program Π and C a partial coloring of Γ. If $AC_\Pi(C) \neq \emptyset$ then $AS_\Pi(C) \neq \emptyset$ by Theorem A.1.1. Let $X \in AS_\Pi(C)$ be an answer set of Π. Then $C_\oplus \subseteq R_\Pi(X)$ and $C_\ominus \cap R_\Pi(X) = \emptyset$. Let be $S = S(\Gamma, C), B = B(\Gamma, C), \overline{S} = \overline{S}(\Gamma, C)$ and $\overline{B} = \overline{B}(\Gamma, C)$. For showing that $\mathcal{P}_\Gamma(C)$ exists, we have to prove that $\mathcal{P}_\Gamma(C)$ is a partial coloring. We prove this by Theorem 3.1.3 and by the fact that C is a partial coloring. We observe

$$\begin{aligned}
\mathcal{P}_\Gamma(C)_\oplus \cap \mathcal{P}_\Gamma(C)_\ominus &= \big(C_\oplus \cup (S \cap \overline{B})\big) \cap \big(C_\ominus \cup \overline{S} \cup B\big) \\
&= (C_\oplus \cap C_\ominus) \cup (C_\oplus \cap \overline{S}) \cup (C_\oplus \cap B) \\
&\quad \cup (S \cap \overline{B} \cap C_\ominus) \cup (S \cap \overline{B} \cap \overline{S}) \cup (S \cap \overline{B} \cap B) \\
&= (C_\oplus \cap \overline{S}) \cup (C_\oplus \cap B) \cup (S \cap \overline{B} \cap C_\ominus) \\
&= \emptyset.
\end{aligned}$$

The last equality follows by Theorem 3.1.3. Thus, $\mathcal{P}_\Gamma(C)$ is a partial coloring and hence, $\mathcal{P}_\Gamma(C)$ exists. ■

Proof 3.3.2 This follows directly from Theorem A.2.1 by the existence of $P(C)$. ■

Proof 3.3.3 (Sketch) Let Γ be the *RDG* of logic program Π. We have to prove that $\mathcal{P}^*_\Gamma((\emptyset, \emptyset))$ exists. By Theorem A.2.1 it is enough to show that $P((\emptyset, \emptyset))$ exists. Clearly, $P^0((\emptyset, \emptyset)) = (\emptyset, \emptyset)$ exists. Assuming that $P^i((\emptyset, \emptyset))$ exists, we have to show $\mathcal{P}_\Gamma(P^i((\emptyset, \emptyset)))_\oplus \cap \mathcal{P}_\Gamma(P^i((\emptyset, \emptyset)))_\ominus = \emptyset$ stating that $P^{i+1}((\emptyset, \emptyset))$ exists. But this holds if

$$P^i((\emptyset, \emptyset))_\oplus \cap \overline{S}(\Gamma, P^i((\emptyset, \emptyset))) = \emptyset \tag{A.5}$$

$$P^i((\emptyset, \emptyset))_\oplus \cap B(\Gamma, P^i((\emptyset, \emptyset))) = \emptyset \tag{A.6}$$

$$P^i((\emptyset, \emptyset))_\ominus \cap S(\Gamma, P^i((\emptyset, \emptyset))) \cap \overline{B}(\Gamma, P^i((\emptyset, \emptyset))) = \emptyset \tag{A.7}$$

We obtain $S(\Gamma, C) \subseteq S(\Gamma, C')$ and $\overline{B}(\Gamma, C) \subseteq \overline{B}(\Gamma, C')$ for all partial colorings C, C' such that $C \sqsubseteq C'$. Hence, we have $P^i((\emptyset, \emptyset))_\oplus \subseteq S(\Gamma, P^i((\emptyset, \emptyset))) \cap \overline{B}(\Gamma, P^i((\emptyset, \emptyset)))$. Thus, A.5 and analogously A.6 and A.7 hold. For this reason we have that $P^{i+1}((\emptyset, \emptyset))$ exists and, by induction, that $\mathcal{P}^*_\Gamma((\emptyset, \emptyset))$ exists. ■

Proof 3.3.4 Let Γ be the *RDG* of logic program Π and let C and C' be partial colorings of Γ such that $AC_\Pi(C') \neq \emptyset$. Furthermore, let be $C \sqsubseteq C'$. Then, we also have $AC_\Pi(C) \neq \emptyset$ by definition.
$\mathcal{P}_\Gamma(C) \sqsubseteq \mathcal{P}_\Gamma(C')$**:** It is easy to see that $C \sqsubseteq C'$ implies $S(\Gamma, C) \subseteq S(\Gamma, C')$, $\overline{S}(\Gamma, C) \subseteq \overline{S}(\Gamma, C')$, $B(\Gamma, C) \subseteq B(\Gamma, C')$, and $\overline{B}(\Gamma, C) \subseteq \overline{B}(\Gamma, C')$. Thus, $\mathcal{P}_\Gamma(C) \sqsubseteq \mathcal{P}_\Gamma(C')$.
$\mathcal{P}^*_\Gamma(C) \sqsubseteq \mathcal{P}^*_\Gamma(C')$**:** Follows by showing $P(C) \sqsubseteq P(C')$ through an induction proof and by Thm. A.2.1. ■

Proof 3.3.5 Let Γ be the *RDG* of logic program Π and C be a partial coloring of Γ.
1: $C \sqsubseteq \mathcal{P}_\Gamma(C)$ implies $AC_\Pi(C) \supseteq AC_\Pi(\mathcal{P}_\Gamma(C))$. Thus, it remains to show $AC_\Pi(C) \subseteq AC_\Pi(\mathcal{P}_\Gamma(C))$. If $AC_\Pi(C) = \emptyset$ then $AC_\Pi(\mathcal{P}_\Gamma(C)) = \emptyset$ by $C \sqsubseteq \mathcal{P}_\Gamma(C)$ and by Equation (3.3). Let be $C' = (R_\Pi(X), \Pi \setminus R_\Pi(X)) \in AC_\Pi(C)$ an admissible coloring of Γ for some answer set X of Π. We have to show that $\mathcal{P}_\Gamma(C)_\oplus \subseteq R_\Pi(X)$ and $\mathcal{P}_\Gamma(C)_\ominus \cap R_\Pi(X) = \emptyset$. But this holds by Theorem 3.1.3 and $C_\oplus \subseteq R_\Pi(X)$ and $C_\ominus \cap R_\Pi(X) = \emptyset$.
2: Holds by Theorem A.2.1. ■

Proof 3.3.6 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of logic program Π and C be a partial coloring of Γ. Furthermore, let be $C' = \mathcal{P}_\Gamma(C)$.
1: Let $(C_\oplus, E)$ be a support graph of (Γ, C) for some $E \subseteq (\Pi \times \Pi)$. We have to show that $((\mathcal{P}^*_\Gamma(C))_\oplus, E')$ is a support graph of $(\Gamma, \mathcal{P}^*_\Gamma(C))$ for some $E' \subseteq (\Pi \times \Pi)$. We show this by construction of $((\mathcal{P}^*_\Gamma(C))_\oplus, E')$ and by Theorem A.2.1. Let $V^0 = P^0(C)_\oplus = C_\oplus$ and $E^0 = E$. Assume that we have constructed the sets $V^i \subseteq \Pi$ and $E^i \subseteq (\Pi \times \Pi)$ for some $i < \omega$ such that $P^i(C)_\oplus = V^i$. Now, we want to construct the sets

$V^{i+1} \subseteq \Pi$ and $E^{i+1} \subseteq (\Pi \times \Pi)$. We define

$$\begin{aligned} V^{i+1} &= V^i \cup \mathcal{P}_\Gamma(P^i(C))_\oplus \\ &= V^i \cup (S(\Gamma, P^i(C)) \cap \overline{B}(\Gamma, P^i(C))) \cup P^i(C)_\oplus \\ &= V^i \cup (S(\Gamma, P^i(C)) \cap \overline{B}(\Gamma, P^i(C))), \end{aligned}$$

$E^{i+1} = E^i \cup \left(\{(r', r) \mid r' \in V^i, r \in V^{i+1}\} \cap E_0\right)$, $V = \bigcup_{i<\omega} V^i$, and $E' = \bigcup_{i<\omega} E^i$. We have to show (1a) $V = P(C)_\oplus$ and (1b) (V, E') is a support graph of $(\Gamma, P(C))$. Then, by Theorem A.2.1 we can conclude that $((\mathcal{P}^*_\Gamma(C))_\oplus, E')$ is a support graph of $(\Gamma, \mathcal{P}^*_\Gamma(C))$ for some $E' \subseteq (\Pi \times \Pi)$.
(1a): This holds by construction of V.
(1b): Clearly, (V, E') is acyclic because E is acyclic and we have only edges from V^i to V^{i+1} for all $i < \omega$. Furthermore, (V, E') is a 0-subgraph of Γ by construction. Now, we have to show that for all $r \in V$ we have $body^+(r) \subseteq \{head(r') \mid (r', r) \in E'\}$. For $r \in V^0$ this holds by $(C_\oplus, E)$ is a support graph of (Γ, C). For $r \in V^i \setminus V^{i-1}$ for some $i > 0$ we have $r \in S(\Gamma, P^i(C)) \cap \overline{B}(\Gamma, P^i(C))$. Thus, if we have $r \in S(\Gamma, P^i(C))$ and by Definition 3.1.2 and construction of E^{i+1} we have $body^+(r) \subseteq \{head(r') \mid (r', r) \in E^{i+1}\}$. Thus, $(\mathcal{P}^*_\Gamma(C)_\oplus, E')$ is a support graph of Γ.
2: Let $(C_\oplus \cup C_\ominus, E_1)|_{S(\Gamma,C)}$ be a blockage graph of (Γ, C) and $C_\oplus \subseteq S(\Gamma, C)$. We have to show that $(C'_\oplus \cup C'_\ominus, E_1)|_{S(\Gamma,C')}$ is a blockage graph of (Γ, C') where $C' = \mathcal{P}^*_\Gamma(C)$. That is, we have to show that (2a) for all $r, r' \in C'_\oplus \cap S(\Gamma, C') : (r, r') \notin E_1 \mid_{S(\Gamma,C')}$, (2b) for all $r \in C'_\ominus \cap S(\Gamma, C') \exists r' \in C'_\oplus \cap S(\Gamma, C')$ s.t. $(r', r) \in E_1 \mid_{S(\Gamma,C')}$. But both conditions hold by C' being closed under $\mathcal{P}_\Gamma(C)$. ■

Proof 3.3.7 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ.
"⇒": By Theorem 3.2.4 it remains to show that $C = \mathcal{P}_\Gamma(C)$. We have $\mathcal{P}_\Gamma(C) = C \sqcup (S(\Gamma, C) \cap \overline{B}(\Gamma, C), \overline{S}(\Gamma, C) \cup B(\Gamma, C))$. By Theorem 3.2.4 we have $C = (S(\Gamma, C) \cap \overline{B}(\Gamma, C), \overline{S}(\Gamma, C) \cup B(\Gamma, C))$ and thus, $C = \mathcal{P}_\Gamma(C)$.
"⇐": By Theorem 3.2.4 it remains to show, that $C_\oplus = S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. By $C = \mathcal{P}_\Gamma(C)$ we have $C_\oplus \supseteq S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. Now, we show $C_\oplus \subseteq S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. Let $r \in C_\oplus$. If $r \in \overline{S}(\Gamma, C)$ or $r \in B(\Gamma, C)$, then $r \in C_\ominus$ by $C = \mathcal{P}_\Gamma(C)$, but this is a contradiction. Thus, $r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. For this reason, we have $C_\oplus = S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. ■

Proof 3.3.8 Let Γ be the *RDG* of program Π and C be a partial coloring of Γ. If there exists a support graph of (Γ, C), then there exists a maximal support graph of (Γ, C) (Thm. 3.2.1). Thus, $\mathcal{U}_\Gamma(C)$ exists. ■

Proof 3.3.9 Let Γ be the *RDG* of logic program Π and let C and C' be partial colorings of Γ such that $AC_\Pi(C) \neq \emptyset$, $AC_\Pi(C') \neq \emptyset$. Note that for C and C', $\mathcal{U}_\Gamma(C)$ and $\mathcal{U}_\Gamma(C')$ exist by Corollary 3.3.8 and Theorem 3.2.2.
1: $C \sqsubseteq \mathcal{U}_\Gamma(C)$ holds by definition of $\mathcal{U}_\Gamma$.
2: Let (V, E) be a maximal support graph of (Γ, C) for some $E \subseteq \Pi \times \Pi$. Then, we have $\mathcal{U}_\Gamma(C) = (C_\oplus, \Pi \setminus V)$. We claim that (V, E) is also a maximal support graph of $(\Gamma, \mathcal{U}_\Gamma(C))$. To see this, observe that (V, E) is a support graph of $\Gamma, \mathcal{U}_\Gamma(C)_\oplus \subseteq V$, and $V \cap (\mathcal{U}_\Gamma(C))_\ominus = V \cap (\Pi \setminus V) = \emptyset$. Furthermore, (V, E) is maximal since $V \cup (\mathcal{U}_\Gamma(C))_\ominus = V \cup (\Pi \setminus V) = \Pi$ and $V \cap (\mathcal{U}_\Gamma(C))_\ominus = \emptyset$. Therefore, we have $\mathcal{U}_\Gamma(\mathcal{U}_\Gamma(C)) = (\mathcal{U}_\Gamma(C)_\oplus, \Pi \setminus V) = (C_\oplus, \Pi \setminus V) = \mathcal{U}_\Gamma(C)$.
3: We have to show that $\mathcal{U}_\Gamma(C)_\oplus \subseteq \mathcal{U}_\Gamma(C')_\oplus$ and $\mathcal{U}_\Gamma(C)_\ominus \subseteq \mathcal{U}_\Gamma(C')_\ominus$. Since $C \sqsubseteq C'$ we have $\mathcal{U}_\Gamma(C)_\oplus \subseteq \mathcal{U}_\Gamma(C')_\oplus$. Because of $C \sqsubseteq C'$, we must have for a maximal support graph (V', E') of (Γ, C') and for a maximal support graph (V, E) of (Γ, C) that $V' \subseteq V$ for some $E, E' \in (\Pi \times \Pi)$. Thus, $\mathcal{U}_\Gamma(C)_\ominus \subseteq \mathcal{U}_\Gamma(C')_\ominus$ holds by definition of $\mathcal{U}_\Gamma$. ■

Proof 3.3.10 Let Γ be the *RDG* of logic program Π and C be a partial coloring of Γ. Clearly, we have $C \sqsubseteq \mathcal{U}_\Gamma(C)$ by Thm. 3.3.9. Thus, we have $AC_\Pi(\mathcal{U}_\Gamma(C)) \subseteq AC_\Pi(C)$. It remains to show that $AC_\Pi(C) \subseteq AC_\Pi(\mathcal{U}_\Gamma(C))$. If $AC_\Pi(C) = \emptyset$ then $AC_\Pi(\mathcal{U}_\Gamma(C)) = \emptyset$ since $C \sqsubseteq \mathcal{U}_\Gamma(C)$ (Theorem 3.3.9). Let be $C' \in AC_\Pi(C)$, where $C' = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ for some answer set X of Π. Let (V, E) be a maximal support graph of (Γ, C) for some $E \subseteq (\Pi \times \Pi)$. The existence of (V, E) is ensured by Theorem 3.2.2. We

have to show, (1) $\mathcal{U}_\Gamma(C)_\oplus \subseteq R_\Pi(X)$, and (2) $\mathcal{U}_\Gamma(C)_\ominus \cap R_\Pi(X) = \emptyset$.
1: This holds by $\mathcal{U}_\Gamma(C)_\oplus = C_\oplus \subseteq R_\Pi(X)$.
2: We have $\mathcal{U}_\Gamma(C)_\ominus = \Pi \setminus V$. Assume that there exists an $r \in (\Pi \setminus V) \cap R_\Pi(X)$. By $r \in R_\Pi(X)$ we have $r \in S(\Gamma, C')$ for $C' \in AC_\Pi(C)$. But this is a contradiction to $r \in (\Pi \setminus V)$ since all rules in $\Pi \setminus V$ can never be supported. Thus, $\mathcal{U}_\Gamma(C)_\ominus \cap R_\Pi(X) = \emptyset$ holds. ■

Proof 3.3.11 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of logic program Π and C be a partial coloring of Γ. Furthermore, let $C' = \mathcal{U}_\Gamma(C)$. Condition 1 holds trivially since $C \sqsubseteq C'$ and $C'_\ominus \cap C_\oplus = \emptyset$. Condition 2 holds by $C'_\oplus = C_\oplus$ and by $(C'_\ominus \setminus C_\ominus) \cap S(\Gamma, C) = \emptyset$. ■

Proof 3.3.12 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ. To show this theorem we show the equivalence to Corollary 3.3.7.
"⇒": It remains to show that $C = \mathcal{U}_\Gamma(C)$. By Corollary 3.3.7, there exists a (maximal) support graph of (Γ, C). That must be $(C_\oplus, E)$ for some $E \subseteq (\Pi \times \Pi)$ since C is a total coloring. Thus $\mathcal{U}_\Gamma(C) = (C_\oplus, \Pi \setminus C_\oplus) = C$.
"⇐": It remains to show that there exists a maximal support graph of (Γ, C). But this exists by the existence of $\mathcal{U}_\Gamma$. ■

Proof 3.3.13 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ.
"⇒": Let C be an admissible coloring of Γ. We have to show that there exists an sequence $(C^i)_{0 \leq i \leq n}$ satisfying the given conditions. Let $m = |C_\oplus|$ and $\langle r_i \rangle_{0 \leq i < m}$ be an arbitrary enumeration of $C_\oplus$. Analogous let $\langle r_j \rangle_{m \leq j < n}$ be an arbitrary enumeration of $\Pi \setminus C_\oplus = C_\ominus$. For $0 \leq i < m$ take $C^{i+1} = \mathcal{C}^\oplus_\Gamma(C^i) = (C^i_\oplus \cup \{r_i\}, C^i_\ominus)$ and for $m \leq i < n$ take $C^{i+1} = \mathcal{C}^\ominus_\Gamma(C^i) = (C^i_\oplus, C^i_\ominus \cup \{r_i\})$. Thus, C^n is a total coloring and conditions 3 and 4 are fulfilled by Corollary 3.3.12.
"⇐": By conditions 3, 4 and 5 we have $C = \mathcal{P}_\Gamma(C)$ and $C = \mathcal{U}_\Gamma(C)$. Thus, C is an admissible coloring of Γ by Corollary 3.3.12. ■

Proof 3.3.14 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ. Let $(C^i)_{0 \leq i \leq n}$ be a sequence satisfying conditions 1-5 in Theorem 3.3.13.
1: We have C^i is a partial coloring for $0 \leq i \leq n$ by Definition 3.3.4.
2: By construction of C^i and by Definition 3.3.4 we have $C^i \sqsubseteq C^{i+1}$.
3: We have $C^i \sqsubseteq C^{i+1}$ and thus we have $AC_\Pi(C^i) \supseteq AC_\Pi(C^{i+1})$.
4: Since we have $C = C^n \sqsupseteq C^i$ for admissible coloring C, we have $C' \in AC_\Pi(C^i) \neq \emptyset$.
5: Since $AC_\Pi(C^i) \neq \emptyset$ we have by Thm. 3.2.2 that there exists a (maximal) support graph of (Γ, C^i). ■

Proof 3.3.15 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ.
"⇐": Since $(\mathcal{PU})^*_\Gamma$ is closed under $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$ we have $C = \mathcal{P}_\Gamma(C)$ and $C = \mathcal{U}_\Gamma(C)$ and thus, we have that C is an admissible coloring of Γ by Corollary 3.3.12.
"⇒": Let $C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ be an admissible coloring of Γ for answer set X of Π. We have to show that there exists a sequence $(C^i)_{0 \leq i \leq n}$ satisfying the given conditions. Let $C^0 = (\mathcal{PU})^*_\Gamma((\emptyset, \emptyset))$. Assume, C^i is defined for some $0 \leq i$. Let be $r \in \Pi \setminus (C^i_\oplus \cup C^i_\ominus)$. If we have $r \in R_\Pi(X)$ then we take $C^{i+1} = (\mathcal{PU})^*_\Gamma(C^i_\oplus \cup \{r\}, C^i_\ominus)$. Otherwise, if we have $r \in \Pi \setminus R_\Pi(X)$ then we take $C^{i+1} = (\mathcal{PU})^*_\Gamma(C^i_\oplus, C^i_\ominus \cup \{r\})$. By construction there exists an $n < \omega$ such that $\Pi = C^n_\oplus \cup C^n_\ominus$. Furthermore, by Theorem 3.3.5, 3.3.10, and by construction of each $C^{i+1}(for 0 \leq i < n)$, we have that $C^n = C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$. ■

Proof 3.3.16 Proof by induction over i by using Theorem A.2.3. ■

Proof 3.3.17 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ. Let $(C^i)_{0 \leq i \leq n}$ be a sequence satisfying conditions 1-3 in Theorem 3.3.15.
1-5: Hold analogous to Theorem 3.3.14 by Definition 3.3.4 and by Theorem A.2.3.
6+7: These conditions hold by definition of $\mathcal{P}_\Gamma$:
$S(\Gamma, C^i) \cap \overline{B}(\Gamma, C^i) \subseteq \mathcal{P}_\Gamma(C^i)_\oplus \subseteq (\mathcal{PU})^*_\Gamma(\mathcal{C}^\circ_\Gamma(C^i))_\oplus = C^{i+1}_\oplus$, and
$\overline{S}(\Gamma, C^i) \cup B(\Gamma, C^i) \subseteq \mathcal{P}_\Gamma(C^i)_\ominus \subseteq (\mathcal{PU})^*_\Gamma(\mathcal{C}^\circ_\Gamma(C^i))_\ominus = C^{i+1}_\ominus$. ■

Proof 3.3.18 Condition 1–3, 6–7 hold analogous to Theorem 3.3.17. By C^i is closed under $\mathcal{U}_\Gamma$, there exists a support graph of (Γ, C^i). Hence, Condition 5 holds. ■

Proof 3.3.19 Let Γ be the *RDG* of logic program Π. First, we show that every sequence over $\mathbb{C}$ satisfying conditions 1 and 2 in Theorem 3.3.15 induces a sequence over $\mathbb{C}$ satisfying conditions 1-4 in Theorem 3.3.13. Let $(C^i)_{i\in J}$ be a sequence satisfying conditions 1 and 2 in Theorem 3.3.15. We obtain an enumeration $\langle r_i\rangle_{i\in I}$ of Π such that we have: if $r_i \in PU^k(C^l)$, $r_j \in PU^e(C^f)$ where $l < f$ or $l = f$ and $k \leq e$ then $i < j$. Thus, we obtain a sequence $(C'^i)_{i\in J'}$ such that $C'^0 = (\emptyset, \emptyset)$ and

$$C'^{i+1} = \begin{cases} (C'^i_\oplus \cup \{r_i\}, C'^i_\ominus) & \text{if } r_i \in R_\Pi(X) \\ (C'^i_\oplus, C'^i_\ominus \cup \{r_i\}) & \text{if } r_i \notin R_\Pi(X). \end{cases}$$

$(C'^i)_{i\in J'}$ clearly satisfies conditions 1-4 in Theorem 3.3.13 by C^i is closed under $\mathcal{P}_\Gamma$ and $\mathcal{U}_\Gamma$ for every $i \in J$ in Theorem 3.3.15. Second, 2 different sequences from Theorem 3.3.15 induces 2 different sequences in Theorem 3.3.13. Hence, $n \leq m$. ■

Proof 3.3.20 Let Γ be the *RDG* of logic program Π and let C be an admissible coloring of Γ. Let $(C_1^i)_{0\leq i\leq m}$ and $(C_2^j)_{0\leq j\leq n}$ be the shortest sequences obtained for C according to Theorem 3.3.13 and Theorem 3.3.15, respectively. Since $|C_1^{i+1} \setminus C_1^i| = 1$ for all $0 \leq i < m$ and $|C_2^{j+1} \setminus C_2^j| \geq 1$ for all $0 \leq j < n$, we have that $n \leq m$ holds where $m = |\Pi|$. ■

Proof 3.3.21 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ. We have to prove:
Plus: C is an admissible coloring of Γ iff there exists a sequence $(C^i)_{0\leq i\leq n}$ with the following properties: $C^0 = (\mathcal{PU})^*_\Gamma((\emptyset,\emptyset))$; $C^{i+1} = (\mathcal{PU})^*_\Gamma(\mathcal{C}^\oplus_\Gamma(C^i))$ for $0 \leq i < n$; $C^n = C$.
Minus: C is an admissible coloring of Γ iff there exists a sequence $(C^i)_{0\leq i\leq n}$ with the following properties: $C^0 = (\mathcal{PU})^*_\Gamma((\emptyset,\emptyset))$; $C^{i+1} = (\mathcal{PU})^*_\Gamma(\mathcal{C}^\ominus_\Gamma(C^i))$ for $0 \leq i < n$; $C^n = C$.
'Plus':
"⇐": C is an admissible coloring of Γ holds by Corollary 3.3.12 since C is closed under $\mathcal{P}_\Gamma$ and closed under $\mathcal{U}_\Gamma$.
"⇒": Let $C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ be an admissible coloring of Γ for answer set X of Π. We have to show the existence of a sequence $(C^i)_{0\leq i\leq n}$ with the given properties. Let $C^0 = (\mathcal{PU})^*_\Gamma((\emptyset,\emptyset))$. For $0 < i < n$ let $C^{i+1} = (\mathcal{PU})^*_\Gamma(\mathcal{C}^\oplus_\Gamma(C^i))$ where $\mathcal{C}^\oplus_\Gamma(C^i) = (C^i_\oplus \cup \{r\}, C^i_\ominus)$ for some $r \in R_\Pi(X) \cap (\Pi \setminus (C^i_\oplus \cup C^i_\ominus))$. It remains to show, that C^n is a total coloring and $C = C^n$ for some $n \geq 0$. Assume, C^n is not total, then $\Pi \setminus (C^n_\oplus \cup C^n_\ominus) \not\subseteq R_\Pi(X)$. Otherwise, we could choose an $r \in R_\Pi(X)$ to extend our sequence of partial colorings. Then, for all $r \in \Pi \setminus (C^n_\oplus \cup C^n_\ominus)$ we have either $body^+(r) \not\subseteq X$ or $body^-(r) \cap X \neq \emptyset$. Observe that $X = head(C^n_\oplus) = head(R_\Pi(X))$ holds by X is an answer set. If there is an $r \in \Pi \setminus (C^n_\oplus \cup C^n_\ominus)$ such that $body^-(r) \cap X \neq \emptyset$, then r is blocked by some $r' \in C^n_\oplus$ and r had to be colored in C^n by C^n is closed under $\mathcal{P}_\Gamma$. Thus, $body^+(r) \not\subseteq X$ for all $r \in \Pi \setminus (C^n_\oplus \cup C^n_\ominus)$. Furthermore, all $r \in \Pi \setminus (C^n_\oplus \cup C^n_\ominus)$ are in a maximal support graph of (Γ, C^n) by C^n is closed under $\mathcal{U}_\Gamma$. Thus, there must exists an $r \in \Pi \setminus (C^n_\oplus \cup C^n_\ominus)$ such that $r \in S(\Gamma, C^n) \cap \overline{B}(\Gamma, C^n)$. But then, r would be colored in C^n by C^n is closed under $\mathcal{P}_\Gamma$. That's a contradiction. Thus, C^n is a total coloring. Furthermore, $C^n = C$ holds by C^n is closed under $\mathcal{P}_\Gamma$ and by Theorem 3.1.3.
'Minus':
"⇐": C is an admissible coloring holds by Corollary 3.3.12 because C is closed under $\mathcal{P}_\Gamma$ and closed under $\mathcal{U}_\Gamma$.
"⇒": Let $C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ be an admissible coloring for answer set X of Π. We have to show the existence of a sequence $(C^i)_{0\leq i\leq n}$ satisfying **'Minus'**. Let $C^0 = (\mathcal{PU})^*_\Gamma((\emptyset,\emptyset))$. For $0 < i < n$ let $C^{i+1} = (\mathcal{PU})^*_\Gamma(\mathcal{C}^\ominus_\Gamma(C^i))$ where $\mathcal{C}^\ominus_\Gamma(C^i) = (C^i_\oplus, C^i_\ominus \cup \{r\})$ for some $r \in (\Pi \setminus R_\Pi(X)) \cap (\Pi \setminus (C^i_\oplus \cup C^i_\ominus))$. It remains to show, that C^n is a total coloring and $C^n = C$, but this can be shown analogous to **'Plus'**. ■

Proof 3.3.22 Let Γ be the *RDG* of logic program Π and C^X be a total coloring of Γ.
"⇒": Let $C^X = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ be an admissible coloring of Γ for answer set X of Π. Clearly,

C^X is a total coloring and $head(C^X_\oplus) = X$ by X is an answer set. Let $|R_\Pi(X)| = n-2$ and $\langle r_i\rangle_{0\leq i<n-2}$ be an enumeration of $R_\Pi(X)$. Given this, we define the sequence $(C^i)_{0\leq i\leq n}$ such that

1. $C^0 = (\emptyset, \emptyset)$,
2. $C^{i+1} = (C^i_\oplus \cup \{r_i\}, C^i_\ominus)$ for $0 \leq i < n-2$,
3. $C^{n-1} = \mathcal{P}^*_\Gamma(C^{n-2})$, and

3′. $C^n = \mathcal{U}_\Gamma(C^{n-1})$.

Clearly, this sequence satisfies conditions 1, 2, and 3 in Theorem 3.3.22. Next, we show that C^n is a total coloring. More precisely, we show that $C^n = C^X$. By definition, we have $C^{n-2} = (R_\Pi(X), \emptyset)$. By Theorem A.1.6 this implies $\mathcal{P}^*_\Gamma(C^{n-2}) = C^{n-1} = (R_\Pi(X), C^{n-1}_\ominus)$ where $B(\Gamma, C^X) \subseteq C^{n-1}_\ominus$ by $\mathcal{P}_\Gamma(C^{n-2}) \sqsubseteq \mathcal{P}^*_\Gamma(C^{n-2})$. By definition of C^X and Theorem 3.1.3, we have that $R_\Pi(X) = S(\Gamma, C^X) \cap \overline{B}(\Gamma, C^X)$. That is, we have $S(\Gamma, C^X) \cap \overline{B}(\Gamma, C^X) = C^{n-1}_\oplus$. Moreover, given that $B(\Gamma, C^X) \subseteq C^{n-1}_\ominus$, we obtain $S(\Gamma, C^X) \cap B(\Gamma, C^X) \subseteq C^{n-1}_\ominus$. Since C^X is a total coloring, we have $C^X = B(\Gamma, C^X) \cup \overline{B}(\Gamma, C^X)$. Consequently, $S(\Gamma, C^X) \subseteq C^{n-1}_\oplus \cup C^{n-1}_\ominus$ and also $\Pi \setminus (C^{n-1}_\oplus \cup C^{n-1}_\ominus) \subseteq \overline{S}(\Gamma, C^X)$ holds. That is, all rules uncolored in C^{n-1} are also unsupported in C^X. Clearly, $(C^X_\oplus, E)$ is a maximal support graph of (Γ, C^X) for some $E \subseteq (\Pi \times \Pi)$. Given that $C^X_\oplus = C^{n-1}_\oplus$ and $C^X_\ominus \setminus C^{n-1}_\ominus \subseteq \overline{S}(\Gamma, C^X)$, $(C^X_\oplus, E)$ is also a maximal support graph of (Γ, C^{n-1}). We get

$$\mathcal{U}_\Gamma(C^{n-1}) = (C^{n-1}_\oplus, (\Pi \setminus C^X_\oplus)) = (C^X_\oplus, (\Pi \setminus C^X_\oplus)) = (C^X_\oplus, C^X_\ominus) = C^X$$

That is, we have $C^n = C^X$. Therefore, we obtain that C^n is a total coloring.
"⇐": According to Corollary 3.3.12, it is sufficient to show that (1) $C = \mathcal{P}_\Gamma(C)$ and (2) $C = \mathcal{U}_\Gamma(C)$.
1: We have to show that $C = C \sqcup (S(\Gamma, C) \cap \overline{B}(\Gamma, C),\ \overline{S}(\Gamma, C) \cup B(\Gamma, C))$. For this, it is enough to show, that $(S(\Gamma, C) \cap \overline{B}(\Gamma, C),\ \overline{S}(\Gamma, C) \cup B(\Gamma, C)) \sqsubseteq C$ that is (1a) $S(\Gamma, C) \cap \overline{B}(\Gamma, C) \subseteq C_\oplus$ and (1b) $\overline{S}(\Gamma, C) \cup B(\Gamma, C) \subseteq C_\ominus$.
(1a): Assume there exists an $r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ such that $r \in C_\ominus$. We have 2 cases where r had to be colored in the sequence $(C^i)_{0\leq i\leq n}$. Case 1: $r \in \mathcal{P}^*_\Gamma(C^{n-2})_\ominus$ and $r \notin C^{n-2}_\ominus$ or Case 2: $r \in \mathcal{U}_\Gamma(\mathcal{P}^*_\Gamma(C^{n-2}))_\ominus$ and $r \notin \mathcal{P}^*_\Gamma(C^{n-2})_\ominus$. In **Case 1** we have $r \in \mathcal{P}^*_\Gamma(C^{n-2})_\ominus = P(C^{n-2})_\ominus$ by Theorem A.2.1. Thus, there exists an $i < \omega$ such that $r \in \overline{S}(\Gamma, P^i(C^{n-2})) \cup B(\Gamma, P^i(C^{n-2}))$. By $P^i(C') \sqsubseteq P(C')$ we have $r \in \overline{S}(\Gamma, C) \cup B(\Gamma, C)$, but this is a contradiction. to $r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. In **Case 2** we have $r \in \mathcal{U}_\Gamma(\mathcal{P}^*_\Gamma(C^{n-2}))_\ominus$. Let (V, E) be a maximal support graph of $(\Gamma, \mathcal{P}^*_\Gamma(C^{n-2}))$ for some $E \subseteq (\Pi \times \Pi)$ Then, we have $r \in \Pi \setminus V$. By $r \in S(\Gamma, C)$ and $C_\oplus = \mathcal{U}_\Gamma(\mathcal{P}^*_\Gamma(C^{n-2}))_\oplus = \mathcal{P}^*_\Gamma(C^{n-2})_\oplus$, we have $r \in S(\Gamma, \mathcal{P}^*_\Gamma(C^{n-2}))$. But then, we have $r \in V$ by V being maximal. Also, this is a contradiction and thus, we have $r \in C_\oplus$.
(1b): Let be $r \in B(\Gamma, C)$ and $r \in C_\oplus$. There are 2 cases where r had to be colored with $\oplus$. Case 1: $r \in C^{n-2}_\oplus$ or Case 2: $r \in \mathcal{P}^*_\Gamma(C^{n-2})$ and $r \notin C^{n-2}_\oplus$. In both cases we have $r \in B(\Gamma, \mathcal{U}_\Gamma(\mathcal{P}^*_\Gamma(C^{n-2})))$. By $\mathcal{U}_\Gamma(\mathcal{P}^*_\Gamma(C^{n-2}))_\oplus = \mathcal{P}^*_\Gamma(C^{n-2})_\oplus$ we have $r \in B(\Gamma, \mathcal{P}^*_\Gamma(C^{n-2}))$. Hence, we have $r \in \mathcal{P}^*_\Gamma(C^{n-2})_\ominus \subseteq C_\ominus$ by $\mathcal{P}^*_\Gamma$ is closed under $\mathcal{P}_\Gamma$. This is a contradiction. Let be $r \in \overline{S}(\Gamma, C)$ and $r \in C_\oplus$. Then, we have $r \in \overline{S}(\Gamma, \mathcal{U}_\Gamma(\mathcal{P}^*_\Gamma(C^{n-2})))$. By Theorem 3.3.9 and $r \in \mathcal{U}_\Gamma(\mathcal{U}_\Gamma(\mathcal{P}^*_\Gamma(C^{n-2})))_\ominus$ we have $r \in \mathcal{U}_\Gamma(\mathcal{P}^*_\Gamma(C^{n-2}))_\ominus$. But, this is a contradiction to $r \in C_\oplus$.
2: $C = \mathcal{U}_\Gamma(C)$ is equivalent to $\mathcal{U}_\Gamma(C') = \mathcal{U}_\Gamma(\mathcal{U}_\Gamma(C'))$ for $C' = \mathcal{P}^*_\Gamma(C^{n-2})$, which is true by virtue of Theorem 3.3.9. ■

Proof 3.3.23 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ.
"⇒": Let $C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ be an admissible coloring of Γ for answer set X of Π. We want to construct a sequence $(C^i)_{0\leq i\leq n}$ with the given properties. Let $\langle r_i\rangle_{1\leq i\leq n-1}$ be an enumeration of $\Pi \setminus R_\Pi(X)$ where $n-1 = |\Pi \setminus R_\Pi(X)|$. Let be $C^0 = (\emptyset, \emptyset)$. For $1 \leq i \leq n-1$ let be $C^i = (C^{i-1}_\oplus, C^{i-1}_\ominus \cup \{r_i\})$. Then, we have $C^{n-1} = (\emptyset, \Pi \setminus R_\Pi(X))$. Note that by Theorem 3.3.2 and by $C \in AC_\Pi(C^{n-1})$ we have

that $C^n = \mathcal{P}^*_\Gamma(C^{n-1})$ is a partial coloring. It remains to show, that $C = C^n$ where $C^n = \mathcal{P}^*_\Gamma(C^{n-1})$. More precisely, we have to show for all $r \in \Pi$: (1) If $r \in R_\Pi(X)$ then $r \in C^n_\oplus = \mathcal{P}^*_\Gamma(C^{n-1})_\oplus$. (2) If $r \notin R_\Pi(X)$ then $r \in C^n_\ominus = \mathcal{P}^*_\Gamma(C^{n-1})_\ominus$.
1: Let be $r \in R_\Pi(X)$, then we have $body^-(r) \cap X = \emptyset$. By $X = head(R_\Pi(X))$ we have $body^-(r) \cap head(R_\Pi(X)) = \emptyset$. Thus, we have $r \in \overline{B}(\Gamma, C^{n-1})$. By Theorem A.1.3, there exists an enumeration of $\langle r_i \rangle_{i \in I}$ of $R_\Pi(X)$ such that for all $i \in I$ we have $body^+(r_i) \subseteq head(\{r_j \mid j < i\})$. Clearly, we have $r_0 \in S(\Gamma, C^{n-1})$. Hence, we have $r_0 \in C^n_\oplus$. By induction over $i \in I$ we can show that $r_i \in S(\Gamma, (C^{n-1}_\oplus \cup \{r_j \mid j < i\}, C^{n-1}_\ominus))$. Thus, $r \in C^n_\oplus$ whenever $r \in R_\Pi(X)$.
2: This follows by $\Pi \setminus R_\Pi(X) = C^{n-1}_\ominus$. Hence, we conclude that $C^n = C$ holds.
"⇐": Let $(C^i)_{0 \leq i \leq n}$ be a sequence with the given properties. By Corollary 3.3.12 we have to show (1) $C^n = \mathcal{P}_\Gamma(C^n)$ and (2) $C^n = \mathcal{U}_\Gamma(C^n)$.
1: This condition is fulfilled by $C^n = \mathcal{P}^*_\Gamma(C^{n-1})$ and by $\mathcal{P}^*_\Gamma$ being closed under $\mathcal{P}_\Gamma$.
2: By $C^n = \mathcal{P}^*_\Gamma(C^{n-1})$ and $(\emptyset, \emptyset)$ is a support graph of $(\Gamma, (\emptyset, \Pi \setminus R_\Pi(X)))$ and Theorem 3.3.6 we have that $(C^n_\oplus, E)$ is a support graph of (Γ, C) for some $E \subseteq (\Pi \times \Pi)$. Then, $(C^n_\oplus, E)$ is a maximal support graph of (Γ, C) and $C^n = \mathcal{U}_\Gamma(C^n)$ by C^n is a total coloring and $C^n_\ominus = \Pi \setminus R_\Pi(X)$. ■

Proof 3.3.24 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of logic program Π and let C be a total coloring of Γ.
"⇒": This follows analogous to the proof "⇒" of Corollary 3.3.26.
"⇐": This follows analogous to the proof of Theorem 3.3.26 by verifying the construction of the maximal support graph of (Γ, C) as follows: Let $R_i = C^{i+1}_\oplus \setminus C^i_\oplus$ for $0 \leq i < n-1$. Let $V^0 = \{r_0\}$ and $E^0 = \emptyset$. Assume, $V^i \subseteq \Pi$ and $E^i \subseteq \Pi \times \Pi$ are defined for some $0 \leq i < n-1$. Define $V^{i+1} = V^i \cup \{R_i\}$ and $E^{i+1} = E^i \cup E^{R_i}$ where
$$E^{R_i} = \begin{cases} \{(r', r_{i+1}) \mid r' \in V^i\} \cap E_0 & \text{if } R_i = \{r_{i+1}\} \text{ for some } r_{i+1} \in \Pi \\ \emptyset & \text{if } R_i = \emptyset. \end{cases}$$ ■

Proof 3.3.25 These properties follow analogous to Theorem 3.3.27 with the modification given in Theorem 3.3.24 in the construction of the support graph of (Γ, C^i) for $0 \leq i \leq n$. ■

Proof 3.3.26 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of logic program Π and let C be a total coloring of Γ.
"⇒": Let $C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ be an admissible coloring of Γ for answer set X of Π. By Theorem A.1.3 there exists an enumeration $\langle r_i \rangle_{0 \leq i < n-1}$ of $R_\Pi(X)$ such that for all $i \in \{0, \ldots, n-2\}$ we have $body^+(r_i) \subseteq head(\{r_j \mid j < i\})$. We define the sequence $(C^i)_{0 \leq i \leq n}$ as follows:

1. $C^0 = (\emptyset, \emptyset)$,
2. $C^{i+1} = (C^i_\oplus \cup \{r_i\}, C^i_\ominus)$ for $0 \leq i < n-1$,
3. $C^n = \mathcal{N}_\Gamma(C^{n-1})$.

By construction and by definition of $\mathcal{N}_\Gamma$, we have that C^n is a total coloring, $C^n_\oplus = R_\Pi(X)$, and $C^n_\ominus = \Pi \setminus R_\Pi(X)$. By definition of C^{i+1} for $0 \leq i < n-1$ we have $C^{i+1} = \mathcal{D}^\oplus_\Gamma(C^i)$ because $r_i \in S(\Gamma, C^i)$ by Theorem A.1.3. Thus, it remains to show that $C^n = \mathcal{P}_\Gamma(C^n)$. $C^n \sqsubseteq \mathcal{P}_\Gamma(C^n)$ holds by definition of $\mathcal{P}_\Gamma$. Hence, it is enough to show that $S(\Gamma, C^n) \cap \overline{B}(\Gamma, C^n) \subseteq C^n_\oplus$ and $\overline{S}(\Gamma, C^n) \cup B(\Gamma, C^n) \subseteq C^n_\ominus$. By Theorem 3.1.3 we have, if $r \in S(\Gamma, C^n) \cap \overline{B}(\Gamma, C^n)$ then $r \in R_\Pi(X) = C^n_\oplus$ and if $r \in \overline{S}(\Gamma, C^n) \cup B(\Gamma, C^n)$ then $r \in (\Pi \setminus R_\Pi(X)) = C^n_\ominus$. Thus, $C^n = \mathcal{P}_\Gamma(C^n)$ and $C^n = C$.
"⇐": By Corollary 3.3.7 we have to show $C = \mathcal{P}_\Gamma(C)$ and there is a support graph of (Γ, C). $C = \mathcal{P}_\Gamma(C)$ holds by property 4 and 5 of the defined sequence $(C^i)_{0 \leq i \leq n}$ in Theorem 3.3.24. Now, we want to construct by induction a support graph $(C^n_\oplus, E)$ of (Γ, C) for some $E \subseteq \Pi \times \Pi$. Let $\{r_i\} = C^{i+1}_\oplus \setminus C^i_\oplus$ for $0 \leq i < n-1$. Let $V^0 = \{r_0\}$ and $E^0 = \emptyset$. Assume, $V^i \subseteq \Pi$ and $E^i \subseteq \Pi \times \Pi$ are defined for some $0 \leq i < n-1$. Define $V^{i+1} = V^i \cup \{r_{i+1}\}$ and $E^{i+1} = E^i \cup E^{r_{i+1}}$ where $E^{r_{i+1}} = \{(r', r_{i+1}) \mid r' \in V^i\} \cap E_0$. Let $V = \bigcup_{i<\omega} V^i$ and $E = \bigcup_{i<\omega} E^i$, then (V, E) is a support graph of (Γ, C) where $V = C^n_\oplus$. Hence, C is an admissible coloring of Γ. ■

Proof 3.3.27 Given the same prerequisites as in Corollary 3.3.26 with $\Gamma = (\Pi, E_0, E_1)$. Let $(C^i)_{0 \leq i \leq n}$ be a sequence satisfying conditions 1-5 in Corollary 3.3.26.
1-5: Hold analogous to Theorem 3.3.14.
8: This follows analogous to the construction of a support graph of (Γ, C) in Corollary 3.3.26.
9: We have to prove (9a) for all $r, r' \in C^i_\oplus \cap S(\Gamma, C^i)$ we have $(r, r') \notin E_1|_{S(\Gamma, C^i)}$, (9b) for all $r \in S(\Gamma, C^i) \cap C^i_\ominus$ exists an $r' \in C^i_\oplus \cap S(\Gamma, C^i)$ such that $(r', r) \in E_1|_{S(\Gamma, C^i)}$.
(9a): Let be $r, r' \in C^i_\oplus \cap S(\Gamma, C^i)$. Assume that we have $(r, r') \in E_1|_{S(\Gamma, C^i)}$. Then, we have $r' \in B(\Gamma, C^i)$. By $C^i \sqsubseteq C^n$ and by C^n is closed under $\mathcal{P}_\Gamma$ we have $r' \in B(\Gamma, C^n)$ and thus $r' \in C^n_\ominus$. But this is a contradiction to $r' \in C^i_\oplus \subseteq C^n_\oplus$.
(9b): For $0 \leq i \leq n-1$ we have $C^i_\ominus = \emptyset$ and thus, there is nothing to show. Let be $i = n$. C^n is closed under $\mathcal{P}_\Gamma$. Assume, there is an $r \in C^n_\ominus \cap S(\Gamma, C^n)$ such that $r \in \overline{B}(\Gamma, C^n)$. Then we have $r \in C^n_\oplus$ by C^n is closed under $\mathcal{P}_\Gamma$. Thus, for $r \in C^n_\ominus \cap S(\Gamma, C^n)$ we have $r \in B(\Gamma, C^n)$. Hence, there must exist an $r' \in C^n_\oplus$ such that $(r', r) \in E_1$. Furthermore, $r' \in S(\Gamma, C^n)$ holds by $C^n_\oplus = \mathcal{P}_\Gamma(C^n)_\oplus = S(\Gamma, C^n) \cap \overline{B}(\Gamma, C^n)$. Thus, we have $(r', r) \in E_1|_{(S(\Gamma, C^n)}$. ■

Proof 3.3.28 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ.
"⇒": Let $C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ be an admissible coloring of Γ for answer set X and $\langle r_i \rangle_{0 \leq i \leq m}$ be an enumeration of $R_\Pi(X)$ such that for all $i \in \{0, \ldots, m\}$ we have $body^+(r_i) \subseteq head(\{r_j \mid j < i\})$. Let be $C^0 = \mathcal{P}^*_\Gamma((\emptyset, \emptyset))$ and for $0 \leq i < n-1$ let be $C^{i+1} = \mathcal{P}^*_\Gamma(\mathcal{D}^\circ_\Gamma(C^i)) = \mathcal{P}^*_\Gamma((C^i_\oplus \cup \{r\}, C^i_\ominus))$, where $r = r_j$ for some $0 \leq j \leq m$ such that $\{r_0, \ldots, r_{j-1}\} \subseteq C^i_\oplus \cup C^i_\ominus$ and $r_j \notin C^i_\oplus \cup C^i_\ominus$. Furthermore, let be $C^n = \mathcal{N}_\Gamma(C^{n-1})$ where $(\Pi \setminus (C^{n-1}_\oplus \cup C^{n-1}_\ominus)) \cap R_\Pi(X) = \emptyset$. Clearly, C^n is a total coloring by definition of $\mathcal{N}_\Gamma$. Next, we show that $C_\oplus = R_\Pi(X)$. Then, we can conclude that $C^n = C$ holds. Let be $r \in C_\oplus$. Case 1: r is the "choice rule" selected to obtain C^{i+1} from C^i or Case 2: r was colored with $\oplus$ by $\mathcal{P}_\Gamma$ at some iterative step from C^i to C^{i+1} for some $0 \leq i < n-1$. In Case 1 we have $r \in R_\Pi(X)$ by construction. In Case 2 we have $r \in S(\Gamma, C') \cap \overline{B}(\Gamma, C')$ for some $C^i \sqsubseteq C' \sqsubseteq C^{i+1}$. Then, by Theorem 3.1.1 we have $r \in R_\Pi(X)$. Let be $r \in R_\Pi(X)$ and assume that we have $r \in C^n_\ominus$. Then, we have one of the following cases: Case 1: $r \in C^n_\ominus \setminus C^{n-1}_\ominus$ (r is colored by $\mathcal{N}_\Gamma$) or Case 2: $r \in C'_\ominus$ for some $C^i \sqsubset C' \sqsubseteq C^{i+1}$ for some $0 \leq i < n-1$ (r is colored by $\mathcal{P}_\Gamma$). Another case don't exists since we only have $\mathcal{D}^\oplus_\Gamma$ as choice. In Case 1 we have then a contradiction to $(\Pi \setminus (C^{n-1}_\oplus \cup C^{n-1}_\ominus)) \cap R_\Pi(X) = \emptyset$. In Case 2 we have $r \in \overline{S}(\Gamma, C'') \cup B(\Gamma, C'')$ where $C' = \mathcal{P}_\Gamma(C'')$ holds for a partial coloring C'' such that $C^i \sqsubseteq C'' \sqsubseteq C^{i+1}$ for some $0 \leq i < n-1$. But then, we get the contradiction $r \notin R_\Pi(X)$ by Theorem 3.1.1. Thus, we have $R_\Pi(X) = C_\oplus$ and hence we have $C = C^n$. It remains to show that $C = \mathcal{P}_\Gamma(C)$. But this holds by Corollary 3.3.7.
"⇐": By Corollary 3.3.7, it remains to show that there exist a support graph of (Γ, C). That must be $(C_\oplus, E)$ for some $E \subseteq \Pi \times \Pi$ since C is total. Each vertex in the sequence $(C^i)_{0 \leq i \leq n}$, which is added to some partial coloring C', is supported in (Γ, C'). Thus, we can construct a support graph of (Γ, C) by inductive adding of vertices according to their occurrence in C'. This works similar to proof "←" of Corollary 3.3.26. ■

Proof 3.3.29 Given the same prerequisites as in Theorem 3.3.28, let $(C^i)_{0 \leq i \leq n}$ be a sequence satisfying conditions 1-5 in Theorem 3.3.28. We have to prove $(\mathcal{N}_\Gamma(C^{n-1})_\ominus \setminus C^{n-1}_\ominus) \subseteq \overline{S}(\Gamma, C)$. Assume, there exists an $r \in \mathcal{N}_\Gamma(C^{n-1})_\ominus \setminus C^{n-1}_\ominus$ such that $r \in S(\Gamma, C)$. By $r \in C_\ominus$ and C is closed under $\mathcal{P}_\Gamma$ we must have $r \in B(\Gamma, C)$. Thus, there must exists an $r' \in C_\oplus$ such that r is blocked by r'. By $\mathcal{N}_\Gamma(C^{n-1})_\oplus = C^{n-1}_\oplus$ we have $r' \in C^{n-1}_\oplus$. Thus, we have $r \in B(\Gamma, C^{n-1})$. Since C^{n-1} is closed under $\mathcal{P}_\Gamma$ we must have $r \in C^{n-1}_\ominus$. But this is a contradiction to $r \notin C^{n-1}_\ominus$. Hence, we have $(\mathcal{N}_\Gamma(C^{n-1})_\ominus \setminus C^{n-1}_\ominus) \subseteq \overline{S}(\Gamma, C)$. ■

Proof 3.3.30 Given the same prerequisites as in Theorem 3.3.24, let $(C^i)_{0 \leq i \leq n}$ be a sequence satisfying conditions 1-5 in Theorem 3.3.24. Properties 1.-5. follow analogous to Theorem 3.3.25 by $C \sqsubseteq \mathcal{P}^*_\Gamma(C)$ for a partial coloring C. Property 8 follows analogous to the proof "⇐" of Theorem 3.3.28. For $0 \leq i < n$ the properties 6 and 7 are fulfilled by C^i being closed under $\mathcal{P}_\Gamma$. For $i = n$ the properties 6 and 7 follow by Theorem 3.1.3, by C^n is total, and $C^n_\oplus = R_\Pi(X)$. ■

Proof 3.3.31 This could be proven analogous to the proofs of Corollary 3.3.26 and Theorem 3.3.28. ■

Proof 3.3.32 Proof analogously to theorems 3.3.27 and 3.3.30. ■

Proof 3.3.33 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of logic program Π and C be a partial coloring of Γ. Furthermore, let $(C_\oplus, E)$ be a support graph of (Γ, C) for some $E \subseteq (\Pi \times \Pi)$. By Theorem A.2.2 we have $\mathcal{T}^*_\Gamma(C) = T(C)$. First, we want to construct a graph $((\mathcal{T}^*_\Gamma(C))_\oplus, E')$ for some $E' \subseteq (\Pi \times \Pi)$. Second, we have to show (SG) $((\mathcal{T}^*_\Gamma(C))_\oplus, E')$ is a support graph of (Γ, C) and (M) $((\mathcal{T}^*_\Gamma(C))_\oplus, E')$ is a maximal one . Let $E^0 = E$ and $V^0 = T^0(C)_\oplus = C_\oplus$. Assume that we have constructed $V^i \subseteq \Pi$ and $E^i \subseteq \Pi \times \Pi$ for some $i < \omega$. Now, we want to construct $V^{i+1} \subseteq \Pi$ and $E^{i+1} \subseteq \Pi \times \Pi$. We define $V^{i+1} = V^i \cup (S(\Gamma, T^i(C)) \setminus T^i(C)_\ominus)$ and $E^{i+1} = E^i \cup \left(\{(r', r'') \mid r' \in V^i, r'' \in V^{i+1} \setminus V^i\} \cap E_0\right)$. We define $E' = \bigcup_{i<\omega} E^i$. Clearly, we have $T^{i+1}(C)_\oplus = T^i(C)_\oplus \cup (S(\Gamma, T^i(C)) \setminus T^i(C)_\ominus) = V^{i+1}$ by $V^i = T^i(C)_\oplus$ for all $i < \omega$. Furthermore, we have $\mathcal{T}^*_\Gamma(C)_\oplus = \bigcup_{i<\omega} V^i$ by construction of each V^i and by Theorem A.2.2. It remains to show, that we have constructed a maximal support graph of (Γ, C).
SG: Clearly, E' is a subset of E_0 and acyclic since E is acyclic and there are only edges from V^i to V^{i+1}. Furthermore, by construction and by $r \in S(\Gamma, T^i(C))$ for all $r \in V^{i+1}$ we have for all $r \in \mathcal{T}^*_\Gamma(C)_\oplus$ that $body^+(r) \subseteq \{head(r') \mid (r', r) \in E'\}$ holds.
M: Assume that there exists an $r \in \Pi \setminus \mathcal{T}^*_\Gamma(C)_\oplus$ where $r \notin C_\ominus$ such that $((\mathcal{T}^*_\Gamma(C))_\oplus \cup \{r\}, E'')$ is a support graph of (Γ, C) for some $E'' \subseteq (\Pi \times \Pi)$. By definition of a support graph we have $r \in S(\Gamma, \mathcal{T}^*_\Gamma(C))$ and thus by Theorem A.2.2 we have $r \in S(\Gamma, T(C))$. Hence, there exists an $i < \omega$ such that $r \in S(\Gamma, T^i(C))$. Because $r \notin C_\ominus$ we must have $r \in T^{i+1}(C)_\oplus$ and thus $r \in \mathcal{T}^*_\Gamma(C))_\oplus$ holds. That's a contradiction to $r \in \Pi \setminus \mathcal{P}^*_\Gamma(C)_\oplus$ and thus, $((\mathcal{T}^*_\Gamma(C))_\oplus, E')$ is a maximal support graph of (Γ, C). ■

Proof 3.3.34 This follows directly from Theorem 3.3.33 and from Definition 3.3.3 and 3.3.7. ■

Proof 3.3.35 Let Γ be the *RDG* of logic program Π and let C be a total coloring of Γ.
"⇐": By Corollary 3.3.12 we have to show $C = \mathcal{P}_\Gamma(C)$ and $C = \mathcal{U}_\Gamma(C)$. $C = \mathcal{P}_\Gamma(C)$ holds since $C = C^n$ is closed under $\mathcal{P}_\Gamma$. Analogous to the proof "←" of Theorem 3.3.28 we can construct $(C_\oplus, E)$ as a support graph of (Γ, C) for some $E \subseteq \Pi \times \Pi$. By Corollary 3.3.34 and $C = C^n$ is closed under $\mathcal{V}_\Gamma$ we conclude $C = \mathcal{U}_\Gamma(C)$.
"⇒": Let $C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ be an admissible coloring of Γ for answer set X of Π. Let $\langle r_i \rangle_{0 \leq i < n}$ be an enumeration of $R_\Pi(X)$ such that for all $0 \leq i \leq n$ we have $body^+(r_i) \subseteq head(\{r_j \mid j < i\})$. We define $C^0 = (\mathcal{PV})^*_\Gamma((\emptyset, \emptyset))$ and $C^{i+1} = (\mathcal{PV})^*_\Gamma(C^i_\oplus \cup \{r\}, C'_\ominus)$ where $r = r_j \in \Pi \setminus (C^i_\oplus \cup C^i_\ominus)$ and $\{r_0, \ldots, r_{j-1}\} \subseteq C^i_\oplus \cup C^i_\ominus$ for some $0 \leq j \leq n$. We have to show $C = C^n$ is a total coloring, and $C_\oplus = R_\Pi(X)$. But this follows analogous to the proof of Corollary 3.3.21, paragraph **"Plus"** "⇒". ■

Proof 3.3.36 Properties 1.–7. follow analogous to proof of Theorem 3.3.17. Property 8 follows from the construction of the support graph $(C_\oplus, E)$ of (Γ, C) for some $E \subseteq \Pi \times \Pi$ (see proof of Thm. 3.3.35). ■

Proof 3.3.37 Proof is analogous to Corollary 3.3.21. ■

Proof 3.3.38 Let $\Gamma = (\Pi, E_0, E_1)$ be the *RDG* of logic program Π. Furthermore, let $n = \mid \Pi \mid$ be the size of Π and C be a partial coloring of Γ.
1: Note that checking whether $r \in \Pi$ is supported, unsupported, blocked, or unblocked in (Γ, C) is in $\mathcal{O}(k)$ where $k = \max_{r \in \Pi} |\{(r', r) \in E_0 \cup E_1 : \ r' \in \Pi\}|$. Hence, it is constant w.r.t. n. $\mathcal{P}_\Gamma(C)$ is defined as $C \sqcup (S(\Gamma, C) \cap \overline{B}(\Gamma, C),\ \overline{S}(\Gamma, C) \cup B(\Gamma, C))$. Hence, for every $r \in \Pi$ it has to be computed whether it belongs to $S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ or $\overline{S}(\Gamma, C) \cup B(\Gamma, C)$. Thus, $\mathcal{P}_\Gamma$ is computable in $\mathcal{O}(n)$.
2: We define $k = \max_{r \in \Pi} |\{(r', r) \in E_0 \cup E_1 : \ r' \in \Pi\}|$ as the maximal number of predecessors of a vertex and $k' = \max_{r \in \Pi} |\{(r, r') \in E_0 \cup E_1 : \ r' \in \Pi\}|$ as the maximal number of successors of a vertex. Checking whether $r \in \Pi$ is supported, unsupported, blocked, or unblocked in (Γ, C) is in $\mathcal{O}(k)$. Hence, computing whether $r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ or $r \in \overline{S}(\Gamma, C) \cup B(\Gamma, C)$ is in $\mathcal{O}(k)$. An algorithm for the computation of $\mathcal{P}^*_\Gamma(C)$ is given in Figure A.1. We call a rule $r \in \Pi$ *decided* if the question of the applicability of r is been resolved that is, we have either $r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ or $r \in \overline{S}(\Gamma, C) \cup B(\Gamma, C)$.

```
(01) function propagation(var C, Γ)
(02)        queue := ∅;
(03)        for all r ∈ Π do status(r) := false; endfor;
(04)        for all r ∈ Π do if r ∈ (S(Γ,C) ∩ B̄(Γ,C)) ∪ S̄(Γ,C) ∪ B(Γ,C)
(05)                        then push(r,queue);
(06)                             status(r) := true;
(07)                        endif;
(08)        endfor;
(09)        while queue is not empty do
(10)              r := pop(queue);
(11)              if r ∈ S(Γ,C) ∩ B̄(Γ,C) and r ∉ C⊕
(12)              then if r ∈ C⊖
(13)                   then return false;
(14)                   else C⊕ := C⊕ ∪ {r};
(15)                        for all r' ∈ Π where status(r') = false, (r,r') ∈ E0 ∪ E1 do
(16)                            if r' ∈ (S(Γ,C) ∩ B̄(Γ,C)) ∪ S̄(Γ,C) ∪ B(Γ,C)
(17)                            then push(r',queue);
(18)                                 status(r') := true;
(19)                            endif;
(20)                        endfor;
(21)                   endif;
(22)              endif;
(23)              if r ∈ S̄(Γ,C) ∪ B(Γ,C) and r ∉ C⊖
(24)              then if r ∈ C⊕
(25)                   then return false;
(26)                   else C⊖ := C⊖ ∪ {r};
(27)                        for all r' ∈ Π where status(r') = false, (r,r') ∈ E0 ∪ E1 do
(28)                            if r' ∈ (S(Γ,C) ∩ B̄(Γ,C)) ∪ S̄(Γ,C) ∪ B(Γ,C)
(29)                            then push(r',queue);
(30)                                 status(r') := true;
(31)                            endif;
(32)                        endfor;
(33)                   endif;
(34)              endif;
(35)        endwhile;
```

Figure A.1: Algorithm for computation of $\mathcal{P}^*_\Gamma(C)$

We call a rule $r \in \Pi$ *undecided* if r is not decided. We use $status(r) = true$ for denoting that r is decided and $status(r) = false$ for denoting that r is undecided. The main idea of the algorithm is to start from the colored vertices and going along the successors of the lastly colored vertices to look for vertices which can be colored in the next step. In Step (2) the queue, which contains all vertices which can be colored next, is initialized. Step (3) initializes the status of each vertex $r \in \Pi$ as false. This works in $\mathcal{O}(n)$. Step (4–8) are putting all decided vertices (**if** condition) into the queue. Their status is becoming true in Step (6). Steps (4–8) are computable in $\mathcal{O}(n)$. The **while** loop in Step (09)–(35) takes (iterative) a (decided) vertex r from the queue and considers the cases for coloring this rule. Step (10) takes a rule from the queue. Step (11–22) are handling the case for $\oplus$ coloring of decided rules. If $r \in C_\ominus$ then Step (13) returns a failure and if r is already colored with $\oplus$ then r is no longer considered. Else, r is colored with $\oplus$ in Step (14). All successors of r, which are now decided (Step (16)) but not yet in the queue (their status is false), are taken into the queue and their status is set to $true$ in Step (18). The **for** loop in Step (15) works in $\mathcal{O}(k' * k)$, hence constant w.r.t. n. Thus, Step (11–22) are computable in constant time w.r.t. n. Analogously, Step (23–34) consider the case for $\ominus$ coloring of decided rules. By construction of the queue and by adding only rules r with $status(r) = false$, every rule is added at most one time in the queue. Note that after adding rules into the queue their status becomes $true$. Hence, the **while** loop in Step (09–35) is passed through at most n times. As a conclusion, the function `propagation` computes $\mathcal{P}^*_\Gamma(C)$ in $\mathcal{O}(n)$.

3: For computing $\mathcal{U}_\Gamma(C)$ we must compute a maximal support graph of (Γ, C). This is done by modifying the linear time algorithm of Dowling and Gallier [75] as follows. Let V be the vertex set of a maximal support graph of (Γ, C) where $\Gamma = (\Pi, E_0, E_1)$. Then, V can be computed by the algorithm given in Figure A.2. This algorithm computes V in linear time in size of Π, since every $r \in \Pi$ is entered at most

function MaxSuppGraph(V,C,Γ)
  $V := \emptyset$;
  $queue := \emptyset$;
  for all $r \in \Pi \setminus C_\ominus$ **do** $counter(r) := |body^+(r)|$ **endfor**;
  for all $r \in \Pi \setminus C_\ominus$ **do if** $counter(r) = 0$ **then** $push(r, queue)$; **endif**; **endfor**;
  while $queue$ is not empty **do**
    $r := pop(queue)$;
    for all $(r, r') \in E_0$ **do**
      if $r' \notin C_\ominus$
      then $counter(r') := counter(r') - 1$;
        if $counter(r') = 0$ **then** $push(r', queue)$; **endif**;
      endif;
    endfor;
    $V := V \cup \{r\}$;
  endwhile;

Figure A.2: Algorithm for computing maximal support graphs

once into the queue and the "for loop" in the "while loop" is executed at most m times, where $m = \max_{r \in \Pi} |\{(r, r') \in E_0 | r' \in \Pi\}|$. Hence, $\mathcal{U}_\Gamma$ is computable in $\mathcal{O}(n)$.

4: By Condition 1 and 3, $(\mathcal{PU})_\Gamma$ is computable in $\mathcal{O}(n)$. Hence, $(\mathcal{PU})^*_\Gamma$ is computable in $\mathcal{O}(n^2)$, since we have to iterate $(\mathcal{PU})_\Gamma$ at most n times.

5: For computing $\mathcal{V}_\Gamma(C)$ we have to compute $\mathcal{T}^*_\Gamma(C)_\oplus$. This is done by a modification of the algorithm given in Condition 3. We start with $V = C_\oplus$ instead of $V = \emptyset$ and consider $C_\oplus$ while initializing the counter for every rule $r \in \Pi$. The algorithm is given in Figure A.3. Analogously to Condition 3, $\mathcal{V}_\Gamma$ is computable in $\mathcal{O}(n)$.

6: This follows directly from Condition 1 and 5.

function $\mathcal{T}_{\oplus}(V,C,\Gamma)$
$V := C_{\oplus}$;
$queue := \emptyset$;
for all $r \in \Pi \setminus C_{\ominus}$ **do** $counter(r) := |body^{+}(r) \setminus \{head(r') \mid r' \in C_{\oplus}\}|$; **endfor**;
for all $r \in \Pi \setminus C_{\ominus}$ **do if** $counter(r) = 0$ **then** $push(r, queue)$; **endif endfor**;
while $queue$ is not empty **do**
$r := pop(queue)$;
for all $(r, r') \in E_0$ **do**
if $r' \notin C_{\ominus}$
then $counter(r') := counter(r') - 1$;
if $counter(r') = 0$ **then** $push(r', queue)$; **endif**;
endif;
endfor;
$V := V \cup \{r\}$;
endwhile;

Figure A.3: Algorithm for computing $\mathcal{V}_{\Gamma}(C)$

7: We have $\mathcal{N}_{\Gamma}(C) = (C_{\oplus}, \Pi \setminus C_{\oplus})$. Hence, computing $\mathcal{N}_{\Gamma}$ is in $\mathcal{O}(n)$. ■

A.3.4 Section 3.4

Proof 3.4.1 Let Γ be the *RDG* of logic program Π. We say that $(X, Y) \subseteq (X', Y')$ if $X \subseteq X'$ and $Y \subseteq Y'$ for any $X, Y, X', Y' \subseteq Atm$. Let $C = \mathcal{P}^{*}_{\Gamma}((\emptyset, \emptyset))$ be a partial coloring of Γ. Furthermore, let $\Phi^{\omega}_{\Pi}(\emptyset, \emptyset) = (X^{\omega}, Y^{\omega})$. We have to prove $(X^{\omega}, Y^{\omega}) = (X_C, Y_C)$. According to Definition 3.4.1 and by Theorem A.2.1 it can be easily seen that $(X_C, Y_C) = \bigcup_{i<\omega}(X_{C^i}, Y_{C^i})$ where $C^0 = (\emptyset, \emptyset)$ and $C^{i+1} = \mathcal{P}_{\Gamma}(C^i)$. By definition of Fitting's operator, we obtain $(X^{\omega}, Y^{\omega}) = \bigcup_{i<\omega}(X^i, Y^i)$ where $(X^0, Y^0) = (\emptyset, \emptyset)$,

$$\begin{aligned} X^i &= X^{i-1} \cup \{head(r) \mid r \in \Pi, body^{+}(r) \subseteq X^{i-1}, body^{-}(r) \subseteq Y^{i-1}\}, \text{ and} \\ Y^i &= Y^{i-1} \cup \{q \mid \text{ for all } r \in \Pi, \text{ if } head(r) = q, \text{ then} \\ &\quad body^{+}(r) \cap Y^{i-1} \neq \emptyset \text{ or } body^{-}(r) \cap X^{i-1} \neq \emptyset\}. \end{aligned}$$

"$(X^{\omega}, Y^{\omega}) \subseteq (X_C, Y_C)$": We prove by induction over i that $(X^i, Y^i) \subseteq (X_C, Y_C)$ for all $i < \omega$ hold. For $i = 0$ we have $(X^0, Y^0) = (\emptyset, \emptyset) \subseteq (X_C, Y_C)$. Assume, $(X^k, Y^k) \subseteq (X_C, Y_C)$ holds for all $k \leq i$ for some $i < \omega$ **(IH)**. We have to prove $(X^{i+1}, Y^{i+1}) \subseteq (X_C, Y_C)$. By induction hypotheses **(IH)**, we have

$$\begin{aligned} X^{i+1} &= X^i \cup \{head(r) \mid r \in \Pi, body^{+}(r) \subseteq X^i \subseteq X_C, body^{-}(r) \subseteq Y^i \subseteq Y_C\} \text{ and} \\ Y^{i+1} &= Y^i \cup \{q \mid \text{ for all } r \in \Pi, \text{ if } head(r) = q, \text{ then} \\ &\quad body^{+}(r) \cap Y^i \neq \emptyset \text{ or } body^{-}(r) \cap X^i \neq \emptyset\}. \end{aligned}$$

Let be $head(r) = a \in X^{i+1} \setminus X^i$, then we have $r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C)$. Thus, $r \in C_{\oplus}$ holds by C is closed under $\mathcal{P}_{\Gamma}$ and hence, we have $a \in X_C$. Let be $a \in Y^{i+1} \setminus Y^i$ and let be $r \in \Pi$ such that $head(r) = a$. If $body^{+}(r) \cap Y^i \neq \emptyset$ then $r \in \overline{S}(\Gamma, C)$. If $body^{-}(r) \cap X^i \neq \emptyset$ then $r \in B(\Gamma, C)$. Thus, $a \in Y_C$ holds by C is closed under $\mathcal{P}_{\Gamma}$. Hence, we have $(X^{i+1}, X^{i+1}) \subseteq (X_C, Y_C)$. Thus, we have $(X^{\omega}, Y^{\omega}) \subseteq (X_C, Y_C)$.
"$(X^{\omega}, Y^{\omega}) \supseteq (X_C, Y_C)$": We show by induction over i that $(X_{C^i}, Y_{C^i}) \subseteq (X^{\omega}, Y^{\omega})$ holds for all $i < \omega$. Then, we conclude $(X_C, Y_C) \subseteq (X^{\omega}, Y^{\omega})$. For $i = 0$ we have $(X_{C^0}, Y_{C^0}) = (\emptyset, Atm \setminus head(\Pi)) \subseteq (X^1, Y^1) \subseteq (X^{\omega}, Y^{\omega})$. Assume, $(X_{C^k}, Y_{C^k}) \subseteq (X^{\omega}, Y^{\omega})$ holds for all $k \leq i$ for some $i < \omega$. We have to show that $(X_{C^{i+1}}, Y_{C^{i+1}}) \subseteq (X^{\omega}, Y^{\omega})$ holds where $C^{i+1} = \mathcal{P}_{\Gamma}(C^i)$. Let be $a \in X_{C^{i+1}}$. Then there

exists an $r \in \Pi$ such that $head(r) = a$ and $r \in S(\Gamma, C^i) \cap \overline{B}(\Gamma, C^i)$. Hence, we have $body^+(r) \subseteq X_{C^i}$ and $body^-(r) \subseteq Y_{C^i}$. Thus, we have $a \in X^\omega$ by $(X_{C^i}, Y_{C^i}) \subseteq (X^\omega, Y^\omega)$. Let be $a \in Y_{C^{i+1}}$ then for all $r \in \Pi$ such that $head(r) = a$ we have $r \in \mathcal{P}_\Gamma(C^i)_\ominus$. Thus, $r \in C^i_\ominus \cup \overline{S}(\Gamma, C^i) \cup B(\Gamma, C^i)$. By $C^k \sqsubseteq C^i$ we have $r \in \overline{S}(\Gamma, C^i) \cup B(\Gamma, C^i)$. If $r \in B(\Gamma, C^i)$ then $body^-(r) \cap X_{C^i} \neq \emptyset$. If $r \in \overline{S}(\Gamma, C^i)$ then $body^+(r) \cap Y_{C^i} \neq \emptyset$. Hence, we have $a \in Y^\omega$. Thus, we have proven $(X^\omega, Y^\omega) \supseteq (X_C, Y_C)$. ■

Proof 3.4.2 Let Γ be the *RDG* of logic program Π and C be a partial coloring of Γ such that $C_\ominus \subseteq \overline{S}(\Gamma, C) \cup B(\Gamma, C)$. Furthermore, let (V, E) be a maximal support graph of (Γ, C) for some $E \subseteq \Pi \times \Pi$. First, we show that $Atm \setminus head(V)$ is an unfounded set of Π w.r.t. (X_C, Y_C). Let be $a \in Atm \setminus head(V)$. For all $r \in \Pi$ such that $head(r) = a$, we have to show that one of the following conditions hold: U1 there exists a $p \in body^+(r)$ such that all rules, with p as head, are in $C_\ominus$ or
there exists a $q \in body^-(r)$ where there exists an $r' \in C_\oplus$ such that $head(r') = q$ (That is, we have $r \in \overline{S}(\Gamma, C) \cup B(\Gamma, C)$); U2 there exists a $p \in body^+(r)$ such that $p \in Atm \setminus head(V)$. If $a \in Atm \setminus head(V)$ then either $a \notin head(\Pi)$ or there are (by Definition 3.2.2) one of the following cases for all rules $r \in \Pi$ where $head(r) = a$: Case 1 $body^+(r) \not\subseteq head(V)$; Case 2 $r \in C_\ominus$. If $a \notin head(\Pi)$ then **U1** and **U2** are trivially fulfilled since $\{r \mid head(r) = a\} = \emptyset$.
Case 1: If $body^+(r) \not\subseteq head(V)$ then there exists a $p \in body^+(r)$ such that $p \in Atm \setminus head(V)$ and condition **U2** is fulfilled.
Case 2: If $r \in C_\ominus$ then either $r \in \overline{S}(\Gamma, C)$ or $r \in B(\Gamma, C)$. If $r \in \overline{S}(\Gamma, C)$ then there exists an $p \in body^+(r)$ such that all rules with p as head are in $C_\ominus$. If $r \in B(\Gamma, C)$ then there exists an $r' \in C_\oplus$ such that $head(r') \in body^-(r)$. Thus, condition **U1** is fulfilled. Thus, $Atm \setminus head(V)$ is an unfounded set of Π w.r.t. (X_C, Y_C). Second, each unfounded set w.r.t. (X_C, Y_C) is in $Atm \setminus head(V)$ by maximality of (V, E) and by Definition 3.2.2. Thus, $Atm \setminus head(V)$ is the greatest unfounded set w.r.t. (X_C, Y_C). ■

Proof 3.4.3 Let Γ be the *RDG* of logic program Π and C be a partial coloring of Γ such that $C_\ominus \subseteq \overline{S}(\Gamma, C) \cup B(\Gamma, C)$. Let $\mathcal{U}_\Gamma(C) = C'$ and let (V, E) be a maximal support graph of (Γ, C) for some $E \subseteq \Pi \times \Pi$. Observe that $Atm \setminus head(\Pi \setminus C'_\ominus) = Atm \setminus head(\Pi \setminus (\Pi \setminus V)) = Atm \setminus head(V)$. Thus, by Theorem 3.4.2, $(Atm \setminus head(\Pi \setminus C'_\ominus))$ is the greatest unfounded set w.r.t. (X_C, Y_C). ■

Proof 3.4.4 Let Γ be the *RDG* of logic program Π. Furthermore, let $C = (\mathcal{PU})^*_\Gamma((\emptyset, \emptyset))$ be a partial coloring. We have to show that (X_C, Y_C) is the well-founded model of Π. First, we show the following Lemma:
[FP]: Let C, C' be partial colorings of Γ such that $C_\oplus \subseteq S(\Gamma, C) \cap \overline{B}(\Gamma, C)$ and $C_\ominus \subseteq \overline{S}(\Gamma, C) \cup B(\Gamma, C)$. Then, if $C' = \mathcal{P}_\Gamma(C)$ then $\Phi_\Pi(X_C, Y_C) = (X_{C'}, Y_{C'})$.
Proof of [FP]: Let be $C' = \mathcal{P}_\Gamma(C)$. Then, $C' = (C_\oplus \cup (S(\Gamma, C) \cap \overline{B}(\Gamma, C)), C_\ominus \cup (\overline{S}(\Gamma, C) \cup B(\Gamma, C)))$. By the preconditions we have $C' = (S(\Gamma, C) \cap \overline{B}(\Gamma, C), \overline{S}(\Gamma, C) \cup B(\Gamma, C))$. We obtain

$$\begin{aligned}
\Phi^+_\Pi(X_C, Y_C) &= \{head(r) \mid r \in \Pi, body^+(r) \subseteq X_C, body^-(r) \subseteq Y_C\} \\
&= \{head(r) \mid r \in S(\Gamma, C) \cap \overline{B}(\Gamma, C)\} \\
&= \{head(r) \mid r \in C'_\oplus\} \\
&= X_{C'}, \text{ and} \\
\Phi^-_\Pi(X_C, Y_C) &= \{q \mid \text{ if } head(r) = q \text{ then } body^+(r) \cap Y_C \neq \emptyset \text{ or } body^-(r) \cap X_C \neq \emptyset\} \\
&= \{q \mid \text{ if } head(r) = q \text{ then } r \in \overline{S}(\Gamma, C) \cup B(\Gamma, C)\} \\
&= \{q \mid \text{ if } head(r) = q \text{ then } r \in C'_\ominus\} \\
&= Y_{C'}.
\end{aligned}$$

Hence, $\Phi_\Pi(X_C, Y_C) = (X_{C'}, Y_{C'})$.**q.e.d.**

Second, we show that (X_C, Y_C) is the well-founded model of Π. For this, we give a definition of the well-founded model [167]. The mapping $\mathcal{U}_\Pi$, which assigns false to every atom in an unfounded set, is defined as follows: $\mathcal{U}_\Pi\langle X, Y\rangle = \langle X', Y'\rangle$ where for all atoms A we have: (i) $A \in X'$ if $A \in X$, (ii)

$A \in Y'$ if A is in the greatest unfounded set (w.r.t. Π and $\langle X, Y\rangle$), (iii) A is undefined otherwise. Then, the well-founded model of Π, $\mathcal{W}^{\omega}_{\Pi}\langle\emptyset,\emptyset\rangle$, is defined as follows:

$$\begin{aligned}
\mathcal{W}^{0}_{\Pi}\langle\emptyset,\emptyset\rangle &= \langle\emptyset,\emptyset\rangle \\
\mathcal{W}^{i+1}_{\Pi}\langle\emptyset,\emptyset\rangle &= \Phi_{\Pi}(\mathcal{U}_{\Pi}(\mathcal{W}^{i}_{\Pi}\langle\emptyset,\emptyset\rangle)) \\
\mathcal{W}^{\omega}_{\Pi}\langle\emptyset,\emptyset\rangle &= \cup_{i<\omega}\mathcal{W}^{i}_{\Pi}\langle\emptyset,\emptyset\rangle,
\end{aligned}$$

where Φ_{Π} denotes Fitting's operator. Since all atoms $Atm \setminus head(\Pi)$ are false in the well-founded model, we have $\mathcal{W}^{\omega}_{\Pi}\langle\emptyset,\emptyset\rangle = \mathcal{W}^{\omega}_{\Pi}\langle\emptyset, Atm \setminus head(\Pi)\rangle$. **[FP]** shows the direct correspondence between greatest unfounded sets and the $\mathcal{U}_{\Gamma}$ operator. Corollary 3.4.3 shows the direct correspondence between Fitting's operator and the propagation operator $\mathcal{P}_{\Gamma}$. By induction one can show that (X_C, Y_C) is the well-founded model of Π. ■

Appendix B

Chapter 4

B.1 Appendix of Section 4.1

The following results are observed in the presence of an answer set obtained from a partial coloring C. Note that these theorems still hold when replacing $X \in AS_{(\Pi,<)}(C)$ with $C' \in AC_{(\Pi,<)}(C)$ where $C' = (R_\Pi(X), \Pi \setminus R_\Pi(X), \emptyset)$ and X is an answer set.

Theorem B.1.1 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$, C be a partial coloring of Ψ and $X \in AS_{(\Pi,<)}(C)$. For $r \in \Pi$, we have*

1. *$body^+(r) \subseteq X$, if $r \in S(\Psi, C)$;*
2. *$body^+(r) \not\subseteq X$, if $r \in \overline{S}(\Psi, C)$;*
3. *$body^-(r) \cap X \neq \emptyset$, if $r \in B(\Psi, C)$;*
4. *$body^-(r) \cap X = \emptyset$, if $r \in \overline{B}(\Psi, C)$.*

Proof B.1.1 The proof of the corresponding theorem 3.1.1 for standard logic programs Π can be transferred to ordered logic programs without loss of validity. ■

For admissible colorings, we may turn the above "if" statements into "iff". Note that $X \in AS^\sigma_{(\Pi,<)}(C)$ implies $X \in AS_{(\Pi,<)}(C)$ for every $\sigma \in \{D, B, W\}$.

Theorem B.1.2 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C be a partial coloring of Ψ. Then, we have for every $X \in AS_{(\Pi,<)}(C)$ that*

1. *$S(\Psi, C) \cap \overline{B}(\Psi, C) \subseteq R_\Pi(X)$;*
2. *$\overline{S}(\Psi, C) \cup B(\Psi, C) \subseteq \Pi \setminus R_\Pi(X)$.*

If C is admissible, we have for $\{X\} = AS_{(\Pi,<)}(C)$ that

3. *$S(\Psi, C) \cap \overline{B}(\Psi, C) = R_\Pi(X)$;*
4. *$\overline{S}(\Psi, C) \cup B(\Psi, C) = \Pi \setminus R_\Pi(X)$.*

Proof B.1.2 The proof of the corresponding theorem for standard logic programs Π (cf. Theorem 3.1.3) can be transferred to ordered logic programs without loss of validity. ■

Next, we provide iterative characterizations of our operators $\mathcal{P}_\Psi$ and $(\mathcal{PU})^*_\Psi$. We define $P(C)$ as $P(C) = \bigsqcup_{i<\omega} P^i(C)$ $P^0(C) = C$ and $P^{i+1}(C) = \mathcal{P}_\Psi(P^i(C))$ for $i < \omega$.

Theorem B.1.3 *Let Ψ be the RDG of ordered logic program $(\Pi, <)$ and C a partial coloring of Ψ. Then,*

1. *$P(C)$ exists, if $AC^D_{(\Pi,<)}(C) \neq \emptyset$,*
2. *$P(C)$ is a partial coloring (if $P(C)$ exists),*
3. *$C' \in AC^D_{(\Pi,<)}(C)$ iff $C' \in AC^D_{(\Pi,<)}(P(C))$,*
4. *$C \sqsubseteq P(C)$,*
5. *$P(C)$ is closed under $\mathcal{P}_\Psi$,*
6. *$P(C)$ is the $\sqsubseteq$- smallest partial coloring closed under $\mathcal{P}_\Psi$,*
7. *$P(C) = \mathcal{P}^*_\Psi(C)$.*

Proof B.1.3 Let Ψ be the *RDG* of ordered logic program $(\Pi, <)$ and C a partial coloring of Ψ.
1+3: By $C \sqsubseteq P(C)$ we have $C' \in AC^D_{(\Pi,<)}(C)$ if $C' \in AC^D_{(\Pi,<)}(P(C))$. Let $C' \in AC^D_{(\Pi,<)}(C)$. We prove by induction over i that $P(C)$ exists and $C' \in AC^D_{(\Pi,<)}(P(C))$.
Base: By $P^0(C) = C$ we have that $P^0(C)$ exists and $C' \in AC^D_{(\Pi,<)}(P^0(C))$.
Step: Assume that $P^k(C)$ exists and $C' \in AC^D_{(\Pi,<)}(P^k(C))$ for all $k < i$. We have to prove that $P^i(C)$ exists and $C' \in AC^D_{(\Pi,<)}(P^i(C))$. The existence of $P^i(C)$ is given by Theorem 4.1.4. To prove $C' \in AC^D_{(\Pi,<)}(P^i(C))$ we have to show (a) $P^i(C)_\oplus \subseteq C'_\oplus$, (b) $P^i(C)_\ominus \subseteq C'_\ominus$, (c) $P^i(C)_\oslash \subseteq \overline{S}(\Psi, C')$. Abbreviatory, we write C'' for $P^{i-1}(C)$.
(a): By definition we have $P^i(C)_\oplus = C''_\oplus \cup (S(\Psi, C'') \cup \overline{B}(\Psi, C'') \cup M(\Psi, C''))$. Since $C' \in AC^D_{(\Pi,<)}(C'')$ we have that $C''_\oplus \subseteq C'_\oplus$ and $S(\Psi, C'') \cup \overline{B}(\Psi, C'') \cup M(\Psi, C'') \subseteq S(\Psi, C'') \cup \overline{B}(\Psi, C'') \subseteq R_\Pi(X) = C'_\oplus$. Hence, $P^i(C)_\oplus \subseteq C'_\oplus$.
(b) holds analogous to (a).
(c) Since $C' \in AC^D_{(\Pi,<)}(C'')$ we have $C''_\oslash \subseteq \overline{S}(\Psi, C')$. Hence, we observe $P^i(C)_\oslash = \mathcal{P}_\Psi(C'')_\oslash = C''_\oslash \subseteq \overline{S}(\Psi, C')$. For this reason, $P(C)$ exists and $C' \in AC^D_{(\Pi,<)}(C)$ iff $C' \in AC^D_{(\Pi,<)}(P(C))$.
2: Clearly, $P(C)$ is a partial coloring by definition.
4: $C \sqsubseteq P(C)$ holds by definition of $P(C)$ and by $C'' \sqsubseteq \mathcal{P}_\Psi(C'')$ for any partial coloring C''.
5: The closedness of $P(C)$ under $\mathcal{P}_\Psi$ follows from the definition of $P(C)$.
6: Assume, $P(C)$ and $Q(C)$ are $\sqsubseteq$- smallest partial colorings closed under $\mathcal{P}_\Psi$ and $P(C) \neq Q(C)$. By $P(C) \neq Q(C)$ and $P^0(C) = C = Q^0(C)$, there exists a minimal $i < \omega$ such that $P^i(C) = Q^i(C)$ and $P^{i+1}(C) \neq Q^{i+1}(C)$. But then, $P^{i+1}(C) = \mathcal{P}_\Psi(P^i(C)) = \mathcal{P}_\Psi(Q^i(C)) = Q^{i+1}(C)$ leads to a contradiction to $P(C) \neq Q(C)$. Hence, $P(C)$ is the $\sqsubseteq$- smallest partial coloring closed under $\mathcal{P}_\Psi$.
7: This follows directly from the above given Conditions and Definition 4.1.6. ■

We define $PU(C)$ as $PU(C) = \bigsqcup_{i<\omega} PU^i(C)$ where $PU^0(C) = C$ and $PU^{i+1}(C) = \mathcal{P}_\Psi(\mathcal{U}_\Psi(PU^i(C)))$ for $i < \omega$. Analogous to Theorem B.1.3, we have $PU(C) = (\mathcal{PU})^*_\Psi(C)$ and $(\mathcal{PU})^*_\Psi$ preserves $<^D$–preserving answer sets.

B.2 Proofs of Section 4.1

B.2.1 Section 4.1.2

Proof 4.1.1 Let $\Psi = (\Pi, E_0, E_1, E_2)$ be the *RDG* of ordered program $(\Pi, <)$ and C be a total coloring of Ψ.
"⇐": By Theorem 3.2.4, Condition 1 and 2a imply that C is an admissible coloring. That is, $C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ for an answer set X of Π. We will show, that X is a $<^D$–preserving answer set, then

C is a $<^D$–preserving admissible coloring. We have to show that there exists an enumeration of Π satisfying Condition (D1)–(D3) in Definition 2.4.1. Given a D–height function h as in Condition 2b of this theorem, we find an enumeration $\langle s_j \rangle_{j \in J}$ of $S(\Psi, C)$ such that for all $k, l \in J$ we have $k < l$ if $h(s_k) \leq h(s_l)$. Furthermore, we obtain an enumeration $\langle r_i \rangle_{i \in I}$ of $\Pi = S(\Psi, C) \cup \overline{S}(\Psi, C)$ extending the enumeration $\langle s_j \rangle_{j \in J}$ such that all unsupported rules are inserted into $\langle s_j \rangle_{j \in J}$ according to the given partial order $<$. That is, if $r_i < r_j$ then $j < i$ for all $i, j \in I$ and $r_i, r_j \in \Pi$. Now, we show that $\langle r_i \rangle_{i \in I}$ fulfills Definition 2.4.1.
(D1): Holds by construction of $\langle r_i \rangle_{i \in I}$ and by Condition 1 of a D–height function in Definition 4.1.3.
(D2): Let $r_i \in R_\Pi(X) = C_\oplus$. Then, r_i is in the support graph given in Condition 2a in Theorem 4.1.1. By Condition 2 in Definition 4.1.3 and by Definition 3.2.2, we have $body^+(r_i) \subseteq \{head(r_j) \mid r_j \in R_\Pi(X), j < i\}$. Thus, (D2) holds.
(D3): Let $r_i \in \Pi \setminus R_\Pi(X) = C_\ominus$. If $r_i \in \overline{S}(\Psi, C) \cap C_\ominus$, then Condition (D3a) holds by Theorem B.1.1. If $r_i \in S(\Psi, C) \cap C_\ominus$, then analogous to (D2) Condition (D3b) holds by Condition 3 in Definition 4.1.3.
"⇒": Let C be a $<^D$–preserving admissible coloring. Then, $C = (R_\Pi(X), \Pi \setminus R_\Pi(X))$ where X is a $<^D$–preserving answer set of Π. By X is an answer set, we have that C is an admissible coloring of Ψ and hence, by Theorem 3.2.4, Condition 1 and 2a hold where $(C_\oplus, E_0')$ is a support graph of (Ψ, C). It remains to show the existence of a D–height function h in Condition 2b of this theorem. Let $\langle r_i \rangle_{i \in I}$ be an enumeration of Π such that Condition (D1)–(D3) in Definition 2.4.1 hold. We define a function $h : S(\Psi, C) \to I\!N$ for all $r_i \in S(\Psi, C)$ as follows:

$$h(r_i) = i. \tag{B.1}$$

Now, we show that h is a D–height function of $(S(\Psi, C), E_0', E_1|_{S(\Psi,C)}, E_2|_{S(\Psi,C)})$ that is, h fulfills Condition 1–3 in Definition 4.1.3.
1: If $(r', r) \in E_2$ then $h(r') < h(r)$ by (D1) and (B.1).
2: Let $r, r' \in C_\oplus$ such that $(r', r) \in E_0'$. Then $r, r' \in R_\Pi(X) = C_\oplus$ and $head(r') \in body^+(r)$. By Condition (D2) and (B.1) we have $h(r') < h(r)$ and thus, Condition 2 holds.
3: Let $r \in C_\ominus \cap S(\Psi, C)$. Then, $r \notin R_\Pi(X)$ and $r \in B(\Psi, C)$ by C is an admissible coloring. Thus, by (D3b), there exists an $r' \in C_\oplus$ such that $(r', r) \in E_1|_{S(\Psi,C)}$ and $h(r') < h(r)$. For this reason, h is a D–height function. ■

Proof 4.1.2 Proof analogous to proof of Theorem 4.1.1. ■

Proof 4.1.3 This Corollary follows by Theorem 3.2.4, Theorem 4.1.1, (Theorem 4.1.2,) and the following theorem proven in [177]: Let Π be a logic program and X a set of atoms. Then, X is a preferred answer set of $(\Pi, \emptyset)$ iff X is an answer set of Π. ■

B.2.2 Section 4.1.3

Proof 4.1.4 Let Ψ be the *RDG* of ordered logic program $(\Pi, <)$ and C a partial coloring of Ψ. Furthermore, let $C'' = (R_\Pi(X), \Pi \setminus R_\Pi(X), \emptyset) \in AC^D_{(\Pi,<)}(C)$ for $<^D$–preserving answer set X. We have to prove that $C' = \mathcal{P}_\Psi(C)$ is a partial coloring that is (1) $C'_\oplus \cap C'_\ominus = \emptyset$, (2) $C'_\ominus \cap C'_\oslash = \emptyset$, and (3) $C'_\oplus \cap C'_\oslash = \emptyset$. Abbreviatory, we write $S, \overline{S}, B, \overline{B}$, and M for $S(\Psi, C), \overline{S}(\Psi, C), B(\Psi, C), \overline{B}(\Psi, C)$, and $M(\Psi, C)$.
1: By C is a partial coloring, we detect

$$\begin{aligned}
C'_\oplus \cap C'_\ominus &= \big[C_\oplus \cup (S \cap \overline{B} \cap M)\big] \cap \big[C_\ominus \cup \overline{S} \cup (B \cap S \cap M)\big] \\
&= (C_\oplus \cap C_\ominus) \cup (C_\oplus \cap \overline{S}) \cup (C_\oplus \cap B \cap S \cap M) \cup \\
&\quad (S \cap \overline{B} \cap M \cap C_\ominus) \cup (S \cap \overline{B} \cap M \cap \overline{S}) \cup (S \cap \overline{B} \cap M \cap B) \\
&= (C_\oplus \cap \overline{S}) \cup (C_\oplus \cap B \cap S \cap M) \cup (S \cap \overline{B} \cap M \cap C_\ominus) \\
&\subseteq (C_\oplus \cap \overline{S}) \cup (C_\oplus \cap B \cap S) \cup (S \cap \overline{B} \cap C_\ominus) \\
&\subseteq (C_\oplus \cap \overline{S}) \cup (C_\oplus \cap B) \cup (S \cap \overline{B} \cap C_\ominus) \\
&= \emptyset
\end{aligned}$$

The last equality follows by Theorem B.1.2 and $C_\oplus \subseteq R_\Pi(X)$. Hence, $C'_\oplus \cap C'_\ominus = \emptyset$.
2: By C is a partial coloring, we find out

$$\begin{array}{rcl} C'_\ominus \cap C'_\oslash & = & \big[C_\ominus \cup \overline{S} \cup (B \cap S \cap M)\big] \cap \big[C_\oslash \setminus \overline{S}\big] \\ & = & \big[C_\ominus \cap (C_\oslash \setminus \overline{S})\big] \cup \big[\overline{S} \cap (C_\oslash \setminus \overline{S})\big] \cup \big[B \cap S \cap M \cap (C_\oslash \setminus \overline{S})\big] \\ & = & B \cap S \cap (C_\oslash \setminus \overline{S}) \cap M \\ & \subseteq & B \cap S \cap (C_\oslash \setminus \overline{S}) \\ & = & \emptyset. \end{array}$$

The last equality we can conclude since $AC^D_{(\Pi,<)}(C) \neq \emptyset$ and $C_\oslash \subseteq \overline{S}(\Psi, C'')$, but $B \cap S \subseteq S(\Psi, C'')$.
3: By C is a partial coloring, we observe

$$\begin{array}{rcl} C'_\oplus \cap C'_\oslash & = & C'_\oplus \cap (C_\oslash \setminus \overline{S}) \\ & \subseteq & C'_\oplus \cap C_\oslash \\ & = & (C_\oplus \cup (S \cap \overline{B} \cap M)) \cap C_\oslash \\ & = & (S \cap \overline{B} \cap M) \cap C_\oslash \\ & \subseteq & (S \cap \overline{B}) \cap C_\oslash \end{array}$$

By Theorem B.1.2 we have $S \cap \overline{B} \subseteq R_\Pi(X)$. By $C_\oslash \cap R_\Pi(X) = \emptyset$ we can conclude that $S \cap \overline{B} \cap C_\oslash = \emptyset$. Hence, $C'_\oplus \cap C'_\oslash = \emptyset$. ■

Proof 4.1.5 The existence of $\mathcal{P}^*_\Psi$ follows directly from Theorem B.1.3. ■

Proof 4.1.6 Let Ψ be the *RDG* of ordered logic program $(\Pi, <)$ and C be a partial coloring of Ψ.
1: Let $AC^D_{(\Pi,<)}(C') \neq \emptyset$ and $C \sqsubseteq C'$. Also, we obtain $AC^D_{(\Pi,<)}(C) \neq \emptyset$. By Theorem B.1.3 we prove by induction over i that $P(C) \sqsubseteq P(C')$. Then, we conclude $\mathcal{P}^*_\Psi(C) \sqsubseteq \mathcal{P}^*_\Psi(C')$.
Base: $P^0(C) = C \sqsubseteq C' = P^0(C')$.
Step: Assume, $P^k(C) \sqsubseteq P^k(C')$ for all $k \leq i < \omega$. Abbreviatory, we write C^1 instead of $P^i(C)$ and C^2 instead of $P^i(C')$. We have to prove that $P^{i+1}(C) \sqsubseteq P^{i+1}(C')$. That is, $\mathcal{P}_\Psi(C^1) \sqsubseteq \mathcal{P}_\Psi(C^2)$. By $C^1 \sqsubseteq C^2$, we have $C^1_\oplus \subseteq C^2_\oplus$, $C^1_\ominus \subseteq C^2_\ominus$ and $C^1_\oslash \subseteq (C^2_\ominus \cap \overline{S}(\Psi, C^2)) \cup C^2_\oslash$. We have to show (1.1) $\mathcal{P}_\Psi(C^1)_\oplus \subseteq \mathcal{P}_\Psi(C^2)_\oplus$, (1.2) $\mathcal{P}_\Psi(C^1)_\ominus \subseteq \mathcal{P}_\Psi(C^2)_\ominus$, and (1.3) $\mathcal{P}_\Psi(C^1)_\oslash \subseteq \big[\mathcal{P}_\Psi(C^2)_\ominus \cap \overline{S}(\Psi, \mathcal{P}_\Psi(C^2))\big] \cup \mathcal{P}_\Psi(C^2)_\oslash$.
(1.1.) and **(1.2.)** hold by $C^1 \sqsubseteq C^2$ and by $S(\Psi, C^1) \subseteq S(\Psi, C^2)$, $\overline{B}(\Psi, C^1) \subseteq \overline{B}(\Psi, C^2)$, $B(\Psi, C^1) \subseteq B(\Psi, C^2)$, $\overline{S}(\Psi, C^1) \subseteq \overline{S}(\Psi, C^2)$, and $M(\Psi, C^1) \subseteq M(\Psi, C^2)$ as consequences of $C^1 \sqsubseteq C^2$.
(1.3): Since $C^1 \sqsubseteq C^2$ we have that $C^1_\oslash \subseteq (C^2_\ominus \cap \overline{S}(\Psi, C^2)) \cup C^2_\oslash$. We have to show that $C^1_\oslash \setminus \overline{S}(\Psi, C^1) = \mathcal{P}_\Psi(C^1)_\oslash \subseteq \big[\mathcal{P}_\Psi(C^2)_\ominus \cap \overline{S}(\Psi, \mathcal{P}_\Psi(C^2))\big] \cup \mathcal{P}_\Psi(C^2)_\oslash$. We obtain

$$\begin{array}{rcl} & & \big[\mathcal{P}_\Psi(C^2)_\ominus \cap \overline{S}(\Psi, \mathcal{P}_\Psi(C^2))\big] \cup \mathcal{P}_\Psi(C^2)_\oslash \\ = & & \big[\big(C^2_\ominus \cup \overline{S}(\Psi, C^2) \cup (B(\Psi, C^2) \cap S(\Psi, C^2) \cap M(\Psi, C^2))\big) \cap \overline{S}(\Psi, \mathcal{P}_\Psi(C^2))\big] \cup (C^2_\oslash \setminus \overline{S}(\Psi, C^2)) \\ \supseteq & & \big[\big(C^2_\ominus \cup \overline{S}(\Psi, C^2)\big) \cap \overline{S}(\Psi, \mathcal{P}_\Psi(C^2))\big] \cup \big(C^2_\oslash \setminus \overline{S}(\Psi, C^2)\big) \\ \supseteq & & \big[\big(C^2_\ominus \cup \overline{S}(\Psi, C^2)\big) \cap \overline{S}(\Psi, C^2)\big] \cup \big(C^2_\oslash \setminus \overline{S}(\Psi, C^2)\big) \\ \supseteq & & \overline{S}(\Psi, C^2) \cup C^2_\oslash \\ \supseteq & & (C^2_\ominus \cap \overline{S}(\Psi, C^2)) \cup C^2_\oslash \\ = & & C^1_\oslash \\ \supseteq & & C^1_\oslash \setminus \overline{S}(\Psi, C^1). \end{array}$$

2: This holds by Theorem B.1.3. ■

Proof 4.1.7 Let Ψ be the *RDG* of ordered logic program $(\Pi, <)$ and C be a partial coloring of Ψ. If (Ψ, C) has a support graph, then there exists a maximal support graph of (Ψ, C) and thus, $\mathcal{U}_\Psi(C)$ exists. ■

Proof 4.1.8 Let $\Psi = (\Pi, E_0, E_1, E_2)$ be the *RDG* of ordered logic program $(\Pi, <)$. Let C be an admissible coloring of Ψ and $(C_\oplus, E_0')$ be a support graph of (Ψ, C) for some $E_0' \subseteq E_0$. Abbreviatory, we write $r \in C'$ if $r \in C'_\oplus \cup C'_\ominus$ for some partial coloring C' of Ψ. Note that $C_\oslash = \emptyset$ and $\mathcal{H}^i_\Psi((\emptyset,\emptyset,\emptyset), C)_\oslash = \emptyset$ for all $i < \omega$.
"⇒": First, we want to define a function $h : S(\Psi, C) \to I\!N$ with the help of $\mathcal{H}^*_\Psi$ and second, we will show that constructed h is a D–height function. For this, we need a specification of the edges E_0' of our support graph. $\mathcal{H}^*_\Psi$ provides us these edges. W.l.o.g. let

$$E_0' = \{(r', r) \mid r' \in \mathcal{H}^j_\Psi((\emptyset,\emptyset,\emptyset), C)_\oplus, r \in \mathcal{H}^{i+1}_\Psi((\emptyset,\emptyset,\emptyset), C)_\oplus, r \in S(\Psi, \mathcal{H}^i_\Psi((\emptyset,\emptyset,\emptyset), C)), j \leq i < \omega\} \cap E_0.$$

We define $h : S(\Psi, C) \to I\!N$ as

$h(r) = 0$ if $r \in S(\Psi, C) \cap \mathcal{H}^0_\Psi((\emptyset,\emptyset,\emptyset), C)$ and

$h(r) = i + 1$ if $r \in S(\Psi, C) \cap (\mathcal{H}^{i+1}_\Psi((\emptyset,\emptyset,\emptyset), C) \setminus \mathcal{H}^i_\Psi((\emptyset,\emptyset,\emptyset), C))$ for $0 < i < \omega$.

By $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = C$ we have the existence of an h- value for all $r \in S(\Psi, C)$. Next, we have to show that h is a D–height function of $(S(\Psi, C), E_0', E_1|_{S(\Psi,C)}, E_2|_{S(\Psi,C)})$, that is, we have to show Condition 1-3 in Definition 4.1.3. Let $r \in S(\Psi, C)$.
1: If $(r', r) \in E_2$ then $r' > r$. Furthermore, there exist $j, k < \omega$ such that $r' \in \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C) \setminus \mathcal{H}^j_\Psi((\emptyset,\emptyset,\emptyset), C)$ and $r \in \mathcal{H}^{k+1}_\Psi((\emptyset,\emptyset,\emptyset), C) \setminus \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C)$. By definition of $\mathcal{H}_\Psi$, r is maximal in $(\Psi, \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C))$ and thus $k > j$, since r' must be colored because r is maximal. Hence, $h(r) > h(r')$ is fulfilled by construction of h since vertices are included in $\mathcal{H}^k_\Psi$ (for some $k < \omega$) if they are maximal.
2: Let $r', r \in C_\oplus$, then by definition of $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C)$ there exist some minimal $j, k < \omega$ such that $r' \in \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C)_\oplus, r' \in S(\Psi, \mathcal{H}^j_\Psi((\emptyset,\emptyset,\emptyset), C)), r \in \mathcal{H}^{k+1}_\Psi((\emptyset,\emptyset,\emptyset), C)_\oplus, r \in S(\Psi, \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C))$. If $(r', r) \in E_0'$, then by construction of E_0' and by definition of h we have $j + 1 = h(r') < h(r) = k + 1$.
3: Let $r \in C_\ominus \cap S(\Psi, C)$. Then by $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = C$, there exists a $k < \omega$ such that $r \in B(\Psi, \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C)) \cap M(\Psi, \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C))$ by $r \notin \overline{S}(\Psi, \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C)) \subseteq \overline{S}(\Psi, C)$. Hence, there exists an $r' \in \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C)_\oplus \subseteq C_\oplus$ such that $(r', r) \in E_1|_{S(\Psi,C)}$ and $h(r') \leq k < k + 1 = h(r)$ by definition of h.
"⇐": Clearly, $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) \sqsubseteq C$ holds by definition of $\mathcal{H}^*_\Psi$. Thus, it remains to show that $C \sqsubseteq \mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C)$ that is for all $r \in C_\oplus \cup C_\ominus$ exists an $i < \omega$ such that $r \in \mathcal{H}^i_\Psi((\emptyset,\emptyset,\emptyset), C)$. Let h be a D–height function of $(S(\Psi, C), E_0', E_1|_{S(\Psi,C)}, E_2|_{S(\Psi,C)})$. For showing $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) \sqsupseteq C$, we look for an extension g of h such that g comprehends Π. Furthermore, g is used to show that every $r \in \Pi$ is in $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C)$. There exists a function $g : \Pi \to I\!N$ such that $g(r) < g(r')$ if $h(r) < h(r')$ for all $r, r' \in S(\Psi, C)$ and $g(r) < g(r')$ if $r > r'$. That is, unsupported vertices in C are inserted according to the given partial order $<$. W.l.o.g. we assume that g is a bijective mapping $g : \Pi \to \{0, \ldots, n\}$ for some $n < \omega$. Next, we prove $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) \sqsupseteq C$ by induction over the values of g.
Base: Let $r \in C_\oplus \cup C_\ominus$ such that $g(r) = 0$. If $r \in C_\oplus$, then $r \in S(\Psi, (\emptyset,\emptyset,\emptyset)) \cap M(\Psi, (\emptyset,\emptyset,\emptyset))$ since g respects the D–height function h and C is admissible. Thus, $r \in \mathcal{H}^1_\Psi((\emptyset,\emptyset,\emptyset), C)_\oplus$. Analogous, if $r \in C_\ominus$, then $r \in \overline{S}(\Psi, (\emptyset,\emptyset,\emptyset)) \cap M(\Psi, (\emptyset,\emptyset,\emptyset))$. Hence, $r \in \mathcal{H}^1_\Psi((\emptyset,\emptyset,\emptyset), C)_\ominus$. For this reason, $r \in \mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C)$ for all $r \in \Pi$ such that $g(r) = 0$.
Step: Assume, for all $r \in C_\oplus \cup C_\ominus$ such that $g(r) \leq j$ we have $r \in \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C)$ for some $k < \omega$. We have to show that for $r \in C_\oplus \cup C_\ominus$ such that $g(r) = j + 1$ we have $r \in \mathcal{H}^{k+1}_\Psi((\emptyset,\emptyset,\emptyset), C)$. Let $r \in C_\oplus$, then we have $r \in S(\Psi, \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C)) \cap M(\Psi, \mathcal{H}^k_\Psi((\emptyset,\emptyset,\emptyset), C))$ by construction of g and because g respects the D–height function h (Condition 1-2 in Definition 4.1.3) and C is admissible. Thus,

$r \in \mathcal{H}_\Psi^{k+1}((\emptyset,\emptyset,\emptyset),C)_\oplus$. Let $r \in C_\ominus$. Then, $r \in M(\Psi, \mathcal{H}_\Psi^k((\emptyset,\emptyset,\emptyset),C))$ by definition of g. Furthermore, either $r \in \overline{S}(\Psi,C)$ or $r \in S(\Psi,C) \cap B(\Psi, \mathcal{H}_\Psi^k((\emptyset,\emptyset,\emptyset),C))$ by Condition 3 of a D–height function in Definition 4.1.3. In both cases we have $r \in \mathcal{H}_\Psi^{k+1}((\emptyset,\emptyset,\emptyset),C)_\ominus$. Thus, $\mathcal{H}_\Psi^*((\emptyset,\emptyset,\emptyset),C) = C$ ■

Proof 4.1.9 Let $\Psi = (\Pi, E_0, E_1, E_2)$ be the *RDG* of ordered logic program $(\Pi, <)$ and let C be a total coloring of Ψ. This theorem follows by Theorem 4.1.8, Theorem 4.1.1 and Theorem 3.3.12 on page 31. Note that in Theorem 3.3.12 we deal with 2-ary colorings and a modified $\mathcal{P}_\Psi$ operator. By C is total and thus $C_\oslash = \emptyset$ and by $\overline{S}(\Psi,C) \cup (B(\Psi,C) \cap S(\Psi,C)) = \overline{S}(\Psi,C) \cup B(\Psi,C)$, we can transfer the result from Theorem 3.3.12to our approach. Thus, it remains to show by Theorem 4.1.1 that there exists a D–height function of $(S(\Psi,C), E_0', E_1|_{S(\Psi,C)}, E_2|_{S(\Psi,C)})$ where $(C_\oplus, E_0')$ is a support graph of (Ψ, C) for some $E_0' \subseteq E_0$ iff $\mathcal{H}_\Psi^*((\emptyset,\emptyset,\emptyset),C) = C$. But this holds by Theorem 4.1.8. ■

Proof 4.1.10 Let $\Psi = (\Pi, E_0, E_1, E_2)$ be the *RDG* of ordered logic program $(\Pi, <)$. Let C be a total coloring of Ψ. Let $\mathcal{H}_\Psi^*((\emptyset,\emptyset,\emptyset),C) = C$, we have to show that $\mathcal{U}_\Psi(C) = C$. For this, we show that $(C_\oplus, E)$ is a maximal support graph of (Ψ, C) for some $E \subseteq \Pi \times \Pi$. Then, we can conclude by C is total that $\mathcal{U}_\Psi(C) = (C_\oplus, \Pi \setminus C_\oplus, C_\oslash \setminus (\Pi \setminus C_\oplus)) = (C_\oplus, C_\ominus, \emptyset) = C$. At first, we define the set of edges for the searched support graph. Let be $E = \bigcup_{i<\omega} E^i \subseteq \Pi \times \Pi$ where $E^0 = \emptyset$ and $E^{i+1} = \{(r', r) \mid r' \in \mathcal{H}_\Psi^j((\emptyset,\emptyset,\emptyset),C)_\oplus, r \in \mathcal{H}_\Psi^{i+1}((\emptyset,\emptyset,\emptyset),C)_\oplus, j \leq i < \omega\} \cap E_0$. Next, we show that $(C_\oplus, E)$ is a support graph of (Ψ, C). We observe (i) $(C_\oplus, E)$ is a 0-subgraph by construction, (ii) $(C_\oplus, E)$ is acyclic by construction, (iii) For all $r \in C_\oplus$ there exists an $j < \omega$ such that $r \in \mathcal{H}_\Psi^{j+1}((\emptyset,\emptyset,\emptyset),C)_\oplus$.
By $r \in S(\Psi, \mathcal{H}_\Psi^j((\emptyset,\emptyset,\emptyset),C))$ we have $body^+(r) \subseteq \{head(r') \mid r' \in \mathcal{H}_\Psi^j((\emptyset,\emptyset,\emptyset),C)_\oplus\} \subseteq \{head(r') \mid (r', r) \in E\}$. Hence, $(C_\oplus, E)$ is a maximal support graph of (Ψ, C). ■

Proof 4.1.11 This corollary follows by Theorem 4.1.9 and 4.1.10. ■

Proof 4.1.12 This theorem follows directly from Corollary 4.1.11. ■

Proof 4.1.13 Given the prerequisites in Theorem 4.1.12, let $(C^i)_{0 \leq i \leq n}$ be a sequence satisfying conditions 1-5 in Theorem 4.1.12. Condition 1 holds by definition of $\mathcal{C}_\Psi^\circ$. Condition 2 holds by definition of $\mathcal{C}_\Psi^\circ$and $C_\oslash^i = \emptyset$ for all $0 \leq i \leq n$. Condition 3 holds by Condition 2. Condition 4 holds by $AC^D_{(\Pi,<)}(C^n) \neq \emptyset$ and by Condition 3. We observe that (Ψ, C^n) has a support graph by $C^n = \mathcal{H}_\Psi^*(C^0, C^n)$ and $C^n = C$. (For details, see construction in proof of Theorem 4.1.10.) By $C^i \sqsubseteq C^n$ and hence, $C_\oplus^i \subseteq C_\oplus^n$ we have that (Ψ, C^i) has a support graph. Thus, Condition 5 holds. ■

Proof 4.1.14 Let $\Psi = (\Pi, E_0, E_1, E_2)$ be the *RDG* of ordered logic program $(\Pi, <)$ and let C be a total coloring of Ψ. Abbreviatory, we write $r \in C'$ if $r \in C'_\oplus \cup C'_\ominus$ for a partial coloring C' of Ψ. Note that $C_\oslash = \emptyset$ and for all $0 \leq i \leq n$ we have $C_\oslash^i = \emptyset$.
"⇒": Let C be a $<^D$–preserving admissible coloring of Ψ. Then, $\mathcal{H}_\Psi^*((\emptyset,\emptyset,\emptyset),C) = C$ holds by Corollary 4.1.11. Next, we construct by using $\mathcal{H}_\Psi^*$ a sequence of partial colorings satisfying Condition 1-3 in Theorem 4.1.14. Then, we will show that C^n is total and $C^n = C$. Let $C^0 = \mathcal{P}_\Psi^*((\emptyset,\emptyset,\emptyset))$ and assume, C^k is constructed for all $k \leq i < \omega$. Let $j < \omega$ such that $C^i \sqsupseteq \mathcal{H}_\Psi^j((\emptyset,\emptyset,\emptyset),C)$ and $C^i \not\sqsupseteq \mathcal{H}_\Psi^{j+1}((\emptyset,\emptyset,\emptyset),C)$. That is $\mathcal{H}_\Psi^{j+1}((\emptyset,\emptyset,\emptyset),C)$ contains vertices which are not detected by C^i but all vertices from $\mathcal{H}_\Psi^j((\emptyset,\emptyset,\emptyset),C)$ are detected by C^i. For our choice operator we choose a vertex r from $\mathcal{H}_\Psi^{j+1}((\emptyset,\emptyset,\emptyset),C)$ which is not detected by C^i. If $r \in \mathcal{H}_\Psi^{j+1}((\emptyset,\emptyset,\emptyset),C)_\oplus$ then $r \in R_\Pi(X)$ and we have to take $\mathcal{C}_\Psi^\oplus(C^i)$. Otherwise, $r \in \mathcal{H}_\Psi^{j+1}((\emptyset,\emptyset,\emptyset),C)_\ominus$ then $r \in \Pi \setminus R_\Pi(X)$ and we have to take $\mathcal{C}_\Psi^\ominus(C^i)$. We define

$$C^{i+1} = \begin{cases} \mathcal{P}_\Psi^*((C_\oplus^i \cup \{r\}, C_\ominus^i, C_\oslash^i)) & \text{if } r \notin C^i, r \in \mathcal{H}_\Psi^{j+1}((\emptyset,\emptyset,\emptyset),C)_\oplus; \\ \mathcal{P}_\Psi^*((C_\oplus^i, C_\ominus^i \cup \{r\}, C_\oslash^i \setminus \{r\})) & \text{if } r \notin C^i, r \in \mathcal{H}_\Psi^{j+1}((\emptyset,\emptyset,\emptyset),C)_\ominus. \end{cases}$$

By construction and by Corollary 4.1.11($\mathcal{H}_\Psi^*((\emptyset,\emptyset,\emptyset),C) = C$) we have that there exists a $n < \omega$ such that

C^n is total. Furthermore, $C^n = (R_\Pi(X), \Pi \setminus R_\Pi(X), \emptyset) = C$ by Thm. B.1.2 and by construction of C^{i+1}.
"⇐": By Corollary 4.1.11 it remains to show $\mathcal{P}_\Psi(C^n) = C^n$, but this holds by C^n is closed under $\mathcal{P}_\Psi$. ■

Proof 4.1.15 Let $\Psi = (\Pi, E_0, E_1, E_2)$ be the *RDG* of ordered logic program $(\Pi, <)$ and C be a total coloring of Ψ.
"⇒": Let C by a $<^D$–preserving admissible coloring of Ψ. Then $\mathcal{H}^*_\Psi((\emptyset,\emptyset,\emptyset), C) = C$ by Corollary 4.1.11. By using $\mathcal{H}^*_\Psi$ We construct a sequence $(C^i)_{0\leq i\leq n}$ such that Condition 1–3 of Theorem 4.1.15 hold. Abbreviatory, we write $r \in C^i$ if $r \in C^i_\oplus \cup C^i_\ominus \cup C^i_\oslash$. Let $C^0 = (\mathcal{PU})^*_\Psi((\emptyset,\emptyset,\emptyset))$. Assume, C^k for all $k \leq i < \omega$ is constructed. Let $j < \omega$ such that $C^i \sqsupseteq \mathcal{H}^j_\Psi((\emptyset,\emptyset,\emptyset), C)$ and $C^i \not\sqsupseteq \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C)$. That is $\mathcal{H}^{j+1}_\Psi$ contains vertices which are not detected by C^i but all vertices from $\mathcal{H}^j_\Psi$ are detected by C^i. Given C^i our choice operator chooses an $r \in \mathcal{H}^{j+1}_\Psi \setminus C^i$. If $r \in \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C)_\oplus$ then we take $\mathcal{D}^\oplus_\Psi$, and if $r \in \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C)_\ominus \cap \overline{S}(\Psi, \mathcal{H}^{j+1}_\Psi)$ then we take $\mathcal{D}^\oslash_\Psi$. Note that all $r \in \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C)_\ominus \setminus \overline{S}(\Psi, \mathcal{H}^{j+1}_\Psi)$ must be colored by $\mathcal{P}_\Psi$. Note that for all $r \in \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C)$ we have that $r \in M(\Psi, \mathcal{H}^j_\Psi((\emptyset,\emptyset,\emptyset), C))$ and thus, $r \in M(\Psi, C^i)$. We define

$$C^{i+1} = \begin{cases} (\mathcal{PU})^*_\Psi((C^i_\oplus \cup \{r\}, C^i_\ominus, C^i_\oslash)) \\ \text{if } r \in R_\Pi(X), r \notin C^i, r \in \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C)_\oplus; \\ (\mathcal{PU})^*_\Psi((C^i_\oplus, C^i_\ominus, C^i_\oslash \cup \{r\})) \\ \text{if } r \in (\Pi \setminus R_\Pi(X)), r \notin C^i, r \in \overline{S}(\Psi, \mathcal{H}^{j+1}_\Psi), r \in \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C)_\ominus. \end{cases}$$

Assume that we could not construct a total C^n. That is, for all $r \in \Pi \setminus C^n$ we have $r \in \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C)_\ominus \setminus \overline{S}(\Psi, \mathcal{H}^{j+1}_\Psi((\emptyset,\emptyset,\emptyset), C))$ and thus they are blocked. But then, these vertices r had to be colored by $\mathcal{P}_\Psi$ or by $\mathcal{U}_\Psi$ since $C_\oplus \subseteq C^n_\oplus$. Hence, there exists an $n < \omega$ such that C^n is total. Thus, we have $C^n = (R_\Pi(X), \Pi \setminus R_\Pi(X), \emptyset) = C$ by C^n is closed under $\mathcal{P}_\Psi$ and $\mathcal{U}_\Psi$, Thm. B.1.2, and by construction of C^{i+1}.
"⇐": For proving that C is a $<^D$–preserving admissible coloring we use Theorem 4.1.1. That is, we have to show Conditions 1-2b in Theorem 4.1.1.
1"⊆": Let C' be a partial coloring $(\emptyset,\emptyset,\emptyset) \sqsubseteq C' \sqsubseteq C$ extracted from the sequence $(C^i)_{0\leq i\leq n}$. Let $r \in C_\oplus$, then r is colored either by $\mathcal{D}^\oplus_\Psi(C')$ or by $\mathcal{P}_\Psi(C')$. If r is colored by $\mathcal{D}^\oplus_\Psi(C')$, then $r \in S(\Psi, C')$. Thus, $r \in S(\Psi, C)$ by $C' \sqsubseteq C$. Assume, $r \in B(\Psi, C)$. Then, r is maximal in (Ψ, C) by $r \in M(\Psi, C') \subseteq M(\Psi, C)$. Since C is closed under $\mathcal{P}_\Psi$, we must have $r \in C_\ominus$. That's a contradiction to $r \in C_\oplus$ and hence, $r \in S(\Psi, C) \cap \overline{B}(\Psi, C)$. If r is colored by $\mathcal{P}_\Psi(C')$, then $r \in S(\Psi, C') \cap \overline{B}(\Psi, C')$. Therefore, $r \in S(\Psi, C) \cap \overline{B}(\Psi, C)$ by $C' \sqsubseteq C$.
1"⊇": Let $r \in S(\Psi, C) \cap \overline{B}(\Psi, C)$. Since $C = C^n$ is closed under $\mathcal{P}_\Psi$ and r is maximal in C, r must be colored with $\oplus$ by $\mathcal{P}_\Psi$. Hence, $r \in C_\oplus$.
2a: Next, we want to construct a support graph. For this we use the given sequence $(C^i)_{0\leq i\leq n}$ of coloring vertices. It can be easily seen that there exists an enumeration $\langle r_i\rangle_{i\in I}$ of $C_\oplus$ such that for all $i, j \in I$ we have $j < i$ whenever r_j is colored before r_i in the sequence $(C^k)_{0\leq k\leq n}$. More precisely, for partial colorings $C' \sqsubseteq C'' \sqsubseteq C$ extracted from the sequence $(C^i)_{0\leq i\leq n}$ we have $r_j \in C'_\oplus$, $r_i \in C''_\oplus$, but $r_i \notin C'_\oplus$. If a set R of vertices is colored at the same time, i.e. by $\mathcal{P}_\Psi$, then their arranging into the enumeration among their selves is arbitrary. If $r_i \in C''_\oplus$, then $r_i \in S(\Psi, C')$. Therefore, $body^+(r_i) \subseteq \{head(r_j) \mid j < i\}$. Furthermore, we define $E'_0 = \{(r_i, r_j) \mid i < j\} \cap E_0$. It can be easily seen that $(C_\oplus, E'_0)$ is a support graph of (Ψ, C).
2b: With the help of Condition 2a we will show that there exists a D–height function. Assume, E'_0 as in (2a) constructed and let $\langle r_i\rangle_{i\in I}$ be an enumeration of $S(\Psi, C)$ which is built as the enumeration of $C_\oplus$ in (2a). Define $h : S(\Psi, C) \to I\!N$ as $h(r_i) = i$. We will show that h is a D–height function of $(S(\Psi, C), E'_0, E_1|_{S(\Psi,C)}, E_2|_{S(\Psi,C)})$. If $(r', r) \in E_2$ and $r', r \in S(\Psi, C)$ then there exists $i, j \in I$ such that $r \widehat{=} r_i$, $r' \widehat{=} r_j$ and $j < i$ by supported rules are colored within the sequence $(C^i)_{0\leq i\leq n}$ only if they are maximal. Thus, $h(r') < h(r)$. Analogous to "2a" we have if $r \in C_\oplus$ then $h(r') < h(r)$ if $(r', r) \in E_0$ and $r' \in C_\oplus$. Let $r \in C_\ominus \cap S(\Psi, C)$, then r is colored by $\mathcal{P}_\Psi$. Thus, $r \in B(\Psi, C) \cap M(\Psi, C)$ and we have

that there exists an $r' \in C_{\oplus}$ such that $(r', r) \in E_1$ and $h(r') < h(r)$. Therefore, h is a D–height function of $(S(\Psi, C), E'_0, E_1|_{S(\Psi,C)}, E_2|_{S(\Psi,C)})$. ■

Proof 4.1.16 Given the prerequisites in Theorem 4.1.15, let $(C^i)_{0\leq i\leq n}$ be a sequence satisfying conditions 1-3 in Theorem 4.1.15. Condition 1–3 hold by definition of $(\mathcal{PU})^*_\Psi$ and $\mathcal{D}^\circ_\Psi$. Condition 4 holds by $AC^D_\Pi(C) = AC^D_\Pi(C^n) \neq \emptyset$ and by Condition 3. By C^i is closed under $\mathcal{U}_\Psi$ we have that there exists a support graph of (Ψ, C^i) (Condition 5). Condition 6 and 7 hold since C^{i+1} is closed under $\mathcal{P}_\Psi$. Condition 8 holds by C^{i+1} is closed under $\mathcal{P}_\Psi$ and $\mathcal{U}_\Psi$ and by definition of $\mathcal{D}^\oslash_\Psi$. Condition 9+10 follow analogous to proof (2a) and (2b) in "⇐" Theorem 4.1.15. ■

Proof 4.1.17 Proof analogous to Theorem 4.1.15. ■

Appendix C

Chapter 5: Proofs

C.1 Section 5.1

Proof 5.1.1 Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered logic programs. Assume that $<_1 = <_2$ does not hold. W.l.o.g., assume that there exist $r, r' \in \Pi_1$ such that $r' <_1 r$ but $r' \not<_2 r$. That is, r is preferred over r' in Π_1 but not in Π_2.

We will first create a Π' such that $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi') \neq \emptyset$.

Let $\Pi' = \{head(r) \leftarrow x \mid r \in \Pi_1 \cup \Pi_2\} \cup \{r^x : x \leftarrow\}$, where x does not occur in Π_1 and Π_2, i.e. $x \notin Atm(\Pi_1 \cup \Pi_2)$. Now we have $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi') = \{X\}$ where $X = \{head(r) \mid r \in \Pi_1 \cup \Pi_2\} \cup \{x\}$. X obviously satisfies all of $(\Pi_1 \cup \Pi')^X$, and $(\Pi_2 \cup \Pi')^X$, and no subset of X can satisfy the reducts. Moreover, any model of $\Pi_1 \cup \Pi'$, and $\Pi_2 \cup \Pi'$ is a superset of X, therefore X is the only answer set.

Furthermore, also $AS^\sigma((\Pi_1 \cup \Pi', <_1)) = AS^\sigma((\Pi_2 \cup \Pi', <_2)) = \{X\}$ holds for $\sigma \in \{D, W, B\}$. Note that $\Pi_1 \cap \Pi' = \Pi_2 \cap \Pi' = \emptyset$, and so no rule of Π' occurs in $<_1$ and $<_2$. We can thus find an rule ordering for $\Pi_1 \cup \Pi'$ (resp. $\Pi_2 \cup \Pi'$), in which all rules of Π' are ordered first (starting with $x \leftarrow$), followed by an ordering of rules in Π_1 (resp. Π_2) which is compatible with $<_1$ (resp. $<_2$). It is easy to verify that such an ordering satisfies all of the conditions for the D–, W–, and B–semantics, respectively.

We will now create another small ordered logic program $(\Pi'', <'')$, which will inhibit X being a preferred answer set if $r' <_1 r$ holds, and do nothing if it does not. We assume that y is a new symbol, i.e. $y \notin Atm(\Pi_1 \cup \Pi_2 \cup \Pi \cup \{x\})$.

$$(\Pi'', <'') = \left\{ \begin{array}{l} r \\ r' \\ r^x : x \leftarrow \\ r^y : y \leftarrow not\ x \\ r^x <'' r' \\ r <'' r^y \end{array} \right\}$$

Let us now examine the preferred answer sets of $P_1 = (\Pi_1^*, <_1^*) = (\Pi_1 \cup \Pi' \cup \Pi'', <_1 \cup <'')$ and $P_2 = (\Pi_2^*, <_2^*) = (\Pi_2 \cup \Pi' \cup \Pi'', <_2 \cup <'')$.

For P_1, observe that no ordering as described above can be extended for X such that the conditions for the D–, W–, and B–semantics are meet: Since $r^x <_1^* r^y$, r^y must occur before r^x in the ordering, but all semantics also require that r^x is ordered before r^y. So $AS^\sigma(P_1) = \emptyset$ for $\sigma \in \{D, W, B\}$.

For P_2, $r^x \not<_2^* r^y$ holds, therefore we can extend the ordering by placing r^y just after r^x. Moreover, r and r' are not necessarily in Π_2; if they are not, they can be placed at the very end of the ordering, and they cannot violate any of the conditions of the three semantics. Therefore $AS^\sigma(P_2) = \{X\}$ for $\sigma \in \{D, W, B\}$.

We obtain that $(\Pi_1, <_1) \not\equiv_s^\sigma (\Pi_2, <_2)$, where $\sigma \in \{D, W, B\}$, from which the theorem follows. ■

Proof 5.1.2 If $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ and $PR((\Pi_1, <_1)) \neq PR((\Pi_2, <_2))$ held, then assume w.l.o.g. that $r \in PR((\Pi_1, <_1))$ but $r \notin PR((\Pi_2, <_2))$. So some $r' \in \Pi_1$ exists, such that $r <_1 r'$ or $r' <_1 r$, but neither $r <_2 r'$ nor $r' <_2 r$, implying $<_1 \neq <_2$, which contradicts Theorem 5.1.1. ■

Proof 5.1.3 Follows directly from Theorem 5.1.1. ■

Proof 5.1.4 Assume that $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ and $AS(\Pi_1) \not\equiv_s AS(\Pi_2)$. From Theorem 5.1.1 it follows that $<_1 = <_2$ holds.

Since $\Pi_1 \not\equiv_s \Pi_2$, there exists some logic program Π such that $AS(\Pi_1 \cup \Pi) \neq AS(\Pi_2 \cup \Pi)$. W.l.o.g., assume that $X \in AS(\Pi_1 \cup \Pi)$ and $X \notin AS(\Pi_2 \cup \Pi)$. X cannot be in $AS^\sigma((\Pi_2 \cup \Pi, <_2))$ and not in $AS^\sigma((\Pi_1 \cup \Pi, <_1))$, since $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$

But, as we will show next, we can construct a Π' such that as before a X' exists such that $X' \in AS(\Pi_1 \cup \Pi')$, $X' \notin AS(\Pi_2 \cup \Pi')$, but now $X' \in AS^\sigma((\Pi_1 \cup \Pi, <_1))$, $X' \notin AS^\sigma((\Pi_2 \cup \Pi, <_2))$, contradicting $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$.

We construct Π' as follows, where x is a new atom, i.e. $x \notin Atm(\Pi_1 \cup \Pi_2 \cup \Pi)$:

$$\Pi' = \Pi \cup \left\{ \begin{array}{ll} r^x : x \leftarrow & \\ r_i' : h \leftarrow B, x & \text{for all } r_i : h \leftarrow B \text{ in } \Pi_1 \cup \Pi \end{array} \right\}$$

Now it is easy to see that $X' = X \cup \{x\}$ is an answer set of $\Pi_1 \cup \Pi'$: It is a model of $\Pi_1 \cup \Pi$ and also satisfies rule r^x and all rules r_i'. Moreover, since no $Y \subset X$ satisfies $(\Pi_1 \cup \Pi)^X$, also no $Y' \subset X'$ can satisfy $(\Pi_1 \cup \Pi')^{X'} \supset (\Pi_1 \cup \Pi)^X$.

We next show that X' is not an answer set of $\Pi_2 \cup \Pi'$: Since X is not an answer set of $\Pi_2 \cup \Pi$, it is (i) either not a model of $\Pi_2 \cup \Pi$, or (ii) X is a model of $\Pi_2 \cup \Pi$, and some $Y \subset X$ is a model of $(\Pi_2 \cup \Pi)^X$. In case (i), a rule $r \in \Pi_2 \cup \Pi$ is not satisfied by X. But r is also in $\Pi_2 \cup \Pi'$ and since x is a new symbol, X' does not satisfy r either and so is neither a model nor an answer set of $\Pi_2 \cup \Pi'$. In case (ii), X' is also a model of $\Pi_2 \cup \Pi'$, and $(\Pi_2 \cup \Pi')^{X'} \supset (\Pi_2 \cup \Pi)^X$. If $Y \subset X$ is a model of $(\Pi_2 \cup \Pi)^X$, then $Y' = Y \cup \{x\} \subset X'$ is a model of $(\Pi_2 \cup \Pi')^{X'}$, and so $X' \notin AS(\Pi_2 \cup \Pi')$.

Since $X' \notin AS(\Pi_2 \cup \Pi')$, also $X' \notin AS^\sigma(\Pi_2 \cup \Pi')$. As a final step, we observe that $X' \in AS^\sigma(\Pi_1 \cup \Pi')$: There exists an enumeration ordering r^x before an enumeration of all r_i', which is compatible with the conditions of the respective semantics, followed by an ordering of $\Pi_1 \cup \Pi$ which is compatible with $<_1$. It is easy to see that it is possible to find such an ordering for the r_i', as $AS(\Pi^*) = AS^\sigma((\Pi^*, \emptyset)$ holds for all $\sigma \in \{D, W, B\}$ and all programs Π^*. For each condition that has to be met for some $r_i \in \Pi_1 \cup \Pi$, there is some r_j' with an appropriate head atom occurring earlier in the enumeration.

Summarizing, we have shown $X' \in AS^\sigma(\Pi_1 \cup \Pi')$ and $X' \notin AS^\sigma(\Pi_2 \cup \Pi')$, contradicting the assumption that $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ holds. ■

Proof 5.1.5 Assume that $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ are ordered logic programs such that $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$, that $Cont(\Pi_1) \neq Cont(\Pi_2)$, and w.l.o.g. that $r \in Cont(\Pi_1)$ but $r \notin Cont(\Pi_2)$. So by definition $X \in AS(\Pi_1 \cup \Pi)$ exists for some Π such that $r \in R_{\Pi_1 \cup \Pi}(X)$. By Theorem 5.1.4 also $X \in AS(\Pi_2 \cup \Pi)$ holds, and therefore $r \notin \Pi_2$, otherwise $r \in R_{\Pi_2 \cup \Pi}(X)$ and therefore $r \in Cont(\Pi_2)$ would hold.

We will now construct a $(\Pi', <')$ based on Π, such that $X' = X \cup \{x\}$ is a preferred answer set of $(\Pi_1, <_1) \cup (\Pi', <')$ as follows.

$$(\Pi', <') = \Pi \cup \left\{ \begin{array}{lll} r^x : x \leftarrow & & \\ r^{a_i} : a \leftarrow x & \text{for all } a_i \in X \setminus \{head(r)\} & \\ r^y : y \leftarrow not\ h & \text{where } h = head(r) & \\ r_h & \text{for all } r_h \in \Pi_1 \cup \Pi_2 & \\ & \text{such that } head(r_h) = head(r) \text{ and } r_h \neq r & \\ r_h <' r^y & \text{for all } r_h \in \Pi_1 \cup \Pi_2 & \\ & \text{such that } head(r_h) = head(r) \text{ and } r_h \neq r & \end{array} \right\}$$

The intention is to create a rule r^y, which is blocked in X' and can only be blocked by r (which is not involved in the preference relation) in an order preserving way. The preference is then extended in a way such that r^y is preferred to all rules, apart from r, that could block r^y, so that Π_1, which contains r, can have X' as preferred answer set by ordering r before r^y, while Π_2 cannot order any rule before r^y which is capable of blocking r^y. Hence, the existence of a preferred answer set depends on the existence of r.

Additionally, we must take care that X' really is a preferred answer set of $(\Pi_1, <_1) \cup (\Pi', <')$. To this end we include all elements of X (apart from the head atom of r) as quasi-facts (rules r^{a_i}), which are not affected by the preference relation. We do not include them as real facts, as such facts may occur in the original program and hence in the preference relation. Note also that r is not involved in any preference order, by virtue of Corollary 5.1.2 and the fact that $r \notin \Pi_2$. We are therefore free to order r as the first of all rules in $\Pi_1 \cup \Pi$, acting also as a quasi-fact. This is possible, since r is a generating rule and the B-semantics fully decouples preference handling from rule application. In this particular case, all atoms in $body^+(r)$ apart from $head(r)$ are derived by quasi-facts, and if $head(r)$ occurs in $body^+(r)$, there must be another generating rule in $\Pi_1 \cup \Pi$ the head of which is equal to $head(r)$. For the B-semantics, this rule may occur after r in the enumeration. For D- and W-semantics, this would not be admissible.

Now, since $X \in AS(\Pi_1 \cup \Pi) = AS(\Pi_2 \cup \Pi)$ and $r \in R_{\Pi_1 \cup \Pi}(X)$, we have that $head(r) \in X$, and so rule r^y is satisfied by X', as are all other rules in $\Pi_1 \cup \Pi'$. We can also verify that no $Y' \subset X'$ is a model of $(\Pi_1 \cup \Pi')^{X'}$, since no $Y \subset X$ is a model of $(\Pi_1 \cup \Pi)^X$. Therefore $X' \in AS(\Pi_1 \cup \Pi')$, and since $\Pi_1 \equiv_s \Pi_2$ (due to Theorem 5.1.4), also $X' \in AS(\Pi_2 \cup \Pi')$.

We can find an enumeration of $\Pi_1 \cup \Pi'$, such that r^x is the first rule, followed by all r^{a_i}. We then order r, followed by r^y, followed by an ordering of all remaining rules which is compatible with $<_1$. This enumeration is clearly compatible with $<_1 \cup <'$, and for each rule $r' \in (\Pi_1 \cup \Pi') \setminus R_{\Pi_1 \cup \Pi'}(X')$ there is either an r^{a_i} or r earlier in the enumeration such that the head of r^{a_i} or r occurs in $body^-(r')$ (note that $\{head(r'') \mid r'' \in \{r^{a_i}, r\}\} \cup \{x\} = X'$). So both conditions of Definition 2.4.3 are satisfied and therefore $X' \in AS^B((\Pi_1, <_1) \cup (\Pi', <'))$.

On the other hand, for $\Pi_2 \cup \Pi'$ such an enumeration cannot be found. One rule r' with $head(r') = head(r)$ must be ordered before r^y, yet since $r \notin \Pi_2 \cup \Pi'$, for any such rule $r' <' r^y$ holds, requiring that r^y occurs before all of these rules. Hence $X' \notin AS^B((\Pi_1, <_1) \cup (\Pi', <'))$, contradicting the assumption $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$. ■

Proof 5.1.6 Assume that $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ are ordered logic programs such that $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ ($\sigma \in \{D, W\}$), that $Cont(\Pi_1) \setminus \{r \in \Pi_1 \mid head(r) \in body^+(r)\} \neq Cont(\Pi_2) \setminus \{r \in \Pi_2 \mid head(r) \in body^+(r)\}$, and w.l.o.g. that $r \in Cont(\Pi_1) \setminus \{r \in \Pi_1 \mid head(r) \in body^+(r)\}$ but $r \notin Cont(\Pi_2) \setminus \{r \in \Pi_2 \mid head(r) \in body^+(r)\}$.

The reasoning is almost the same as in the proof of Theorem 5.1.5. We observe that some $X \in AS(\Pi_1 \cup \Pi)$ exists for some Π such that $r \in R_{\Pi_1 \cup \Pi}(X)$. By Theorem 5.1.4 also $X \in AS(\Pi_2 \cup \Pi)$ holds, and therefore $r \notin \Pi_2$.

We construct the same $(\Pi', <')$ as in the proof of Theorem 5.1.5, and observe that for $X' = X \cup \{x\}$, $X' \in AS(\Pi_1 \cup \Pi')$ and $X' \in AS(\Pi_2 \cup \Pi')$ hold.

Again, we find the same enumeration of $\Pi_1 \cup \Pi'$ as in the proof of Theorem 5.1.5: r^x is the first rule, followed by all r^{a_i}, r, r^y, followed by an ordering of all remaining rules which is compatible with $<_1$. Again, this enumeration is compatible with $<_1 \cup <'$, and for each rule $r' \in (\Pi_1 \cup \Pi') \setminus R_{\Pi_1 \cup \Pi'}(X')$ with non-empty $body^-(r')$ there is either an r^{a_i} or r earlier in the enumeration such that the head of r^{a_i} or r occurs in $body^-(r')$, so conditions 1 and 3 of Definition 2.4.1 resp. Definition 2.4.2 are satisfied. Concerning condition 2, we note first that, for rule r which is in $R_{\Pi_1 \cup \Pi'}(X')$, $head(r) \notin body^+(r)$. So each atom in $body^+(r)$ occurs in the head of some r^{a_i}. For all other rules in $R_{\Pi_1 \cup \Pi'}(X')$, each atom in $body^+(r)$ occurs in the head of some r^{a_i} or in the head of r. Here, it is necessary to require $head(r) \notin body^+(r)$ since in the D-and W-semantics all atoms in $body^+(r)$ must be derived by a rule which occurs earlier in the enumeration. In our enumeration, $head(r)$ is indeed not derived by a rule occurring earlier. So all conditions of Definition 2.4.1 resp. Definition 2.4.2 are satisfied and therefore $X' \in AS^\sigma((\Pi_1, <_1) \cup (\Pi', <'))$ for

$\sigma \in \{D, W\}$.

On the other hand, for $\Pi_2 \cup \Pi'$ such an enumeration cannot be found. One rule r' with $head(r') = head(r)$ must be ordered before r^y, yet since $r \notin \Pi_2 \cup \Pi'$, for any such rule $r' <' r^y$ holds, requiring that r^y occurs before all of these rules. Therefore $X' \notin AS^\sigma((\Pi_1, <_1) \cup (\Pi', <'))$, contradicting the assumption $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$, for $\sigma \in \{D, W\}$. ■

Proof 5.1.7 Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered logic programs such that

C 1: $\Pi_1 \equiv_s \Pi_2$,

C 2: $<_1 = <_2$, and

C 3: $Cont(\Pi_1) = Cont(\Pi_2)$.

Assume that $(\Pi_1, <_1) \equiv_s^B (\Pi_2, <_2)$ does not hold. Then, there exists a $(\Pi, <)$ such that $AS^B((\Pi_1 \cup \Pi, <_1 \cup <)) \neq AS^B((\Pi_2 \cup \Pi, <_2 \cup <))$. W.l.o.g. assume $X \in AS^B((\Pi_1 \cup \Pi, <_1 \cup <))$ such that $X \notin AS^B((\Pi_2 \cup \Pi, <_2 \cup <))$. Note that $X \in AS((\Pi_2 \cup \Pi))$ since $\Pi_1 \equiv_s \Pi_2$ and $X \in AS(\Pi_1 \cup \Pi)$. Furthermore, there exists an enumeration $E_1 = \langle s_j \rangle_{j \in J}$ of $\Pi_1 \cup \Pi$ for which X is $<^B$- preserving.

Next, we construct an enumeration of $\Pi_2 \cup \Pi$ for which X is $<^B$– preserving, which is a contradiction to the assumption $X \notin AS^B((\Pi_2 \cup \Pi, <_2 \cup <))$.

There exists an enumeration $E_2 = \langle r_i \rangle_{i \in I_1}$ of $(\Pi_2 \cap \Pi_1) \cup \Pi$ such that for all $i, j \in I_1, k, l \in J$ where $r_i \hat{=} s_k, r_j \hat{=} s_l$ we have $i < j$ iff $k < l$. That is, $\langle r_i \rangle_{i \in I_1}$ is the restriction of E_1 to the rules which $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have in common. Note that in this enumeration all rules r are included, which satisfy at least one of the following conditions:

- $r \in \Pi$,
- $r \in R_{\Pi_2 \cup \Pi}(X)$ (because of Condition C 3), or
- r is involved in $<_2$ that is there exists an $r' \in \Pi_2$ such that $r' <_2 r$ or $r <_2 r'$ holds (because of Condition C 2).

Hence, for the remaining rules $r \in \Pi_2 \setminus (\Pi_1 \cup \Pi)$ we have that r is not involved in $<_2 \cup <$ and r fulfills one of the following conditions:

Case 1: $r \notin R_{\Pi_2 \cup \Pi}(X)$ and $body^+(r) \not\subseteq X$, or

Case 2: $r \notin R_{\Pi_2 \cup \Pi}(X)$, $body^+(r) \subseteq X$ and $body^-(r) \cap X \neq \emptyset$.

We can add rules from Case 1 arbitrarily into the enumeration $E_2 = \langle r_i \rangle_{i \in I_1}$. In Case 2 we have that $body^-(r) \cap \{head(r_j) \mid r_j \in E_2\} \neq \emptyset$, since all generating rules for X are in E_2. Hence, we can insert all rules $r \in \Pi_2 \setminus (\Pi_1 \cup \Pi)$ at the end of the enumeration E_2. Thus, we get an $<^B$–preserving enumeration $E = \langle r_i \rangle_{i \in I}$ of $\Pi_2 \cup \Pi$, which is a contradiction to the assumption. ■

Proof 5.1.8 This proof is similar to the proof for Theorem 5.1.7. We assume that the conditions hold and that the programs are not strongly $<^\sigma$-equivalent for $\sigma \in \{D, W\}$. Take a preferred answer set of an admissible extension of one program, which is not a preferred answer set of the same admissible extension of the other program. Starting from the order preserving enumeration of the preferred answer set, we construct an order preserving enumeration for the same preferred answer set of the admissible extension of the other program, contradicting our assumption. The difference to Theorem 5.1.7 is the way how we construct the order preserving enumeration.

Let $\sigma \in \{D, W\}$ be fixed, and let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be ordered logic programs such that

C 1: $\Pi_1 \equiv_s \Pi_2$,

C 2: $<_1 = <_2$, and

C 3: $Cont(\Pi_1)\backslash\{r \in \Pi_1 \cup \Pi \mid head(r) \in body^+(r)\} = Cont(\Pi_2)\backslash\{r \in \Pi_2 \cup \Pi \mid head(r) \in body^+(r)\}$.

Assume that $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$ does not hold. Then, there exists a $(\Pi, <)$ such that $AS^\sigma((\Pi_1 \cup \Pi, <_1 \cup <)) \neq AS^\sigma((\Pi_2 \cup \Pi, <_2 \cup <))$. W.l.o.g. assume $X \in AS^\sigma((\Pi_1 \cup \Pi, <_1 \cup <))$ such that $X \notin AS^\sigma((\Pi_2 \cup \Pi, <_2 \cup <))$. Note that $X \in AS((\Pi_2 \cup \Pi))$ since $\Pi_1 \equiv_s \Pi_2$ and $X \in AS(\Pi_1 \cup \Pi)$. Furthermore, there exists an enumeration $E_1 = \langle s_j \rangle_{j \in J}$ of $\Pi_1 \cup \Pi$ for which X is $<^\sigma$- preserving.

Next, we construct an enumeration of $\Pi_2 \cup \Pi$ for which X is $<^\sigma$– preserving, which is a contradiction to the assumption $X \notin AS^\sigma((\Pi_2 \cup \Pi, <_2 \cup <))$.

There exists an enumeration $E_2 = \langle r_i \rangle_{i \in I_1}$ of $(\Pi_2 \cap \Pi_1) \cup \Pi$ such that for all $i, j \in I_1, k, l \in J$ where $r_i \hat{=} s_k, r_j \hat{=} s_l$ we have $i < j$ iff $k < l$. That is, $\langle r_i \rangle_{i \in I_1}$ is the restriction of E_1 to the rules which $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have in common. Note that in this enumeration all rules r are included, which satisfy at least one of the following conditions:

- $r \in \Pi$,
- $r \in R_{\Pi_2 \cup \Pi}(X) \setminus \{r \in \Pi_2 \cup \Pi \mid head(r) \in body^+(r)\}$ (because of Condition C 3), or
- r is involved in $<_2$ that is there exists an $r' \in \Pi_2$ such that $r' <_2 r$ or $r <_2 r'$ holds (because of Condition C 2).

Hence, for the remaining rules $r \in \Pi_2 \setminus (\Pi_1 \cup \Pi)$ we have that r is not involved in $<_2 \cup <$ and r fulfills one of the following conditions:

Case 1: $r \in R_{\Pi_2 \cup \Pi}(X)$ and $head(r) \in body^+(r)$,

Case 2: $r \notin R_{\Pi_2 \cup \Pi}(X)$ and $body^+(r) \not\subseteq X$, or

Case 3: $r \notin R_{\Pi_2 \cup \Pi}(X)$, $body^+(r) \subseteq X$ and $body^-(r) \cap X \neq \emptyset$.

In Case 1 we have that $body^+(r) \subseteq \{head(r_j) \mid r_j \in E_2\}$, hence, such rules can be inserted at the end of enumeration E_2. Rules from Case 2 can be added arbitrarily to the enumeration E_2. In Case 3 we have that $body^-(r) \cap \{head(r_j) \mid r_j \in E_2\} \neq \emptyset$. Hence, we can insert all rules $r \in \Pi_2 \setminus (\Pi_1 \cup \Pi)$ at the end of the enumeration E_2. Thus, we get an $<^\sigma$–preserving enumeration $E = \langle r_i \rangle_{i \in I}$ of $\Pi_2 \cup \Pi$, which is a contradiction to the assumption. For this reason we have $(\Pi_1, <_1) \equiv_s^\sigma (\Pi_2, <_2)$. ■

Proof 5.1.9 This Corollary follows from Theorem 5.1.4, 5.1.5, 5.1.1 and from 5.1.7. ■

Proof 5.1.10 This Corollary follows from Theorem 5.1.4, 5.1.6, 5.1.1 and from 5.1.8. ■

Proof 5.1.11 Let $P_1 = (\Pi_1, <_1)$ and $P_2 = (\Pi_2, <_2)$.

If $\sigma = B$, then $SOE^B(P_1) = SOE^B(P_2)$ is equivalent to the facts that $\Pi_1 \equiv_s \Pi_2$, $<_1 = <_2$, and $Cont(\Pi_1) = Cont(\Pi_2)$. By Corollary 5.1.9 these conditions are equivalent to $P_1 \equiv_s^B P_2$.

If $\sigma \in \{D, W\}$, then $SOE^\sigma(P_1) = SOE^\sigma(P_2)$ is equivalent to the facts that $\Pi_1 \equiv_s \Pi_2$, $<_1 = <_2$, and $Cont(\Pi_1) \setminus \{r \in \Pi_1 \mid head(r) \in body^+(r)\} = Cont(\Pi_2) \setminus \{r \in \Pi_2 \mid head(r) \in body^+(r)\}$. By Corollary 5.1.10 these conditions are equivalent to $P_1 \equiv_s^B P_2$. ■

Proof 5.1.12 (a) follows from Corollary 5.1.9 and 5.1.10. (b) follows from Corollary 5.1.10. ■

Proof 5.1.13 It is common knowledge that checking whether an interpretation I is an answer set of a given program is feasible in polynomial time, cf. [57].

In [34], it was shown that checking whether an interpretation I is a $<^B$-preferred answer set is polynomial, by virtue of an algorithm called "FULL-ORDER," which is an enhanced topological sorting method. Here, we will give deterministic variants of this algorithm, dealing also with $<^D$- and $<^W$- preferred answer sets. In these algorithms, the initial graphs $G(P, A)$ consist of rules as vertices, with an arc (r, r') if $r' < r$. The arcs therefore point from preferred rules to non-preferred rules. A source vertex of a graph is a vertex to which no arc points.

Algorithm Enumerate-D
Input: ordered logic program $P = (\Pi, <)$,
$A \in AS(\Pi)$
Output: yes, if $A \in AS^D(P)$, with a supporting enumeration of rules;
no otherwise
Method:
1.a. Construct graph $G(P, A) = (V, E)$, where $V = \Pi$ and
$E = \{(r, r') \mid r' < r, r, r' \in \Pi\}$, labeling each vertex r as follows:
g (generating) if $r \in R_\Pi(A)$
i (irrelevant) if $body^+(r) \not\subseteq A$
z (zombie) otherwise
1.b. Initialize $T := \emptyset$ and $E := \langle\rangle$
2. If $G(P, A)$ has no vertices, output yes and E, and stop.
3. Let O be the set of source vertices in $G(P, A)$, such that
r is labeled i or
r is labeled g and $body^+(r) \subseteq T$ or
r is labeled z and $body^-(r) \cap T \neq \emptyset$
4. If $O = \emptyset$, output no and stop.
5. a. $T := T \cup \{head(r) \mid r \in O$ and r labeled $g\}$
5. b. Append an arbitrary enumeration of O to E.
5. c. Delete from $G(P, A)$ all vertices in O and all arcs with vertices in O.
6. Continue at 2.

Algorithm Enumerate-W
Input: ordered logic program $P = (\Pi, <)$,
$A \in AS(\Pi)$
Output: yes, if $A \in AS^W(P)$, with a supporting enumeration of rules;
no otherwise
Method:
1.a. Construct graph $G(P, A) = (V, E)$, where $V = \Pi$ and
$E = \{(r, r') \mid r' < r, r, r' \in \Pi\}$, labeling each vertex r as follows:
g (generating) if $r \in R_\Pi(A)$
i (irrelevant) if $body^+(r) \not\subseteq A$
z (zombie) otherwise
1.b. Initialize $T := \emptyset$ and $E := \langle\rangle$
2. If $G(P, A)$ has no vertices, output yes and E, and stop.
3. Let O be the set of source vertices in $G(P, A)$, such that
r is labeled i or
r is labeled g and $body^+(r) \subseteq T$ or $head(r) \in T$ or
r is labeled z and $body^-(r) \cap T \neq \emptyset$ or $head(r) \in T$
4. If $O = \emptyset$, output no and stop.
5. a. $T := T \cup \{head(r) \mid r \in O$ and r labeled $g\}$
5. b. Append an arbitrary enumeration of O to E.
5. c. Delete from $G(P, A)$ all vertices in O and all arcs with vertices in O.
6. Continue at 2.

Algorithm Enumerate-B
Input: ordered logic program $P = (\Pi, <)$,
$A \in AS(\Pi)$
Output: yes, if $A \in AS^B(P)$, with a supporting enumeration of rules;

no otherwise

Method:

1.a. Construct graph $G(P, A) = (V, E)$, where $V = \Pi$ and
$E = \{(r, r') \mid r' < r, r, r' \in \Pi\}$, labeling each vertex r as follows:
g (generating) if $r \in R_\Pi(A)$
i (irrelevant) if $body^+(r) \not\subseteq A$ or $head(r) \in A$
z (zombie) otherwise

1. b. Initialize $T := \emptyset$ and $E := \langle\rangle$

2. If $G(P, A)$ has no vertices, output yes and E, and stop.

3. Let O be the set of source vertices in $G(P, A)$, such that
r is labeled i or
r is labeled g or
r is labeled z and $body^-(r) \cap T \neq \emptyset$

4. If $O = \emptyset$, output no and stop.

5. a. $T := T \cup \{head(r) \mid r \in O$ and r labeled $g\}$

5. b. Append an arbitrary enumeration of O to E.

5. c. Delete from $G(P, A)$ all vertices in O and all arcs with vertices in O.

6. Continue at 2.

Each of these algorithms computes an enumeration for an ordered logic program $P = (\Pi, <)$ and an answer set of Π for the respective semantics if one exists or returns "no," if none exists. Each of these algorithms runs within a polynomial time bound. ■

Proof 5.1.14 Membership: Let $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$ be two ordered logic programs. To show that the two programs are not σ-equivalent, we guess an interpretation I, and verify in polynomial time that either (a) $I \in AS(\Pi_1)$ and $I \in AS^\sigma((\Pi_1, <_1))$ and that $I \notin AS(\Pi_2)$ or $I \notin AS^\sigma((\Pi_2, <_2))$, or (b) $I \notin AS(\Pi_1)$ or $I \notin AS^\sigma((\Pi_1, <_1))$ and that $I \in AS(\Pi_2)$ and $I \notin AS^\sigma((\Pi_2, <_2))$. All of these subchecks are feasible in polynomial time (cf. [57, 34]), hence the whole check can be done in polynomial time. Testing $\Pi_1 \not\equiv^\sigma \Pi_2$ is therefore in NP, and testing $\Pi_1 \equiv^\sigma \Pi_2$ in co-NP.

For hardness, it is sufficient to observe that, for all $\sigma \in \{D, W, B\}$, $AS^\sigma((\Pi, \emptyset)) = AS(\Pi)$ holds for all logic programs Π. Therefore, for two logic programs Π_1 and Π_2, testing whether $\Pi_1 \equiv \Pi_2$ is not harder than testing $(\Pi_1, <_1) \equiv^\sigma (\Pi_2, <_2)$ for two ordered logic programs $(\Pi_1, <_1)$ and $(\Pi_2, <_2)$. Testing $\Pi_1 \not\equiv \Pi_2$ is well-known to be co-NP. This result per se follows directly from [14] and [148], who showed that deciding whether an answer set exists for a program is NP-complete, but was, to the best of our knowledge, reported formally only recently in Theorem 6.17 of [82] in the context of more complex settings. ■

Proof 5.1.15 For any given Π, $r \in \Pi$, Π', and $X \in AS(\Pi \cup \Pi')$, we construct $X^* = X \cap Atm(\Pi)$ and $\Pi^* = \{a \leftarrow \mid a \in X^*\}$. We first note that $X^* \in AS(\Pi \cup \Pi^*)$: As $\Pi^X = \Pi^{X^*}$, and since X is closed under Π^X, also X^* is closed under Π^{X^*}. X^* is trivially closed under Π^*, and no subset of X^* is closed under Π^*, and hence also no subset of X^* is closed under $\Pi \cup \Pi^*$. Finally, we observe that $R_{\Pi\cup\Pi'}(X) \cap \Pi = R_\Pi(X) = R_\Pi(X^*) = R_{\Pi\cup\Pi^*}(X^*) \cap \Pi$ and hence $r \in R_{\Pi\cup\Pi^*}(X^*)$. ■

Proof 5.1.16 Membership: Because of Lemma 5.1.15, it is sufficient to check whether some program $\Pi' \in facts(Atm(\Pi))$ exists such that there is an answer set $X' = \{a \mid a \leftarrow\in \Pi'\}$ of $\Pi \cup \Pi'$ such that $r \in R_{\Pi\cup\Pi'}(X')$. We guess appropriate Π' and X' out of an exponential number of candidates and verify in polynomial time that $X' \in AS(\Pi \cup \Pi')$ and that $r \in R_{\Pi\cup\Pi'}(X')$. The problem is thus in NP.

Hardness: We give a reduction from $3SAT$, the propositional satisfiability problem over 3CNF formulas. Let $\phi = C_1 \wedge \cdots \wedge C_n$ be a 3CNF, i.e. a conjunction of clauses $\bigwedge_{i=1}^n L_{i,1} \vee L_{i,2} \vee L_{i,3}$ where the $L_{i,j}$ are classical literals over propositional atoms $X = \{x_1, \ldots, x_m\}$. For a literal l, we define $\neg.l = \neg l$ if l is an atom and $\neg.l = x$ if $l = \neg x$. We construct the following logic program Π_ϕ over atoms

$\{v_1, \ldots, v_m, nv_1, \ldots, nv_m, c_1, \ldots, c_n, inc, sat\}$, and use the notation $\nu(x_i) = v_i$ and $\nu(\neg x_i) = nv_i$.

$$\Pi_\phi = \left\{ \begin{array}{l} r_0 : sat \leftarrow c_1, \ldots, c_n, not\ inc. \\ r_{1,1} : inc \leftarrow c_1, not\ \nu(L_{1,1}), not\ \nu(L_{1,2}), not\ \nu(L_{1,3}). \\ \vdots \\ r_{1,n} : inc \leftarrow c_n, not\ \nu(L_{n,1}), not\ \nu(L_{n,2}), not\ \nu(L_{n,3}). \\ r_{2,1} : inc \leftarrow v_1, nv_1. \\ \vdots \\ r_{2,m} : inc \leftarrow v_m, nv_m. \\ r_{3,1} : inc \leftarrow not\ v_1, not\ nv_1. \\ \vdots \\ r_{3,m} : inc \leftarrow not\ v_m, not\ nv_m. \end{array} \right\}$$

If we have $r_0 \in Cont(\Pi_\phi)$, then, by Lemma 5.1.15, there exists a program $\Pi' \in facts(Atm(\Pi_\phi))$ and $X' \in AS(\Pi_\phi \cup \Pi')$ such that $r_0 \in R_{\Pi_\phi \cup \Pi'}(X')$. So $\{sat, c_1, \ldots, c_n\} \subseteq X'$ and $inc \notin X'$. Since $X' \in AS(\Pi_\phi \cup \Pi')$, we get that for each $1 \leq i \leq n$, $\nu(L_{i,1}) \in X'$, $\nu(L_{i,2}) \in X'$, or $\nu(L_{i,3}) \in X'$ because of $r_{1,i}$. Moreover, for each $1 \leq j \leq m$, exactly one of v_j, nv_j is in X' because of rules $r_{2,j}$ and $r_{3,j}$. We then get $\Pi' = \{a. \mid a \in X'\}$. It is now easy to see that X' represents a satisfying truth assignment for ϕ.

In turn, for a satisfying truth assignment μ for ϕ, we construct

$$X' = \{sat, c_1, \ldots, c_n\} \cup \{\nu(x) \mid x \text{ true in } \mu\} \cup \{\nu(\neg x) \mid x \text{ false in } \mu\}.$$

With $\Pi' = \{a \leftarrow \mid a \in X'\}$, we can verify that $X' \in AS(\Pi_\phi \cup \Pi')$ and that $r_0 \in R_{\Pi_\phi \cup \Pi'}(X')$, so $r_0 \in Cont(\Pi_\phi)$. ■

Proof 5.1.17 Assume that $r \in Cont(\Pi_1)$ and $r \notin Cont(\Pi_2)$. Then there exists some Π' and $X \in AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$, such that $r \in R_{\Pi_1 \cup \Pi'}(X)$. Since the condition for being a generating rule does not depend on the program context, but only on X, $r \in \Pi_1$ and $r \notin \Pi_2$ must hold. A symmetric argument can be given for $r \notin Cont(\Pi_1)$ and $r \in Cont(\Pi_2)$ ■

Proof 5.1.18 We show that deciding $Cont(\Pi_1) \neq Cont(\Pi_2)$ is NP-complete, and as in the proof for Theorem 5.1.16, we exploit Lemma 5.1.15.

Membership: Guess a rule $r \in (\Pi_1 \cup \Pi_2) \setminus (\Pi_1 \cap \Pi_2)$, an interpretation X' and program P'. We check in polynomial time that $X' \in AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ and that r is generating in X'.

Hardness: We give a reduction from $3SAT$, the propositional satisfiability problem over 3CNF formulas. For any 3CNF ϕ, let $\Pi_1 = \Pi_\phi \cup \{a \leftarrow\}$, where Π_ϕ is as in the proof of Theorem 5.1.16, while $\Pi_2 = \Pi_\phi \cup \{a \leftarrow\} \cup \{r_0'\}$, where r_0' is r_0 of Π_ϕ with an additional body literal a. Obviously, $\Pi_1 \equiv_s \Pi_2$. But if ϕ is satisfiable, an extension and an answer set X of the extended program exist such that both r_0 and r_0' are generating rules. Since r_0' occurs only in Π_2, $Cont(\Pi_1) \neq Cont(\Pi_2)$ holds in this case. If ϕ is unsatisfiable, no extension has an answer set, hence $Cont(\Pi_1) = Cont(\Pi_2)$ holds. ■

Proof 5.1.19 Membership: According to Corollary 5.1.9 and 5.1.10, we have to check whether the underlying programs are strongly equivalent, the ordered programs coincide on their preference relations, and the underlying programs coincide on their rules contributing to answer sets.

- Checking whether $\Pi_1 \equiv_s \Pi_2$ is in co-NP[134].
- Checking whether $<_1 = <_2$ is in linear time in size of $\Pi_1 \cup \Pi_2$.
- Checking whether Π_1 and Π_2 coincide on their rules contributing to answer sets is co-NP-complete (see Theorem 5.1.18), provided that $\Pi_1 \equiv_s \Pi_2$.

Hardness: Follows from Theorem 5.1.18. ■

Proof 5.1.20 This follows from Corollaries 5.1.9 and 5.1.10, since $(\Pi, <) \equiv_s^\sigma (\Pi \setminus \{r\}, <)$, the preference relation is equal on $(\Pi, <)$ and $(\Pi \setminus \{r\}, <)$, and since $Cont(\Pi) = Cont(\Pi \setminus \{r\})$, which implies also $Cont(\Pi) \setminus \{r \in \Pi \mid head(r) \in body^+(r)\} = Cont(\Pi \setminus \{r\}) \setminus \{r \in \Pi \mid head(r) \in body^+(r)\}$. ■

Proof 5.1.21 This follows from Corollary 5.1.10. ■

C.2 Section 5.2

Proof 5.2.1 Follows directly from Definition 5.2.1. ■

Proof 5.2.2 Follows directly from definitions of strong and n-strong order equivalence. ■

Proof 5.2.3 Proof follows directly from that of strong order equivalence. ■

Proof 5.2.4 Follows directly from Corollary 5.1.9. ■

Proof 5.2.5 Follows directly from Corollary 5.1.10. ■

Proof 5.2.6 The counterexamples are given in Section 5.2.1 directly after this theorem. ■

Proof 5.2.7 Let $(\Pi, <)$ be an ordered logic program, $r_1, r_2 \in \Pi$ such that $body(r_1) = \emptyset$, $head(r_1) \in body(r_2)$, $r_2 < r_1$, and $\sigma \in \{D, W, B\}$. Assume $(\Pi, <) \not\equiv_n^\sigma (\Pi, <')$ holds for $<' = < \setminus \{r_2 < r_1\}$. Then, there exists an Π' such that $(\Pi \cup \Pi', <) \not\equiv^\sigma (\Pi \cup \Pi', <')$. Since $<' \subseteq <$, we have $AS^\sigma((\Pi \cup \Pi', <)) \subseteq AS^\sigma((\Pi \cup \Pi', <'))$ [177]. That is, there exists an $X \in AS^\sigma((\Pi \cup \Pi', <'))$ such that $X \notin AS^\sigma((\Pi \cup \Pi', <))$. Hence, the preference relation $r_2 < r_1$ is "responsible" for making X non-preferred. Since $X \in AS^\sigma((\Pi \cup \Pi', <'))$, there exists an $<^\sigma$-preserving enumeration E of $\Pi \cup \Pi'$ w.r.t. X. Whenever r_1 precedes r_2 in this enumeration, E is also an $<^\sigma$-preserving enumeration for $(\Pi \cup \Pi', <)$, which is a contradiction to $X \notin AS^\sigma((\Pi \cup \Pi', <))$. Hence, for all $<^\sigma$-preserving enumeration E w.r.t. $(\Pi \cup \Pi', <')$ and X we have that r_2 precedes r_1. All rules that are higher preferred than r_1 are higher preferred than r_2, since $<$ is transitive. Hence, all rules higher preferred than r_1 must precede r_2 in an order preserving enumeration. Thus, r_1 could precede r_2 in an order preserving enumeration w.r.t. $<$, which leads to a contradiction to the assumption. ■

Proof 5.2.8 Let $(\Pi, <)$ be an ordered logic program and $r_1, r_2 \in \Pi$ such that $head(r_1) \in body^+(r_2)$, and $r_1, r_2 \in Appl(\Pi)$. Assume, $(\Pi, <) \not\equiv_n^B (\Pi, <')$ holds for $<' = < \setminus \{r_2 < r_1, r_1 < r_2\}$. Then, there exists an Π' such that $AS^B((\Pi \cup \Pi', <)) \neq AS^B((\Pi \cup \Pi', <'))$. Since $<' \subseteq <$, we have that $AS^B((\Pi \cup \Pi', <)) \subseteq AS^B((\Pi \cup \Pi', <'))$. Then, there exists an $X \in AS^B((\Pi \cup \Pi', <'))$ such that $X \notin AS^B((\Pi \cup \Pi', <))$.

Let be $<' = < \setminus \{r_2 < r_1\}$. Since $r_1 \in Appl(\Pi)$ and all rules higher preferred than r_1 are also higher preferred than r_2, there always exists an $<^B$- preserving enumeration where r_1 is enumerated before r_2. Hence, r_2 can be enumerated after r_1 and hence, there exists a $<^B$- preserving enumeration of $(\Pi \cup \Pi', <)$ that is also order preserving for $(\Pi \cup \Pi', <')$ w.r.t. X. Hence, $X \in AS^B((\Pi \cup \Pi', <))$. The case $<' = < \setminus \{r_1 < r_2\}$ is analogous. ■

Proof 5.2.9 Assume $(\Pi, <) \not\equiv_n^B (\Pi, <')$ holds for $<' = < \setminus \{r_1 < r_2\}$. That is, there exists an Π' such that $AS^B((\Pi \cup \Pi', <)) \neq AS^B((\Pi \cup \Pi', <'))$. Since $<' \subseteq <$ we have $AS^B((\Pi \cup \Pi', <)) \subseteq AS^B((\Pi \cup \Pi', <'))$. Then, there exists an $X \in AS^B((\Pi \cup \Pi', <'))$ such that $X \notin AS^B((\Pi \cup \Pi', <))$. Let E be an $<^B$-preserving enumeration w.r.t. $(\Pi \cup \Pi', <')$ and X. We observe that r_2 is blocked by r_1 and $head(r_2) \in A(\Pi)$. Hence, $head(r_2) \in X$ since $A(\Pi) \subseteq X$. Thus, r_2 can be enumerated directly before r_1 . For this reason $X \in AS^B((\Pi \cup \Pi', <))$. ■

Proof 5.2.10 Assume $(\Pi, <) \not\equiv_n^\sigma (\Pi, \emptyset)$. Then, there exists an Π' and $X \in AS^\sigma((\Pi \cup \Pi', \emptyset))$ such that $X \notin AS^\sigma((\Pi \cup \Pi', <))$. More precisely, $\Pi \cup \Pi'$ has no order preserving enumeration w.r.t. $<$. Since all

rules involved in $<$ are in $Appl(\Pi)$, there exists an order preserving enumeration of $Appl(\Pi)$. Furthermore, rules cannot be blocked by lower ranked ones, since $<$ discards no answer set as non-preferred and all non-applicable rules involved in $<$ are blocked from $A(\Pi)$. Hence, there exists an order preserving enumeration of $\Pi \cup \Pi'$ w.r.t. X and $<$. ■

Proof 5.2.11 Assume $(\Pi, <) \not\equiv_n^\sigma (\Pi \setminus \{r_1\}, <)$. Then, there exists an Π', $r_1 \notin \Pi'$, such that $(\Pi \cup \Pi', <) \not\equiv^\sigma (\Pi \cup \Pi' \setminus \{r_1\}, <)$. Abbreviatory, we write Π^* for $\Pi \cup \Pi'$. There are 2 cases. Case 1: There exists an $X \in AS^\sigma((\Pi^*, <))$ such that $X \notin AS^\sigma((\Pi^* \setminus \{r_1\}, <))$; Case 2: There exists an $X \in AS^\sigma((\Pi^* \setminus \{r_1\}, <))$ such that $X \notin AS^\sigma((\Pi^*, <))$. In Case 1, we have an $<^\sigma$- preserving enumeration of Π^* w.r.t. X, but not for $\Pi^* \setminus \{r_1\}$. That is, r_1 is used to derive rules or to block rules in an order preserving way, which can not be done by r_2. That is, r_1 precedes r_2 in any order-preserving enumeration (otherwise r_2 can be used to derive rules or to block rules). But this is a contradiction to $body(r_2) \subseteq body(r_1)$ and $r_1, r_2 \notin PR((\Pi, <))$. In Case 2, r_1 can always be inserted at the end of the enumeration E and we get an $<^\sigma$- preserving enumeration of Π^* w.r.t. X, which is a contradiction to the assumption. Thus, $(\Pi, <) \equiv_n^\sigma (\Pi \setminus \{r_1\}, <)$. ■

Proof 5.2.12 Analogously to Lemma 5.2.11. ■

Proof 5.2.13 Assume, $(\Pi_1, <_1) \not\equiv_n^\sigma (\Pi_2, <_2)$. Hence, there exists a Π' such that $AS^\sigma((\Pi_1 \cup \Pi', <_1)) \neq AS^\sigma((\Pi_2 \cup \Pi', <_2))$. W.l.o.g., there exists an $X \in AS^\sigma((\Pi_1 \cup \Pi', <_1))$ such that $X \notin AS^\sigma((\Pi_2 \cup \Pi', <_2))$. Then, Π' may have rules from Π_1, Π_2, and rules S, which are not contained in both programs, i.e. $\Pi' = S \cup \Pi_1^S \cup \Pi_2^S$, where $\Pi_1^S \subseteq \Pi_1$, $\Pi_2^S \subseteq \Pi_2$, and $S \cap (\Pi_1 \cup \Pi_2) = \emptyset$. Let S' be a logic program, which contains rules $head(r) \leftarrow$, where r belongs to S and r is generating w.r.t. the answer set X, i.e. $S' = \{head(r) \leftarrow x \mid r \in R_S(X)\}$.

Then, $\Pi^* = S' \cup \{x \leftarrow\} \cup \Pi_1^S \cup \Pi_2^S$ has the form (5.6). Next, we show that $X \in AS^\sigma((\Pi_1 \cup \Pi^*, <_1))$ and $X \notin AS^\sigma((\Pi_2 \cup \Pi^*, <_2))$ hold. But this holds trivially, since Π^* comprises the heads of the generating rules from Π'. ■

Proof 5.2.14

Membership: We show that deciding $(\Pi_1, <_1) \not\equiv_n^\sigma (\Pi_2, <_2)$ is in NP. By Lemma 5.2.13, we have to find a Π^* of the form (5.6) such that $X \in AS^\sigma((\Pi_1 \cup \Pi^*, <_1))$ and $X \notin AS^\sigma((\Pi_2 \cup \Pi^*, <_2))$. Hence, we guess appropriate Π^* and X^* and verify in polynomial time that $X^* \in AS^\sigma((\Pi_1 \cup \Pi^*, <_1))$ and $X^* \notin AS^\sigma((\Pi_2 \cup \Pi^*, <_2))$, or $X^* \notin AS^\sigma((\Pi_1 \cup \Pi^*, <_1))$ and $X^* \in AS^\sigma((\Pi_2 \cup \Pi^*, <_2))$. The problem is thus in NP.

Hardness: We give a reduction from strong equivalence for normal logic programs. Let Π_1 and Π_2 be logic programs. Then, deciding $\Pi_1 \equiv_s \Pi_2$ is co-NP-complete. By Lemma 5.2.1, we have $\Pi_1 \equiv_s \Pi_2$ iff $(\Pi_1, \emptyset) \equiv_n^\sigma (\Pi_2, \emptyset)$. Hence, $(\Pi_1, <_1) \equiv_n^\sigma (\Pi_2, <_2)$ is co-NP-hard for $\sigma \in \{D, W, B\}$.

In total, we obtain that the problem is co-NP-complete. ■

Appendix D

Chapter 6: Proofs

D.1 Section 6.1

Proof 6.1.1 Let $\mathcal{R}$ be a partial strict order and X be a set of candidates, where $x, y, z \in X$.
$rank_R^{min}(x)$**:** Let $R_x^{\uparrow}$ be the following extension of R:

$$(\uparrow)\ R_x^{\uparrow} = R \cup \{(x, z) \mid z \neq x, not\ (z >_R x)\}$$

That is, all candidates which are not initially strictly preferred to x are now less preferred: for all $z \neq x$, $x > z$ holds in $R_x^{\uparrow}$ as soon as $(z >_R x)$ does not hold that is, as soon as it is possible to enforce $x > z$. Note also that it can be easily checked that $R_x^{\uparrow}$ is transitive. Next, we show that $R_x^{\uparrow}$ has at least one complete extension. For this, let $\langle z_i \rangle_{i \in L}$ be an enumeration of $\{z \mid z \neq x, not(z >_R x)\}$ such that $i < j$ if $z_i >_R z_j$ holds for all $i, j \in L$. Analogously, let $\langle y_i \rangle_{i \in K}$ be an enumeration of $\{y \mid y >_R x\}$ such that $i < j$ if $y_i >_R y_j$ holds for all $i, j \in K$. Then, let $\langle r_i \rangle_{i \in J}$ be an enumeration of X such that $\langle r_i \rangle_{i \in J} = \langle \langle y_i \rangle_{i \in K}, x, \langle z_i \rangle_{i \in L} \rangle$. Then, R', defined as $r_i \geq_{R'} r_j$ iff $i \leq j$, is an complete extension of $R_x^{\uparrow}$.

For all complete extensions of $R_x^{\uparrow}$ we have that x has the rank $|\{y \mid y >_R x\}| + 1$ since all other candidates are ranked lower than x. For all complete extensions T which do not satisfy $(\uparrow)$ we have that x has at least the rank $|\{y \mid y >_R x\}| + 2$ since there is at least one other candidate who is additionally higher ranked than x. Hence, we have $rank_R^{min}(x) = |\{y \mid y >_R x\}| + 1$.
$rank_R^{max}(x)$**:** Proof is similar by taking $(\downarrow)\ R_x^{\downarrow} = R \cup \{(z, x) \mid z \neq x, not\ (x >_R z)\}$ ■

Proof 6.1.2 Let $\mathcal{R}$ be a preference profile, where each R_i is a partial order and F_s be a voting procedure.
1,⇐: Suppose that x is not a necessary winner for $\mathcal{R}$ w.r.t. F_s. Then there exists an extension $\mathcal{T}$ of $\mathcal{R}$ and a $y \neq x$ such that $S(y, \mathcal{T}) > S(x, \mathcal{T})$, thus $S_{\mathcal{R}}^{max}(y) \geq S(y, \mathcal{T}) > S(x, \mathcal{T}) \geq S_{\mathcal{R}}^{min}(x)$, which contradicts the assumption that $S_{\mathcal{R}}^{min}(x) \geq S_{\mathcal{R}}^{max}(y)$ holds for all $y \neq x$.
1, ⇒: Let x be a necessary winner. Then, there don't exists an $\mathcal{T} \in Ext(\mathcal{R})$ and there don't exists an $y \neq x$ such that $S(x, \mathcal{T}) < S(y, \mathcal{T})$. Since $S_{\mathcal{R}}^{min}(x) \leq S(x, \mathcal{T})$ and $S_{\mathcal{R}}^{max}(y) \geq S(y, \mathcal{T})$, we have that there don't exists an $y \neq x$ such that $S_{\mathcal{R}}^{min}(x) < S_{\mathcal{R}}^{max}(y)$.
2: analogously to 1. ■

Proof 6.1.3 For each partial order R_i and each candidate x, we just have to compute the number of candidates dominated by x and dominating x in R_i, which lead to the exact bound $\mathcal{O}(n * m^2)$, where n is the number of voters and m the number of candidates. ■

Proof 6.1.4 Let $\mathcal{R}$ be a partial preference profile and x, y two distinct candidates from X.
1: Let us first show that $\sum_{i=1}^{n} N_{R_i}^{min}(x, y) = \min_{\mathcal{T} \in Ext(\mathcal{R})} N_{\mathcal{T}}(x, y)$. We have

$$\sum_{i=1}^{n} N_{R_i}^{min}(x, y) = |\{i \mid x >_i y\}| - |\{i \mid not(x \geq_i y)|\}$$

Let $\mathcal{T} = \langle T_1, \ldots, T_n \rangle \in Ext(\mathcal{R})$. We have $N_{\mathcal{T}}(x,y) = |\{i \mid x >_{T_i} y\}| - |\{i \mid y >_{T_i} x\}|$. Now, $x >_i y$ implies $x >_{T_i} y$, therefore

$$\text{(a) } |\{i \mid x >_{T_i} y\}| \geq |\{i \mid x >_i y\}|.$$

Next, $y >_{T_i} x$ implies not($x \geq_i y$), therefore

$$\text{(b) } |\{i \mid y >_{T_i} x\}| \leq |\{i \mid \text{not}(x \geq_i y)\}|.$$

(a) and (b) give

$$\text{(c) } |\{i \mid x >_{T_i} y\}| - |\{i \mid y >_{T_i} x)\}| \geq |\{i \mid x >_i y\}| - |\{i \mid not(x \geq_i y)\}|,$$

which is equivalent to $N_{\mathcal{T}}(x,y) \geq |\{i \mid x >_i y\}| - |\{i \mid not(x \geq_i y)\}|$. Since this holds for all $\mathcal{T} \in Ext(\mathcal{R})$, we get $\min_{\mathcal{T} \in Ext(\mathcal{R})} N_{\mathcal{T}}(x,y) \geq \sum_{i=1}^{n} N_{R_i}^{min}(x,y)$. To show the inequality on the reverse direction, we consider, as in the proof of Proposition 6.1.1, the worst-case (for x) complete extension of R_i: for each i, let $(R_i)_x^{\downarrow} = R_i \cup \{(z,x) \mid z \neq x, not\ (x >_i z)\}$ and $\mathcal{R}_x^{\downarrow} = \langle (R_1)_x^{\downarrow}, \ldots, (R_n)_x^{\downarrow} \rangle$. We have

$$\begin{array}{rcl} N_{\mathcal{R}_x^{\downarrow}}(x,y) & = & |\{i \mid x >_{(R_i)_x^{\downarrow}} y\}| - |\{i \mid y >_{(R_i)_x^{\downarrow}} x)\}| \\ & = & |\{i \mid x >_{(R_i)_x^{\downarrow}} y\}| - |\{i \mid \text{not}(x \geq_{(R_i)_x^{\downarrow}} y)\}| \\ & = & |\{i \mid x >_{R_i} y\}| - |\{i \mid \text{not}(x \geq_{R_i} y)\}| \\ & = & \sum_{i=1}^{n} N_{R_i}^{min}(x,y) \end{array}$$

Therefore, $\min_{\mathcal{T} \in Ext(\mathcal{R})} N_{\mathcal{T}}(x,y) \leq \sum_{i=1}^{n} N_{R_i}^{min}(x,y)$. The proof for $N_{\mathcal{R}}^{max}(x,y) = \sum_{i=1}^{n} N_{R_i}^{max}(x,y)$ is similar.

2, $\Leftarrow$: Assume that x is not a necessary Condorcet winner. Then, there exists an $\mathcal{T} \in Ext(\mathcal{R})$ and an $y \neq x$ such that $|\{i \mid x >_{T_i} y\}| \leq |\{i \mid y >_{T_i} x\}|$. Hence, $N_{\mathcal{T}}(x,y) \leq 0$. That implies that $\min_{\mathcal{T} \in Ext(\mathcal{R})} N_{\mathcal{T}}(x,y) \leq 0$ and thus, $N_{\mathcal{R}}^{min}(x,y) \leq 0$ which is a contradiction to the assumption that $N_{\mathcal{R}}^{min}(x,y) > 0$ holds. Hence, x is a necessary Condorcet winner.

2, $\Rightarrow$: Let x be a necessary Condorcet winner. Assume that there exists an $y \neq x$ such that $N_{\mathcal{R}}^{min}(x,y) \leq 0$ holds for some x. That is, we have $|\{i \mid x >_i y\}| \leq |\{i \mid not(x \geq_i y)\}|$. Hence, there exists an $\mathcal{T} \in Ext(\mathcal{R})$ such that $N_{\mathcal{T}}(x,y) \leq 0$. Hence, x is no Condorcet winner, which is a contradiction to the assumption.

3: similar to the proof for necessary Condorcet winners. ■

Proof 6.1.5 Proof analogously to Corollary 6.1.3. ■

Proof 6.1.6 Follows directly from the definition of necessary and possible winners in Definition 6.1.1 and Definition 6.1.2. ■

Proof 6.1.7 Follows directly from the definition of necessary and possible winners in Definition 6.1.1 and Definition 6.1.2. ■

D.2 Section 6.2

Proof 6.2.1 (Sketch) Let be given a set of candidates X, a set of voters V, partial preference profiles for each voter over the set of candidates, and a voting procedure $VP \in \{Borda, Plurality, Condorcet\}$. Furthermore, let Π be the logic program, consisting of rules (6.1)- (6.26).

First, we show that Π has exactly one answer set. We obtain that rules (6.1)- (6.8) are in the well-founded model of Π (cf. [42] for well-founded semantics of aggregate functions). Hence, recursively, also rules (6.9)-(6.26) are in the well-founded model of Π. Thus, the well-founded model is total and for this reason, there exists exactly one answer set Y of Π.

By the semantics of the aggregate functions we can easily verify that the set $\{X : possible(VP, X) \in Y\}$ is the set of all possible winners and $\{X : necessary(VP, X) \in Y\}$ is the set of all necessary winner w.r.t. voting procedure VP. ■

D.3 Section 6.3

Proof 6.3.1 (Sketch) Let $\mathcal{M} = \langle D, X, NA, k\rangle$ be a meeting scheduling problem and $\Pi_{\mathcal{M}}$ as described in Rules (6.27)-(6.37).

Rules (6.27)-(6.31) are initialized as facts. Rule (6.32) gives the availabilities of each person for each date. Rule (6.33) for group g and date d becomes applicable if and only if $|\{P \in g : P \text{ is available for } d\}| \geq k$. Hence, rule (6.34) becomes applicable for date d if and only if there exists a group g such that $|\{P \in g : P \text{ is available for } d\}| < k$.

For this reason, by rules (6.35)-(6.37), Π has an answer set iff $|\{P \in g : P \text{ is available for } d\}| \geq k$ holds for some date d, where $meeting(d)$ is true. On the other hand, Π has no answer set iff $absentgroup(d)$ is derivable for all dates d, i.e there exists no $d \in D$ such that for all groups g we have $|\{P \in g : P \text{ is available for } d\}| \geq k$. ■

Proof 6.3.2 (Sketch) Let $\mathcal{M} = \langle D, X, NA, k\rangle$ be a meeting scheduling problem and $\Pi^{\mathcal{D}}_{\mathcal{M}}$ be the corresponding diagnostic model.

Whenever there is a meeting schedulable (according to Definition 6.3.1), rules (6.44), (6.45), (6.47), (6.48), and (6.49) become not applicable. So, $E_X = \emptyset$ and $R_X = \emptyset$.

Whenever no meeting is schedulable according to Definition 6.3.1, then the rules (6.44)–(6.50) catch all possible reasons. Hence, $E_X \neq \emptyset$. ■

Proof 6.3.3 Follows from theorems 6.3.1 and 6.2.1. ■

Bibliography

[1] J. J. Alferes and L. M. Pereira. Updates plus preferences. *Proceedings of the European Workshop on Logics in Artificial Intelligence (JELIA 2000)*, 1919:345–360, 2000.

[2] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub. The `nomore++` approach to answer set solving. In G. Sutcliffe and A. Voronkov, editors, *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *Lecture Notes in Computer Science*, pages 95–109. Springer-Verlag, 2005.

[3] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub. The `nomore++` system. In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 422–426. Springer-Verlag, 2005.

[4] C. Anger, K. Konczak, and T. Linke. `NoMoRe`: Non-monotonic reasoning with logic programs. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proceedings of the Eighth European Conference on Logics in Artificial Intelligence (JELIA'02)*, volume 2424 of *Lecture Notes in Artificial Intelligence*, pages 521–524. Springer-Verlag, 2002.

[5] C. Anger, K. Konczak, T. Linke, and T. Schaub. A glimpse of answer set programming. *Künstliche Intelligenz*, 19(1):12–17, 2005.

[6] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann Publishers, 1987.

[7] M. Balduccini. USA-smart: Improving the quality of plans in answer set planning. In B. Jayaraman, editor, *Practical Aspects of Declarative Languages, 6th International Symposium, PADL 2004*, volume 3057 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 2004.

[8] M. Balduccini and M. Gelfond. Logic programs with consistency-restoring rules. In P. Doherty, J. McCarthy, and M.-A. Williams, editors, *International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series*, March 2003.

[9] M. Balduccini and V. Mellarkod. CR-prolog with ordered disjunction. In J. Delgrande and T. Schaub, editors, *10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Whistler, Canada, June 6-8, 2004, Proceedings*, pages 34–40, 2004.

[10] C. Baral. *Knowledge representation, reasoning and declarative problem solving with Answer sets*. Cambridge University Press, 2003.

[11] C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 12:1–80, 1994.

[12] C. Baral and C. Uyan. Declarative specification and solution of combinatorial auctions using logic programming. In T. Eiter, W. Faber, and M. Truszczyński, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, volume 2173 of *Lecture Notes in Computer Science*, pages 186–199. Springer, 2001.

[13] P. Besnard, G. Fanselow, and T. Schaub. Optimality theory as a family of cumulative logics. *Journal of Logic, Language and Information*, 12(2):153–182, April 2003.

[14] N. Bidoit and C. Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78(1):85–112, 1991.

[15] P. Bonatti. Proof systems for default and autoepistemic logic. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings Tableaux'96*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 127–142. Springer-Verlag, 1996.

[16] P. Bonatti. Reasoning with infinite stable models. In B. Nebel, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 603–610. Morgan Kaufmann, 2001.

[17] P. Bonatti. Resolution for skeptical stable model semantics. *Journal of Automated Reasoning*, 27(4):391–421, 2001.

[18] P. Bonatti and N. Olivetti. A sequent calculus for skeptical default logic. In D. Galmiche, editor, *Proceedings Tableaux'97*, volume 1227 of *Lecture Notes in Artificial Intelligence*, pages 107–121. Springer-Verlag, 1997.

[19] A. Bösel, T. Linke, and T. Schaub. Profiling answer set programming: The visualization component of the `noMoRe` system. In J. Alferes and J. Leite, editors, *Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA'04)*, 2004.

[20] C. Boutilier. A logical approach to qualitative decision theory. In *Proceedings of UAI-94*, 1994.

[21] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. Preference-based constraint optimization with CP-nets. *Computational Intelligence*, 20(2):137–157, May 2004.

[22] C. Boutilier, R. Brafman, H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In K. Laskey and H. Prade, editors, *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 71–80. Morgan Kaufmann, 1999.

[23] C. Boutilier, R. I. Brafman, C. Domshlak, D. Poole, and H. Hoos. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

[24] S. Brams and P. Fishburn. Voting procedures. In K. J. Arrow, A. K. Sen, and K. Suzumura, editors, *Handbook of Social Choice and Welfare*, volume 1. Elsevier, 2003.

[25] S. Brass and J. Dix. Semantics of (disjunctive) logic programs based on partial evaluation. *Journal of Logic Programming*, 40(1):1–46, 1999.

[26] S. Brass, J. Dix, B. Freitag, and U. Zukowski. Transformation-based bottom-up computation of the well-founded model. *Theory and Practice of Logic Programming*, 1(5):497–538, September 2001.

[27] G. Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, 4:19–36, 1996.

[28] G. Brewka. Logic programming with ordered disjunction. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 100–105. AAAI Press, 2002.

[29] G. Brewka. Logic programming with ordered disjunction. In S. Benferhat and E. Giunchiglia, editors, *9th International Workshop on Non-Monotonic Reasoning (NMR 2002)*, pages 67–76, 2002.

[30] G. Brewka. Answer sets: From constraint programming towards qualitative optimization. In V. Lifschitz and I. Niemelä, editors, *Logic Programming and Nonmonotonic Reasoning, 7th International Conference (LPNMR 2004)*, volume 2923 of *Lecture Notes in Computer Science*, pages 34–46. Springer, 2004.

[31] G. Brewka. Complex preferences for answer set optimization. In D. Dubois, C. Welty, and M. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, pages 213–223. AAAI Press, 2004.

[32] G. Brewka. A rank based description language for qualitative preferences. In J. Delgrande and T. Schaub, editors, *10th International Workshop on Non-Monotonic Reasoning (NMR 2004)*, pages 88–93, 2004.

[33] G. Brewka, S. Benferhat, and D. Le Berre. Qualitative choice logic. *Artificial Intelligence*, 157(1-2):203–237, 2004.

[34] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.

[35] G. Brewka, I. Niemelä, and T. Syrjänen. Implementing ordered disjunction using answer set solvers for normal programs. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Logics in Artificial Intelligence, European Conference, JELIA*, volume 2424 of *Lecture Notes in Computer Science*, pages 444–455, 2002.

[36] G. Brewka, I. Niemelä, and T. Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence*, 20(2):335–357, May 2004.

[37] G. Brewka, I. Niemelä, and M. Truszczyński. Answer set optimization. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 867–872. Morgan Kaufmann, 2003.

[38] G. Brewka, I. Niemelä, and M. Truszczyński. Prioritized component systems. In M. Veloso and S. Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 596–601. AAAI Press / The MIT Press, 2005.

[39] G. Brignoli, S. Costantini, O. D'Antona, and A. Provetti. Characterizing and computing stable models of logic programs: the non-stratified case. In C. Baral and H. Mohanty, editors, *Proceedings of the Conference on Information Technology, Bhubaneswar, India*, pages 197–201. AAAI Press, 1999.

[40] F. Buccafurri, N. Leone, and P. Rullo. Adding weak constraints to disjunctive datalog. In M. Falaschi, M. Navarro, and A. Policriti, editors, *Joint Conference on Declarative Programming, APPIA-GULP-PRODE'97*, pages 557–568, 1997.

[41] F. Buccafurri, N. Leone, and P. Rullo. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[42] F. Calimeri, W. Faber, N. Leone, and S. Perri. Declarative and computational properties of logic programs with aggregates. In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 406–411, 2005.

[43] F. Calimeri, W. Faber, N. Leone, and G. Pfeifer. Pruning operators for answer set programming systems. Technical Report INFSYS RR-1843-01-07, Technische Universität Wien, May 2001.

[44] F. Calimeri, G. Ianni, G. Ielpa, A. Pietramala, and M. Santoro. A system with template answer set programs. In J. Alferes and J. Leite, editors, *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004*, volume 3229 of *Lecture Notes in Computer Science*, pages 693–697. Springer, 2004.

[45] P. Cholewiński, V. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with nonmonotonic reasoning. In L. Sterling, editor, *Proceedings of the International Conference on Logic Programming*, pages 267–281. The MIT Press, 1995.

[46] K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[47] `Cmodels`. `http://www.cs.utexas.edu/users/tag/cmodels.html`.

[48] V. Conitzer, J. Lang, and T. Sandholm. How many candidates are required to make an election hard to manipulate? In *Proceedings of TARK-03*, pages 201–214, 2003.

[49] V. Conitzer and T. Sandholm. Complexity of manipulating an election with few candidates. In *Proceedings of AAAI-02*, pages 314–319, 2002.

[50] V. Conitzer and T. Sandholm. Vote elicitation: complexity and strategy-proofness. In *Proceedings of AAAI-02*, pages 392–397, 2002.

[51] V. Conitzer and T. Sandholm. Universal voting protocols tweaks to make manipulation hard. In *Proceedings of IJCAI-03*, pages 781–788, 2003.

[52] S. Costantini. Comparing different graph representations of logic programs under the answer set semantics. In T. C. Son and A. Provetti, editors, *Proc. of the AAAI Spring 2001 Symposium on: Answer Set Programming, Towards Efficient and Scalable Knowledge Representation and Reasoning*, Stanford, CA, USA, 2001. AAAI Press.

[53] S. Costantini, O. D'Antona, and A. Provetti. On the equivalence and range of applicability of graph-based representations of logic programs. *Information Processing Letters*, 84(5):241–249, 2002.

[54] S. Costantini and A. Provetti. Normal forms for answer sets programming. *Theory and Practice of Logic Programming*, 5(6):747–760, 2005.

[55] CRmodels3. `http://krlab.cs.ttu.edu/~marcy/crmodels3/index.html`.

[56] B. Cui and T. Swift. Preference logic grammars: Fixed-point semantics and application to data standardization. *Artificial Intelligence*, 138(1-2):117–147, 2002.

[57] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[58] A. Davenport and J. Kalagnanam. A computational study of the kemeny rule for preference aggregation. In *Proceedings of AAAI-04*, pages 697–702, 2004.

[59] A. Davison. A survey of logic programming-based object-oriented languages. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research directions in concurrent object-oriented programming*, pages 42–106. MIT Press, Cambridge, MA, USA, 1993.

[60] J. Delgrande and T. Schaub. Expressing preferences in default logic. *Artificial Intelligence*, 123(1-2):41–87, 2000.

[61] J. Delgrande, T. Schaub, and H. Tompits. Logic programs with compiled preferences. In W. Horn, editor, *Proceedings of the European Conference on Artificial Intelligence*, pages 392–398. IOS Press, 2000.

[62] J. Delgrande, T. Schaub, and H. Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129–187, March 2003.

[63] J. Delgrande, T. Schaub, and H. Tompits. Domain-specific preferences for causal reasoning and planning. In D. Dubois, C. Welty, and M.A. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*, pages 673–682. The AAAI Press/The MIT Press, 2004.

[64] J. Delgrande, T. Schaub, H. Tompits, and K. Wang. Towards a classification of preference handling approaches in nonmonotonic reasoning. In U. Junker, editor, *Proceedings of the Workshop on Preferences in Artificial Intelligence and Constraint Programming: Symbolic Approaches*, pages 16–24. AAAI Press, 2002.

[65] J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence*, 20(2):308–334, 2004.

[66] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in dlv. In G. Gottlob and T. Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 847–852. Morgan Kaufmann, 2003.

[67] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in dlv. Technical report, TU Wien, 2003. INFSYS RR-1843-03-07.

[68] M. Denecker and A. Kakas. Abduction in logic programming. In A. Kakas and F. Sadri, editors, *Computational Logic. Logic Programming and Beyond*, volume 2407 of *Lecture Notes in Computer Science*, pages 402–436. Springer-Verlag, 2002.

[69] M. Denecker, V. Marek, and M. Truszczyński. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In *Logic-based artificial intelligence*, pages 127–144. Kluwer Academic Publishers, 2000.

[70] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for xml. *Computer Networks*, 31(11-16):1155–1169, 1999.

[71] Y. Dimopoulos and C. Kakas. Logic programming without negation as failure. In J. Lloyd, editor, *Proceedings of the International Symposium of Logic Programming*, pages 369–383. The MIT Press, 1995.

[72] Y. Dimopoulos and A. Torres. Graph theoretical structures in logic programs and default theories. *Theoretical Computer Science*, 170:209–244, 1996.

[73] `DLV`. `http://www.dbai.tuwien.ac.at/proj/dlv/`.

[74] `meta-interpreter`. `http://www.dbai.tuwien.ac.at/proj/dlv/preferred/`.

[75] W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1:267–284, 1984.

[76] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.

[77] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Computing preferred answer sets by meta-interpretation in answer set programming. *Theory and Practice of Logic Programming*, 3(4-5):463–498, 2003.

[78] T. Eiter and M. Fink. Uniform equivalence of logic programs under the stable model semantics. In C. Palamidessi, editor, *Logic Programming, 19th International Conference, ICLP 2003*, volume 2916 of *Lecture Notes in Computer Science*, pages 224–238. Springer, 2003.

[79] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. A Generic Approach for Knowledge-Based Information Site Selection. In D. Fensel, F. Giunchiglia, D. McGuiness, and M.-A. Williams, editors, *Proceedings Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR-02), April 22-25, Toulouse, France*, pages 459–469. Morgan Kaufmann, 2002. Extended version Technical Report INFSYS RR-1843-02-09, TU Wien, 2002.

[80] T. Eiter, M. Fink, H. Tompits, and S. Woltran. Simplifying logic programs under uniform and strong equivalence. In V. Lifschitz and I. Niemelä, editors, *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Computer Science*, pages 87–99. Springer-Verlag Heidelberg, 2004.

[81] T. Eiter, M. Fink, H. Tompits, and S. Woltran. Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In M. M. Veloso and S. Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference (AAAI'05)*, pages 695–700. AAAI Press AAAI Press / The MIT Press, 2005.

[82] T. Eiter, M. Fink, and S. Woltran. Semantical Characterizations and Complexity of Equivalences in Answer Set Programming. *ACM Transactions on Computational Logic*. To appear.

[83] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15:289–323, 1995.

[84] T. Eiter, G. Gottlob, and N. Leone. Abduction from logic programs: semantics and complexity. *Theoretical Computer Science*, 189(1–2):129–177, 1997.

[85] I. Engan, T. Langholm, E. Lian, and A. Waaler. Default reasoning with preference within only knowing logic. In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 304–316. Springer, 2005.

[86] E. Erdem, V. Lifschitz, L. Nakhleh, and D. Ringe. Reconstructing the evolutionary history of indo-european languages using answer set programming. In *Practical Aspects of Declarative Languages, 5th International Symposium, PADL 2003*, volume 2562 of *Lecture Notes in Computer Science*, pages 160–176. Springer, 2003.

[87] W. Faber. *Enhancing Efficiency and Expressiveness in Answer Set Programming Systems*. PhD thesis, Institut für Informationssysteme, Technische Universität Wien, 2002.

[88] W. Faber. Unfounded sets for disjunctive logic programs with arbitrary aggregates. In Ch. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Logic Programming and Nonmonotonic Reasoning, 8th International Conference (LPNMR 2005)*, volume 3662 of *Lecture Notes in Comupter Science*, pages 40–52. Springer, 2005.

[89] W. Faber and K. Konczak. Strong equivalence for logic programs with preferences. In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 430–435, 2005.

[90] W. Faber and K. Konczak. Strong order equivalence. *Special Issue of Annals of Mathematics and Artificial Intelligence on Answer Set Programming*, 2006. To appear.

[91] W. Faber, N. Leone, and G. Pfeifer. Representing school timetabling in a disjunctive logic programming language. In *13th Workshop on Logic Programming (WLP '98)*, pages 43–52, 1998.

[92] W. Faber, N. Leone, and G. Pfeifer. Pushing goal derivation in dlp computations. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 177–191, El Paso, Texas, USA, 1999. Springer-Verlag.

[93] W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In J. Alferes and J. Leite, editors, *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004*, volume 3229 of *Lecture Notes in Computer Science*, pages 200–212. Springer, 2004.

[94] F. Fages. Consistency of clark's completion and the existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[95] M. Fitting. Fixpoint semantics for logic programming a survey. *Theoretical Computer Science*, 278(1-2):25–51, 2002.

[96] P. Flach and A. Kakas. *Abduction and Induction: Essays on their Relation and Integration*. Applied Logic Series. Kluwer Academic Publishers, 2000.

[97] `GCasp`. `http://www.cs.uni-potsdam.de/~konczak/system/gcasp/`.

[98] `GCplp`. `http://www.cs.uni-potsdam.de/~konczak/system/GCplp`.

[99] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the International Conference on Logic Programming*, pages 1070–1080. The MIT Press, 1988.

[100] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[101] M. Gelfond and T. Son. Reasoning with prioritized defaults. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Third International Workshop on Logic Programming and Knowledge Representation*, volume 1471 of *Lecture Notes in Computer Science*, pages 164–223. Springer-Verlag, 1997.

[102] A. Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41(4):587–601, July 1973.

[103] S. Grell, K. Konczak, and T. Schaub. nomore$^<$: A system for computing preferred answer sets. In C. Baral, G. Greco, N. Leona, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 394–398. Springer, 2005.

[104] B. Grosof. Prioritized conflict handling for logic programs. In J. Maluszynsk, editor, *Logic Programming: Proceedings of the 1997 International Symposium*, pages 197–211. The MIT Press, 1997.

[105] S.O. Hansson. What is ceteris paribus preference. *Journal of Philosophical Logic*, 25(3):307–332, 1996.

[106] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of kemeny elections. Technical report, Mathematik und Informatik, 2004.

[107] G. Ianni, G. Ielpa, A. Pietramala, M. Santoro, and F. Calimeri. Enhancing answer set programming with templates. In J. Delgrande and T. Schaub, editors, *10th International Workshop on Non-Monotonic Reasoning (NMR 2004)*, pages 233–239, 2004.

[108] The internet movie database. `http://imdb.com/`.

[109] K. Inoue and C. Sakama. Abducing priorities to derive intended conclusions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 44–49. Morgan Kaufmann Publishers Inc., 1999.

[110] K. Inoue and C. Sakama. Equivalence of logic programs under updates. In J. Alferes and J. Leite, editors, *Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA'04)*, volume 3229 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2004.

[111] K. Inoue and C. Sakama. Equivalence in abductive logic. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 472–477, 2005.

[112] T. Janhunen. Comparing the expressive powers of some syntactically restricted classes of logic programs. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. Pereira, Y. Sagiv, and P. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic (CL 2000)*, volume 1861 of *Lecture Notes in Computer Science*, pages 852–866. Springer-Verlag, 2000.

[113] R. Kager. *Optimality Theory*. Cambridge University Press, 1999.

[114] A. Kakas, R. Kowalski, and F. Toni. The role of abduction in logic programming. In *Handbook of logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press, 1998.

[115] A. C. Kakas and P. Mancarella. Database updates through abduction. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proceedings of the 16th International Conference on very large databases (VLDB'90)*, pages 650–661. Morgan Kaufmann, 1990.

[116] A. C. Kakas and P. Mancarella. Generalized stable models: A semantics for abduction. In *9th European Conference on Artificial Intelligence, ECAI '90*, pages 385–391, 1990.

[117] K. Konczak. Graphbasierte Behandlung geordneter logischer Programme. Master's thesis, Institute for Computer Science, University of Potsdam, 2002. Diplomarbeit.

[118] K. Konczak. Voting theory in answer set programming. In M. Fink, H. Tompits, and S. Woltran, editors, *20th Workshop on Logic Programmin (WLP 2006)*, number 1843-06-02 in INFSYS Research Report, pages 45–53. Technische Universität Wien, 2006.

[119] K. Konczak. Weak order equivalence for logic programs with preferences. In M. Fink, H. Tompits, and S. Woltran, editors, *20th Workshop on Logic Programmin (WLP 2006)*, number 1843-06-02 in INFSYS Research Report, pages 154–163. Technische Universität Wien, 2006.

[120] K. Konczak and J. Lang. Voting procedures with incomplete preferences. In R. Brafman and U. Junker, editors, *IJCAI-05 Workshop on Advances in Preference Handling*, pages 124–129, 2005.

[121] K. Konczak, T. Linke, and T. Schaub. Graphs and colorings for answer set programming: Abridged report. In V. Lifschitz and I. Niemelä, editors, *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Computer Science*, pages 127 – 140. Springer-Verlag Heidelberg, 2003.

[122] K. Konczak, T. Linke, and T. Schaub. Graphs and colorings for answer set programming: Abridged report. In M. De Voss and A. Provetti, editors, *Proceedings of the Second International Workshop on Answer Set Programming (ASP03)*, volume 78, pages 137–150, Messina, 2003. CEUR Workshop Proceedings.

[123] K. Konczak, T. Linke, and T. Schaub. Graphs and colorings for answer set programming. In *Computing Research Repository (CoRR)*. `http://arxiv.org/abs/cs.AI/0502082`, 2005.

[124] K. Konczak, T. Linke, and T. Schaub. Graphs and colorings for answer set programming. *Theory and Practice of Logic Programming*, 6(1):61–106, 2006.

[125] K. Konczak, T. Schaub, and T. Linke. Graphs and colorings for answer set programming with preferences. *Fundamenta Informaticae*, 57(2-4):393–421, 2003.

[126] K. Konczak, T. Schaub, and T. Linke. Graphs and colorings for answer set programming with preferences: Preliminary report. In M. De Voss and A. Provetti, editors, *Proceedings of the Second International Workshop on Answer Set Programming (ASP'03)*, volume 78, pages 43–56, Messina, 2003. CEUR Workshop Proceedings.

[127] K. Konczak and R. Vogel. Abduction and preferences in linguistics. In M. de Vos and A. Provetti, editors, *Answer Set Programming: Advances in Theory and Implementation (ASP'05)*, pages 263–276. Research Press International, 2005.

[128] K. Konczak and R. Vogel. Abduction and preferences in linguistics: Extended abstract. In C. Baral, G. Greco, N. Leona, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 384–388. Springer, 2005.

[129] J. Lang. Logical preference representation and combinatorial vote. *Annals of Mathematics and Artificial Intelligence*, 42(1):37–71, 2004.

[130] N. Leone, W. Faber, G. Pfeifer, T. Eiter, G. Gottlob, C. Koch, C. Mateis, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*. To appear.

[131] V. Lifschitz. Foundations of logic programming. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 69–127. CSLI Publications, 1996.

[132] V. Lifschitz. Answer set planning. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[133] V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

[134] F. Lin. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In D. Fensel, F. Giunchiglia, D. McGuiness, and M.-A. Williams, editors, *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR'02)*, pages 170–176. Morgan Kaufmann, 2002.

[135] F. Lin and Y. Chen. Discovering classes of strongly equivalent logic programs. In L. P. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 516–521. Professional Book Center, 2005.

[136] F. Lin and J. You. Abduction in logic programming: a new definition and an abductive procedure based on rewriting. *Artificial Intelligence*, 140(1-2):175–205, 2002.

[137] F. Lin and J. Zhao. On tight logic programs and yet another translation from normal logic programs to propositional logic. In G. Gottlob and T. Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 853–858. Morgan Kaufmann, 2003.

[138] F. Lin and Y. Zhao. Assat: computing answer sets of a logic program by sat solvers. *Artificial Intelligence*, 157(1-2):115–137, 2004.

[139] T. Linke. Graph theoretical characterization and computation of answer sets. In B. Nebel, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 641–645. Morgan Kaufmann Publishers, 2001.

[140] T. Linke. Using nested logic programs for answer set programming. In M. de Vos and A. Provetti, editors, *Proceedings of the Second International Workshop on Answer Set Programming (ASP'03)*, volume 78, pages 181–194. CEUR Workshop Proceedings, 2003.

[141] T. Linke, C. Anger, and K. Konczak. More on `nomore`. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proceedings of the Eighth European Conference on Logics in Artificial Intelligence (JELIA'02)*, volume 2424 of *Lecture Notes in Artificial Intelligence*, pages 468–480. Springer-Verlag, 2002.

[142] T. Linke and T. Schaub. An approach to query-answering in Reiter's default logic and the underlying existence of extensions problem. In J. Dix, L. Fariñas del Cerro, and U. Furbach, editors, *Logics in Artificial Intelligence, Proceedings of the Sixth European Workshop on Logics in Artificial Intelligence*, volume 1489 of *Lecture Notes in Artificial Intelligence*, pages 233–247. Springer-Verlag, 1998.

[143] T. Linke and T. Schaub. Alternative foundations for Reiter's default logic. *Artificial Intelligence*, 124(1):31–86, 2000.

[144] J. Lloyd. *Foundations of Logic Programming*. Symbolic Computation. Springer-Verlag, 2nd edition, 1987.

[145] Z. Lonc and M. Truszczyński. On the problem of computing the well-founded semantics. *Theory and Practice of Logic Programming*, 1(5):591–609, 2001.

[146] `Lparse`. Available at http://saturn.tcs.hut.fi/Software/smodels/.

[147] V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In K. Apt, W. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[148] V. W. Marek and M. Truszczyński. Autoepistemic Logic. *Journal of the ACM*, 38(3):588–619, 1991.

[149] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.

[150] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303. The MIT Press, 1996.

[151] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An a-prolog decision support system for the space shuttle. In *Practical Aspects of Declarative Languages, Third International Symposium, PADL 2001*, volume 1990 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2001.

[152] `nomore++`. `http://www.cs.uni-potsdam.de/nomore`.

[153] `NoMoRe`. `http://www.cs.uni-potsdam.de/~linke/nomore`.

[154] M. Osorio, J.A. Navarro, and J. Arrazola. Equivalence in answer set programming. In *Logic Based Program Synthesis and Transformation, 11th International Workshop (LOPSTR 2001)*, volume 2372 of *Lecture Notes in Computer Science*, pages 57–75. Springer, 2001.

[155] C. Papadimitriou and M. Sideri. Default theories that always have extensions. *Artificial Intelligence*, 69:347–357, 1994.

[156] Ch. Papadimitriou. *Computational complexity*. Addison–Wesley, 1994.

[157] D. Pearce. Simplifying logic programs under answer set semantics. In B. Demoen and V. Lifschitz, editors, *Logic Programming, 20th International Conference, ICLP 2004*, volume 3132 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 2004.

[158] D. Pearce, H. Tompits, and S. Woltran. Encodings for equilibrium logic and logic programs with nested expressions. In P. Brazdil and A. Jorge, editors, *Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving (EPIA 2001)*, volume 2258 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2001.

[159] D. Pearce and A. Valverde. Some types of equivalence for logic programs and equilibrium logic. In *Proceedings of International Joint Conference on Declarative Programming, APPIA-GULP-PRODE 2003*, pages 350–361. Universitá degli Studi di Reggio Calabria, 2003.

[160] I. Pivkina, E. Pontelli, and T. Son. Revising knowledge in multi-agent systems using revision programming with preferences. In J. Dix and J. Leite, editors, *Computational Logic in Multi-Agent Systems, 4th International Workshop, CLIMA IV*, volume 3259 of *Lecture Notes in Computer Science*, pages 134–158. Springer, 2004.

[161] `plp`. `http://www.cs.uni-potsdam.de/~torsten/plp`.

[162] S. Pradhan and J. Minker. Using priorities to combine knowledge bases. *International Journal of Cooperative Information Systems*, 5(2-3):333–364, 1996.

[163] T. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann Publishers, 1988.

[164] T. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–463, 1990.

[165] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 55–76. Plenum Press, New York, 1978.

[166] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

[167] K. Ross. A prodedural semantics for well-founded negation in logic programs. *Journal of Logic Programming*, 13(1):1–22, May 1992.

[168] F. Rossi, K. Venable, and T. Walsh. μCP-nets: representing and reasoning with preference of multiple agents. In *Proceedings of AAAI-04*, 2004.

[169] J. Rothe, H. Spakowski, and J. Vogel. Exact complexity of the winner determination problem in young elections. *Theory of Computing Systems*, 36:375–386, 2003.

[170] D. Saccá and C. Zaniolo. Stabel models and non-determinism in logic programs with negation. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 205–217. ACM Press, 1990.

[171] K. Sagonas, T. Swift, and D. Warren. An abstract machine for computing the well-founded semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 274–288. The MIT Press, 1996.

[172] C. Sakama and K. Inoue. Representing priorities in logic programs. In M. Maher, editor, *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 82–96, Cambridge, 1996. The MIT Press.

[173] C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123(1-2):185–222, 2000.

[174] M. A. Satterthwaite. Strategy-proofness and arrow's condition: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, pages 187–217, 1975.

[175] A. Schaerf. A survey of automated timetabling. *Artificial Intelligence*, 13(2):87–127, 1999.

[176] T. Schaub and K. Wang. Preferred well-founded semantics for logic programming by alternating fixpoints: Preliminary report. In *Proceedings of the Workshop on Nonmonotonic Reasoning'2002*, pages 238–246, 2002.

[177] T. Schaub and K. Wang. A semantic framework for preference handling in answer set programming. *Theory and Practice of Logic Programming*, 3(4-5):569–607, 2003.

[178] T. Schmid and R. Vogel. Dialectal variation in german 3-verb-clusters. a surface-oriented optimality theoretic account. *Journal of Comparative Germanic Linguistics*, 7(3):235–274, 2004.

[179] S. Sen, Th. Haynes, and N. Arora. Satisfying user preference while negotiating meetings. *International Journal of Human-Computer Studies*, 47:407–427, 1997.

[180] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

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

[182] T. Soininen. *An Approach to Knowledge Representation and Reasoning for Product Configuration*. Phd thesis, Helsinki University of Technology, Finland, 2000.

[183] T. Son and E. Pontelli. Planning with preferences using logic programming. In V. Lifschitz and I. Niemelä, editors, *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004*, volume 2923 of *Lecture Notes in Computer Science*, pages 247–260. Springer, 2004.

[184] V. Subrahmanian, D. Nau, and C. Vago. Wfs + branch and bound = stable models. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):362–377, 1995.

[185] T. Syrjänen. A rule-based formal model for software configuration. Research Report A55, Helsinki University of Technology, Laboratory for Theoretical Computer Science, December 1999.

[186] T. Syrjänen. Including diagnostic information in configuration models. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. Pereira, Y. Sagiv, and P. Stuckey, editors, *First International Conference on Computational Logic (CL 2000)*, volume 1861 of *Lecture Notes in Computer Science*, pages 837–851. Springer, 2000.

[187] S.-W. Tan and J. Pearl. Qualitative decision theory: specification and representation of preferences under uncertainty. In *Proceedings of KR-94*, 1994.

[188] H. Turner. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 3(4-5):609–622, 2003.

[189] A. van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Science*, 47:185–120, 1993.

[190] A. van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[191] T. Wakaki, K. Inoue, C. Sakama, and K. Nitta. Computing preferred answer sets in answer set programming. In M. Vardi and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 10th International Conference, LPAR 2003*, volume 2850 of *Lecture Notes in Computer Science*, pages 259–273. Springer, 2003.

[192] T. Wakaki, K. Inoue, C. Sakama, and K. Nitta. The plp system. In J. Alferes and J. Leite, editors, *Logics in Artificial Intelligence, 9th European Conference, (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, pages 706–709. Springer, 2004.

[193] K. Wang and L. Zhou. Comparisons and computation of well-founded semantics for disjunctive logic programs. *ACM Transactions on Computational Logic*, 6(2):295–327, 2005.

[194] K. Wang, L. Zhou, and F. Lin. Alternating fixpoint theory for logic programs with priority. In *Proceedings of the First International Conference on Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 164–178. Springer-Verlag, 2000.

[195] J. Ward and J. S. Schlipf. Answer set programming with clause learning. In V. Lifschitz and I. Niemelä, editors, *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Computer Science*, pages 302–313. Springer-Verlag, 2003.

[196] S. Woltran. Characterizations for relativized notions of equivalence in answer set programming. In J. Alferes and J. Leite, editors, *Logics in Artificial Intelligence, 9th European Conference (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, pages 161–173. Springer, 2004.

[197] Y. Zhang. Logic program based updates. *ACM Transactions on Computational Logic*. To appear.

[198] Y. Zhang. Two results for prioritized logic programming. *Theory and Practice of Logic Programming*, 3(2):223–242, 2003.

[199] Y. Zhang and N. Foo. Answer sets for prioritized logic programs. In J. Maluszynski, editor, *Proceedings of the International Symposium on Logic Programming (ILPS-97)*, pages 69–84. The MIT Press, 1997.

[200] Y. Zhang, C-M. Wu, and Y. Bai. Implementing prioritized logic programming. *AI Communications*, 14(4), 2001.

Index