GENERATING EXPLANATIONS FOR COMPLEX BIOMEDICAL QUERIES

by

Umut Öztok

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
August, 2012

GENERATING EXPLANATIONS FOR COMPLEX BIOMEDICAL QUERIES

APPROVED BY:

Asst. Prof. Dr. Esra Erdem (Dissertation Supervisor)	.....
Assoc. Prof. Dr. Hans Tompits	.....
Asst. Prof. Dr. Hüsnü Yenigün	.....
Assoc. Prof. Dr. Uğur Sezerman	.....
Asst. Prof. Dr. Volkan Patoğlu	.....

DATE OF APPROVAL: .....

GENERATING EXPLANATIONS FOR COMPLEX BIOMEDICAL QUERIES

Umut Öztok

Computer Science and Engineering, MS Thesis, 2012

Thesis Supervisor: Esra Erdem

Keywords: answer set programming, biomedical query answering, explanation generation

Abstract

Recent advances in health and life sciences have led to generation of a large amount of biomedical data. To facilitate access to its desired parts, such a big mass of data has been represented in structured forms, like databases and ontologies. On the other hand, representing these databases and ontologies in different formats, constructing them independently from each other, and storing them at different locations have brought about many challenges for answering queries about the knowledge represented in these ontologies and databases.

One of the challenges for the users is to be able to represent such a biomedical query in a natural language, and get its answers in an understandable form. Another challenge is to extract relevant knowledge from different knowledge resources, and integrate them appropriately using also definitions, such as, chains of gene-gene interactions, cliques of genes based on gene-gene relations, or similarity/diversity of genes/drugs. Furthermore, once an answer is found for a complex query, the experts may need further explanations about the answer. The first two challenges have been addressed earlier using Answer Set Programming (ASP), with the development of a software system (called BIOQUERY-ASP). This thesis addresses the third challenge: explanation generation in ASP.

In this thesis, we extend the earlier work on the first two challenges, to new forms of biomedical queries (e.g., about drug similarity) and to new biomedical knowledge resources. We introduce novel mathematical models and algorithms to generate (shortest or k different) explanations for queries in ASP, and provide a comprehensive theoretical analysis of these methods. We implement these algorithms and integrate them in BIOQUERY-ASP, and provide an experimental evaluation of our methods with some complex biomedical queries over the biomedical knowledge resources PHARMGKB, DRUGBANK, BIOGRID, CTD, SIDER, DISEASEONTOLOGY and ORPHADATA.

KARMAŞIK BİYOMEDİKAL SORGULAR İÇİN AÇIKLAMA ÜRETME

Umut Öztok

Bilgisayar Bilimi ve Mühendisliği, Yüksek Lisans Tezi, 2012

Tez Danışmanı: Esra Erdem

Anahtar Kelimeler: çözüm kümesi programlama, sorgu cevaplama, açıklama üretme

Özet

Özellikle son yıllarda sağlık ve yaşam bilimleri alanlarındaki gelişmeler çok büyük miktarda bir veri üretimine yol açmıştır. Bu verinin gerekli kısımlarına ulaşımı kolaylaştırmak amacıyla, veritabanları ve ontolojiler gibi veri saklama yöntemleri kullanılmaktadır. Öte yandan, veritabanlarının ve ontolojilerin farklı formatlarda ve birbirinden bağımsız şekilde oluşturulması ve farklı yerlerde saklanması, bu veritabanları ve ontolojilerde gösterilen bilgi ile ilgili karmaşık sorguları cevaplamayı farklı açılardan zorlaştırmıştır.

Bu zorluklardan biri, biyomedikal bir sorguyu doğal bir dilde göstermek ve cevaplarını anlaşılabilir bir biçimde uzmanlara sunmaktır. Başka bir zorluk ise, farklı bilgi kaynaklarından ilgili bilginin çıkartılıp, uygun bir şekilde biraraya getirilmesidir. Farklı bilgi kaynaklarının entegrasyonu sırasında, genler arası ilişkilerden oluşan zincirler, genler arası ilişkilere bağlı olan klikler, birbirine benzer veya birbirinden farklı genler ya da ilaçlar gibi tanımların da göz önünde bulundurulması gerekmektedir. Bunlara ek olarak, bir başka zorluk ise karmaşık bir sorgu için bulunan yanıt hakkında ilgili açıklamaların üretilmesidir. Yukarıda bahsedilen ilk iki problem daha önce Çözüm Kümesi Programlama (ÇKP) kullanılarak çalışılmıştır. Bu çalışmaların sonucu BIOQUERY-ASP adı ver-

ilen bir yazılım sistemi geliştirilmiştir. Bu tez çalışması, yukarıda bahsedilen üçüncü problem üstündedir: açıklama üretme.

Bu tez kapsamında, ilk iki problemle alakalı olarak önceden yapılan çalışmalar, ilaç benzerlikleri hakkında sorgular gibi yeni biyomedikal sorgulara da uygulanacak şekilde ve yeni biyomedikal bilgi kaynaklarından da faydalanacak şekilde genişletilmiştir. ÇKP'de gösterimi yapılan sorgulara (en kısa veya k tane farklı) açıklama üretebilmek için, yeni matematiksel modeller ve algoritmalar geliştirilmiştir. Bu algoritmaların kapsamlı olarak teorik analizi yapılmış, yazılımları BIOQUERY-ASP ile bütünleştirilmiştir. ÇKP'ye dayanan bu yöntemlerimizin, PHARMGKB, DRUGBANK, BIOGRID, CTD, SIDER, DISEASEONTOLOGY ve ORPHADATA gibi biyomedikal bilgi kaynakları üzerinden bazı karmaşık biyomedikal sorguları yanıtlayarak deneysel bir değerlendirmesi de yapılmıştır.

Acknowledgements

I want to thank my supervisor, Esra Erdem, without whose invaluable support this work would have been of much lesser quality. I have learned a lot from her about writing academic papers and presenting theoretical results.

I thank Alev Topuzoğlu for keeping me in contact with the beautiful world of mathematics during my master's studies.

I thank the members of my thesis committee, Hans Tompits, Hüsnü Yenigün, Uğur Sezerman, and Volkan Patoğlu, for their valuable comments and suggestions.

I thank the Scientific and Technological Research Council of Turkey (TUBITAK) for the financial support provided to me through the BIDEB scholarship during my master's studies.

I would like to thank Aysu Okbay and Firat H. Tahaoğlu, who showed me that I can live anywhere, even in Tuzla, if I am surrounded with my awesome friends.

I would like to give special thanks to Erdi Aker and Suha O. Mutluergil for the "humorous" moments we had in the gloomy aura of FENS 2014 late in the nights.

I furthermore thank İnanç Arın, Uğur Bağcı, and Yaşar Tüzel for their exceptional friendships.

I wish to thank Doğa Gizem Kısa who made me to see the world through rose-colored glassess again as in my childhood. With her, the grass is always greener.

Obviously, the most important supporters of not just this work but anything related to me are my parents, Gülçin and Necdet Öztok, and my lovely sister, Başak Öztok. I am indebted to them for their unprecedented endurance, endless support and unconditional love for me. Without them, it would have been impossible for me, such a lazy boy, to complete this work.

TABLE OF CONTENTS

Page

1 INTRODUCTION **1**

1.1 Contributions of the Thesis 3

1.2 Thesis Outline 5

2 ANSWER SET PROGRAMMING **6**

2.1 Programs 7

2.2 Generate-And-Test Representation Methodology with Special ASP Constructs 9

2.3 Presenting Programs to Answer Set Solvers 11

2.4 Example: Graph Coloring Problem 12

3 EXTENDING BIOQUERY-ASP TO ANSWER NEW QUERIES **15**

3.1 New Types of Biomedical Queries 15

3.1.1 Negation in Queries 15

3.1.2 Queries about Symptoms of Diseases 17

3.1.3 Similarity/Diversity of Drugs 17

3.1.4 Finding Close/Distant Drugs to a Given Drug 19

3.2 New Biomedical Knowledge Resources 23

3.3 New Experiments 23

4 EXPLANATION GENERATION IN ASP **28**

4.1 Explanations in ASP 28

4.2 Generating Shortest Explanations 34

4.3 Generating k Different Explanations 39

4.4 Presenting Explanations in a Natural Language 44

4.5 Experiments with Biomedical Queries 45

4.6 Implementation of Explanation Generation Algorithms 48

5 RELATING EXPLANATIONS TO JUSTIFICATIONS **50**

5.1 Offline Justifications 51

5.2 From Justifications to Explanations 57
5.3 From Explanations to Justifications 60

6 PROOFS **63**
6.1 Generating Shortest Explanations 63
6.1.1 Proof of Proposition 2 – Termination of Algorithm 2 63
6.1.2 Proof of Proposition 3 – Soundness of Algorithm 2 64
6.1.3 Proof of Proposition 4 – Complexity of Algorithm 2 73
6.2 Generating k Different Explanations 74
6.2.1 Proof of Proposition 5 – Termination of Algorithm 6 74
6.2.2 Proof of Proposition 6 – Soundness of Algorithm 6 74
6.2.3 Some Properties of Algorithm 6 77
6.2.4 Proof of Proposition 8 – Complexity of Algorithm 6 81
6.3 Relations between Explanations and Justifications 81
6.3.1 Proof of Proposition 10 – Soundness of Algorithm 8 81
6.3.2 Proof of Proposition 11 – Soundness of Algorithm 9 85

7 DISCUSSION AND CONCLUSION **89**

A SHORTEST EXPLANATIONS FOR BIOMEDICAL QUERIES **93**

BIBLIOGRAPHY **102**

LIST OF TABLES

Page

1.1 A list of complex biomedical queries. 2

2.1 Applications of ASP. 7
2.2 ASP solvers. 8

3.1 Grammar of BIOQUERY-CNL*. 16
3.2 Special functions used in BIOQUERY-CNL*. 17
3.3 Knowledge resources and their relations. 24
3.4 Experimental results (using CLASP). 25
3.5 Experimental Results (using DLV). 26
3.6 Experimental results for closeness/distantness queries. 27

4.1 Predicate look-up table used while expressing explanations in natural language. 45
4.2 Experimental results for generating shortest explanations for some biomedical queries, using Algorithm 2. 46
4.3 Experimental results for generating different explanations for some biomedical queries, using Algorithm 6. 48

7.1 Comparison of SLAP and our greedy method for finding closest drugs. . . 90

LIST OF FIGURES

Page

1.1 System overview of BIOQUERY-ASP. 4

2.1 Presenting COLORING to GRINGO. 13
2.2 Presenting a COLORING instance to GRINGO. 14

3.1 ASP program $\Pi^{\Delta_S}_{\text{grdy}}(i, d)$. 22

4.1 The and-or explanation tree for Example 2. 32
4.2 Explanation trees for Example 3. 33
4.3 Explanations for Example 4. 33
4.4 (a) The and-or explanation tree for a and (b) an explanation for a. 34
4.5 An explanation for Q8. 35
4.6 Another explanation for Q8. 36
4.7 A generic execution of Algorithm 2. 37
4.8 An explanation for Q5. 39
4.9 Another explanation for Q5. 40
4.10 A generic execution of Algorithm 6. 43
4.11 A shortest explanation for Q8. 44
4.12 A shortest explanation for Q1. 47
4.13 A snapshot of BIOQUERY-ASP showing its explanation generation facility. 49

5.1 An offline justification for Example 7. 50
5.2 An e-graph for Example 9. 53
5.3 An offline justification for Example 14. 57
5.4 (a) An offline justification and (b) its corresponding explanation tree obtained by using Algorithm 8. 60
5.5 (a) An explanation tree and (b) its corresponding offline justification obtained by using Algorithm 9. 62

6.1 Part of the recursion tree for createTree$(\Pi, X, d, \emptyset)$. 67

A.1 A shortest explanation for Q1 93
A.2 A shortest explanation for Q2 94
A.3 A shortest explanation for Q3 95
A.4 A shortest explanation for Q4 96
A.5 A shortest explanation for Q5 97
A.6 A shortest explanation for Q8 98
A.7 A shortest explanation for Q10 99
A.8 A shortest explanation for Q11 100
A.9 A shortest explanation for Q12 101

List of Algorithms

1 Generating n Closest Drugs 21
2 Generating Shortest Explanations 37
3 createTree 38
4 calculateWeight 39
5 extractExp 40
6 Generating k Different Explanations 41
7 calculateDifference 42
8 Justification to Explanation 58
9 Explanation to Justification 61

CHAPTER 1

INTRODUCTION

Recent advances in health and life sciences have led to generation of a large amount of biomedical data. To facilitate access to its desired parts, such a big mass of data has been represented in structured forms, like databases and ontologies. On the other hand, representing these databases and ontologies in different formats, constructing them independently from each other, and storing them at different locations have brought about many challenges for answering queries about the knowledge represented in these ontologies and databases.

One of the challenges for the users is to be able to represent such a biomedical query in a natural language, and get its answers in an understandable form. Another challenge is to extract relevant knowledge from different knowledge resources, and integrate them appropriately using also definitions, such as, chains of gene-gene interactions, cliques of genes based on gene-gene relations, or similarity/diversity of genes/drugs. Furthermore, once an answer is found for a complex query, the experts may need further explanations about the answer.

For instance, consider the query Q6 in Table 1.1 which displays a list of complex biomedical queries that are important from the point of view of drug discovery.[1] New molecule synthesis by changing substitutes of parent compound may lead to different biochemical and physiological effects; and each trial may lead to different indications. Such studies are important for fast inventions of new molecules. For example, while developing Lovastatin (a member of the drug class of statins, used for lowering cholesterol) from Aspergillus terreus (a sort of fungus) in 1979, scientists at Merck derived a new molecule named Simvastatin (a hypolipidemic drug used to control elevated cholesterol). Therefore, identifying genes targeted by a group of drugs automatically by means of queries like Q6 may be useful for experts.

Once an answer to a query is computed, the experts may ask for an explanation to

[1] In Table 1.1, drug-drug interactions present negative interactions among drugs. Gene-gene interactions present both negative and positive interactions among genes.

Table 1.1: A list of complex biomedical queries.

Q1	What are the drugs that treat the disease Asthma and that target the gene ADRB1?
Q2	What are the side effects of the drugs that treat the disease Asthma and that target the gene ADRB1?
Q3	What are the genes that are targeted by the drug Epinephrine and that interact with the gene DLG4?
Q4	What are the genes that interact with at least 3 genes and that are targeted by the drug Epinephrine?
Q5	What are the drugs that treat the disease Asthma or that react with the drug Epinephrine?
Q6	What are the genes that are targeted by all the drugs that belong to the category Hmg-coa reductase inhibitors?
Q7	What are the cliques of 5 genes, that contain the gene DLG4?
Q8	What are the genes that are related to the gene ADRB1 via a gene-gene interaction chain of length at most 3?
Q9	What are the 3 most similar genes that are targeted by the drug Epinephrine?
Q10	What are the genes that are related to the gene DLG4 via a gene-gene interaction chain of length at most 3 and that are targeted by the drugs that belong to the category Hmg-coa reductase inhibitors?
Q11	What are the drugs that treat the disease Depression and that do not target the gene ACYP1?
Q12	What are the symptoms of diseases that are treated by the drug Triadimefon?
Q13	What are the 3 most similar drugs that target the gene DLG4?
Q14	What are the 3 closest drugs to the drug Epinephrine?

have a better understanding. For instance, an answer for the query Q3 in Table 1.1 is "ADRB1". A shortest explanation for this answer is computed as follows.

> The drug Epinephrine targets the gene ADRB1 according to CTD.
> The gene DLG4 interacts with the gene ADRB1 according to BIOGRID.

Most of the existing biomedical querying systems (e.g., web services built over the available knowledge resources) support keyword search but not complex queries like the queries in Table 1.1. Some of these complex queries, such as Q3 or Q6, can be represented in a formal query language (e.g., SQL/SPARQL) and then answered using Semantic Web technologies. However, queries, like Q8, that require auxiliary recursive definitions (such as transitive closure) cannot be directly represented in these languages; and thus such queries cannot be answered directly using Semantic Web technologies. The experts usually compute auxiliary relations externally, for instance, by enumerating all gene-gene interaction chains or gene cliques, and then use these auxiliary relations to represent and answer a query like Q7 or Q8. Similarity/diversity queries, like Q9 or Q13, cannot be represented directly in these languages either, and require a sophisticated reasoning algorithm. Also, none of the existing systems can provide informative explanations about the

answers, but point to related web pages of the knowledge resources available online.

To address the challenges described above, novel methods and a software system, called BIOQUERY-ASP [36] (Figure 1.1), have been developed using Answer Set Programming(ASP) [63, 16]. In particular, the following studies have been completed for the first two challenges:

- Erdem and Yeniterzi [39] developed a controlled natural language, BIOQUERY-CNL, for expressing biomedical queries related to drug discovery. For instance, queries Q1–Q10 in Table 1.1 are in this language. They also developed an algorithm to translate a given query in BIOQUERY-CNL to a program in ASP.
- Bodenreider et al. [11] introduced methods to extract biomedical information from various knowledge resources and integrate them by a rule layer. This rule layer not only integrates those knowledge resources but also provides definitions of auxiliary concepts.
- Erdem et al. [35] have introduced an algorithm for query answering by identifying the relevant parts of the rule layer and the knowledge resources with respect to a given query.

The focus of this thesis is about the last challenge: generating explanations for biomedical queries.

1.1 Contributions of the Thesis

The contributions of the thesis can be summarized in two parts.

- Extension of the earlier work on BIOQUERY-ASP to new forms of queries and new knowledge resources:
  - We have extended the grammar of BIOQUERY-CNL to construct negative queries (e.g., query Q11), queries about symptoms of diseases (e.g., query Q12) and similarity/diversity queries about drugs (e.g., query Q13). We call this extended controlled natural language as BIOQUERY-CNL*.
  - We have modified the system BIOQUERY-ASP to allow users for constructing such new queries as well. We have also extended BIOQUERY-ASP to most recent biomedical knowledge resources about drugs, genes, and diseases,

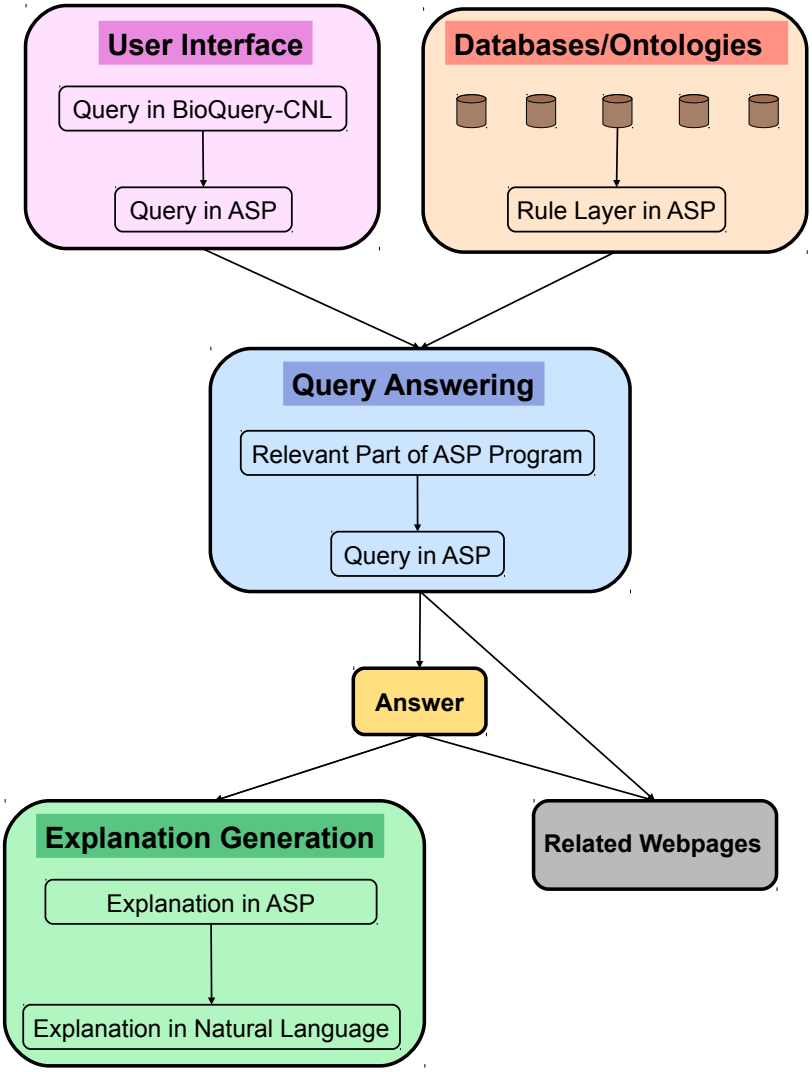


Figure 1.1: System overview of BIOQUERY-ASP.

such as PHARMGKB[2] [67], DRUGBANK[3] [55], BIOGRID[4] [89], CTD[5] [22], SIDER[6] [57], DISEASEONTOLOGY[7], and ORPHADATA[8].

– To study queries about similarity of drugs (e.g., finding the closest/distant drugs to a given drug), we have introduced new distance measures for drugs and naive/greedy algorithms to compute queried drugs. We have also related the problem to the problem similar/diverse solutions studied in [29].

[2] http://www.pharmgkb.org/
[3] http://www.drugbank.ca/
[4] http://thebiogrid.org/
[5] http://ctd.mdibl.org/
[6] http://sideeffects.embl.de/
[7] http://disease-ontology.org
[8] http://www.orphadata.org/cgi-bin/index.php/

- Adding a new feature to BIOQUERY-ASP: Explanation Generation.
  - We have formally defined "explanations" in ASP, utilizing properties of programs and graphs. We have also defined variations of explanations, such as "shortest explanations" and "k different explanations".
  - We have introduced novel generic algorithms to generate explanations for biomedical queries. These algorithms can compute shortest or k different explanations. We have analyzed the termination, soundness, and complexity of those algorithms.
  - We have developed a computational tool, called EXPGEN-ASP, that implements these explanation generation algorithms.
  - We have showed the applicability of our methods to generate explanations for answers of complex biomedical queries.
  - We have embedded EXPGEN-ASP into BIOQUERY-ASP so that the experts can obtain explanations regarding the answers of biomedical queries, in a natural language.

1.2 Thesis Outline

The rest of the thesis is organized as follows. In Chapter 2, we provide a summary of Answer Set Programming. Next, in Chapter 3, we give an overview of BIOQUERY-ASP, the earlier work done, and the new extensions in the scope of this thesis. Then, in Chapter 4, we describe explanation generation in ASP. After discussing related work in Chapter 5, we provide in Chapter 6 the proofs of the theorems stated throughout the thesis and conclude the thesis in Chapter 7 by summarizing our contributions and pointing out some possible future work.

CHAPTER 2

ANSWER SET PROGRAMMING

Answer Set Programming (ASP) [7, 63, 16] is a form of declarative programming paradigm oriented towards solving, in general Σ_2^{P}, combinatorial search and knowledge-intensive problems. The idea is to represent a problem as a "program" whose models (called "answer sets" [50]) correspond to the solutions. The answer sets for the given program can then be computed by special implemented systems called answer set solvers. ASP has an high-level representation language that allows recursive definitions, aggregates, weight constraints, optimization statements, and default negation.

ASP also provides efficient solvers (see Table 2.2), such as CLASP [46], which has won first places at competitions like ASP'07/09/11/12, PB'09/11/12, and SAT'/09/11/12. Due to the continuous improvement of the ASP solvers and highly expressive representation language of ASP which is supported by a strong theoretical background that results from a years of intensive research, ASP has been applied fruitfully to a wide range of areas (see Table 2.1). Here are, for instance, three applications of ASP used in industry:

- *Decision Support Systems*: An ASP-based system was developed to help flight controllers of space shuttle solve some planning and diagnostic tasks [73] (used by United Space Alliance).

- *Automated Product Configuration*: A web-based commercial system uses an ASP-based product configuration technology [92] (used by Variantum Oy).[1]

- *Workforce Management*: An ASP-based system is developed to build teams of employees to handle incoming ships by taking into account a variety of requirements, e.g., skills, fairness, regulations [80] (used by Gioia Tauro seaport).

Let us briefly explain the syntax and semantics of ASP programs and describe how a computational problem can be solved in ASP.

[1] `http://www.variantum.com/en/`

Table 2.1: Applications of ASP.

Area	**References**
planning	[24] [62] [87]
theory update/revision	[54]
preferences	[83] [15]
diagnosis	[30] [6] [27]
learning	[81]
description logics and semantic web	[17] [32] [91]
probabilistic reasoning	[8]
data integration and question answering	[2] [59]
multi-agent systems	[87] [88] [96]
wire routing	[38] [26]
decision support systems	[73]
bounded model checking	[53]
game theory	[66] [97]
logic puzzles	[42]
phylogenetics	[28] [19] [37] [33]
combinatorial auctions	[9]
haplotype inference	[34] [95]
systems biology	[93] [43] [82] [48]
automatic music composition	[13] [12]
verification of cryptographic protocols	[23]
assisted living	[68] [69]
context	[27]
scheduling	[5]
team-building	[80]
package-configuration	[45]
e-tourism	[79]
indoor positioning	[70]
ontologies	[76]
information extraction	[71]

2.1 Programs

Syntax The input language of ASP programs are composed of three sets namely *constant symbols*, *predicate symbols*, and *variable symbols* where intersection of constant symbols and variable symbols is empty. The basic elements of the ASP programs are *atoms*. An atom $p(\vec{t})$ is composed of a predicate symbol p and *terms* $\vec{t} = t_1, \ldots, t_k$ where each t_i $(1 \leq i \leq k)$ is either a constant or a variable. A *literal* is either an atom $p(\vec{t})$ or its negated form $not\ p(\vec{t})$.

An ASP program is a finite set of *rules* of the form:

$$A \leftarrow A_1, \ldots, A_k, not\ A_{k+1}, \ldots, not\ A_m \tag{2.1}$$

where $m \geq k \geq 0$ and each A_i is an atom; whereas, A is an atom or $\bot$.

For a rule r of the form (2.1), A is called the *head* of the rule and denoted by $H(r)$. The conjunction of the literals $A_1, \ldots, A_k, not\ A_{k+1}, \ldots, not\ A_m$ is called the *body* of r.

Table 2.2: ASP solvers.

Solver	Year	University	Reference
SMODELS	1996	Helsinki University of Technology	[72]
DLV	1997	University of Calabria	[60]
CMODELS	2002	University of Texas-Austin	[51]
ASSAT	2003	Hong Kong University of Science and Technology	[64]
PBMODELS	2005	University of Kentucky	[65]
DLVHEX	2006	Vienna University of Technology	[31]
CLASP	2006	University of Potsdam	[46]
ASPERIX	2008	University of Angers	[58]
SUP	2008	University of Kentucky	[61]
WASP	2011	University of Calabria	[25]

The set $\{A_1, ..., A_k\}$ of atoms (called the positive part of the body) is denoted by $B^+(r)$, and the set $\{A_{k+1}, ..., A_m\}$ of atoms (called the negative part of the body) is denoted by $B^-(r)$, and all the atoms in the body are denoted by $B(r) = B^+(r) \cup B^-(r)$.

We say that a rule r is a *fact* if $B(r) = \emptyset$, and we usually omit the $\leftarrow$ sign. Furthermore, we say that a rule r is a *constraint* if the head of r is $\bot$, and we usually omit the $\bot$ sign.

Semantics (Answer Sets) Answer sets of a program are defined over *ground programs*. We call an atom, rule, or program *ground*, if it does not contain any variables. Given a program Π, the set $\mathcal{U}_\Pi$ represents all the constants in Π, and the set $\mathcal{B}_\Pi$ represents all the ground atoms that can be constructed from atoms in Π with constants in $\mathcal{U}_\Pi$. Also, $Ground(\Pi)$ denotes the set of all the ground rules which are obtained by substituting all variables in rules with the set of all possible constants in $\mathcal{U}_\Pi$.

A subset I of $\mathcal{B}_\Pi$ is called an *interpretation* for Π. A ground atom p is true with respect to an interpretation I if $p \in I$; otherwise, it is false. Similarly, a set S of atoms is true (resp., false) with respect to I if each atom $p \in S$ is true (resp., false) with respect to I. An interpretation I *satisfies* a ground rule r, if $H(r)$ is true with respect to I whenever $B^+(r)$ is true and $B^-(r)$ is false with respect to I. An interpretation I is called a *model* of a program Π if it satisfies all the rules in Π.

The *reduct* Π^I of a program Π with respect to an interpretation I is defined as follows:

$$\Pi^I = \{H(r) \leftarrow B^+(r) \mid r \in Ground(\Pi)\ s.t.\ I \cap B^-(r) = \emptyset\}$$

An interpretation I is an *answer set* for a program Π, if it is a subset-minimal model for Π^I, and $AS(\Pi)$ denotes the set of all the answer sets of a program Π.

For example, consider the following program Π_1:

$$p \leftarrow not\ q \tag{2.2}$$

and take an interpretation $I = \{p\}$. The reduct Π_1^I is as follows:

$$p \tag{2.3}$$

I is a model of the reduct (2.3). Let us take a strict subset I' of I which is $\emptyset$. Then, reduct $\Pi_1^{I'}$ is again equal to (2.3); however, I' does not satisfy (2.3). Therefore, $I = \{p\}$ is a subset-minimal model; hence an answer set of Π_1. Note also that $\{p\}$ is the only answer set of Π.

The not in ASP programs is called *negation as failure* and is different from classical negation in SAT [10] in terms of its non-monotonicity. Let the *conclusion* of a program be the intersection of its all answer sets. To understand the non-monotonicity of ASP programs, we need to observe the changes in the conclusion of programs when we extend them.

Consider the following program Π_2:

$$\begin{array}{l} p \leftarrow not\ q \\ q \leftarrow not\ p \end{array} \tag{2.4}$$

Note that Π_2 has one extra rule compared to Π_1 and has two answer sets $\{p\}$ and $\{q\}$. Adding a rule to the program Π_1 removes the element in its conclusion; hence decreases the size of its conclusion from $\{p\}$ to $\emptyset$. Now, consider that we add a constraint to Π_2 and obtain the following program Π_3:

$$\begin{array}{l} p \leftarrow not\ q \\ q \leftarrow not\ p \\ \leftarrow p \end{array} \tag{2.5}$$

Π_3 has a single answer set $\{q\}$. Note that the size of conclusion of Π_2 increases from $\emptyset$ to $\{q\}$ when we add the new constraint. We can observe that when we extend an ASP program by adding new rules, the change in the size of its conclusion is neither monotonic nor anti-monotonic. This is why the semantics of ASP is considered to be non-monotonic unlike SAT.

2.2 Generate-And-Test Representation Methodology with Special ASP Constructs

The idea of ASP [63] is to represent a computational problem as a program whose answer sets correspond to the solutions of the problem, and to find the answer sets for that program using an answer set solver.

When we represent a problem in ASP, two kinds of rules play an important role: those that "generate" many answer sets corresponding to "possible solutions", and those

that can be used to "eliminate" the answer sets that do not correspond to solutions. Rules (2.4) are of the former kind: they generate the answer sets $\{p\}$ and $\{q\}$. Constraints are of the latter kind. For instance, adding the constraint

$$\leftarrow p$$

to program (2.4) as in (2.5) eliminates the answer sets for the program that contain p.

In ASP, we use special constructs of the form

$$\{A_1, \ldots, A_n\}^c \tag{2.6}$$

(called *choice expressions*), and of the form

$$l \leq \{A_1, \ldots, A_m\} \leq u \tag{2.7}$$

(called *cardinality expressions*) where each A_i is an atom and l and u are nonnegative integers denoting the "lower bound" and the "upper bound" [85]. Programs using these constructs can be viewed as abbreviations for normal nested programs defined in [41]. For instance, the following program

$$1 \leq \{p, q\}^c \leq 1 \leftarrow$$

stands for program (2.4). The constraint

$$\leftarrow \ 2 \leq \{p, q, r\}$$

stands for the constraints

$$\begin{array}{l} \leftarrow \ p, q \\ \leftarrow \ p, r \\ \leftarrow \ q, r \end{array}$$

Expression (2.6) describes subsets of $\{A_1, \ldots, A_n\}$. Such expressions can be used in heads of rules to generate many answer sets. For instance, the answer sets for the program

$$\{p, q, r\}^c \ \leftarrow \tag{2.8}$$

are arbitrary subsets of $\{p, q, r\}$. Expression (2.7) describes the subsets of the set $\{A_1, \ldots, A_m\}$ whose cardinalities are at least l and at most u. Such expressions can be used in constraints to eliminate some answer sets. For instance, adding the constraint

$$\leftarrow \ 2 \leq \{p, q, r\}$$

to program (2.8) eliminates the answer sets for (2.8) whose cardinalities are at least 2. Adding the constraint

$$\leftarrow \ not \ (1 \leq \{p, q, r\}) \tag{2.9}$$

to program (2.8) eliminates the answer sets for (2.8) whose cardinalities are not at least 1. Similarly, adding the constraint

$$\leftarrow \ not\ (\{p, q, r\} \leq 1) \tag{2.10}$$

to program (2.8) eliminates the answer sets for (2.8) whose cardinalities are not at most 1.

We abbreviate the rules

$$\begin{array}{l} \{A_1, \ldots, A_m\}^c \leftarrow \mathit{Body} \\ \leftarrow not\ (l \leq \{A_1, \ldots, A_m\}) \\ \leftarrow not\ (\{A_1, \ldots, A_m\} \leq u) \end{array}$$

by

$$l \leq \{A_1, \ldots, A_m\}^c \leq u \leftarrow \mathit{Body}.$$

For instance, rules (2.8), (2.9) and (2.10) can be written as

$$1 \leq \{p, q, r\}^c \leq 1 \leftarrow$$

whose answer sets are the singleton subsets of $\{p, q, r\}$.

In ASP, there are also special constructs that are useful for optimization problems. For instance, to compute answer sets that contain the maximum number of elements from the set $\{A_1, \ldots, A_m\}$, we can use the following optimization statement.

$$maximize\langle\{A_1, \ldots, A_m\}\rangle$$

2.3 Presenting Programs to Answer Set Solvers

Once we represent a computational problem as a program whose answer sets correspond to the solutions of the problem, we can use an answer set solver to compute the solutions of the problem. To present a program to an answer set solver, like CLASP, we need to make some syntactic modifications.

Recall that answer sets for a program are defined over ground programs. Thus, the input of ASP solvers should be ground instantiations of the programs. For that, programs go through a "grounding" phase in which variables in the program (if exists) are substituted by constants. For CLASP, we use the "grounder" GRINGO [44].

Although the syntax of the input language of GRINGO is somewhat more restricted than the class of programs defined above, it provides a number of useful special constructs. For instance, the head of a rule can be an expression of one of the forms

$$\begin{array}{l} \{A_1, \ldots, A_n\}^c \\ l \leq \{A_1, \ldots, A_n\}^c \\ \{A_1, \ldots, A_n\}^c \leq u \\ l \leq \{A_1, \ldots, A_n\}^c \leq u \end{array}$$

but the superscript ^{c} and the sign $\leq$ are dropped. The body can also contain cardinality expressions but the sign $\leq$ is dropped. In the input language of GRINGO, `:-` stands for $\leftarrow$, and each rule is followed by a period. For facts $\leftarrow$ is dropped. For instance, the rule

$$1 \leq \{p, q, r\}^c \leq 1 \leftarrow$$

can be presented to GRINGO as follow:

```
1{p,q,r}1.
```

Variables in a program are represented by strings whose initial letters are capitalized. The constants and predicate symbols, on the other hand, start with a lowercase letter. For instance, the program Π_n

$$p_i \leftarrow \mathit{not}\ \ p_{i+1} \qquad (1 \leq i \leq n)$$

can be presented to GRINGO as follows:

```
index(1..n).
p(I) :- not p(I+1), index(I).
```

Here, the auxiliary predicate `index` is a "domain predicate" used to describe the ranges of variables. Variables can be also used "locally" to describe the list of formulas. For instance, the rule

$$1 \leq \{p_1, \ldots, p_n\} \leq 1$$

can be expressed in GRINGO as follows

```
index(1..n).
1{p(I) : index(I)}1.
```

2.4 Example: Graph Coloring Problem

Given a set $C = \{c_1, \ldots, c_n\}$ of colors and a graph $G = \langle V, E\rangle$ where V is the set of vertices and E is the set of edges, the graph coloring problem, COLORING, is to decide whether there exists an assignment of colors in C to vertices in V such that the following hold:

(i) every vertex in V is assigned to exactly one color,

(ii) no two vertices connected by an edge in E are assigned to the same color.

```
% Assign exactly one color to every vertex
1{assign(V,C) : color(C)}1 :- vertex(V).
% Ensure that no two adjacent vertices have the same color
:- assign(V1,C), assign(V2,C), edge(V1,V2), V1 < V2.
```

Figure 2.1: Presenting COLORING to GRINGO.

To represent COLORING in ASP, we can use the generate-and-test methodology described above. We can describe a solution by utilizing the atoms of the form $assign(v, c)$; meaning that a vertex v in V is assigned to a color c in C.

In the "generate" part of the ASP program, we assign exactly one color to every vertex in V as follows:

$$1 \leq \{assign(v, c_1), \ldots, assign(v, c_n)\}^c \leq 1 \leftarrow vertex(v) \qquad (v \in V) \tag{2.11}$$

In the "test" part, we ensure that no adjacent vertices have the same color by using the following constraints:

$$\leftarrow assign(v, c), assign(v', c), edge(v, v') \qquad (v \neq v') \tag{2.12}$$

Then, answer sets of the program consisting of the rules (2.11) $\cup$ (2.12) (along with a description of a set of colors and a graph) describe assignments of colors to vertices that satisfy COLORING.

As an example, the program consisting of the rules (2.11) $\cup$ (2.12) describing COLORING can be represented in the input language of GRINGO as in Figure 2.1. The expression `{assign(V,C) : color(C)}` is an abbreviation for

$$\{assign(v, c_1), \ldots, assign(v, c_n)\}$$

where $v \in V$. To use this program, we need to combine it with a description of a set of colors and a graph, like in Figure 2.2. The first rule indicates that the input graph has four vertices. The rules in the second line correspond to the edges of the graph. Rules in the last line represent the possible colors that can be used in a color assignment, i.e., the set of colors.

Then, CLASP finds the following answer set for the union of these programs:

$\{vertex(1), vertex(2), vertex(3), vertex(4),$
$edge(1, 2), edge(2, 3), edge(3, 4), edge(1, 4),$
$color(yellow), color(blue), color(white),$
$assign(4, white), assign(3, yellow), assign(2, blue), assign(1, yellow)\}$

The `vertex` and `edge` predicates correspond to the set of vertices and edges of the given graph, respectively. The `assign` predicates correspond to color assignments to vertices.

```
vertex(1..4).
edge(1,2). edge(2,3). edge(3,4). edge(1,4).
color(yellow). color(blue). color(white).
```

Figure 2.2: Presenting a COLORING instance to GRINGO.

According to this answer set, the color assignment where vertices 1 and 3 are colored to yellow, and vertices 2 and 4 to blue, satisfies COLORING.

In a variation of this problem, suppose that we want to maximize the number of vertices colored in blue. In ASP, it can be represented by the program consisting of the rules (2.11) $\cup$ (2.12) and the following optimization statement.

$$maximize\langle\{v : assign(v, blue)\}\rangle$$

We represent this problem in the input language of GRINGO by the rules in Figure 2.1 and the following optimization statement.

```
#maximize [ assign(V,blue) ].
```

Then, CLASP finds following optimal answer set:

$$\{vertex(1), vertex(2), vertex(3), vertex(4),\\ edge(1,2), edge(2,3), edge(3,4), edge(1,4),\\ color(yellow), color(blue), color(white),\\ assign(4, white), assign(3, blue), assign(2, yellow), assign(1, blue)\}$$

Observe that this answer set contains two vertices colored to blue, whereas the previously computed one (without optimization statement) has a single vertex colored to blue.

CHAPTER 3

EXTENDING BIOQUERY-ASP TO ANSWER NEW QUERIES

We have earlier developed the software system BIOQUERY-ASP [36] (see Figure 1.1) that answers complex queries requiring appropriate integration of relevant knowledge from different knowledge resources and auxiliary definitions such as chains of drug-drug interactions, cliques of genes based on gene-gene relations, or similar/diverse genes. As depicted in Figure 1.1, BIOQUERY-ASP takes a query in a controlled natural language and transforms it into ASP. Meanwhile, it extracts knowledge from biomedical databases and ontologies, and integrates them in ASP. Afterwards, it computes an answer to the given query using an ASP solver.

We extend BIOQUERY-ASP to answer new types of queries (e.g., negative queries, queries about symptoms of diseases, similarity/diversity queries about drugs) by incorporating new biomedical knowledge resources related to drugs, genes, and diseases (e.g., DISEASEONTOLOGY, ORPHADATA).

3.1 New Types of Biomedical Queries

3.1.1 Negation in Queries

BIOQUERY-ASP allows us to construct positive queries such that various positive relations between instances of drugs, genes and diseases can be answered. Sometimes, an expert might be interested in discovering negative relations (or combinations of both positive and negative relations) among those concepts. Consider, for instance, the following query Q11 from Table 1.1.

Q11 What are the drugs that treat the disease Depression and that do not target the gene ACYP1?

Table 3.1: Grammar of BIOQUERY-CNL*.

QUERY →	WHATQUERY QUESTIONMARK
WHATQUERY →	What are OFRELATIONINSTANCE
WHATQUERY →	What are OFRELATION NESTEDPREDICATERELATION
WHATQUERY →	What are the $Type()$ SIMPLEALLRELATION NESTEDPREDICATERELATION
WHATQUERY →	What are the $Type()$ NESTEDPREDICATERELATION
WHATQUERY →	What are the $PositiveInteger()$ most SD $Type()$ NESTEDPREDICATERELATION
WHATQUERY →	What are the cliques of $PositiveInteger()$ $Type()$ CONTAIN NESTEDPREDICATERELATION
CONTAIN →	, that (NEG)? contain the $Type()$ $Instance()$,
OFRELATIONINSTANCE →	$Noun()$ of $Type()$ $Instance()$
OFRELATION →	$Noun()$ of $Type()$
NESTEDPREDICATERELATION →	(that SIMPLERELATION)* that PREDICATERELATION
SIMPLERELATION →	(NEG)? $Verb()$ the $Type()$
SIMPLEALLRELATION →	that $Verb()$ all the $Type()$
PREDICATERELATION →	INSTANCERELATION (CONNECTOR NESTEDPREDICATERELATION)*
INSTANCERELATION →	(NEG)? $Verb()$ the $Type()$ $Instance()$
INSTANCERELATION →	$Verb()$ GENERALISEDQUANTOR $PositiveInteger()$ $Type()$
INSTANCERELATION →	are related to the $Type()$ $Instance()$ via a $Type()$-$Type()$ relation chain of length at most $PositiveInteger()$
SD →	similar | diverse
CONNECTOR →	and | or
GENERALISEDQUANTOR →	at least | at most | exactly
NEG →	$Neg()$
QUESTIONMARK →	?

This type of queries might be important in terms of drug repurposing [21] which has achieved a number of successes in drug development, including the famous example of Pfizer's Viagra [52].

With this motivation, we have incorporated negative queries into BIOQUERY-ASP. For that, first we have extended the grammar of BIOQUERY-CNL as to allow users to construct negative queries, like the query Q11. The extended grammar, called BIOQUERY-CNL*, is presented in Table 3.1, where red colored parts reflect the extensions about negative queries. A detailed description of some special functions to extract knowledge from the given biomedical databases and ontologies, which are denoted in italic in the grammar, is given in Table 3.2. Then we can present negative queries as an ASP program by using negation as failure. For example, we can transform the query Q11 into the following ASP program. The negative queries in BIOQUERY-CNL* (like the other forms of queries) are automatically translated into ASP.

```
what_be_drugs(DRG) :-  cond1(DRG), cond2(DRG).
cond1(DRG) :- drug_disease(DRG,"Depression").
cond2(DRG) :- drug_name(DRG), not drug_gene(DRG,"ACYP1").
answer_exists :- what_be_drugs(DRG).
:- not answer_exists.
```

Since the algorithm for identifying relevant parts is able to cover programs with negation as failure, we do not need to modify query answering part of BIOQUERY-ASP.

Table 3.2: Special functions used in BIOQUERY-CNL*.

$Type()$	Returns a suitable type, ex. gene, disease, drug
$Instance()$	Returns a suitable instance according to the given type, ex. Asthma, Epinephrine
$Verb()$	Returns a suitable verb according to the given types, ex. treat, interact, are related to
$Noun()$	Returns a suitable noun according to the given type, ex. side-effect, symptom
$Neg()$	Returns a suitable negation phrase, ex. do not, are not

3.1.2 Queries about Symptoms of Diseases

The earlier work on BIOQUERY-ASP has considered queries about side effects of drugs, like the query Q2 in Table 1.1. One of the newly added knowledge resources, DISEASEONTOLOGY, provides information about symptoms of diseases. To exploit this information, we have further modified the grammar of BIOQUERY-CNL. More specifically, we have added the word "symptom" to the list of possible nouns that can be returned by the special function $Noun()$ used in BIOQUERY-CNL, as shown in Table 3.2 with the blue part. In this way, the users are able to construct queries about symptoms of diseases, like the query Q12 in Table 1.1.

We then can represent queries about symptoms of diseases in ASP. Similar to the representation of queries about side effects of drugs in ASP. The only difference is to use convenient predicates that correspond to symptoms of diseases, instead of side effects of drugs. For instance, the following program is used to represent the query Q12.

```
what_be_symptoms(X) :- disease_symptom(DIS,X), cond0(DIS).
cond0(DIS) :- drug_disease("Epinephrine",DIS).
answer_exists :- what_be_symptoms(X).
:- not answer_exists.
```

3.1.3 Similarity/Diversity of Drugs

Some queries may have too many answers. In such cases, it might be more desirable to compute a subset of answers which are similar/diverse to each other with respect to some given distance measure. Motivated by that, similarity/diversity queries related to genes are studied in [35] by utilizing Online Method 3 of [29]. Similar to that, in this thesis, we study similarity/diversity queries related to drugs.

First, in order to answer similarity/diversity queries about drugs, one needs to find a way to measure similarities between drugs. Most of the drugs have side effects and the number of common side effects among drugs might be a strong indication of how similar those drugs are [18]. Thus, we have introduced the following distance measure. Assume that c is a large constant, larger than the number of all side effects of drugs in databases and ontologies.

Definition 1 (the side effect distance)**.** *Given two drugs d_1 and d_2, the distance $\Delta_S(d_1, d_2)$ between d_1 and d_2 is defined as*

$$\Delta_S(d_1, d_2) = c - |\{s_1 \mid s_1 \textit{ is a sideeffect of } d_1\} \cap \{s_2 \mid s_2 \textit{ is a sideeffect of } d_2\}| \quad (3.1)$$

Note that Δ_S is always positive due to large value of c. Intuitively, having more common side effects decreases the distance between two drugs, like in Example 1.

Example 1. *Let d_1, d_2 and d_3 be pairwise different drugs such that the number of common side effects is 15 between d_1 and d_2, and 9 between d_1 and d_3. Assume that c is 20. Then, $\Delta_S(d_1, d_2) = 5$ and $\Delta_S(d_1, d_3) = 11$. This shows that more common side effects implies more similar drugs.*

Next, we need to define the corresponding similarity/diversity problem in the context of ASP. In [29], the authors define the following problem to finding n solutions which are k similar with respect to a given distance measure.

Definition 2 (n k-similar (resp., diverse) solutions)**.** *Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure δ that maps a set of solutions for P to a nonnegative integer, and two nonnegative integers n and k, decide whether a set S of n solutions for P exists such that $\delta(S) \leq k$ (resp., $\delta(S) \geq k$).*

Analogous to n k-similar/diverse solutions, we have defined n k-similar/diverse drugs.

Definition 3 (n k-similar (resp., diverse) drugs)**.** *Given an ASP program $\mathcal{P}$ that formulates a drug finding problem P, a distance measure δ that maps a set of drugs for P to a nonnegative integer, and two nonnegative integers n and k, decide whether a set S of n drugs for P exists such that $\delta(S) \leq k$ (resp., $\delta(S) \geq k$).*

To compute similar/diverse drugs, we need to define a distance measure δ for a set D of drugs. We define the distance measure δ for a set D of similar drugs as follows:

$$\delta(D) = \textit{max}\{\Delta_S(d_1, d_2) \mid d_1, d_2 \in D\}$$

Similarly, we define the distance measure δ for a set D of diverse drugs as follows:

$$\delta(D) = \textit{min}\{\Delta_S(d_1, d_2) \mid d_1, d_2 \in D\}$$

Note that the problems n k-similar/diverse solutions are NP-complete [29], for the kind of ASP programs considered in this thesis, which are ASP programs where the head of a rule can be an atom, $\bot$ or a choice expression, and the distance function $\delta(D)$, which is computable in polynomial time with respect to the size of D. Therefore, the problems n k-similar/diverse drugs are inherently intractable.

Finally, we have represented similarity/diversity queries about drugs in ASP. For instance, let us describe how we can solve the query Q13 from Table 1.1. Take the ASP program $\mathcal{P}$ as follows:

```
1{similardrugs(DRG):cond1(DRG)}1.
cond1(DRG) :- drug_gene(DRG,"DLG4").
answer_exists :- similardrugs(DRG).
:- not answer_exists.
```

This program generates a single drug that targets the gene "DLG4". Take the distance function $\delta(D)$ as defined above for similar drugs, i.e., maximum distance over the pairwise drug distances in D. Take $n = 3$ and $k = 100$. Then, to compute an answer to the query Q13, we have used CLASP-NK.[1] CLASP-NK computes an answer to the query Q13 incrementally. For that, we need an admissible heuristic function that estimates the value of $\delta(D)$ from a partially computed answer set. We take the heuristic function as the distance function itself. Then, we can compute an answer to the query Q13 using CLASP-NK with the following command line:

$ gringo Q13.gr | CLASP-NK

where Q13.gr is a file which contains the ASP program $\mathcal{P}$ shown above. With a binary search, one can try to minimize the value of k; however, the underlying online method of CLASP-NK is not complete (due to the choice of the first answer set).

Also, as the grammar of BIOQUERY-CNL allows to construct similarity/diversity queries about drugs, we have integrated those queries into BIOQUERY-ASP.

3.1.4 Finding Close/Distant Drugs to a Given Drug

Another type of query, which might be useful in analyzing relations between drugs, is to finding drugs that are close/distant to a previously computed/known set of drugs. For that, we are interested in answering queries in the following format.

What are the n closest drugs to the drug d ?

where n is a positive integer and d is a drug name. The query Q14 from Table 1.1 is an example of such a query. We model such queries mathematically as follows.

Definition 4 (n closest drugs)**.** *Given a positive integer n, a drug d and a distance measure Δ that maps two drugs to a nonnegative integer,* n closest drugs *to the drug d with respect to Δ, $(n, d, \Delta)_{MCD}$ for short, is a set S of n drugs such that $\sum_{d_s \in S} \Delta(d, d_s)$ is less than or equal to $\sum_{d'_s \in S'} \Delta(d, d_{s'})$ for any set S' of n drugs, where $d \notin S, S'$.*

[1] http://krr.sabanciuniv.edu/projects/SimilarDiverseSolnsInASP

Note that the queries that ask for distant drugs can be modeled similarly. The only difference is to maximize the distance function instead of minimizing.

To compute solutions for the above problem, we have developed two methods, one with a naive approach and the other with a greedy approach. In the former method, we have solely used ASP to find n closest drugs. The latter method is a greedy algorithm that makes use of ASP.

Naive Method In this approach, we have applied generate-and-test methodology of ASP to find n closest drugs to a given drug d. Assume that the predicate `drug_name(DRG)` represents a drug `DRG` and the predicate `drug_side(DRG,SE)` represents a side effect `SE` of the drug `DRG`. These predicates are defined in the rule layer over biomedical knowledge resources.

In the ASP formulation, we first generate n drugs, which are potentially the n closest drugs to the given drug `d`:

```
index(1..n).
1 { close_drugs(N,DRG) : drug_name(DRG) : DRG != d } 1 :- index(N).
```

Here, the predicate `close_drugs(N,DRG)` represents a potential drug `DRG` that could be one of the n closest drugs to `d`. Then, for each generated drug `DRG`, we define common side effects between `DRG` and `d`.

```
common_sideeffect(N,DRG,SE) :- close_drugs(N,DRG),
                               drug_sideeffect(DRG,SE),
                               drug_sideeffect(d,SE).
```

We need to make sure that each generated drug `DRG` is unique.

```
:- close_drugs(N1,DRG), close_drugs(N2,DRG), N1 < N2.
```

Finally, we maximize the number of common side effects to ensure that the generated set of drugs are the closest drugs to the drug `d`.

```
#maximize [ common_sideeffect(N,DRG,SE) ].
```

Greedy Method For some problems, the naive method cannot find n closest drugs to a given drug efficiently, even for small values of n. This is probably due to generation of n drugs at once, in the program. To increase the computational efficiency in time, and to find solutions for bigger values of n, we have designed an ASP-based greedy algorithm (Algorithm 1).

The idea of the algorithm is to generate one drug at a time, instead of generating n drugs at once, and to ensure that the total number of different common side effects is

Algorithm 1: Generating n Closest Drugs

Input: n is the number of drugs, d is a drug name.

Output: N represents a set of n closest drugs to the drug d

```
1 N ← ∅
2 for i ← 1 to n do
3 |   X ← AS(N ∪ Π^{Δ_S}_{grdy}(i, d))
4 |   N ← X
5 return N|_{drugs/2}
```

maximized while generating a new drug. In other words, in Algorithm 1, at the end of the i^{th} iteration, X contains i closest drugs to the given drug d. To compute X, we use an ASP program $\Pi^{\Delta_S}_{\text{grdy}}(i, d)$ shown in Figure 3.1. This encoding is similar to the one used in the naive method. It generates a single drug instead of n drugs:

```
1 { close_drugs(i,DRG) : drug_name(DRG) : DRG != d } 1.
```

Then, the common side effects between the generated drug and the given drug `d` is defined:

```
common_sideeffect(i,DRG,SE) :- close_drugs(i,DRG),
                               drug_sideeffect(DRG,SE),
                               drug_sideeffect(d,SE).
```

Next, we ensure that the generated drug is not the same as the previously generated drugs:

```
:- close_drugs(i,DRG), close_drugs(N,DRG), i != N.
```

To guarantee that the union of the previously computed drugs and the generated drug is $(i, d, \Delta)_{MCD}$, i.e., i closest drugs to the drug d, we maximize the number of common side effects.

```
#maximize [ common_sideeffect(N,DRG,SE) ].
```

Other Possible Methods Recall that computing similar/diverse solutions for a given problem, in the context of ASP, is studied in [29]. By reducing the problem $(n, d, \Delta)_{MCD}$ to some of these problems, we can use computational methods of [29] to solve $(n, d, \Delta)_{MCD}$.

One of the problems studied in [29] is the following:

Definition 5 (n most similar solutions)**.** *Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure δ that maps a set of solutions for P to a nonnegative integer, and a nonnegative integer n, find a set S of n solutions for P with the minimum distance $\delta(S)$.*

We can reduce the problem $(n, d, \Delta)_{MCD}$ to n most similar solutions as follows. Clearly, n comes from $(n, d, \Delta)_{MCD}$. Since there are n drugs in $(n, d, \Delta)_{MCD}$ and we look for n solutions, each solution of the computational problem P must consist of a single drug. Also d cannot be in $(n, d, \Delta)_{MCD}$. Then, P can be defined simply as "find a drug different from d". This problem can be represented by the following ASP program $\mathcal{P}$.

```
1 { drug(DRG) : drug_name(DRG) : DRG != d } 1.
```

In the problem $(n, d, \Delta)_{MCD}$, the goal is to find a set S of n drugs such that $\sum_{s \in S} \Delta(d, s)$ is minimized. Therefore, we define the distance measure δ as follows:

$$\delta(S) = \sum_{s \in S} \Delta(d, s) \tag{3.2}$$

Notice that this function is valid, since a solution for P consists of a single drug. Then, with these parameters (P, δ and n), a solution for the problem n most similar solutions corresponds to a solution for the problem $(n, d, \Delta)_{MCD}$.

Another problem studied in [29], which can be related to the problem $(n, d, \Delta)_{MCD}$, is as follows.

Definition 6 (k-close solution)**.** *Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure δ that maps a set of solutions for P to a nonnegative integer, a set S of solutions for P, and a nonnegative integer k, decide whether some solution s ($s \notin S$) for P exists such that $\delta(S \cup \{s\}) \leq k$.*

Its optimization version is then described as:

Definition 7 (closest solution)**.** *Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure δ that maps a set of solutions for P to a nonnegative integer, and a set S of solutions for P, find a solution s ($s \notin S$) for P with the minimum $\delta(S \cup \{s\})$.*

```
1 { close_drugs(i,DRG) : drug_name(DRG) :DRG != d } 1.

common_sideeffect(i,DRG,SE) :- close_drugs(i,DRG),
                               drug_sideeffect(DRG,SE),
                               drug_sideffect(d,SE).

:- close_drugs(i,DRG), close_drugs(N,DRG), i != N.

#maximize [ common_sideeffect(N,DRG,SE) ].
```

Figure 3.1: ASP program $\Pi^{\Delta_S}_{\text{grdy}}(i, d)$.

By carefully defining the inputs of the problem closest solution, one can also show that the problem $(n, d, \Delta)_{MCD}$ can be reduced to the problem closest solution.

Let n, d and Δ be the parameters of the problem $(n, d, \Delta)_{MCD}$. Then, the problem P is defined as "find a set of n drugs, not including d". This problem can be represented by the following ASP program $\mathcal{P}$.

```
index(1..n).
1 { drugs(N,DRG) : drug_name(DRG) : DRG != d } 1 :- index(N).
```

Given a set S of solutions for P, the distance measure δ is defined as

$$\delta(S) = min\{\sum_{d' \in s} \Delta(d, d') \mid s \in S\}. \tag{3.3}$$

Finally, the parameter $S = \emptyset$. Then, a solution s for P with minimum $\delta(S \cup \{s\})$ corresponds to a solution for the problem $(n, d, \Delta)_{MCD}$.

3.2 New Biomedical Knowledge Resources

The knowledge base of BIOQUERY-ASP is built over large biomedical knowledge resources about drugs, genes and diseases, such as PHARMGKB, DRUGBANK, BIOGRID, CTD and SIDER. After updating the biomedical information gathered from these resources, we have also expanded the knowledge base of BIOQUERY-ASP by including knowledge extracted from DISEASEONTOLOGY and ORPHADATA. In Table 3.3, we provide the relations extracted from each knowledge resource together with the number of corresponding ASP facts.

These knowledge resources are in different formats. For instance, DISEASEONTOLOGY keeps the knowledge in OBO format, whereas ORPHADATA keeps the knowledge in XML format. The other knowledge resources use their own formats, which are generally in text formats where related fields are separated by a delimiter such as the tab character. In order to transform these knowledge resources to ASP, we have developed basic parsers for every format accordingly.

3.3 New Experiments

To show the usefulness of our methods with the new extensions, we conducted experiments on the queries listed in Table 1.1 over large biomedical knowledge resources about genes, drugs and diseases such as PHARMGKB, DRUGBANK, SIDER, BIOGRID, CTD, DISEASEONTOLOGY and ORPHADATA. Experiments were run on a workstation with two 1.60GHz Intel Xeon E5310 Quad-core Processor and 16GB RAM.

Table 3.3: Knowledge resources and their relations.

Source	Relation	# of ASP facts
BIOGRID	*gene-gene*	372.293
CTD	*disease-gene*	8.909.071
	drug-disease	704.590
	drug-gene	259.048
DISEASEONTOLOGY	*disease-symptom*	1.752
DRUGBANK	*drug-category*	4.743
	drug-drug	21.756
ORPHADATA	*disease-gene*	1.452
PHARMGKB	*disease-gene*	9.417
	drug-disease	3.740
	drug-gene	15.805
SIDER	*drug-sideeffect*	61.102
		Total: $\approx$ 10.3 M

First, we considered the queries that can be generated by the grammar of BIOQUERY-CNL*. Those queries are the queries Q1–Q13. Among those queries, for the ones that are not concerned about similarity/diversity of genes/drugs, we used two ASP solvers, CLASP version 2.0.3 (together with the grounder GRINGO version 3.0.3) and DLV version 21.12.2011.[2] For similarity/diversity queries, we used the ASP solver CLASP-NK version 2, a variant of CLASP.

Table 3.4 presents the results of the experiments when CLASP was used. In this table, the first column consists of the queries we used in the experiments. In the second column you can find the computation times (in seconds) and program sizes of the corresponding queries in case of using the complete rule layer (i.e., considering all possible relations extracted from the databases/ontologies).[3] To make the computation more efficient, we applied the method described in [35] for answering queries with respect to the relevant parts of knowledge resources and the rule layer. Essentially, we identified the relevant predicates that the query-predicates depend on (using a dependency graph), and considered the rules that contain these relevant predicates. The results obtained after using that method (i.e., considering only relevant relations with respect to given query) can be found in the third column. The smallest computation time is shown in bold-face. For instance, for the query Q2, CLASP takes 249 seconds to find an answer with the complete program which contains 21 million rules, whereas it takes 12.5 seconds to find an answer with relevant part of the program which contains 2 million rules. As seen from the other results, it is advantageous to identify the relevant part of the program while answering queries.

[2] DLV has its own grounder.

[3] While using CLASP, we take the program size as the number of rules in the ground instantiation of the ASP program. In case of DLV, structural size is used for the program size.

Table 3.4: Experimental results (using CLASP).

Query	Complete	Relevant
Q1	259.65s Rules : 21.070.086	**11.69s** Rules : 1.964.429
Q2	249.02s Rules : 21.070.672	**12.50s** Rules : 2.087.219
Q3	261.00s Rules : 21.067.622	**9.37s** Rules : 1.567.652
Q4	**250.51s** Rules : 21.090.279	303.64s Rules : 19.476.119
Q5	244.91s Rules : 21.074.836	**8.26s** Rules : 1.465.817
Q6	**253.27s** Rules : 21.119.996	271.23s Rules : 19.515.322
Q7	246.60s Rules : 21.067.721	**6.33s** Rules : 1.020.378
Q8	259.17s Rules : 21.107.280	**6.92s** Rules : 1.060.288
Q9	271.75s Rules : 21.059.597	**3.35s** Rules : 547.545
Q10	246.89s Rules : 21.102.612	**10.39s** Rules : 1.612.128
Q11	255.81s Rules : 21.078.277	**12.90s** Rules : 2.158.684
Q12	255.91s Rules : 21.067.704	**83.14s** Rules : 10.338.474
Q13	258.89s Rules : 21.059.455	**3.36s** Rules : 547.332

Table 3.5 reports the results of the experiments when DLV was used. In this table, the first column is the same as the first column of Table 3.4. In the second column, results for the complete rule layer can be found. An advanced query optimization technique for logic programs (known as dynamic magic-sets [1]) is embedded into DLV. To see whether this technique can improve computation times for our queries, we applied it on the complete rule layer. Results when magic sets method applied on the complete rule layer can be seen in the third column. Also, we applied the method of [35] for answering queries with respect to the relevant parts of knowledge resources and the rule layer. Results are depicted in the fourth column. Finally, with the goal of obtaining more efficient computation times, we applied both optimization methods: we first extracted the relevant knowledge with respect to the given query, and then we applied the magic sets method on the relevant part. The results are shown in the fifth column. The magic set method is not applicable for some ASP programs (the ones containing aggregates, constraints etc.). For queries that are represented by such programs, we cannot apply the magic set method.

Table 3.5: Experimental Results (using DLV).

Query	Complete	Complete (Magic)	Relevant	Relevant (Magic)
Q1	221.93s Size: 10.364.769	91.47s Size: 10.364.769	15.00s Size: 983.183	**9.60s** Size: 983.183
Q2	221.76s Size: 10.364.769	91.94s Size: 10.364.769	16.27s Size: 1.044.285	**10.82s** Size: 1.044.285
Q3	224.97s Size: 10.364.769	91.81s Size: 10.364.769	13.74s Size: 647.146	**6.29s** Size: 647.146
Q4	250.79s Size: 10.364.769	NA	**231.48s** Size: 9.567.086	NA
Q5	221.86s Size: 10.364.769	91.65s Size: 10.364.769	11.00s Size: 730.086	**7.34s** Size: 730.086
Q6	217.79s Size: 10.364.769	NA	**83.52s** Size: 9.571.829	NA
Q7	233.85s Size: 10.370.353	NA	**13.95s** Size: 652.730	NA
Q8	238.89s Size: 10.364.771	NA	**9.49s** Size: 372.295	NA
Q10	238.14s Size: 10.364.771	NA	**14.57s** Size: 651.891	NA
Q11	226.96s Size: 10.364.769	98.50s Size: 10.364.769	**16.33s** Size: 1.070.784	16.72s Size: 1.070.784
Q12	226.00s Size: 10.364.769	97.33s Size: 10.364.769	182.25s Size: 9.630.022	**80.09s** Size: 9.630.022

Those ones are shown by "NA", an abbreviation for "Not Applicable". The smallest computation time is shown in bold-face. For instance, for the query Q5, DLV takes 221.8 seconds to find an answer with the complete program which contains 10.3 million rules, whereas it takes 7.3 seconds to find an answer with the magic set method applied on the relevant part of the program which contains 730 thousand rules. According to the results, the most advantageous way of answering queries is to apply magic set method on the identified relevant part of the program, if applicable.

Among the queries listed in Table 1.1, the only query that cannot be represented in the grammar of BIOQUERY-CNL* is the query Q14. Notice that it is a query about close drugs to the drug "Epinephrine". Thus, we represent the query as an instance of $(n, d, \Delta)_{MCD}$ and apply the two methods described in Section 3.1.4.

In our experiments, we used the distance measure Δ_S to compute the distance between two drugs. For the ASP solver, we chose CLASP version $2.0.3$. Table 3.6 presents the results of the experiments. The first column shows the number of closest drugs. The second (resp., third) column denotes computation times for the naive method (resp., the greedy method) together with the distances of the drugs in the solution. Observe that the

Table 3.6: Experimental results for closeness/distantness queries.

n	**Naive**	**Greedy**
1	11.992s $\Delta_S = 18$	11.550s $\Delta_S = 18$
2	900.000s $\Delta_S = 35$	23.088s $\Delta_S = 35$
3	900.000s $\Delta_S = 50$	34.588s $\Delta_S = 52$
4	900.000s $\Delta_S = 55$	46.049s $\Delta_S = 69$
5	900.000s $\Delta_S = 64$	57.912s $\Delta_S = 86$
10	900.000s $\Delta_S = 113$	115.307s $\Delta_S = 167$

distances are different when n is not equal to 1 and 2. Due to the maximization statement in the encoding of the naive method, CLASP finds answer sets incrementally towards the optimized answer sets. Then, although it finds some answer sets, it may not generate the optimized ones within a reasonable amount of time. Therefore, we put a 10 minutes time limit for each run of CLASP. In the table, the distances denote the best solutions computed within 10 minutes. In fact, in the greedy method, CLASP is able to find optimized answer sets for every n. That is, for every n, the results for the greedy method are indeed the solution of $(n, Epinephrine, \Delta_S)_{MCD}$. However, in the naive method, CLASP could not guarantee the optimization of the results for n is equal to $2, 3, 4, 5$ and 10. Although the result for $n = 2$ is optimized, CLASP continues its search as it could not succeed in guaranteeing the optimality.

CHAPTER 4

EXPLANATION GENERATION IN ASP

Once an answer is found for a complex biomedical query, the experts may need informative explanations about the answer. For instance, consider the following query.

What are the drugs that treat the disease Asthma and that targets the gene ADRB1?

An answer for this query is "Xenobiotics". At this point, it might be useful to explain why "Xenobiotics" is an answer for the query. The following explanation could help experts investigate knowledge resources which justify the answer.

The drug Xenobiotics treats the disease Asthma according to PHARMGKB.
The drug Xenobiotics targets the gene ADRB1 according to PHARMGKB.

With this motivation, we study generating explanations for complex biomedical queries. Since the queries, knowledge extracted from databases and ontologies, and the rule layer are in ASP, our studies focus on explanation generation within the context of ASP.

In the following, we first introduce definitions regarding explanations in ASP. Next, we provide a method to generate shortest explanations with respect to given queries. Then, we present another method which allows to generate k different explanations for a given query. After that, we discuss how to present explanations to the users in a natural language. Finally, we discuss the implementation of these algorithms, and its integration in BIOQUERY-ASP.

4.1 Explanations in ASP

Let Π be the relevant part of a ground ASP program with respect to a given biomedical query Q (also a ground ASP program), that contains rules describing the knowledge extracted from biomedical ontologies and databases, the knowledge integrating them, and

the background knowledge. Rules in $\Pi \cup Q$ generally do not contain cardinality/choice expressions in the head; therefore, we assume that in $\Pi \cup Q$ only bodies of rules contain cardinality expressions. Let X be an answer set for $\Pi \cup Q$. Let p be an atom that characterizes an answer to the query Q. The goal is to find an "explanation" as to why p is computed as an answer to the query Q, i.e., why is p in X? Before we introduce a definition of an explanation, we need the following notations and definitions.

We say that a set X of atoms *satisfies* a cardinality expression C of the form

$$l \leq \{A_1, \ldots, A_m\} \leq u$$

if the cardinality of $X \cap \{A_1, \ldots, A_m\}$ is within the lower bound l and upper bound u. Also X *satisfies* a set SC of cardinality expressions (denoted by $X \models SC$), if X satisfies every element of SC.

Let Π be a ground ASP program, r be a rule in Π, p be an atom in Π, and Y and Z be two sets of atoms. Let $B_{card}(r)$ denote the set of cardinality expressions that appear in the body of r. We say that r *supports* an atom p using atoms in Y but not in Z (or with respect to Y but Z), if the following hold:

$$\begin{array}{l} H(r) = p, \\ B^+(r) \subseteq Y \backslash Z, \\ B^-(r) \cap Y = \emptyset, \\ Y \models B_{card}(r) \end{array} \tag{4.1}$$

We denote the set of rules in Π that support p with respect to Y but Z, by $\Pi_{Y,Z}(p)$.

We now introduce definitions about explanations in ASP. We first define a generic tree whose vertices are labeled by either atoms or rules.

Definition 8 (Vertex-labeled tree)**.** *A* vertex-labeled tree $\langle V, E, l, \Pi, X \rangle$ *for a program* Π *and a set* X *of atoms is a tree* $\langle V, E \rangle$ *whose vertices are labeled by a function* l *that maps* V *to* $\Pi \cup X$*. In this tree, the vertices labeled by an atom (resp., a rule) are called* atom vertices *(resp.,* rule vertices*).*

For a vertex-labeled tree $T = \langle V, E, l, \Pi, X \rangle$ and a vertex v in V, we introduce the following notations:

- $anc_T(v)$ denotes the set of atoms which are labels of ancestors of v.
- $des_T(v)$ denotes the set of rule vertices which are descendants of v.
- $child_E(v)$ denotes the set of children of v.
- $sibling_E(v)$ denotes the set of siblings of v.
- $out_E(v)$ denotes the set of out-going edges of v.

- $deg_E(v)$ denotes the degree of v and equals to $|out_E(v)|$.
- If $deg_E(v) = 0$, then v is a *leaf* vertex.
- $leaf(T)$ denotes the set of leaves of T.
- The *root* of T is the root of $\langle V, E \rangle$.
- T is *empty* if $\langle V, E \rangle = \langle \emptyset, \emptyset \rangle$.

We now define a specific class of vertex-labeled trees which contains all possible "explanations" for an atom.

Definition 9 (And-or explanation tree)**.** *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X. The* and-or explanation tree *for p with respect to Π and X is a vertex-labeled tree $T = \langle V, E, l, \Pi, X \rangle$ that satisfies the following:*

(i) *for the root $v \in V$ of the tree, $l(v) = p$;*

(ii) *for every atom vertex $v \in V$,*

$$out_E(v) = \{(v, v') \mid (v, v') \in E,\ l(v') \in \Pi_{X, anc_T(v')}(l(v))\};$$

(iii) *for every rule vertex $v \in V$,*

$$out_E(v) = \{(v, v') \mid (v, v') \in E,\ l(v') \in B^+(l(v))\};$$

(iv) *each leaf vertex is a rule vertex.*

Let us explain conditions $(i) - (iv)$ in Definition 9 in detail.

(i) The root of the and-or explanation tree T is labeled by the atom p. Intuitively, T contains all possible explanations for p.

(ii) For every atom vertex $v \in V$, there is an out-going edge (v, v') to a rule vertex $v' \in V$ under the following conditions: the rule that labels v' supports the atom that labels v, using atoms in X but not any atom that labels an ancestor of v'. We want to exclude the atoms labeling ancestors of v' to ensure that the height of the and-or explanation tree is finite (e.g., otherwise, due to cyclic dependencies the tree may be infinite).

(iii) For every rule vertex $v \in V$, there is an out-going edge (v, v') to an atom vertex if the atom that labels v' is in the positive body of the rule that labels v. In this way, we make sure that every atom in the positive body of the rule that labels v takes part in explaining the head of the rule that labels v.

(iv) Together with Conditions (ii) and (iii) above, this condition guarantees that the leaves of the and-or explanation tree are rule vertices that are labeled by facts in the reduct of the given ASP program Π with respect to the given answer set X. Intuitively, this condition expresses that the leaves are self-explanatory.

Example 2. *Let* Π *be the program*

$$a \leftarrow b, c$$
$$a \leftarrow d$$
$$d \leftarrow$$
$$b \leftarrow c$$
$$c \leftarrow$$

and $X = \{a, b, c, d\}$. *The and-or explanation tree for* a *with respect to* Π *and* X *is shown in Figure 4.1. Intuitively, the and-or explanation tree includes all possible "explanations" for an atom. For instance, according to Figure 4.1, the atom* a *has two explanations:*

- *One explanation is characterized by the rules that label the vertices in the left subtree of the root:* a *is in* X *because the rule*

  $$a \leftarrow b, c$$

  support a. *Moreover, this rule can be "applied to generate* a*" because* b *and* c*, the atoms in its positive body, are in* X. *Further,* b *is in* X *because the rule*

  $$b \leftarrow c$$

  supports b. *Further,* c *is in* X *because* c *is supported by the rule*

  $$c \leftarrow$$

  which is self-explanatory.

- *The other explanation is characterized by the rules that label the vertices in the right subtree of the root:* a *is in* X *because the rule*

  $$a \leftarrow d$$

  supports a. *Further, this rule can be "applied to generate* a*" because* d *is in* X. *In addition,* d *is in* X *because* d *is supported by the rule*

  $$d \leftarrow$$

  which is self-explanatory.

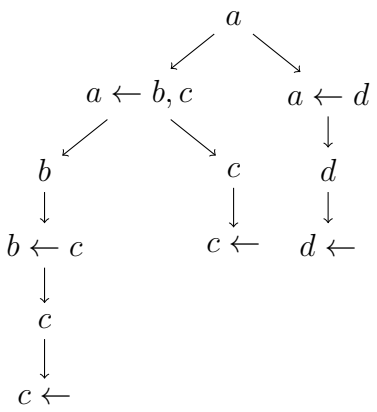


Figure 4.1: The and-or explanation tree for Example 2.

Proposition 1. *Let Π be a ground ASP program and X be an answer set for Π. For every p in X, the and-or explanation tree for p with respect to Π and X is not empty.*

Note that in the and-or explanation tree, atom vertices are the "or" vertices, and rule vertices are the "and" vertices. Then, we can obtain a subtree of the and-or explanation tree that contains an explanation, by visiting only one child of every atom vertex and every child of every rule vertex, starting from the root of the and-or explanation tree. Here is precise definition of such a subtree, called an explanation tree.

Definition 10 (Explanation tree)**.** *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and $T = \langle V, E, l, \Pi, X\rangle$ be the and-or explanation tree for p with respect to Π and X. An* explanation tree *in T is a vertex-labeled tree $T' = \langle V', E', l, \Pi, X\rangle$ such that*

(i) *$\langle V', E'\rangle$ is a subtree of $\langle V, E\rangle$;*

(ii) *the root of $\langle V', E'\rangle$ is the root of $\langle V, E\rangle$;*

(iii) *for every atom vertex $v' \in V'$, $deg_{E'}(v') = 1$;*

(iv) *for every rule vertex $v' \in V'$, $out_E(v') \subseteq E'$.*

Example 3. *Let T be the and-or explanation tree in Figure 4.1. Then, Figure 4.2 illustrates the explanation trees in T. These explanation trees characterize the two explanations for a explained in Example 2.*

After having defined the and-or explanation tree and an explanation tree for an atom, let us now define an explanation for an atom.

Definition 11 (Explanation)**.** *Let Π be a ground ASP program, X be an answer set for Π, and p be an atom in X. A vertex-labeled tree $\langle V', E', l, \Pi, X\rangle$ is an* explanation *for p*

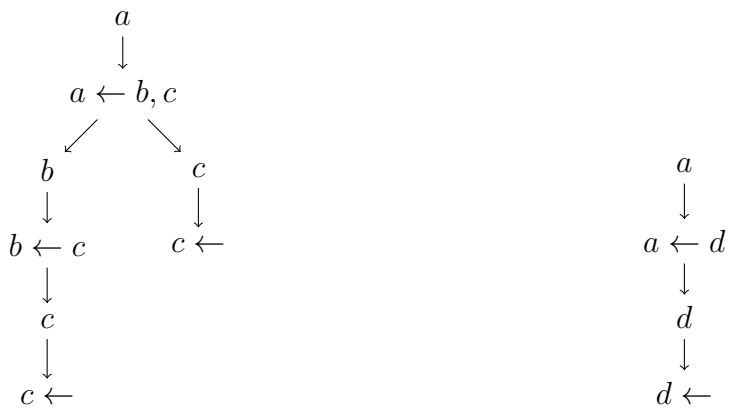


Figure 4.2: Explanation trees for Example 3.

with respect to Π *and* X *if there exists an explanation tree* $\langle V, E, l, \Pi, X\rangle$ *in the and-or explanation tree for* p *with respect to* Π *and* X *such that*

(i) $V' = \{v \mid v$ *is a rule vertex in* $V\}$*;*

(ii) $E' = \{(v_1, v_2) \mid (v_1, v), (v, v_2) \in E,$ *for some atom vertex* $v \in V\}$.

Intuitively, an explanation can be obtained from an explanation tree by "ignoring" its atom vertices.

Example 4. *Let* Π *and* X *be defined as in Example 2. Then, Figure 4.3 depicts two explanations for* a *with respect to* Π *and* X*, described in Example 2.*

Figure 4.3: Explanations for Example 4.

So far, we have considered only positive programs in the examples. Our definitions can also be used in programs that contain negation and aggregates in the bodies of rules.

Example 5. *Let* Π *be the program*

$$
\begin{aligned}
&a \leftarrow b, c, not\ e\\
&a \leftarrow d, not\ b\\
&a \leftarrow d, 1 \leq \{b, c\} \leq 2\\
&d \leftarrow\\
&b \leftarrow c\\
&c \leftarrow
\end{aligned}
$$

and $X = \{a, b, c, d\}$. The and-or explanation tree for a with respect to Π and X is shown in Figure 4.4(a). Here, the rule $a \leftarrow d, not\ b$ is not included in the tree as b is in X, whereas the rule $a \leftarrow b, c, not\ e$ is in the tree as e is not in X and, b and c are in X. Also, the rule $a \leftarrow d, 1 \leq \{b, c\} \leq 2$ is in the tree as d is in X and the cardinality expression $1 \leq \{b, c\} \leq 2$ is satisfied by X. An explanation for a with respect to Π and X is shown in Figure 4.4(b).

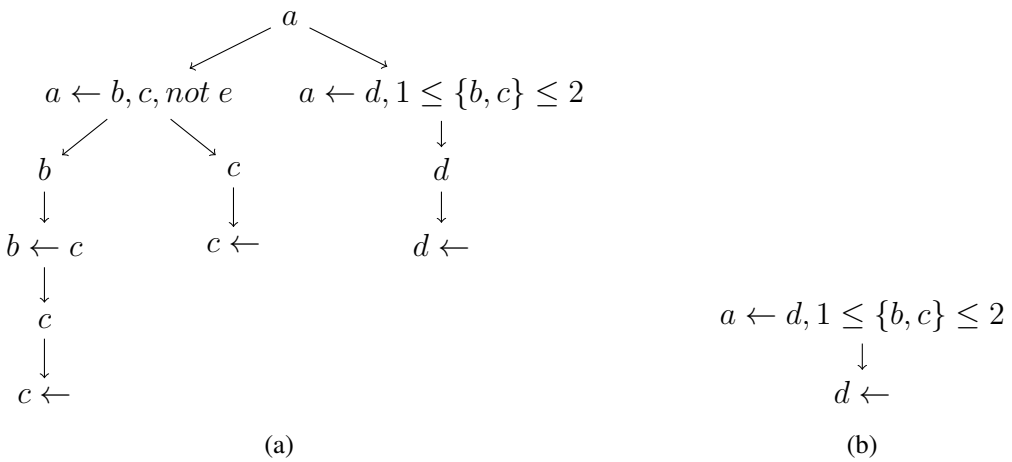


Figure 4.4: (a) The and-or explanation tree for a and (b) an explanation for a.

4.2 Generating Shortest Explanations

As can be seen in Figure 4.3, there might be more than one explanation for a given atom. Hence, it is not surprising that one may prefer some explanations to others. Consider biomedical queries about chains of gene-gene interactions like the query Q8 in Table 1.1. Answers of such queries may contain chains of gene-gene interactions with different lengths. For instance, an answer for this query is "CASK". Figure 4.5 shows an explanation for this answer. Here, "CASK" is related to "ADRB1" via a gene-gene chain interaction of length 2 (the chain "CASK"–"DLG4"–"ADRB1"). Another explanation is shown in Figure 4.6. Now, "CASK" is related to "ADRB1" via a gene-gene chain interaction of length 3 (the chain "CASK"–" DLG1"–"DLG4"–"ADRB1"). Since gene-gene interactions are important for drug discovery, it may be more desirable for the experts to reason about chains with shortest lengths. Motivated by that, we consider generating shortest explanations. Intuitively, an explanation S is shorter than another explanation S' if the number of rule vertices involved in S is less than the number of rule vertices involved in S'. Then we can define shortest explanations as follows.

Definition 12 (Shortest explanation)**.** *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and S be an explanation (with vertices V) for p with respect*

```
what_be_genes("CASK") :- gene_reachable_from("CASK",2).

   gene_reachable_from("CASK",2) :- gene_gene("CASK","DLG4"), gene_reachable_from("DLG4",1)...

      gene_gene("CASK","DLG4") :- gene_gene_biogrid("CASK","DLG4").

          gene_gene_biogrid("CASK","DLG4").

      gene_reachable_from("DLG4",1) :- gene_gene("DLG4","ADRB1"), start_gene("ADRB1").

          gene_gene("DLG4","ADRB1") :- gene_gene_biogrid("DLG4","ADRB1").

              gene_gene_biogrid("DLG4","ADRB1").

          start_gene("ADRB1").
```

Figure 4.5: An explanation for Q8.

to Π *and* X*. Then,* S *is a* shortest explanation *for* p *with respect to* Π *and* X *if there exists no explanation* S' *(with vertices* V'*) for* p *with respect to* Π *and* X *such that* $|V'| < |V|$.

Example 6. *Let* Π *and* X *be defined as in Example 2. Then, Figure 4.3(b) is the shortest explanation for* a *with respect to* Π *and* X.

To compute shortest explanations, we define a weight function that assigns weights to the vertices of the and-or explanation tree. Basically, the weight of an atom vertex ("or" vertex) is equal to the minimum weight among weights of its children and the weight of a rule vertex ("and" vertex) is equal to sum of weights of its children plus 1. Then the idea is to extract a shortest explanation by propagating up the weights of the leaves and then traversing the vertices that contribute to the weight of the root. Let us define the weight of vertices in the and-or explanation tree.

Definition 13 (Weight function)**.** *Let* Π *be a ground ASP program,* X *be an answer set for* Π*,* p *be an atom in* X*, and* $T = \langle V, E, l, \Pi, X\rangle$ *be the and-or explanation tree for* p *with respect to* Π *and* X*. The weight function* W_T *for* T *maps vertices in* V *to a positive integer and it is defined as follows.*

$$W_T(v) = \begin{cases} \mathit{min}\{W_T(c) \mid c \in \mathit{child}_E(v)\} & \text{if } v \text{ is an atom vertex in } V; \\ 1 + \sum_{c \in \mathit{child}_E(v)} W_T(c) & \text{otherwise.} \end{cases}$$

We develop Algorithm 2 to generate shortest explanations. Let us describe this algorithm. Algorithm 2 starts by creating the and-or explanation tree T for p with respect to Π and X (Line 1); for that it uses Algorithm 3. If T is not empty, then Algorithm 2 assigns weights to the vertices of T (Line 4), using Algorithm 4. As the final step, Algorithm 2 extracts a shortest explanation from T (Line 5), using Algorithm 5. The idea is to traverse

```
what_be_genes("CASK") :- gene_reachable_from("CASK",3).

   gene_reachable_from("CASK",3) :- gene_gene("CASK","DLG1"), gene_reachable_from("DLG1",2)...

      gene_gene("CASK","DLG1") :- gene_gene("DLG1","CASK").

          gene_gene("DLG1","CASK") :- gene_gene_biogrid("DLG1","CASK").

              gene_gene_biogrid("DLG1","CASK").

      gene_reachable_from("DLG1",2) :- gene_gene("DLG1","DLG4"), gene_reachable_from("DLG4",1)...

          gene_gene("DLG1","DLG4") :- gene_gene("DLG4","DLG1").

              gene_gene("DLG4","DLG1") :- gene_gene_biogrid("DLG4","DLG1").

                  gene_gene_biogrid("DLG4","DLG1").

          gene_reachable_from("DLG4",1) :- gene_gene("DLG4","ADRB1"),start_gene("ADRB1").

              gene_gene("DLG4","ADRB1") :- gene_gene_biogrid("DLG4","ADRB1").

                  gene_gene_biogrid("DLG4","ADRB1").

              start_gene("ADRB1").
```

Figure 4.6: Another explanation for Q8.

an explanation tree of T, by the help of the weight function, and construct an explanation, which would be a shortest one, by contemplating only the rule vertices in the traversed explanation tree. If T is empty, Algorithm 2 returns an empty vertex-labeled tree.

The execution of Algorithm 2 is also illustrated in Figure 4.7. First, the and-or explanation tree is generated, which has a generic structure as in Figure 4.7(a). Here, yellow vertices denote atom vertices and blue vertices denote rule vertices. Then, this tree is weighted as in Figure 4.7(b). Then, starting from the root, a subtree of the and-or explanation tree is traversed by visiting minimum weighted child of every atom vertex and every child of every rule vertex. This process is shown in Figure 4.7(c), where red vertices form the traversed subtree. From this subtree, an explanation is extracted by ignoring atom vertices and keeping the parent-child relationship of the tree as it is. The resulting explanation is depicted in Figure 4.7(d).

Proposition 2. *Given a ground ASP program Π, an answer set X for Π, and an atom p in X, Algorithm 2 terminates.*

Proposition 3. *Given a ground ASP program Π, an answer set X for Π, and an atom p in X, Algorithm 2 either finds a shortest explanation for p with respect to Π and X or returns an empty vertex-labeled tree.*

Algorithm 2: Generating Shortest Explanations

Input: Π : ground ASP program, X : answer set for Π, p : atom in X.

Output: a shortest explanation for p w.r.t Π and X, or an empty vertex-labeled tree.

```
⟨V, E, l, Π, X⟩ := createTree(Π, X, p, {})
if ⟨V, E⟩ is not empty then
    v ← root of ⟨V, E⟩
    calculateWeight(Π, X, V, l, v, E, W_T)
    ⟨V', E', l, Π, X⟩ := extractExp(Π, X, V, l, v, E, W_T, ∅, min)
    return ⟨V', E', l, Π, X⟩
else
    return ⟨∅, ∅, l, Π, X⟩
```

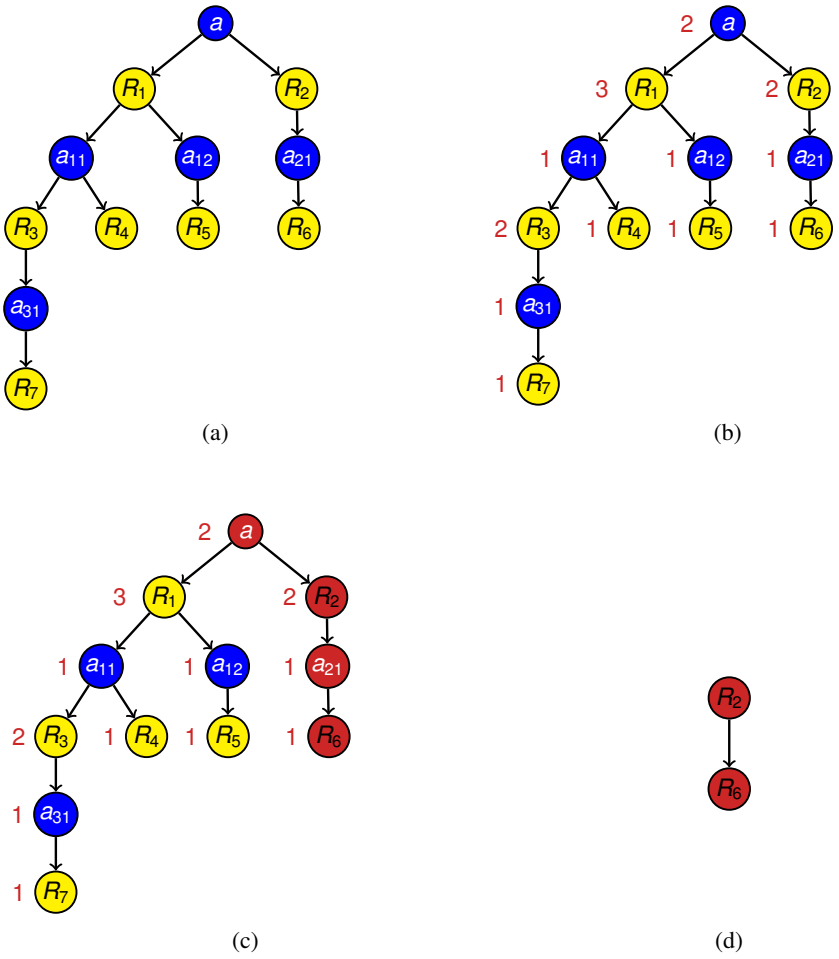


Figure 4.7: A generic execution of Algorithm 2.

Proposition 4. *Given a ground ASP program* Π*, an answer set* X *for* Π*, and an atom* p *in* X*, the time complexity of Algorithm 2 is* $O(|\Pi|^{|X|} \times |\mathcal{B}_\Pi|)$.

Algorithm 3: createTree

Input: Π : ground ASP program, X : answer set for Π, d : an atom in X or a rule in Π, L : set of atoms in X.

Output: A vertex-labeled tree.

```
1  V := ∅, E := ∅
2  if d ∈ X\L then
3      v ← Create an atom vertex s.t. l(v) = d
4      L := L ∪ {d}, V := V ∪ {v}
5      foreach r ∈ Π_{X,L}(d) do
6          ⟨V', E', l, Π, X⟩ := createTree(Π, X, r, L)
7          if ⟨V', E'⟩ ≠ ⟨∅, ∅⟩ then
8              v' ← root of ⟨V', E'⟩ s.t. l(v') = r
9              V := V ∪ V', E := E ∪ {(v, v')} ∪ E'
10     if E = ∅ then return ⟨∅, ∅, l, Π, X⟩
11 else if d ∈ Π then
12     v ← Create a rule vertex s.t. l(v) = d
13     foreach a ∈ B+(d) do
14         ⟨V', E', l, Π, X⟩ := createTree(Π, X, a, L)
15         if ⟨V', E'⟩ = ⟨∅, ∅⟩ then return ⟨∅, ∅, l, Π, X⟩
16         v' ← root of ⟨V', E'⟩ s.t. l(v') = a
17         V := V ∪ V', E := E ∪ {(v, v')} ∪ E'
18 return ⟨V, E, l, Π, X⟩
```

We generate the complete and-or explanation tree while finding a shortest explanation. In fact, we can find a shortest explanation by creating a partial and-or explanation tree using a branch-and-bound idea. In particular, the idea is to compute the weights of vertices during the creation of the and-or explanation tree and, in case there exists a branch of the and-or explanation tree that exceeds the weight of a vertex computed so far, to stop branching on unnecessary parts of the and-or explanation tree. Then, a shortest explanation can be extracted by the same method used previously, i.e., by traversing a subtree of the and-or explanation tree and ignoring the atom vertices in this subtree. For instance, consider Figure 4.7(b). Assume that we first create the right branch of the root. Since the weight of an atom vertex is equal to the minimum weight among its children weights, we know that the weight of the root is at most 2. Now, we check whether it is necessary to branch on the left child of the root. Note that the weight of a rule vertex is equal to 1 plus the sum of its children weights. As R_1 has two children, its weight is at

Algorithm 4: calculateWeight

Input: Π : ground ASP program, X : answer set for Π, V : set of vertices, $l : V \to \Pi \cup X$, v : vertex in V, E : set of edges, W_T : candidate weight function.

Output: Weight of v.

```
if l(v) ∈ X then
    foreach c ∈ child_E(v) do W_T(c) := calculateWeight(Π, X, V, l, c, E, W_T)
    W_T(v) := min{W_T(c) | c ∈ child_E(v)}
else if l(v) ∈ Π then
    W_T(v) := 1
    foreach c ∈ child_E(v) do
    W_T(v) := W_T(v) + calculateWeight(Π, X, V, l, c, E, W_T)
return W_T(v)
```

least 3. Therefore, it is redundant to branch on the left child of the root. This improvement is not implemented and is a future work.

4.3 Generating k Different Explanations

When there is more than one explanation for an answer of a query, it might be useful to provide the experts with several more explanations that are different from each other. For instance, consider the query Q5 in Table 1.1. An answer for this query is "Doxepin". An explanation for this answer is shown in Figure 4.8. According to this explanation, "Doxepin" reacts with "Epinephrine". At this point, the expert may need to learn whether "Doxepin" also treats "Asthma". Another explanation shown in Figure 4.9 confirms that "Doxepin" treats "Asthma". Motivated by this example, we study generating different explanations.

We introduce an algorithm (Algorithm 6) to compute k different explanations for an

```
what_be_drugs("Doxepin") :- drug_drug("Doxepin","Epinephrine").

   drug_drug("Doxepin","Epinephrine") :- drug_drug("Epinephrine","Doxepin").

      drug_drug("Epinephrine","Doxepin") :- drug_drug_drugbank("Epinephrine","Doxepin").

          drug_drug_drugbank("Epinephrine","Doxepin").
```

Figure 4.8: An explanation for Q5.

Algorithm 5: extractExp

Input: Π : ground ASP program, X : answer set for Π, V_t : set of vertices, $l : V_t \to \Pi \cup X$, v : vertex in V_t, E_t : set of edges, W_T : weight function of T, r : rule vertex in V_t or $\emptyset$, *op*: string *min* or *max*.

Output: A vertex-labeled tree $\langle V, E, l, \Pi, X \rangle$.

```
1  V := ∅, E := ∅
2  if l(v) ∈ X then
3      c ← Pick op weighted child of v
4      if r ≠ ∅ then E := E ∪ {(r, c)}
5      ⟨V', E', l, Π, X⟩ := extractExp(Π, X, V_t, l, c, E_t, W_T, r, op)
6      V := V ∪ V', E := E ∪ E'
7  else if l(v) ∈ Π then
8      V := V ∪ {v}
9      foreach c ∈ child_{E_t}(v) do
10         ⟨V', E', l, Π, X⟩ := extractExp(Π, X, V_t, l, c, E_t, W_T, v, op)
11         V := V ∪ V', E := E ∪ E'
12 return ⟨V, E, l, Π, X⟩
```

atom p in X with respect to Π and X. For that, we define a distance measure Δ_D between a set Z of (previously computed) explanations, and an (to be computed) explanation S. We consider the rule vertices R_Z and R_S contained in Z and S, respectively. Then, we define the function Δ_D that measures the distance between Z and S as follows:

$$\Delta_D(Z, S) = |R_S \backslash R_Z|.$$

In the following, we sometimes use R_Z and R_S instead of Z and S in Δ_D. Also, we denote by *RVertices*(S) the set of rule vertices of a vertex-labeled tree S.

Let us now explain Algorithm 6. It computes a set K of k different explanations iteratively. Initially, $K = \emptyset$. First, we compute the and-or explanation tree T (Line 2). Then, we enter into a loop that iterates at most k times (Line 4). At each iteration i, an explanation K_i that is most distant from the previously computed $i - 1$ explanations is

```
what_be_drugs("Doxepin") :- drug_disease("Doxepin","Asthma").

   drug_disease("Doxepin","Asthma") :- drug_disease_ctd("Doxepin","Asthma").

      drug_disease_ctd("Doxepin","Asthma").
```

Figure 4.9: Another explanation for Q5.

Algorithm 6: Generating k Different Explanations

Input: Π: ground ASP program, X: answer set for Π, p: atom in X, k : a positive integer. Assume there are n different explanations for p w.r.t Π and X.

Output: *min*$\{n, k\}$ different explanations for p with respect to Π and X.

1 $K := \emptyset$, $R_0 := \emptyset$
2 $\langle V, E, l, \Pi, X\rangle :=$ createTree$(\Pi, X, p, \{\})$
3 $v \leftarrow$ root of $\langle V, E\rangle$
4 **for** $i = 1, 2, \ldots k$ **do**
5 calculateDifference$(\Pi, X, V, l, v, E, R_{i-1}, W_{T,R_{i-1}})$
6 **if** $W_{T,R_{i-1}}(v) = 0$ **then return** K
7 $\langle V', E', l, \Pi, X\rangle :=$ extractExp$(\Pi, X, V, l, v, E, W_{T,R_{i-1}}, \emptyset, \mathit{max})$
8 $K_i \leftarrow \langle V', E', l, \Pi, X\rangle$
9 $K := K \cup \{K_i\}$
10 $R_i := R_{i-1} \cup \{v \mid \text{rule vertex } v \in V'\}$
11 **return** K

extracted from T. Let us denote the rule vertices included in the previously computed $i-1$ explanations by R_{i-1}. Then, essentially, at each iteration we pick an explanation K_i such that $\Delta_D(R_{i-1}, \mathit{RVertices}(K_i))$ is maximum. To be able to find such a K_i, we need to define the "contribution" of each vertex v in T to the distance measure $\Delta_D(R_{i-1}, \mathit{RVertices}(K_i))$ if v is included in explanation K_i:

$$W_{T,R_{i-1}}(v) = \begin{cases} \mathit{max}\{W_{T,R_{i-1}}(v') \mid v' \in \mathit{child}_E(v)\} & \text{if } v \text{ is an atom vertex;} \\ \sum_{v' \in \mathit{child}_E(v)} W_{T,R_{i-1}}(v') & \text{if } v \text{ is a rule vertex and } v \in R_{i-1}; \\ 1 + \sum_{v' \in \mathit{child}_E(v)} W_{T,R_{i-1}}(v') & \text{otherwise.} \end{cases}$$

Note that this function is different from W_T. Intuitively, v contributes to the distance measure if it is not included in R_{i-1}. The contributions of vertices in T are computed by Algorithm 7 (Line 5) by propagating up the contributions in the spirit of Algorithm 4. Then, K_i is extracted from weighted-T by using Algorithm 5 (Line 7).

The execution of Algorithm 6 is also illustrated in Figure 4.10. Similar to Algorithm 2, which generates shortest explanations, first the and-or explanation tree is created, which has a generic structure as shown in Figure 4.10(a). Recall that yellow vertices denote atom vertices and blue vertices denote rule vertices. For the sake of example, assume that $R = \{R_2, R_6\}$. Then, the goal is to generate an explanation that contains different rule vertices from the rule vertices in R as much as possible. For that, the weights of vertices are assigned according to the weight function $W_{T,R}$ as depicted in Figure 4.10(b). Here, the weight of the root implies that there exists an explanation which contains 4 different rule vertices from the rule vertices in R and this explanation is the most different

Algorithm 7: calculateDifference

Input: Π : ground ASP program, X : answer set for Π, V : set of vertices, $l : V \to \Pi \cup X$, v : vertex in V, E : set of edges, R : set of rule vertices in V, D_R : candidate distance function.

Output: distance of v.

```
if l(v) ∈ X then
    foreach c ∈ child_E(v) do
        D_R(c) := calculateDifference(Π, X, V, l, c, E, R, D_R)
    D_R(v) := max{D_R(c) | c ∈ child_E(v)}
else if l(v) ∈ Π then
    if v ∉ R then D_R(v) := 1
    else D_R(v) := 0
    foreach c ∈ child_E(v) do
        D_R(v) := D_R(v) + calculateDifference(Π, X, V, l, c, E, R, D_R)
return D_R(v)
```

one. Then, starting from the root, a subtree of the and-or explanation tree is traversed by by visiting maximum weighted child of every atom vertex, and every child of every rule vertex. This subtree is shown in Figure 4.10(c) by red vertices. Finally, an explanation is extracted by ignoring the atom vertices and keeping the parent-child relationship as it is, from this subtree. This explanation is illustrated in Figure 4.10(d).

Proposition 5. *Given a ground ASP program Π, an answer set X for Π, an atom p in X, and a positive integer k, Algorithm 6 terminates.*

Proposition 6. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and k be a positive integer. Let n be the number of different explanations for p with respect to Π and X. Then, Algorithm 6 returns min$\{n, k\}$ different explanations for p with respect to Π and X.*

Furthermore, at each iteration i of the loop in Algorithm 6 the distance $\Delta_D(R_{i-1}, K_i)$ is maximized.

Proposition 7. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and k be a positive integer. Let n be the number of explanations for p with respect to Π and X. Then, at the end of each iteration i ($1 \leq i \leq$ min$\{n, k\}$) of the loop in Algorithm 6, $\Delta_D(R_{i-1}, RVertices(K_i))$ is maximized, i.e., there is no other explanation K' such that $\Delta_D(R_{i-1}, RVertices(K_i)) < \Delta_D(R_{i-1}, RVertices(K'))$.*

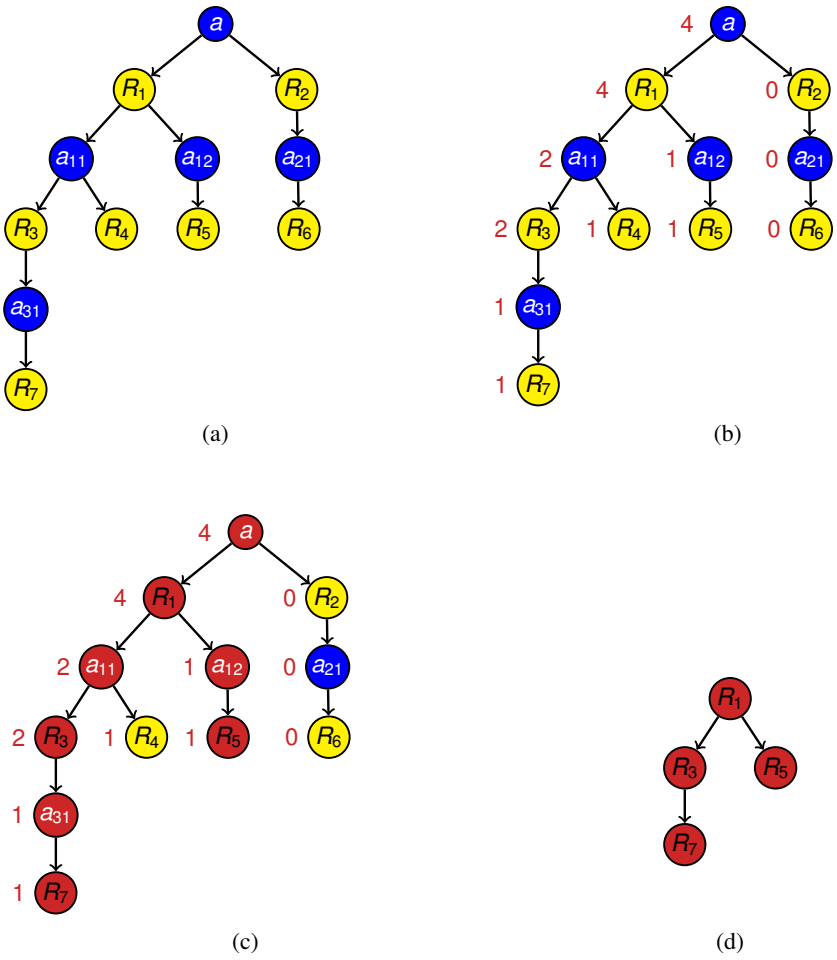


Figure 4.10: A generic execution of Algorithm 6.

This result leads us to some useful consequences. First, Algorithm 6 computes "longest" explanations if $k = 1$. The following corollary shows how to compute longest explanations.

Corollary 1. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and $k = 1$. Then, Algorithm 6 computes a longest explanation for p with respect to Π and X.*

Next, we show that Algorithm 6 computes k different explanations such that for every i ($1 \leq i \leq k$) the i^{th} explanation is the most distant explanation from the previously computed $i - 1$ explanations.

Corollary 2. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and k be a positive integer. Let n be the number of explanations for p with respect to Π and X. Then, Algorithm 6 computes $min\{n, k\}$ different explanations $K_1, \ldots, K_{min\{n,k\}}$ for p with respect to Π and X such that for every j ($2 \leq j \leq min\{n, k\}$) $\Delta_D(\bigcup_{z=1}^{j-1} RVertices(K_z), K_j)$ is maximized.*

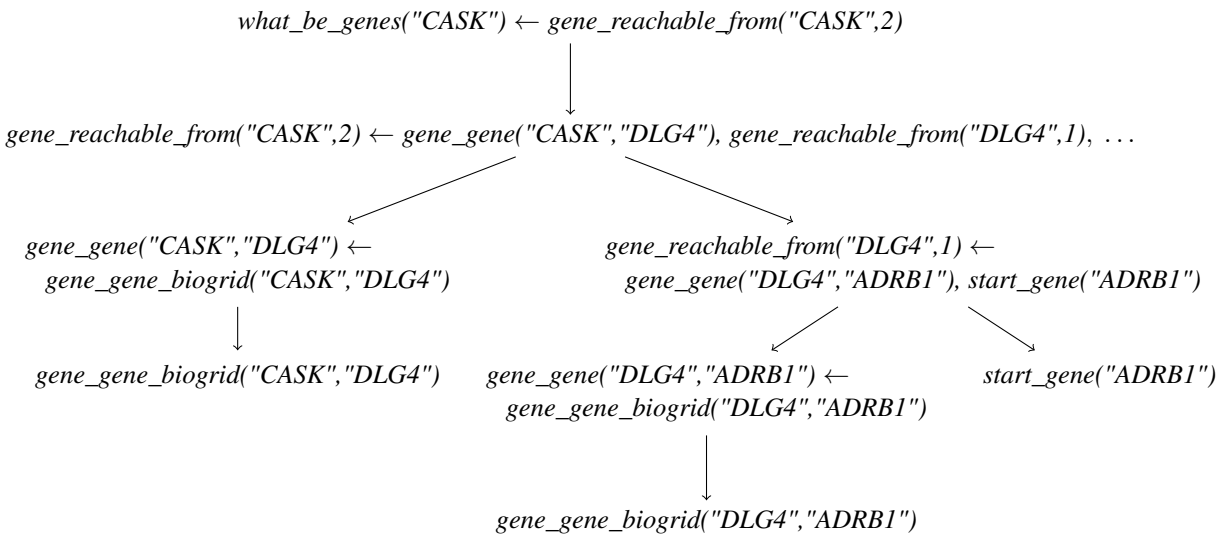


Figure 4.11: A shortest explanation for Q8.

The following proposition shows that the time complexity of Algorithm 6 is exponential in the size of the given answer set.

Proposition 8. *Given a ground ASP program* Π*, an answer set* X *for* Π*, an atom* p *in* X *and a positive integer* k*, the time complexity of Algorithm 6 is* $O(k \times |\Pi|^{|X|+1} \times |\mathcal{B}_\Pi|)$.

4.4 Presenting Explanations in a Natural Language

An explanation for an answer of a biomedical query may not be clear to the experts. For instance, an answer of the query Q8 in Table 1.1 is "CASK". A shortest explanation for this answer is shown in Figure 4.11. As it is seen, the meaning of such an explanation, which states that "CASK" is related to "ADRB1" via a gene-gene chain interaction of length 2 (the chain "CASK"–"DLG4"–"ADRB1"), is not obvious for the experts. To this end, it is better to present explanations to the experts in a natural language.

Observe that leaves of an explanation denote facts. That is, they represent the knowledge base of the program. In case of an explanation for an answer of a biomedical query, we know that they correspond to some predicates used for representing the knowledge that are obtained from biomedical knowledge resources. Similar to that, some internal vertices contain informative explanations such as the position of a drug in a chain of drug-drug interactions. Thus, we make use of a predicate look-up table, illustrated in Table 4.1, to express explanations in a natural language. Basically, we perform a depth-first traversal in an explanation and look for predicates that have a corresponding expression in the look-up table. For instance, the explanation in Figure 4.11 is expressed in natural language as follows.

The gene CASK interacts with the gene DLG4 according to BIOGRID.
The gene DLG4 interacts with the gene ADRB1 according to BIOGRID.
ADRB1 is the start gene.
The distance of the gene DLG4 from the start gene is 1.
The distance of the gene CASK from the start gene is 2.

4.5 Experiments with Biomedical Queries

Our algorithms for generating explanations are applicable to the queries Q1, Q2, Q3, Q4, Q5, Q8, Q10, Q11 and Q12 in Table 1.1. The ASP programs for the other queries involve choice expressions. For instance, the query Q7 asks for cliques of 5 genes. We use the following rule to generate a possible set of 5 genes that might form a clique.

```
5{clique(GEN):gene_name(GEN)}5.
```

Our algorithms apply to ASP programs that contain a single atom in the heads of the rules, and negation and cardinality expressions in the bodies of the rules. Therefore, our methods are not applicable to the queries which are presented by ASP programs that include choice expressions.

In Table 4.2, we present the results for generating shortest explanations for the queries Q1, Q2, Q3, Q4, Q5, Q8, Q10, Q11 and Q12. In this table, the second column denotes the CPU timings to generate shortest explanations in seconds. The third column consists of the sizes of explanations, i.e., the number of rule vertices in an explanation. In the fourth column, the sizes of answer sets, i.e., the number of atoms in an answer set, are

Table 4.1: Predicate look-up table used while expressing explanations in natural language.

Predicate	**Expression in Natural Language**
gene_gene_biogrid(x,y)	The gene x interacts with the gene y according to BIOGRID.
drug_disease_ctd(x,y)	The disease y is treated by the drug x according to CTD.
drug_gene_ctd(x,y)	The drug x targets the gene y according to CTD.
gene_disease_ctd(x,y)	The disease y is related to the gene x according to CTD.
disease_symptom_do(x,y)	The disease x has the symptom y according to DISEASEONTOLOGY.
drug_category_drugbank(x,y)	The drug x belongs to the category y according to DRUGBANK.
drug_drug_drugbank(x,y)	The drug x reacts with the drug y according to DRUGBANK.
drug_sideeffect_sider(x,y)	The drug x has the side effect y according to SIDER.
disease_gene_orphadata(x,y)	The disease x is related to the gene y according to ORPHADATA.
drug_disease_pharmgkb(x,y)	The disease y is treated by the drug x according to PHARMGKB.
drug_gene_pharmgkb(x,y)	The drug x targets the gene y according to PHARMGKB.
disease_gene_pharmgkb(x,y)	The disease x is related to the gene y according to PHARMGKB.
start_drug(x)	The drug x is the start drug.
start_gene(x)	The gene x is the start gene.
drug_reachable_from(x, l)	The distance of the drug x from the start drug is l.
gene_reachable_from(x, l)	The distance of the gene x from the start gene is l.

Table 4.2: Experimental results for generating shortest explanations for some biomedical queries, using Algorithm 2.

Query	CPU Time	Explanation Size	Answer Set Size	And-Or Tree Size	Calling GRINGO
Q1	52.78s	5	1.964.429	16	0
Q2	67.54s	7	2.087.219	233	1
Q3	31.15s	6	1.567.652	15	0
Q4	1245.83s	6	19.476.119	6690	4
Q5	41.75s	3	1.465.817	16	0
Q8	40.96s	14	1.060.288	28	4
Q10	1601.37s	14	1.612.128	3419	193
Q11	113.40s	6	2.158.684	5528	5
Q12	327.22s	5	10.338.474	10	1

given. The fifth column presents the sizes of the and-or explanation trees, i.e., the number of vertices in the tree. Before telling what the last column presents, let us clarify an issue regarding the computation of explanations. Since answer sets contain millions of atoms, the corresponding grounded programs are also huge. Thus, first grounding the programs and then generating explanations over those grounded programs is an overkill in terms of computational efficiency. To this end, we apply another method and do grounding when it is necessary. To better explain the idea, let us present our method by an example. At the beginning, we have a grounded atom p for which we are looking shortest explanations. Assume that p is *what_be_genes("ADRB1")*. Then, we need to find rules whose heads are *what_be_genes("ADRB1")*. Since we work on non-ground programs, we look for rules whose heads are *what_be_genes(GN)*. Assume that the following rule r exists.

$$\textit{what\_be\_genes(GN)} \leftarrow \textit{drug\_gene(DRG,GN)}$$

Then, we propagate the corresponding constants due to *what_be_genes("ADRB1")* both in the head and the body of r. In this way, we obtain the following rule r'.

$$\textit{what\_be\_genes("ADRB1")} \leftarrow \textit{drug\_gene(DRG,"ADRB1")}$$

Next, we need to process over r'. As r' is non-ground, we ground it using the grounder GRINGO and process over the grounded rules. This allows us to deal with a relevant subset of the rules while generating explanations. If r' would be grounded after the propagation, we would not call GRINGO and continue the computation over r'. Then, in the last column of Table 4.2, we present the number of times GRINGO is called during the computation. For instance, for the queries Q1, Q3 and Q5, GRINGO is never called. However, GRINGO is called 193 times during the computation of a shortest explanation for the query Q10.

Shortest explanations for the queries in Table 4.2 can be found in Appendix A. For instance, for the query Q1, the explanation computed by EXPGEN-ASP is as follows:

```
what_be_drugs("Epinephrine") :- drug_gene("Epinephrine","ADRB1"),
                                drug_disease("Epinephrine","Asthma").

    drug_gene("Epinephrine","ADRB1") :-
          drug_gene_ctd("Epinephrine","ADRB1").

        drug_gene_ctd("Epinephrine","ADRB1").

    drug_disease("Epinephrine","Asthma") :-
          drug_disease_ctd("Epinephrine","Asthma").

        drug_disease_ctd("Epinephrine","Asthma").
```

which is essentially the tree in Figure 4.12. It took 52.71 CPU seconds for EXPGEN-ASP to generate this shortest explanation. As seen from the results presented in Table 4.2, the computation time is not very much related to the size of the explanation. As also suggested by the complexity results of Algorithm 2 (i.e., $O(|\Pi|^{|X|} \times |\mathcal{B}_\Pi)|)$, the computation time for generating shortest explanations greatly depends on the sizes of the answer set and the and-or explanation tree. For instance, for the query Q4, the answer set contains approximately 19 million atoms, the size of the and-or explanation tree is 6690, and it takes 1245 CPU seconds to compute a shortest explanation, whereas for the query Q8, the answer set approximately contains 1 million atoms, the and-or explanation tree has 28 vertices, and it takes 40 CPU seconds to compute a shortest explanation. Also, the number of times GRINGO is called during the computation affects the computation time. For instance, for the query Q10 the answer set approximately contains 1.6 million atoms, the and-or explanation tree has 3419 vertices, and it takes 1600 CPU seconds to compute a shortest explanation.

Table 4.3 shows the computation times for generating different explanations for the answers of the same queries, if exists. As seen from these results, the time for computing 2 and 4 different explanations is slightly different than the time for computing shortest explanations.

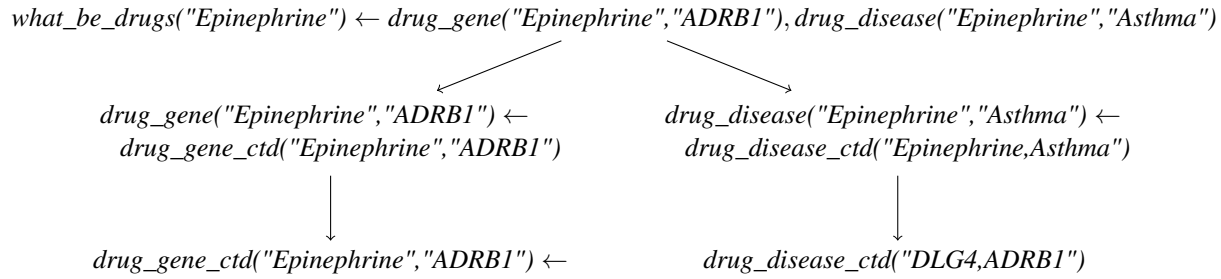


Figure 4.12: A shortest explanation for Q1.

Table 4.3: Experimental results for generating different explanations for some biomedical queries, using Algorithm 6.

Query	**CPU Time**		
	2 different	4 different	Shortest
Q1	53.73s	-	52.78s
Q2	66.88s	67.15s	67.54s
Q3	31.22s	-	31.15s
Q4	1248.15s	1251.13s	1245.83s
Q5	-	-	41.75s
Q8	-	-	40.96s
Q10	1600.49s	1602.16s	1601.37s
Q11	113.25s	112.83s	113.40s
Q12	-	-	327.22s

4.6 Implementation of Explanation Generation Algorithms

Based on the algorithms introduced above, we have developed a computational tool called EXPGEN-ASP, using the programming language C++. Given an ASP program and its answer set, EXPGEN-ASP generates shortest explanations as well as k different explanations.

The input of EXPGEN-ASP are

- an ASP program Π,
- an answer set X for Π,
- an atom p in X,
- an option that is used to generate either a shortest explanation or k different explanations,
- a predicate look-up table,

and the output are

- a shortest explanation for p with respect to Π and X in a natural language (if shortest explanation option is chosen),
- k different explanations for p with respect to Π and X in a natural language (if k different explanations option is chosen).

For generating shortest explanations (resp., k different explanations), EXPGEN-ASP utilizes Algorithm 2 (resp., Algorithm 6).

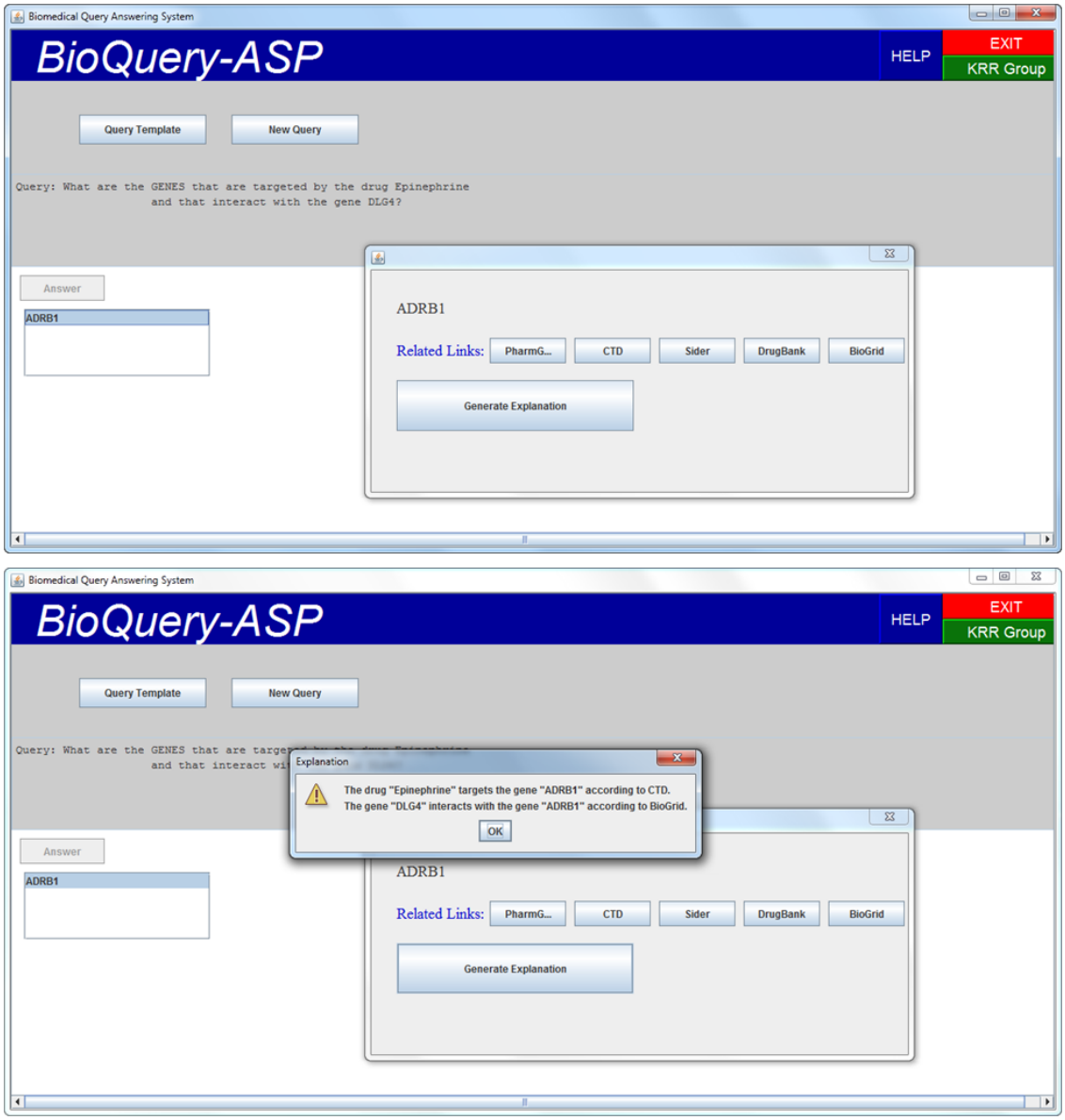


Figure 4.13: A snapshot of BIOQUERY-ASP showing its explanation generation facility.

To provide experts with further informative explanations about the answers of biomedical queries, we have embedded EXPGEN-ASP into BIOQUERY-ASP by utilizing Table 4.1 as the predicate look-up table of the system. Figure 4.13 shows a snapshot of the explanation generation mechanism of BIOQUERY-ASP.

CHAPTER 5

RELATING EXPLANATIONS TO JUSTIFICATIONS

The most similar work to ours is [78] that study the question "why is an atom p in an answer set X for an ASP program Π". As an answer to this question, the authors of [78] finds a "justification", which is a labeled graph that provides an explanation for the truth values of atoms with respect to an answer set.

Example 7. *Let* Π *be the program*

$$a \leftarrow b, c$$
$$a \leftarrow d$$
$$d \leftarrow$$
$$b \leftarrow c$$
$$c \leftarrow$$

and $X = \{a, b, c, d\}$. *Figure 5.1 is an offline justification of* a^+ *with respect to* X *and* $\emptyset$. *Intuitively,* a *is in* X *since* b *and* c *are also in* X *and there is a rule in* Π *that supports* a *using the atoms* b *and* c. *Furthermore,* b *is in* X *since* c *is in* X *and there is a rule in* Π *that supports* b *using the atom* c. *Finally,* c *is in* X *as it is a fact in* Π.

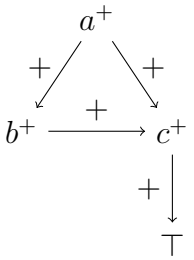


Figure 5.1: An offline justification for Example 7.

To relate offline justifications and explanations, we need to introduce the following definitions and notations about justifications defined in [78].

5.1 Offline Justifications

First, let us introduce notations related to ASP programs used in [78]. The class of ASP programs studied is normal programs, i.e., programs that consist of the rules of the form

$$A \leftarrow A_1, \ldots, A_k, not\ A_{k+1}, \ldots, not\ A_m$$

where $m \geq k \geq 0$ and A and each A_i is an atom. Therefore, the programs we consider are more general. Let Π be a normal ASP program. Then, $\mathcal{A}_\Pi$ is the Herbrand base of Π. An *interpretation* I for a program Π is defined as a pair $\langle I^+, I^- \rangle$, where $I^+ \cup I^- \subseteq \mathcal{A}_\Pi$ and $I^+ \cap I^- = \emptyset$. Intuitively, I^+ denotes the set of atoms that are true, while I^- denotes the set of atoms that are false. I is a *complete interpretation* if $I^+ \cup I^- = \mathcal{A}_\Pi$. The *reduct* Π^I of Π with respect to I is defined as

$$\Pi^I = \{H(r) \leftarrow B^+(r) \,|\, r \in \Pi,\ B^-(r) \cap I^+ = \emptyset\}$$

A complete interpretation M for a program Π is an answer set for Π if M^+ is an answer set for Π^M. Also, a *literal* is either an atom or a formula of the form $not\ a$ where a is an atom. The set of atoms which appear as negated literals in Π is denoted by $NANT(\Pi)$. For an atom a, a^+ denotes that the atom a is true and a^- denotes that a^- is false. Then, a^+ and a^- are called the *annotated* versions of a. Moreover, it is defined that $atom(a^+) = a$ and $atom(a^-) = a$. For a set S of atoms, the following sets of annotated atoms are defined.

- $S^p = \{a^+ \,|\, a \in S\}$
- $S^n = \{a^- \,|\, a \in S\}$

Finally, the set $not\ S$ is defined as $not\ S = \{not\ a \,|\, a \in S\}$.

Apart from the answer set semantics, there is another important semantics of logic programs, called the well-founded semantics [49]. Since this semantics is important to build the notion of a justification, we now briefly describe the well-founded semantics. We consider the definition proposed in [3], instead of the original definition proposed in [49], as the authors of [78] considered.

Definition 14 (Immediate consequence)**.** *Let Π be a normal ASP program, and S and V be two sets of atoms from $\mathcal{A}_\Pi$. Then, the* immediate consequence *of S with respect to Π and V, denoted by $T_{\Pi,V}(S)$ is the set defined as follows:*

$$T_{\Pi,V}(S) = \{a \,|\, \exists\ r \in \Pi,\ H(r) = a,\ B^+(r) \subseteq S,\ B^-(r) \cap V = \emptyset\}.$$

We denote by $lfp(T_{\Pi,V})$ the least fixpoint of $T_{\Pi,V}$ when V is fixed.

Definition 15 (The well-founded model)**.** *Let* Π *be a normal ASP program,* $\Pi^+ = \{r \mid r \in \Pi,\ B^-(r) = \emptyset\}$*. The sequence* $\langle K_i, U_i \rangle_{i \geq 0}$ *is defined as follows:*

$$\begin{array}{ll} K_0 = lfp(T_{\Pi^+}), & U_0 = lfp(T_{\Pi,K_0}), \\ K_i = lfp(T_{\Pi,U_{i-1}}), & U_i = lfp(T_{\Pi,K_i}). \end{array}$$

Let j *be the first index of the computation such that* $\langle K_j, U_j \rangle = \langle K_{j+1}, U_{j+1} \rangle$*. Then,* the well-founded model *of* Π *is* $WF_\Pi = \langle W^+, W^- \rangle$ *where* $W^+ = K_j$ *and* $W^- = \mathcal{A}_\Pi \backslash U_j$*.*

Example 8. *Let* Π *be the program*

$$\begin{array}{l} a \leftarrow b, not\ d \\ d \leftarrow b, not\ a \\ b \leftarrow c \\ c \leftarrow \end{array}$$

Then, the well-founded model of Π *is computed as follows:*

$$\begin{array}{rcl} K_0 & = & \{b, c\}, \\ U_0 & = & \{a, b, c, d\}, \\ K_1 & = & \{b, c\}, \\ U_0 & = & \{a, b, c, d\}. \end{array}$$

Thus, $WF_\Pi = \langle \{b, c\}, \emptyset \rangle$*.*

We now provide definitions regarding the notion of an offline justification. First, we introduce the basic building of an offline justification, a labeled graph called as e-graph.

Definition 16 (e-graph)**.** *Let* Π *be a normal ASP program. An* e-graph *for* Π *is a labeled, directed graph* (N, E)*, where* $N \subseteq \mathcal{A}^p_\Pi \cup \mathcal{A}^n_\Pi \cup \{assume, \top, \bot\}$ *and* $E \subseteq N \times N \times \{+, -\}$*, which satisfies following properties:*

(i) *the only sinks (i.e., nodes without out-going edges) in the graph are* $assume, \top, \bot$*;*

(ii) *for every* $b \in N \cap \mathcal{A}^p_\Pi$*,* $(b, assume, -) \notin E$ *and* $(b, \bot, -) \notin E$*;*

(iii) *for every* $b \in N \cap \mathcal{A}^n_\Pi$*,* $(b, assume, +) \notin E$ *and* $(b, \top, +) \notin E$*;*

(iv) *for every* $b \in N$*, if for some* $l \in \{assume, \top, \bot\}$ *and* $s \in \{+, -\}$*,* $(b, l, s) \in E$*, then* (b, l, s) *is the only out-going edge originating from* b*.*

According to this definition, an edge of an e-graph connects two annotated atoms or an annotated atom with one of the nodes in $\{assume, \top, \bot\}$ and is marked by a label from $\{+, -\}$. An edge is called as *positive* (resp., *negative*) if it is labeled by $+$ (resp., $-$). Also, a path in an e-graph is called as *positive* if it has only positive edges, whereas it

is called as *negative* if it has at least one negative edge. The existence of a positive path between two nodes v_1 and v_2 is denoted by $(v_1, v_2) \in E^{*,+}$. In the offline justification, $\top$ is used to explain facts, $\bot$ to explain atoms which do not have defining rules, and $assume$ is for atoms for which explanations are not needed, i.e., they are assumed to be true or false.

Example 9. *Let* Π *be the program:*

$$
\begin{aligned}
a &\leftarrow b, c \\
a &\leftarrow d \\
d &\leftarrow \\
b &\leftarrow c \\
c &\leftarrow
\end{aligned}
$$

and $X = \{a, b, c, d\}$*. Then, Figure 5.2 is an e-graph for* Π*. Intuitively, the true state of* a *depends on the true state of* b *and the false state of* c*, where* b *is assumed to be true and* c *is assumed to be false.*

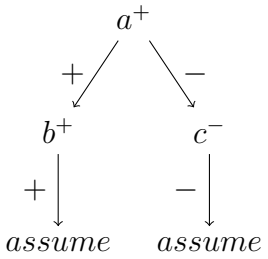


Figure 5.2: An e-graph for Example 9.

In an e-graph, a set of elements that directly contributes to the truth value of an atom can be obtained through the out-going edges of a corresponding node. This set is defined as follows.

Definition 17 ($support(b, G)$)**.** *Let* Π *be a normal ASP program,* $G = (N, E)$ *be an e-graph for* Π *and* $b \in N \cap (\mathcal{A}^p_\Pi \cup \mathcal{A}^n_\Pi)$ *be a node in* G*. Then,* $support(b, G)$ *is defined as follows.*

- $support(b, G) = \{l\}$*, if for some* $l \in \{assume, \top, \bot\}$ *and* $s \in \{+, -\}$*,* (b, l, s) *is in* E*;*
- $support(b, G) = \{atom(c) \mid (b, c, +) \in E\} \cup \{not\, atom(c) \mid (b, c, -) \in E\}$*, otherwise.*

Example 10. *Let* G *be the e-graph in Figure 5.2. Then,* $support(a, G) = \{b, not\ c\}$*,* $support(b, G) = \{assume\}$ *and* $support(c, G) = \{assume\}$*.*

To define the notion of a justification, an e-graph should be enriched with explanations of truth values of atoms that are derived from the rules of the program. For that, the concept of one step justification of a literal is defined as follows.

Definition 18 (Local Consistent Explanation (LCE)). *Let Π be a normal ASP program, b be an atom, J be a possible interpretation for Π, $U \subseteq \mathcal{A}_\Pi$ be a set of atoms, and $S \subseteq \mathcal{A}_\Pi \cup not\, \mathcal{A}_\Pi \cup \{assume, \top, \bot\}$ be a set of literals. We say that*

- *S is an* LCE *of b^+ with respect to (J, U), if $b \in J^+$ and*
  - *$S = \{assume\}$ or*
  - *$S \cap \mathcal{A}_\Pi \subseteq J^+$, $\{c \,|\, not\, c \in S\} \subseteq J^- \cup U$, and there is a rule $r \in \Pi$ such that $H(r) = b$ and $B(r) = S$. In case, $B(r) = \emptyset$, S is denoted by the set $\{\top\}$.*
- *S is an* LCE *of b^- with respect to (J, U), if $b \in J^- \cup U$ and*
  - *$S = \{assume\}$ or*
  - *$S \cap \mathcal{A} \subseteq J^- \cup U$, $\{c \,|\, not\, c \in S\} \subseteq J^+$, and S is a minimal set of literals such that for every rule $r \in \Pi$ if $H(r) = b$, then $B^+(r) \cap S \neq \emptyset$ or $B^-(r) \cap \{c \,|\, not\, c \in S\} \neq \emptyset$. In case, $S = \emptyset$, S is denoted by the set $\{\bot\}$.*

The set of all the LCEs of b^+ with respect to (J, U) is denoted by $LCE^p_\Pi(b, J, U)$ and the set of all the LCEs of b^- with respect to (J, U) is denoted by $LCE^n_\Pi(b, J, U)$.

Here, a possible interpretation J denotes an answer set. The set U consists of atoms that are assumed to be false (which will be called as Assumptions in the notion of justification later on). The need for U comes from the fact that the truth value of some atoms is first guessed while computing answer sets. Intuitively, if an atom a is true, an LCE consists of the body of a rule which is satisfied by J and has a in its head; if a is false, an LCE consists of a set of literals that are false in J and falsify all rules whose head are a.

Example 11. *Let Π and X be defined as in Example 9. Then, the LCEs of the atoms with respect to $(X, \emptyset)$ is as follows.*

$$
\begin{aligned}
LCE^p_\Pi(a, X, \emptyset) &= \{\{b, c\}, \{d\}, \{assume\}\} \\
LCE^p_\Pi(b, X, \emptyset) &= \{\{c\}, \{assume\}\} \\
LCE^p_\Pi(c, X, \emptyset) &= \{\{\top\}, \{assume\}\} \\
LCE^p_\Pi(d, X, \emptyset) &= \{\{\top\}, \{assume\}\}
\end{aligned}
$$

Accordingly, a class of e-graphs where edges represent LCEs of the corresponding nodes are defined as follows.

Definition 19 ((J,U)-based e-graph)**.** *Let* Π *be a normal ASP program,* J *be a possible interpretation for* Π*,* $U \subseteq \mathcal{A}_\Pi$ *be a set of atoms and* b *be an element in* $\mathcal{A}^p_\Pi \cup \mathcal{A}^n_\Pi$*. A* (J,U)-based e-graph $G = (N,E)$ *of* b *is an e-graph such that*

(i) *every node* $c \in N$ *is reachable from* b*,*

(ii) *for every* $c \in N \backslash \{assume, \top, \bot\}$*,* $support(c,G)$ *is an LCE of* c *with respect to* (J,U)*;*

A (J,U)*-based e-graph* (N,E) *is* safe *if for all* $b^+ \in N, (b^+,b^+) \notin E^{*,+}$*, i.e., there is no positive cycle in the graph.*

We now introduce a special class of (J,U)-based e-graphs where only false elements can be assumed.

Definition 20 (Offline e-graph)**.** *Let* Π *be a normal ASP program,* J *be a partial interpretation for* Π*,* $U \subseteq \mathcal{A}_\Pi$ *be a set of atoms and* b *be an element in* $\mathcal{A}^p \cup \mathcal{A}^n$*. An* offline e-graph $G = (N,E)$ *of* b *with respect to* J *and* U *is a* (J,U)*-based e-graph of* b *that satisfies following properties:*

(i) *there exists no* $p^+ \in N$ *such that* $(p^+, assume, +) \in E$*;*

(ii) $(p^-, assume, -) \in E$ *if and only if* $p \in U$*.*

$\mathcal{E}(b,J,U)$ *is the set of all offline e-graphs of* b *with respect to* J *and* U*.*

Here, the roles of J and U are the same as their roles in Definition 18. Observe that true atoms cannot be assumed due to the first condition and only elements in the set U are assumed due to the second condition.

We said earlier that in a (J,U)-based e-graph J represents an answer set and U consists of atoms that are assumed to be false. Here, U is chosen based on some characteristics of J. In particular, we want U to be a set of atoms such that when its elements are assumed to be false, the truth value of other atoms in the program can be uniquely determined and leads to J. We now introduce regarding definitions formally.

Definition 21 (Tentative Assumptions)**.** *Let* Π *be a normal ASP program,* M *be an answer set for* Π *and* $WF_\Pi =< WF^+_\Pi, WF^-_\Pi >$ *be the well-founded model of* Π*. The* tentative assumptions $\mathcal{TA}_\Pi(M)$ *of* Π *with respect to* M *are defined as*

$$\mathcal{TA}_\Pi(M) = \{a \,|\, a \in NANT(\Pi) \wedge a \in M^-, a \notin (WF^+_\Pi \cup WF^-_\Pi)\} \tag{5.1}$$

Example 12. *Let* Π *be the program:*

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

Then, $X = \{a, b, c\}$ *is an answer set for* Π *and* $\langle \{a, b\}, \emptyset \rangle$ *is the well-founded model of* Π*. Given that,* $\mathcal{TA}_\Pi(X) = \{d\}$ *as* $d \in NANT(\Pi)$*,* $d \notin X$ *and* $d \notin (WF^+_\Pi \cup WF^-_\Pi)$*.*

In fact, tentative assumptions is a set of atoms whose subsets might "potentially" form U.

We provide a definition that would allow one to obtain a program from a given program Π and a set V of atoms by assuming all the atoms in V as false.

Definition 22 (Negative Reduct)**.** *Let* Π *be a normal ASP program,* M *be an answer set for* Π*, and* $U \subseteq \mathcal{TA}_\Pi(M)$ *be a set of tentative assumption atoms. The* negative reduct $NR(\Pi, U)$ *of* Π *with respect to* U *is the set of rules defined as*

$$NR(\Pi, U) = \Pi \backslash \{r \mid H(r) \in U\} \tag{5.2}$$

Finally, the concept of assumptions can be introduced formally.

Definition 23 (Assumption)**.** *Let* Π *be a normal ASP program,* M *be an answer set for* Π*. An* assumption *of* Π *with respect to* M *is a set* U *of atoms that satisfies the following properties:*

(i) $U \subseteq \mathcal{TA}_\Pi(M)$*;*

(ii) *the well-founded model of* $NR(\Pi, U)$ *is equal to* M*.*

$Assumptions(\Pi, M)$ *is the set of all assumptions of* Π *with respect to* M*.*

Example 13. *Let* Π *and* X *be defined as in Example 12. Let* $U = \{d\}$*. Then,* $NR(\Pi, U)$ *is:*

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

and $\langle \{a, b, c\}, \emptyset \rangle$ *is the well-founded model of* $NR(\Pi, U)$*. Thus,* U *is an assumption of* Π *with respect to* X*.*

Note that assumptions are nothing but subsets of tentative assumptions that would allow to obtain the answer set J.

At last, we are ready to define the notion of offline justification.

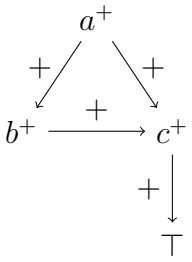


Figure 5.3: An offline justification for Example 14.

Definition 24 (Offline Justification)**.** *Let* Π *be a normal ASP program,* M *be an answer set for* Π*,* U *be an assumption in* $Assumptions(\Pi, M)$ *and* b *be an annotated atom in* $\mathcal{A}^p \cup \mathcal{A}^n$*. An* offline justification *of* b *with respect to* M *and* U *is an element* (N, E) *of* $\mathcal{E}(b, M, U)$ *which is safe.*

According to the definition, a justification is a (J, U)-based e-graph where J is an answer set and U is an assumption. Also, justifications do not allow the creation of positive cycles in the justification of true atoms.

Example 14. *Let* Π *and* X *be defined as in Example 9. Figure 5.3 is an offline justification of* a^+ *with respect to* X *and* $\emptyset$*.*

In [78], the authors prove the following proposition which shows that for every atom in the program, there exists an offline justification.

Proposition 9. *Let* Π *be a ground normal ASP program,* X *be an answer for* Π*. Then, for each atom* a *in* Π*, there is an offline justification of with respect to* X *and* $X^- \backslash WF_\Pi^-$ *which does not contain negative cycles.*

5.2 From Justifications to Explanations

We relate a justification to an explanation. In particular, given an offline justification, we show that one can obtain an explanation tree whose atom vertices are formed by utilizing the "annotated atoms" of the justification and rule vertices are formed by utilizing the "support" of annotated atoms. To compute such explanation trees, we develop Algorithm 8.

Let us now explain the algorithm in detail. Algorithm 8 takes as input a ground normal ASP program Π , an answer set X for Π, an atom p in X, and a justification (V, E) of p^+ with respect to X and some $U \in Assumptions(\Pi, X)$. Our goal is to obtain an explanation tree in the and-or explanation tree for p with respect to Π and X from the justification (V, E). The algorithm starts by creating two sets V' and E' (Line 1). Here, V' and E' corresponds to the set of vertices and the set of edges of the explanation

Algorithm 8: Justification to Explanation

Input: Π : ground normal ASP program, X : answer set for Π, p : atom in X, (V, E) : justification of p^+ w.r.t X and some $U \in Assumptions(\Pi, X)$.

Output: A vertex-labeled tree $\langle V', E', l, \Pi, X \rangle$.

```
1  V' := ∅, E' := ∅
2  v ← Create a vertex v s.t. l(v) = p
3  Q ← v
4  while Q ≠ ∅ do
5      v' ← Dequeue an element from Q
6      V' := V' ∪ {v'}
7      if l(v') ∈ Π then // v' is a rule vertex
8          foreach a ∈ B+(l(v')) do
9              v'' ← Create a vertex v'' s.t. l(v'') = a
10             E' := E' ∪ {(v', v'')} // edge from rule vertex to atom vertex
11             Enqueue v'' to Q
12     else if l(v') ∈ X then // v' is an atom vertex
13         r ← Create a rule r s.t. H(r) = l(v') and B(r) = support(l(v')+, G)
14         v'' ← Create a vertex v'' s.t. l(v'') = r
15         E' := E' ∪ {(v', v'')} // edge from atom vertex to rule vertex
16         Enqueue v'' to Q
17 return ⟨V', E', l, Π, X⟩
```

tree, respectively. By Condition (ii) in Definition 10 and Condition (i) in Definition 9, we know that the label of the root of an explanation tree for p with respect to Π and X is p. Thus, a vertex v with label p is defined (Line 2), and added into the queue Q (Line 3). Then, the algorithm enters into a "while" loop which executes until Q becomes empty. At every iteration of the loop, an element v from Q is first extracted (Line 5) and added into V' (Line 6). This implies that every element added into Q is also added into V'. For instance, the vertex defined at Line 2 is the first vertex extracted from Q and also added into V', which makes sense since we know that the root of an explanation tree is an atom vertex with label v. Then, according to the type of the extracted vertex, its out-going edges are defined. Let v' be a vertex extracted from Q at Line 5 in some iteration of the loop. Consider the following two cases.

Case (1) Assume that v' is an atom vertex. Then, the algorithm directly goes to Line 13. By Condition (i) in Definition 10, we know that an explanation tree is a subtree of the

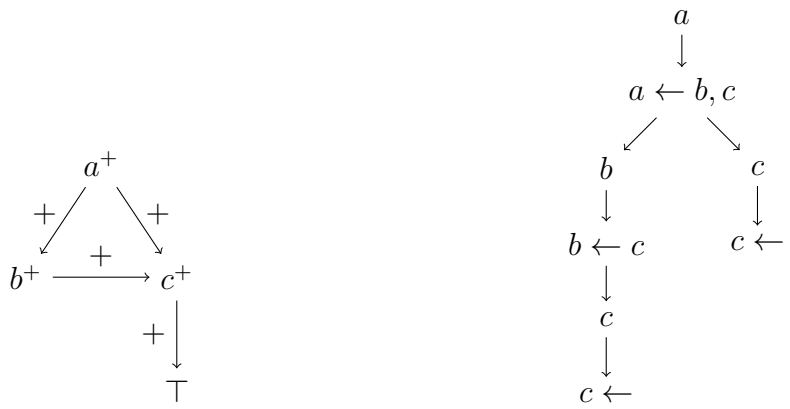
and-or explanation tree. Hence, we need to define out-going edges of v' by taking into account Condition (ii) in Definition 9, which implies that a child of v' must be a rule vertex v'' such that the rule that labels v'' "supports" the atom that labels v'. Thus, a rule r that supports the atom that labels v' is created (Line 13). We ensure "supportedness" property by utilizing the annotated atoms in the given offline justification which supports the annotated version of the atom that labels v'. Then, a vertex v'' with label r is created (Line 14), and a corresponding child of v' is added into E' (Line 15). By Condition (iii) in Definition 10, we know that every atom vertex of an explanation tree has a single child. Therefore, another child of v' is not created. Then, before finishing the iteration of the loop, the child v'' of v' is added into Q so that its children can be formed in the next iterations of the loop.

Case (2) Assume that v' is a rule vertex. Then, the condition at Line 7 is satisfied and the algorithm goes to Line 8. In this case, while forming the children of v', we should consider Condition (iii) in Definition 9, which implies that a child v'' of v' must be an atom vertex such that the atom that labels v'' is in the positive body of the rule that labels v'. Also, by Condition (iv) in Definition 10, we should ensure that for every atom a in the positive body of the rule that labels v', there exists a child v_a of v' such that the atom that labels v_a is equal to a. Thus, the loop between Lines 8–11 iterates for every atom a in the positive body of the label of v' and a vertex v'' with label a is created (Line 9). Then, v'' becomes a child of v' (Line 10). To form the children of v'' in the next iterations of the "while" loop, the child v'' of v' is added into Q (Line 11).

When the algorithm finishes processing the elements of Q, i.e., Q becomes empty, the "while" loop terminates. Then, the algorithm returns a vertex-labeled tree (Line 17). We now provide the proposition about the soundness of Algorithm 8.

Proposition 10. *Given a ground normal ASP program Π, an answer set X for Π, an atom p in X, an assumption U in $Assumption(\Pi, X)$, and an offline justification $G = (V, E)$ of p^+ with respect to X and U, Algorithm 8 returns an explanation tree $\langle V', E', l, \Pi, X\rangle$ in the and-or explanation tree for p with respect to Π and X.*

Example 15. *Let Π and X be defined as in Example 9. Figure 5.4(a) is an offline justification of a^+ with respect to X and $\emptyset$. Figure 5.4(b) shows a corresponding explanation tree in the and-or explanation tree for a with respect to Π and X that is obtained by using Algorithm 8.*



(a) (b)

Figure 5.4: (a) An offline justification and (b) its corresponding explanation tree obtained by using Algorithm 8.

5.3 From Explanations to Justifications

We relate an explanation to a justification. In particular, given an explanation tree whose labels of vertices are unique, we show that one can obtain an offline justification by utilizing the labels of atom vertices of the explanation tree. For that, we design Algorithm 9.

Let us now describe the algorithm in detail. Algorithm 9 takes as input a ground normal ASP program Π, an answer set X for Π, an atom p in X, and an explanation tree $T' = \langle V', E', l, \Pi, X \rangle$ in the and-or explanation tree for p with respect to Π and X. Our goal is to obtain an offline justification (V, E) of p^+ in Π^X with respect to X and $\emptyset$. The reason to obtain the offline justification in the reduct of Π with respect to X is that our definition of explanation is not defined for the atoms that are not in the answer set. Algorithm 9 starts by creating two sets V and E which will correspond to the set of nodes and the set of edges of the offline justification, respectively (Line 1). Then, the root of $\langle V', E' \rangle$ is added into the queue Q (Line 2) and we enter into a "while" loop that iterates until Q becomes empty. At every iteration of the loop, first an element v is extracted from Q (Line 4) and $l(v)^+$ is added into V (Line 5). Then, we form the out-going edges of $l(v)^+$. Due to Condition (iii) in Definition 10, every atom vertex in an explanation tree has a single child, which is a rule vertex due to Condition (i) in Definition 10 and Condition (ii) in Definition 9. Then, we extract the child v' of v at Line 6 and consider two cases. Note that v' is a rule vertex.

Case (1) Assume that $l(v')$ is a fact in Π^X. Then, $l(v')$ satisfies the condition at Line 7 and we add $(l(v)^+, \top, +)$ into E at Line 8. The key insight behind that is as follows. Due to Condition (ii) in Definition 19, $support(l(v)^+, (V, E))$ must be an LCE of $l(v)^+$. Due

Algorithm 9: Explanation to Justification

Input: Π : ground normal ASP program, X : answer set for Π, p : atom in X, $\langle V', E', l, \Pi, X\rangle$: an explanation tree in the and-or explanation tree for p w.r.t Π and X.

Output: (V, E) : justification of p^+ w.r.t X and $\emptyset$.

1 $V := \emptyset,\ E := \emptyset$
2 $Q \leftarrow$ root of $\langle V', E'\rangle$
3 **while** $Q \neq \emptyset$ **do**
4 $v \leftarrow$ Dequeue an element from Q
5 $V := V \cup \{l(v)^+\}$
6 $v' \leftarrow$ child of v in $\langle V', E'\rangle$
7 **if** $l(v')$ *is a fact in* Π^X **then**
8 $E := E \cup \{(l(v)^+, \top, +)\}$
9 **foreach** $v'' \in$ *child*$_{E'}(v')$ **do**
10 $E := E \cup \{(l(v)^+, l(v'')^+, +)\}$
11 Enqueue v'' to Q
12 $V := V \cup \{\top\}$
13 **return** (V, E)

to Condition (i) in Definition 10 and Condition (ii) in Definition 9, the head of $l(v')$ is $l(v)$. As $l(v')$ is a fact in Π^X, i.e., its body is empty in Π^X, $\{\top\}$ becomes an LCE of $l(v)^+$ with respect to $(X, \emptyset)$, due to Definition 18. Thus, by adding $(l(v)^+, \top, +)$ to E, $support(l(v)^+, (V, E))$ becomes $\{\top\}$.

Case (2) Assume that $l(v')$ is not a fact in Π^X. Then, for every child v'' of v', we add $(l(v)^+, l(v'')^+, +)$ into E at Line 10. The intuition behind this is to make sure that $support(l(v)^+, (V, E))$ is an LCE of $l(v)^+$. Due to Condition (i) in Definition 10, an explanation tree is a subtree of the corresponding and-or explanation tree. Then, due to Condition (ii) in Definition 9, for every atom vertex v in an explanation tree, the atoms in the positive body of the rule that labels the child v' of v are in the given answer set X. Thus, due to Definition 18, adding $(l(v)^+, l(v'')^+, +)$ into E for every child v'' of v' ensures that $support(l(v)^+, (V, E))$ is an LCE of $l(v)^+$ with respect to $(X, \emptyset)$. Also, we add v'' into V so that its children in V are formed in the next iterations of the "while" loop.

Due to Line 8, there are incoming edges of $\top$. But, $\top$ is not added into V inside the "while" loop. Thus, when the "while" loop terminates, before returning (V, E) at Line 13, we add $\top$ into V.

Algorithm 9 creates an offline justification of the given atom in the reduct of the given ASP program with respect to the given answer set, provided that labels of the vertices of the given explanation tree are unique.

Proposition 11. *Given a ground normal ASP program* Π *, an answer set* X *for* Π*, an atom* p *in* X*, and an explanation tree* $\langle V', E', l, \Pi, X\rangle$ *in the and-or explanation tree for* p *with respect to* Π *and* X *such that for every* $v, v' \in V'$*,* $l(v) = l(v')$ *if and only if* $v = v'$*, Algorithm 9 returns an offline justification of* p^+ *in* Π^X *with respect to* X *and* $\emptyset$*.*

Example 16. *Let* Π *be the program:*

$$a \leftarrow b, c, not\ d$$
$$b \leftarrow not\ e$$
$$c \leftarrow$$

and $X = \{a, b, c\}$*. Figure 5.5(a) is an explanation tree* T *in the and-or explanation tree for* a *with respect to* Π *and* X*. Then, given* Π, X, a *and* T*, Algorithm 9 creates an offline justification of* a^+ *in* Π^X *with respect to* Π *and* $\emptyset$ *as in Figure 5.5(b).*

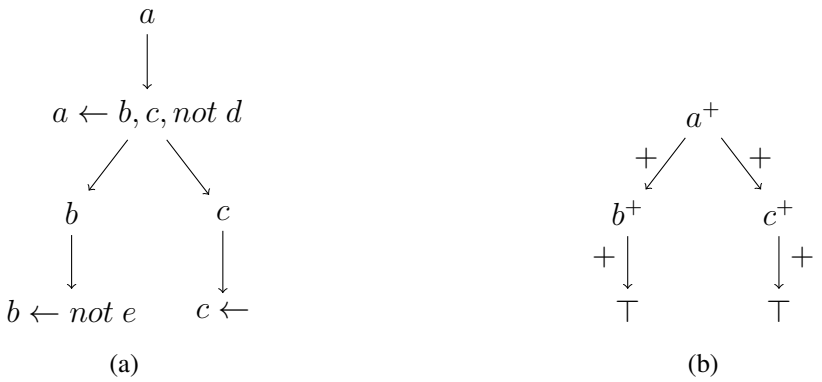


Figure 5.5: (a) An explanation tree and (b) its corresponding offline justification obtained by using Algorithm 9.

CHAPTER 6

PROOFS

In this chapter, we provide proofs of the theoretical results presented throughout the thesis: the algorithmic analysis for generating shortest explanations (Propositions 2, 3, and 4), the algorithmic analysis for generating k different explanations (Propositions 5, 6, and 8), and the analysis for the relationship between an explanation and a justification. (Propositions 10 and 11).

6.1 Generating Shortest Explanations

In Section 4.2, we have analyzed three properties of Algorithm 2, namely termination, soundness and complexity, resulting in Propositions 2, 3, and 4. In the following, we show the proofs of these results.

6.1.1 Proof of Proposition 2 – Termination of Algorithm 2

To prove that Algorithm 2 terminates we need the following lemma.

Lemma 1. *Given a ground ASP program Π, an answer set X for Π, an atom p in X, and a set L of atoms in X, Algorithm 3 terminates.*

Proof of Lemma 1. It is sufficient to show that the recursion tree generated by Algorithm 3 is finite. That is, the branching factor of the recursion tree and the height of the recursion tree are finite. Note that each node of the recursion tree denotes a call to createTree(Π, X, d, L) for some atom or rule d and a set L of atoms.

Part (1) We show that the branching factor of the tree is finite. Branches in the tree are created in loops at Lines 5 and 13. The loop at Line 5 iterates at most the number of rules in Π and the loop at Line 13 iterates at most the number of atoms in X. As Π and X are finite, the branching factor of the tree is finite.

Part (2) We show that the height of the recursion tree is finite. Let us first make the following observation. Consider a path $\langle v_1, \ldots \rangle$ in the recursion tree. For every node v_i that denotes a call to createTree(Π, X, d_i, L_i) where d_i is an atom in X, the following holds for v_{i+2} (if exists): $L_i \subset L_{i+2}$. This follows from the two consecutive calls to createTree: L_i increases at every other call, due to Line 4. Now, assume that the height of the recursion tree is infinite. Then, there exists an infinite path $\langle v_1, \ldots \rangle$ in the recursion tree. Thus, $L_1 \subset L_3 \subset \ldots \subset L_{(2\times|X|)+1} \subset L_{(2\times|X|)+3} \subset \ldots$. Recall that we add elements into L_is only from X (Line 4). Thus, $L_{(2\times|X|)+1} = X$. Then, it is not possible to have $L_{(2\times|X|)+1} \subset L_{(2\times|X|)+3}$. As we reach a contradiction, the height of the recursion tree is finite.

□

Now, we show that Algorithm 2 terminates.

Proposition 2. *Given a ground ASP program Π, an answer set X for Π, and an atom p in X, Algorithm 2 terminates.*

Proof of Proposition 2. Algorithm 2 terminates only if Algorithms 3, 4 and 5 terminate. By Lemma 1, we know that Algorithm 3 terminates and that the vertex-labeled tree T returned by Algorithm 3 is finite. Since Algorithm 4 and Algorithm 5 simply traverse T (cf. Lines 2 and 6 in Algorithm 4, and Lines 5 and 10 in Algorithm 5), they also terminate. Thus, Algorithm 2 terminates.

□

6.1.2 Proof of Proposition 3 – Soundness of Algorithm 2

To show the proof of Proposition 3, we need the following necessary lemmas.

Lemma 2. *Let Π be a ground ASP program, X be an answer set for Π, d be an atom in X or a rule in Π and L be a subset of X. If the vertex-labeled tree $\langle V, E, l, \Pi, X \rangle$ returned by createTree(Π, X, d, L) is not empty, then the following hold:*

(i) *the root of $\langle V, E \rangle$ is created in createTree(Π, X, d, L) and is mapped to d by l;*

(ii) *for every rule vertex $v \in V$,*

$$out_E(v) = \{(v, v') \mid (v, v') \in E,\ l(v') \in B^+(l(v))\};$$

(iii) *each leaf vertex is a rule vertex.*

Proof of Lemma 2. Let $\langle V, E, l, \Pi, X \rangle$ be the non-empty vertex-labeled tree returned by createTree(Π, X, d, L). We show one by one that each condition in the lemma holds.

(i) Assume that d is an atom in X. Note that $d \notin L$ because otherwise $\langle V, E\rangle = \langle \emptyset, \emptyset\rangle$. Due to the call createTree(Π, X, d, L), the algorithm, at Line 3, creates a vertex v such that $l(v) = d$. We know that $E \neq \emptyset$. Then, there exists some out-going edges of v. At Line 9, the out-going edges of v are formed. Due to Line 8, each vertex v' in (v, v') is the root of a vertex-labeled tree. Moreover, there is no part of the algorithm that adds a "parent" to a vertex. Therefore, v is the root of $\langle V, E\rangle$.

Similar reasoning can be applied for the case where d is a rule in Π.

(ii) Let $v \in V$ be a rule vertex. Let $S_v = \{(v, v') \,|\, (v, v') \in E,\, l(v') \in B^+(l(v))\}$ denoting the set of out-going edges of v to atom vertices whose labels are in the positive body of the rule that labels v. We show that $\mathit{out}_E(v) = S_v$.

Let $(v, v') \in \mathit{out}_E(v)$. Then, $(v, v') \in E$. Edges are created at Lines 9 and 17. As v is a rule vertex, (v, v') must be created at Line 17. Then, by Line 16 and the condition at Line 13, $l(v') \in B^+(l(v))$. So, $(v, v') \in S_v$ (i.e., $\mathit{out}_E(v) \subseteq S_v$).

Let $(v, v') \in S_v$. Then, by the definition of S_v, $(v, v') \in E$. Thus, $(v, v') \in \mathit{out}_E(v)$ (i.e., $S_v \subseteq \mathit{out}_E(v)$).

(iii) Assume otherwise. Then, there exists an atom vertex $v \in V$ such that it is a leaf vertex. Vertices are created at Lines 3 and 12. As v is an atom vertex, it must be created at Line 3. Since it is a leaf vertex, condition at Line 7 never holds. But, then condition at Line 10 holds. Then, the call where v is created returns an empty set of vertices. That is, v cannot be in V.

□

Lemma 3. *Let Π be a ground ASP program and X be an answer set for Π. For the two subsequent calls createTree(Π, X, d, L) and createTree(Π, X, d', L') on a path in the recursion tree for some execution of Algorithm 3, the following hold*

(i) *If d is an atom, then $L' = L \cup \{d\}$;*

(ii) *If d is a rule, then $L' = L$.*

Proof of Lemma 3. Let createTree(Π, X, d, L) and createTree(Π, X, d', L') be two subsequent calls on a path in the recursion tree for some execution of Algorithm 3. We show one by one that the conditions in the lemma hold.

(i) Assume that d is an atom. Then, createTree(Π, X, d', L') is called at Line 6. Due to Line 4, $L' = L \cup \{d\}$.

(ii) Assume that d is a rule. Then, createTree(Π, X, d', L') is called at Line 13. As L is not modified prior to this call, $L' = L$.

□

Lemma 4. *Let Π be a ground ASP program, X be an answer set for Π and $\langle V, E, l, \Pi, X\rangle$ be the vertex-labeled tree returned by execution Exc of Algorithm 3. Let createTree(Π, X, d, L) and createTree(Π, X, d', L') be the two subsequent calls on a path in the recursion tree for Exc, where each call on the path returns a nonempty vertex-labeled tree. Let v and v' be two vertices created by createTree(Π, X, d, L) and createTree(Π, X, d', L'), respectively, such that $l(v) = d$ and $l(v') = d'$. Then, $(v, v') \in E$.*

Proof of Lemma 4. Notice that either d is an atom and d' is a rule or vice versa. We show that the lemma holds for the former case. The latter case can be shown similarly. Assume that d is an atom and d' is a rule. Then, createTree(Π, X, d', L') must be called at Line 6 within createTree(Π, X, d, L). As none of the calls on the path returns empty vertex-labeled tree, the condition at Line 7 holds in createTree(Π, X, d, L). Then, an edge (v, v') is added to E at Line 9. Note that v is created in createTree(Π, X, d, L) at Line 3 and $l(v) = d$. Also, due to Line 8, v' is the root of the vertex-labeled tree returned by createTree(Π, X, d', L'). Then, by Lemma 2, v' is a vertex created in createTree(Π, X, d', L') and $l(v') = d'$.

□

Lemma 5. *Let Π be a ground ASP program, X be an answer set for Π, d be an atom in X and $T = \langle V, E, l, \Pi, X\rangle$ be the vertex-labeled tree returned by createTree$(\Pi, X, d, \emptyset)$. If T is not empty, then for every node createTree(Π, X, d', L') in the recursion tree for createTree$(\Pi, X, d, \emptyset)$, where createTree(Π, X, d', L') and its ancestors return nonempty vertex-labeled trees, $L' = anc_T(v)$ where $l(v) = d'$.*

Proof of Lemma 5. Assume that T is not empty. Then, we prove the lemma by induction on the depth of a node in the recursion tree for createTree$(\Pi, X, d, \emptyset)$.

Base case: Note that the node at depth 0 in the recursion tree for createTree$(\Pi, X, d, \emptyset)$ returns a nonempty vertex-labeled tree and it has no ancestors. By Lemma 2, the root v of $\langle V, E\rangle$ is mapped to d by l, i.e., $l(v) = d = d'$. As the root of a tree does not have any ancestors, $anc_T(v) = \emptyset = L'$.

Induction step: As an induction hypothesis, assume that for every node createTree(Π, X, d', L') at depth less than n in the recursion tree for createTree$(\Pi, X, d, \emptyset)$, where createTree(Π, X, d', L') and its ancestors return nonempty

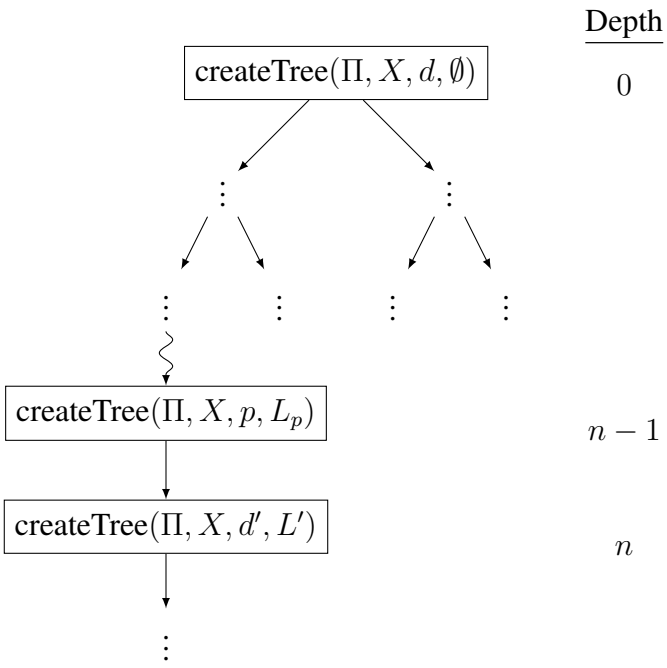


Figure 6.1: Part of the recursion tree for createTree$(\Pi, X, d, \emptyset)$.

vertex-labeled trees, $L' = \mathit{anc}_T(v)$ where $l(v) = d'$. Let createTree(Π, X, d', L') be a node at depth n in the recursion tree for createTree$(\Pi, X, d, \emptyset)$, where createTree(Π, X, d', L') and its ancestors return nonempty vertex-labeled trees. We show that $L' = \mathit{anc}_T(v)$ where $l(v) = d'$. For that, we need to consider two cases: d' is an atom and d' is a rule. Let createTree(Π, X, p, L_p) be the parent of createTree(Π, X, d', L'), as illustrated in Figure 6.1.

Case (1) Assume that d is an atom. Then, p must be a rule. By Lemma 3, $L' = L_p$. Notice that the depth of the parent node is $n - 1$. Then, by the induction hypothesis, $L_p = \mathit{anc}_T(v)$ where $l(v) = p$. Also, by Lemma 4, $(v, v') \in E$ where $l(v') = d'$. Since v is a rule vertex, $\mathit{anc}_T(v) = \mathit{anc}_T(v')$. Thus, $L' = \mathit{anc}_T(v')$ where $l(v') = d'$.

Case (2) Assume that d is a rule. Then, p must be an atom. By Lemma 3, $L' = L_p \cup \{p\}$. Notice that the depth of the parent node is $n - 1$. Then, by the induction hypothesis, $L_p = \mathit{anc}_T(v)$ where $l(v) = p$. Also, by Lemma 4, $(v, v') \in E$ where $l(v') = d'$. Since v is an atom vertex, $\mathit{anc}_T(v) \cup \{l(v)\} = \mathit{anc}_T(v')$. Thus, $L' = \mathit{anc}_T(v')$ where $l(v') = d'$.

□

Proposition 12. (Soundness of Algorithm 3) *Let Π be a ground ASP program, X be an answer set for Π, d be an atom in X and $T = \langle V, E, l, \Pi, X\rangle$ be the vertex-labeled tree returned by createTree$(\Pi, X, d, \emptyset)$. If T is not empty, then T is the and-or explanation tree for d with respect to Π and X.*

Proof of Proposition 12. Suppose that T is not empty. We want to show that T is the and-or explanation tree for d with respect to Π and X. For that, T must satisfy conditions $(i) - (iv)$ in Definition 9. As T is not empty, conditions $(i), (iii)$ and (iv) hold due to Lemma 2. To complete the proof, we show condition (ii) also holds in the sequel.

Let $S_v = \{(v, v') \mid (v, v') \in E,\, l(v') \in \Pi_{X, anc_T(v')}(l(v))\}$. Our goal is to show that for every atom vertex $v \in V$, $\mathit{out}_E(v) = S_v$. To do so, we show that $\mathit{out}_E(v) \subseteq S_v$ and that $S_v \subseteq \mathit{out}_E(v)$.

Let v be an atom vertex in V. Now, we show that $\mathit{out}_E(v) \subseteq S$. Take an arbitrary element $(v, v') \in \mathit{out}_E(v)$. Then, $(v, v') \in E$. Throughout the algorithm edges are created at Lines 9 and 17. Since v is an atom vertex, (v, v') must be created at Line 9. Then, due to the condition at Line 5 and Lemma 5, $l(v') \in \Pi_{X, anc_E(v')}(l(v))$. Thus, $(v, v') \in S_v$. As the last step, we show that $S_v \subseteq \mathit{out}_E(v)$. Let $(v, v') \in S_v$ be an arbitrary element. Then, by the definition of S_v, $(v, v') \in E$. Thus, trivially, $(v, v') \in \mathit{out}_E(v)$.

□

Lemma 6. *Let Π be a ground ASP program, X be an answer set for Π, d be an atom in X, $T = \langle V_T, E_T, l, \Pi, X\rangle$ be the and-or explanation tree for d with respect to Π and X, v be the root of T and $\langle V, E, l, \Pi, X\rangle$ be the vertex-labeled tree returned by extractExp$(\Pi, X, V_T, l, v, E_T, W_T, \emptyset, \mathit{min})$. Then, for each $v' \in V$, we have*

$$W_T(v') \leq \mathit{min}\{W_T(s) \mid s \in \mathit{sibling}_{E_T}(v')\}.$$

Proof of Lemma 6. Let $v' \in V$. Vertices are added to V at Line 8. Then, due to Line 7, v' is a rule vertex. Also, each vertex added to V corresponds to the 5^{th} parameter of the algorithm. Note that recursive calls are made at Lines 5 and 10. Since the 5^{th} parameter is a rule vertex only in the call at Line 5, the call where v' is added to V must be initiated at Line 5. Then, due to Line 3, $W_T(v') = \mathit{min}\{W_T(s) \mid s \in \mathit{sibling}_{E_T}(v')\}$.

□

Lemma 7. *Let Π be a ground ASP program, X be an answer set for Π, d be an atom in X, $T = \langle V_T, E_T, l, \Pi, X\rangle$ be the and-or explanation tree for d with respect to Π and X, v be the root of T, and $\langle V, E, l, \Pi, X\rangle$ be the vertex-labeled tree returned by extractExp$(\Pi, X, V_T, l, v, E_T, W_T, \emptyset, \mathit{min})$. Let $v_1, v_2 \in V$. If $(v_1, v), (v, v_2) \in E_T$ for some $v \in V_T$, then $(v_1, v_2) \in E$.*

Proof of Lemma 7. Since $v_1 \in V$, it is added to V at Line 8. Then, for each child of v_1, the algorithm is recursively called at Line 10. As $(v_1, v) \in E_T$, we make a call extractExp$(\Pi, X, V_T, l, v, E_T, W_T, v_1, \mathit{min})$. Inside that call, we add (v_1, c) to E (at Line 4) where c is a minimum weighted child of v (due to Line 3). As $(v, v_2) \in E_T$ and v_2 is a minimum weighted child of v due to Lemma 6, c is equal to v_2. Thus, $(v_1, v_2) \in E$.

□

Lemma 8. *Let Π be a ground ASP program, X be an answer set for Π, d be an atom in X, $T = \langle V_T, E_T, l, \Pi, X\rangle$ be the and-or explanation tree for d with respect to Π and X, and op be a string min. Then, Algorithm 5 returns an explanation $\langle V, E, l, \Pi, X\rangle$ for d with respect to Π and X.*

Proof of Lemma 8. Let v be the root of T and $S = \langle V, E, l, \Pi, X\rangle$ be the output of extractExp($\Pi, X, V_T, l, v, E_T, W_T, \emptyset$, *min*). To prove that S is an explanation for d with respect to Π and X, we need to show that there exists an explanation tree $\langle V', E', l, \Pi, X\rangle$ in T which satisfies Conditions (i) and (ii) in Definition 11. That is, the following hold.

(i) $V = \{v \mid v \text{ is a rule vertex in } V'\}$;

(ii) $E = \{(v_1, v_2) \mid (v_1, v), (v, v_2) \in E' \text{ for some atom vertex } v \in V'\}$.

To do so, we construct a vertex-labeled tree and show that it is an explanation tree in T which satisfies above conditions. Thus, let us define a vertex-labeled tree $T' = \langle V', E', l, \Pi, X\rangle$ where

$$V' = V \cup \{v \mid v \in V_T \text{ s.t. } (v, v') \in E_T \text{ for some } v' \in V\} \tag{6.1}$$

$$E' = \{(v, v') \mid v, v' \in V' \text{ s.t. } (v, v') \in E_T\} \tag{6.2}$$

We now show that T' is an explanation tree in T, i.e., T' satisfies Conditions (i)–(iv) in Definition 10.

(i) Due to Line 8 and that $v \in V_T$, $V \subseteq V_T$. So, by (6.1), $V' \subseteq V_T$. Also, by (6.2), $E' \subseteq E_T$.

(ii) In the first call of Algorithm 5, v is the root of T. Then, at Line 5, the algorithm is called with a child c of v. Note that c is a rule vertex. Due to Line 8, for some $v' \in V$, $(v, v') \in E_T$.

(iii) Let v' be an atom vertex in V'. Then, by (6.1) and (6.2), $(v', v'') \in E'$ for some $v'' \in V$. This ensures that $deg'_E(v') \geq 1$. Assume that $deg'_E(v') > 1$. Then, for some $v''' \in V'$ ($v'' \neq v'''$), $(v', v''') \in E'$. Then, $v''' \in V$. This is not possible due to Line 3. Thus, $deg_E(v') = 1$.

(iv) Let v' be a rule vertex in V'. Then, by (6.1) $v' \in V$ and v is added into V at Line 8. Due to Line 9 and (6.1), every child c of v is in V'. Then, by (6.2), $(v, c) \in E'$. That is, $out_{E_T}(v) \subseteq E'$.

As a last step, we show that T' satisfies Conditions (i) and (ii) in Definition 11.

(i) Note that every element in the set $\{v \mid v \in V_T \text{ s.t. } (v, v') \in E_T \text{ for some } v' \in V\}$ is an atom vertex. Then, by (6.1), $V = \{v \mid v \text{ is a rule vertex in } V'\}$;

(ii) Let $S = \{(v_1, v_2) \mid (v_1, v), (v, v_2) \in E' \text{ for some atom vertex } v \in V'\}$. We show that $E = S$.

Let $(v_1, v_2) \in E$. Then, (v_1, v_2) is added into E at Line 4. So, due to Lines 2 and 3, we know that there exists an atom vertex $v \in V_T$ such that $(v, v_2) \in E_T$. Then, by (6.1), $v \in V'$ and, by (6.2), $(v, v_2) \in E'$. Also, by Line 9, we know that $(v_1, v) \in E_T$. Then, by (6.2), $(v_1, v) \in E'$. Thus, $(v_1, v_2) \in S$. That is, $E \subseteq S$.

Let $(v_1, v_2) \in S$. Then, for some atom vertex $v \in V'$, $(v_1, v), (v, v_2) \in E'$. By (6.2), $v_1, v_2 \in V$ and $(v_1, v), (v, v_2) \in E_T$. Then, due to Lemma 7, $(v_1, v_2) \in E$. That is, $S \subseteq E$.

□

Lemma 9. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, $T = \langle V, E, l, \Pi, X\rangle$ be the and-or explanation tree for p with respect to Π and X, $T' = \langle V', E', l, \Pi, X\rangle$ be an explanation tree in T and v be a rule vertex in V'. Then, the following inequality holds for the weight of v*

$$W_T(v) \leq 1 + |\{u' \mid u' \in \mathit{des}_{T'}(v)\}|. \tag{6.3}$$

Proof of Lemma 9. We prove the lemma by induction on the height of a rule vertex in the explanation tree.

Base case: Let u be a rule vertex in V' at height 0. Then, u is a leaf vertex. By the definition of the weight function, $W_T(u) = 1$. Then, (6.3) holds.

Induction step: As an induction hypothesis, assume that for every rule vertex $i \in V'$ at height less than n, (6.3) holds. We show that (6.3) holds for every rule vertex $w \in V'$ at height $n + 1$. Let w be a rule vertex at height $n + 1$. Then, by the definition of the weight function, $W_T(w) = 1 + \sum_{w' \in \mathit{child}_E(w)} W_T(w')$. Let w' be a child of w. Note that w' is an atom vertex. By the definition of an explanation tree, $w' \in V'$ and w' has exactly one child $w'' \in V'$ which is a rule vertex. Then, by the definition of the weight function, $W_T(w') = \mathit{min}\{W_T(c) \mid c \in \mathit{child}_E(w')\}$ and thus $W_T(w') \leq W_T(w'')$. On the other hand, as the height of w'' is $n - 1$, by the induction hypothesis, $W_T(w'') \leq 1 + |\{d \mid d \in \mathit{des}_{T'}(w'')\}|$. Since w' has exactly one child $w'' \in V'$, we have

$$1 + |\{d \mid d \in \mathit{des}_{T'}(w'')\}| = |\{u \mid u \in \mathit{des}_{T'}(w')\}|.$$

That is, $W_T(w') \leq |\{u \mid u \in \mathit{des}_{T'}(w')\}|$. Then, we have

$$\begin{aligned} W_T(w) = & \quad 1 + \textstyle\sum_{w' \in \mathit{child}_E(w)} W_T(w') \\ \leq & \quad 1 + \textstyle\sum_{w' \in \mathit{child}_E(w)} |\{u \mid u \in \mathit{des}_{T'}(w')\}| \\ = & \quad 1 + |\{u' \mid u' \in \mathit{des}_{T'}(w)\}|. \end{aligned}$$

□

Lemma 10. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, T be the and-or explanation tree (with edges E) for p with respect to Π and X, v be the root of T, and T' be an explanation tree (with vertices V') in T. Then,*

$$W_T(v) \leq |\{u \mid u \textit{ is a rule vertex in } V'\}|.$$

Proof of Lemma 10. We want to show that the weight of v, $W_T(v)$, is at most the number of rule vertices in V'. Note that v is the root of T' and there exists exactly one vertex $v' \in V'$ such that $v' \in \mathit{child}_E(v)$ (due to Definition 10). Then, we have

$$\begin{aligned} W_T(v) = & \quad \mathit{min}\{W_T(c) \mid c \in \mathit{child}_E(v)\} && \text{(by Definition 13)} \\ \leq & \quad W_T(v') && \text{(as } v' \in \mathit{child}_E(v)\text{)} \\ \leq & \quad 1 + |\{v'' \mid v'' \in \mathit{des}_{T'}(v')\}| && \text{(by Lemma 9)} \\ = & \quad |\{u \mid u \text{ is a rule vertex in } V'\}|. && \text{(as } v' \text{ is the only child of } v \text{ in } T'\text{)} \end{aligned}$$

□

Lemma 11. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, T be the and-or explanation tree for p with respect to Π and X, v be the root of T, and $\langle V, E, l, \Pi, X\rangle$ be an explanation for p with respect to Π and X. Then, $W_T(v) \leq |V|$.*

Proof of Lemma 11. We show that the weight of v, $W_T(v)$, is at most $|V|$. By the definition of an explanation, there exists an explanation tree T' (with vertices V') of T such that $|V| = |\{v' \mid v' \text{ is a rule vertex in } V'\}|$. By Lemma 10, $W_T(v) \leq |\{v' \mid v' \text{ is a rule vertex in } V'\}|$. Thus, $W_T(v) \leq |V|$.

□

Lemma 12. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, $T = \langle V, E, l, \Pi, X\rangle$ be the and-or explanation tree for p with respect to Π and X, and $T' = \langle V', E', l, \Pi, X\rangle$ be an explanation tree in T. If we have*

$$W_T(v) \leq \mathit{min}\{W_T(s) \mid s \in \mathit{sibling}_E(v)\}. \tag{6.4}$$

for every rule vertex $v \in V'$, then the following holds for every rule vertex $v \in V'$.

$$W_T(v) = 1 + |\{u' \mid u' \in \mathit{des}_{T'}(v)\}|. \tag{6.5}$$

Proof of Lemma 12. Assume that (6.4) holds for every rule vertex $v \in V'$. Then, we prove the lemma by induction on the height of a rule vertex in the explanation tree.

Base case: Let u be a rule vertex in V' at height 0. Then, u is a leaf vertex. By the definition of the weight function, $W_T(u) = 1$. As u has no descendants, (6.5) holds.

Induction step: As an induction hypothesis, assume that for every rule vertex $i \in V'$ at height less than n, (6.5) holds. We show that (6.5) holds for every rule vertex $w \in V'$ at height $n+1$. Let $w \in V'$ be a rule vertex at height $n+1$. Then, by the definition of the weight function, $W_T(w) = 1 + \sum_{w' \in \mathit{child}_E(w)} W_T(w')$. Let w' be a child of w. Note that w' is an atom vertex. By the definition of an explanation tree, $w' \in V'$ and w' has exactly one child $w'' \in V'$, which is a rule vertex. Then, by the definition of the weight function, $W_T(w') = \mathit{min}\{W_T(c) \mid c \in \mathit{child}_E(w')\}$. Also, by (6.4), $W_T(w'') \leq \mathit{min}\{W_T(s) \mid s \in \mathit{sibling}_E(w'')\}$. Then, $W_T(w'') = \mathit{min}\{W_T(c) \mid c \in \mathit{child}_E(w')\}$. Thus, $W_T(w') = W_T(w'')$. As the height of w'' is $n-1$, by the induction hypothesis, $W_T(w'') = 1 + |\{u \mid u \in \mathit{des}_{T'}(w'')\}|$. As w'' is the only child of w', $W_T(w') = |\{u \mid u \in \mathit{des}_{T'}(w')\}|$. Then, we have

$$\begin{aligned} W_T(w) = & \quad 1 + \textstyle\sum_{w' \in \mathit{child}_E(w)} W_T(w') \\ = & \quad 1 + \textstyle\sum_{w' \in \mathit{child}_E(w)} |\{u \mid u \in \mathit{des}_{T'}(w')\}| \\ = & \quad 1 + |\{u' \mid u' \in \mathit{des}_{T'}(w)\}|. \end{aligned}$$

□

Lemma 13. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, T be the and-or explanation tree (with edges E) for p with respect to Π and X, v be the root of T, and T' be an explanation tree (with vertices V') in T. If we have*

$$W_T(v') \leq \mathit{min}\{W_T(s) \mid s \in \mathit{sibling}_E(v')\} \tag{6.6}$$

for every rule vertex $v' \in V'$, then, the following holds

$$W_T(v) = |\{u \mid u \text{ is a rule vertex in } V'\}|.$$

Proof of Lemma 13. Assume that (6.4) holds for every rule vertex $v' \in V'$. Then, we want to show that the weight of v, $W_T(v)$, is equal to the number of rule vertices in V'. Note that v is the root of T' and there exists exactly one vertex $v' \in V'$ such that $v' \in \mathit{child}_E(v)$ (due to Definition 10). Then, we have

$$\begin{aligned} W_T(v) = & \quad \mathit{min}\{W_T(c) \mid c \in \mathit{child}_E(v)\} && \text{(by Definition 13)} \\ = & \quad W_T(v') && \text{(by (6.6))} \\ = & \quad 1 + |\{u' \mid u' \in \mathit{des}_{T'}(v')\}| && \text{(by Lemma 12)} \\ = & \quad |\{u \mid u \text{ is a rule vertex in } V'\}|. && \text{(as } v' \text{ is the only child of } v \text{ in } T') \end{aligned}$$

□

We are now ready to prove Proposition 3.

Proposition 3. *Given a ground ASP program Π, an answer set X for Π, and an atom p in X, Algorithm 2 either finds a shortest explanation for p with respect to Π and X or returns an empty vertex-labeled tree.*

Proof of Proposition 3. Algorithm 2 has two return statements; Lines 6 and 8. At Line 8, it returns an empty vertex-labeled tree. We show that what Algorithm 2 returns at Line 6, $S = \langle V', E', l, \Pi, X \rangle$, is an explanation for p with respect to Π and X and that there is no other explanation for p with respect to Π and X, with vertices V'', such that $|V''| < |V'|$.

Due to the condition at Line 2, the vertex-labeled tree found at Line 1 is not empty. Then, by Proposition 12, we know that T is the and-or explanation tree for p with respect to Π and X. Then, by Lemma 8, we know that $\langle V', E', l, \Pi, X \rangle$ found at Line 5 is an explanation for p with respect to Π and X.

Now, suppose that there exists another explanation for p with respect to Π and X, with vertices V'', such that $|V''| < |V'|$. Let v be the root of T. Then, by Lemma 11, $W_T(v) \leq |V''|$. Also, since S is an explanation for p with respect to Π and X, there exists an explanation tree with vertices V''' in T such that $V' = \{u \mid u \text{ is a rule vertex in } V'''\}$. Due to Lemma 6, $W_T(v') \leq \mathit{min}\{W_T(s) \mid s \in \mathit{sibling}_E(v')\}$ for each $v' \in V'$. Then, by Lemma 13, $W_T(v) = |\{u \mid u \text{ is a rule vertex in } V'''\}|$. This implies that $|V'| \leq |V''|$. Since this is a contradiction, S is a shortest explanation for p with respect to Π and X.

□

6.1.3 Proof of Proposition 4 – Complexity of Algorithm 2

We prove Proposition 4 which shows that the time complexity of Algorithm 2 is exponential in the size of the given answer set.

Proposition 4. *Given a ground ASP program Π, an answer set X for Π, and an atom p in X, the time complexity of Algorithm 2 is $O(|\Pi|^{|X|} \times |\mathcal{B}_\Pi|)$.*

Proof of Proposition 4. In Algorithm 2, all the lines, expect 1, 4 and 5, take constant amount of time. At Lines 1, 4 and 5, three different algorithms are called. At Line 1, Algorithm 3 is called. This algorithm creates the and-or explanation tree recursively. As shown in Proof of Lemma 1, the branching factor of a vertex in the recursion tree is $O(\mathit{max}\{|X|, |\Pi|\})$ and the height of the tree is $O(|X|)$. Also, at Line 5, for an atom $d \in X$, we check whether a rule in Π supports d in $O(|\mathcal{B}_\Pi|)$. Thus, the time complexity of Algorithm 3, in the worst case, is $O(\mathit{max}\{|X|, |\Pi|\}^{|X|} \times |\mathcal{B}_\Pi|)$. As $|X| \leq |\Pi|$, it is $O(|\Pi|^{|X|} \times |\mathcal{B}_\Pi|)$. At Lines 4 and 5, Algorithm 4 and Algorithm 5 are called, respectively. Algorithm 4 and Algorithm 5 simply traverse the tree T created by Algorithm 3 (cf.

Lines 2 and 6 in Algorithm 4, and Lines 5 and 10 in Algorithm 5). By Proposition 12, we know that T is the and-or explanation tree for p with respect to Π and X. Since the height of T is $O(|X|)$ and the branching factor of a vertex in T is $max\{|X|, |\Pi|\}$, the time complexity of Algorithm 3 dominates the time complexities of others. Thus, the time complexity of Algorithm 2, in the worst case, is $O(|\Pi|^{|X|} \times |\mathcal{B}_\Pi|)$.

□

6.2 Generating k Different Explanations

In Section 4.3, we have first analyzed three properties of Algorithm 6, namely termination, soundness and complexity, resulting in Propositions 5, 6, and 8. Then, we show some characteristics of its output, resulting in Proposition 7, Corollary 1 and Corollary 2. In the following, we show the proofs of these results.

6.2.1 Proof of Proposition 5 – Termination of Algorithm 6

We now prove Proposition 5 which shows that Algorithm 6 terminates.

Proposition 5. *Given a ground ASP program Π, an answer set X for Π, an atom p in X, and a positive integer k, Algorithm 6 terminates.*

Proof of Proposition 5. Algorithm 6 calls Algorithm 3 at Line 2. By Lemma 1, we know that Algorithm 3 terminates. Then, to show that Algorithm 6 terminates, we need to show that the loop between Lines 4–10 terminates. Due to Line 4, the loop iterates at most k times. If it iterates less than k times, it means that it is terminated at Line 6. Thus, assume that it iterates k times. Then, it is enough to show that every iteration of the loop terminates. Consider the i^{th} $(1 \leq i \leq k)$ iteration of the loop. First, at Line 5, Algorithm 7 is called. Observe that this algorithm simply traverses the and-or explanation tree T created at Line 2 (cf. Lines 2, 3, 8 and 9 in Algorithm 7). Since T is finite, Algorithm 7 terminates. Next, Algorithm 5 is called at Line 7. Similar to Algorithm 7, this algorithm also simply traverses a portion of T (cf. Lines 3, 5, 9 and 10 in Algorithm 5). Hence, Algorithm 5 also terminates. As the rest of the loop consists of some assignment statements, the i^{th} $(1 \leq i \leq k)$ iteration of the loop terminates. Since every iteration of the loop terminates, Algorithm 6 terminates.

□

6.2.2 Proof of Proposition 6 – Soundness of Algorithm 6

Before showing the soundness property of Algorithm 6, we provide some necessary lemmas.

Proposition 13. (Completeness of Algorithm 3) *Let* Π *be a ground ASP program,* X *be an answer set for* Π*,* p *be an atom in* X*. Let* T *be the and-or explanation tree* p *with respect to* Π *and* X*. Then, createTree*$(\Pi, X, p, \{\})$ *returns* T*.*

Proof of Proposition 13. Let $\langle V, E, l, \Pi, X\rangle$ be the output of Algorithm 3. By Condition (i) in Definition 9, the root of T is an atom vertex with label p. Since p is in X and $L = \emptyset$ at the beginning of Algorithm 3, a vertex v with label p is created at Line 3 and added into V at Line 4.

Take an atom vertex $v \in V$. Then, there should be an out-going edge (v, v') of v such that $l(v') \in \Pi_{X,anc_T(v')}(l(v))$ (due to Condition (ii) in Definition 9). Note that, the set L in Algorithm 3 is essentially $anc_T(v')$, and $l(v') \in \Pi_{X,L}(l(v))$ is checked at Line 5. Moreover, we need to ensure that v' is a rule vertex. This condition is checked at Line 6 by recursive calls.

Take a rule vertex $v \in V$. Then, for every atom a in $B^+(l(v))$, there should be an out-going edge (v, v') of v such that $l(v') = a$ (due to Condition (iii) in Definition 9). This is satisfied by the condition at Line 13. Moreover, we need to make sure that v' is atom vertex. For that, there is a recursive call at Line 14.

Take a leaf vertex $v \in V$. Then, v must be a rule vertex (due to Condition (iv) in Definition 9). Assume that v is an atom vertex. Thus, no out-going edge is defined for v at Line 9. Then, when the loop at Line 5 terminates, the condition at Line 10 is satisfied. This implies that v is not in V. As it is a contradiction, v must be a rule vertex.

□

By this proposition and Proposition 1, we know that Algorithm 3 returns the and-or explanation tree. Thus, at Line 1 of Algorithm 6, the and-or explanation tree for p with respect to Π and X is created. We prove our statements by keeping this in mind.

Lemma 14. *Let* Π *be a ground ASP program,* X *be an answer set for* Π*,* p *be an atom in* X*, and* k *be a positive integer. Let* n *be the number of different explanations for* p *with respect to* Π *and* X*. Then, for the root* v_r *of* T*, at each iteration* i *(*$1 \leq i \leq min\{n, k\}$*) of the loop in Algorithm 6,* $W_{T,R_{i-1}}(v_r) = |RVertices(K_i) \backslash R_{i-1}|$*.*

Proof of Lemma 14. Consider the i^{th} $(1 \leq i \leq min\{n, k\})$ iteration of the loop. At Line 5, we call Algorithm 7 and calculate $W_{T,R_{i-1}}$ for every vertex v in V. According to Algorithm 7, for an atom vertex $v \in V$, due to Line 4, $W_{T,R_{i-1}}(v) = max\{W_{T,R_{i-1}}(v') \mid v' \in child_E(v)\}$ and for a rule vertex u, due to Lines 6–9, $W_{T,R_{i-1}}(u) = x_u + \sum_{u' \in child_E(u)} W_{T,R_{i-1}}(u')$ where $x_u = 1$ if $u \notin R_{i-1}$, $x_u = 0$ otherwise. Let v be an atom vertex in V. Let $\{v'_1, \ldots, v'_{v_z}\}$ be a set of rule vertices that "contribute" to $W_{T,R_{i-1}}(v)$, i.e., rule vertices that appear in the expanded formula of $W_{T,R_{i-1}}(v)$. This implies that $W_{T,R_{i-1}} = x_{v'_1} + \ldots + x_{v'_{v_z}}$ where $x_{v'_j} = 1$ if $v'_j \notin R_{i-1}$, $x_{v'_j} = 0$ otherwise,

for $1 \leq j \leq v_z$. Also, at Line 7 in Algorithm 6, we extract an explanation using Algorithm 5. Then, at Line 8, we assign the output $\langle V', E', l, \Pi, X\rangle$ of Algorithm 5 to K_i. Let *RVertices*$(K_i) = \{v_1, \ldots, v_z\}$. Observe in Algorithm 5 that for every atom vertex, we process its maximum weighted child recursively (Line 3), and for every rule vertex we choose every child of it recursively (Line 9). Then, due to the observation and the calculation of $W_{T,R_{i-1}}$, *RVertices*(K_i) is a set of rule vertices that contribute to $W_{T,R_{i-1}}(v_r)$. Then, $W_{T,R_{i-1}}(v_r) = x_1 + \ldots + x_z$ where $x_j = 1$ if $v_j \notin R_{i-1}$, $x_j = 0$ otherwise, for $1 \leq j \leq z$. That is, $W_{T,R_{i-1}}(v_r) = |$*RVertices*$(K_i) \backslash R_{i-1}|$.

□

Now, we prove Proposition 6.

Proposition 6. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and k be a positive integer. Let n be the number of different explanations for p with respect to Π and X. Then, Algorithm 6 returns min$\{n, k\}$ different explanations for p with respect to Π and X.*

Proof of Proposition 6. First, assume that $n \geq k$. Then, we show that Algorithm 6 returns at Line 11 a set K of k different explanations for p with respect to Π and X. Note that an element is added into K at each iteration of the loop between Lines 4–10. Since that loop iterates k times, we need to show that two properties hold: **(i)** for every iteration i $(1 \leq i \leq k)$, K_i (the element added into K at the i^{th} iteration) is an explanation for p with respect to Π and X. **(ii)** for all iterations i, j $(1 \leq i < j \leq k)$, K_i and K_j are different. In the following, we consider those two properties.

(i) For every iteration i $(1 \leq i \leq k)$, K_i is formed at Line 8. According to Line 7, K_i is an output of Algorithm 5. Then, due to Lemma 8, we conclude that K_i is an explanation for p with respect to Π and X.

(ii) Assume otherwise. Then, there exists i, j $(1 \leq i < j \leq k)$ such that $K_i = K_j$. Consider the j^{th} iteration of the loop. At Line 5, we call Algorithm 7 and calculate $W_{T,R_{j-1}}$ for every vertex v in V. Then, at Line 7, we extract an explanation using Algorithm 5. Let $K_j = \langle V', E', l, \Pi, X\rangle$, v_r be the root of $\langle V', E'\rangle$ and *RVertices*$(K_j) = \{v_1, \ldots, v_l\}$. Then, by Lemma 14, $W_{T,R_{j-1}}(v_r) = x_1 + \ldots + x_l$ where $x_z = 1$ if $v_z \notin R_{j-1}$, $x_z = 0$ otherwise, for $1 \leq z \leq l$. Since $K_i = K_j$, $\{v_1, \ldots, v_l\} =$ *RVertices*(K_i). Due to Line 10, we know that $RVertices(K_i) \subseteq R_{j-1}$. Then, for $1 \leq z \leq l$, $v_z \in R_{j-1}$. Thus, $W_{T,R_{j-1}}(v_r) = 0$. This implies that for every vertex $v \in V$, $W_{T,R_{j-1}}(v) = 0$. That is, every rule vertex in V is also in R_{j-1}. However, as we are at the j^{th} iteration and at each iteration one explanation is computed, R_{j-1} might contain rule vertices for at most $j - 1$

explanations. We know that $n \geq k$. Thus, for some $v' \in V$, the following should hold: $v' \notin R_{j-1}$. As we reach a contradiction, $K_i \neq K_j$.

Now, assume that $n < k$. Then, we show that Algorithm 6 returns at Line 6 a set K of n different explanations for p with respect to Π and X. Note that at the end of n^{th} iteration of the loop, K contains n different explanations (due to the same reasoning above). Then, in the next iteration, $W_{T,R_{n+1}}(v) = 0$. This is because R_n contains the rule vertices of n different explanations and the total number of explanations for p with respect to Π and X is n. Thus, the condition at Line 6 is satisfied and n different explanations are returned.

□

6.2.3 Some Properties of Algorithm 6

Before presenting some characteristics of the output of Algorithm 6, we provide some necessary definitions and lemmas.

Definition 25 ($W_{T,R}$)**.** *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, $T = \langle V, E, l, \Pi, X\rangle$ be the and-or explanation tree for p with respect to Π and X, and R be a set of rule vertices in V. Then, $W_{T,R}$ is a function that maps vertices in V to nonnegative integers as follows.*

$$W_{T,R}(v) = \begin{cases} max\{W_{T,R}(v') \mid v' \in child_E(v)\} & \text{if } v \text{ is an atom vertex;} \\ \sum_{v' \in child_E(v)} W_{T,R}(v') & \text{if } v \text{ is a rule vertex and } v \in R; \\ 1 + \sum_{v' \in child_E(v)} W_{T,R}(v') & \text{otherwise.} \end{cases}$$

Lemma 15. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, $T = \langle V, E, l, \Pi, X\rangle$ be the and-or explanation tree for p with respect to Π and X, $T' = \langle V', E', l, \Pi, X\rangle$ be an explanation tree in T and R be a set of rule vertices in V. Then, for every rule vertex v in V', the following holds*

$$W_{T,R}(v) \geq \begin{cases} 1 + |\{u' \mid u' \in (des_{T'}(v) \backslash R)\}| & \text{if } v \notin R; \\ |\{u' \mid u' \in (des_{T'}(v) \backslash R)\}| & \text{otherwise.} \end{cases} \tag{6.7}$$

Proof of Lemma 15. We prove the lemma by induction on the height of a rule vertex in the explanation tree.

Base case: Let v be a rule vertex in V' at height 0. Then, v is a leaf vertex. By Definition 25 , $W_{T,R}(v) = x_v$ where $x_v = 1$ if $v \notin R$, $x_v = 0$ otherwise. Then, (6.7) holds.

Induction step: As an induction hypothesis, assume that for every rule vertex $i \in V'$ at height less than $n+1$, (6.7) holds. We show that (6.7) holds for every rule vertex $v \in V'$ at height $n+1$. Let v be a rule vertex in V' at height $n+1$. Then, by Definition 25, $W_{T,R}(v) = x_v + \sum_{v' \in \mathit{child}_E(v)} W_{T,R}(v')$ where $x_v = 1$ if $v \notin R$, $x_v = 0$ otherwise. Let v' be a child of v. Note that v' is an atom vertex. By Conditions (iii) and (iv) in Definition 10, $v' \in V'$ and v' has exactly one child $v'' \in V'$ which is a rule vertex. Then, by Definition 25, $W_{T,R}(v') = \mathit{max}\{W_{T,R}(c) \mid c \in \mathit{child}_E(v')\}$ and thus $W_{T,R}(v') \geq W_{T,R}(v'')$. On the other hand, as the height of v'' is $n-1$, by the induction hypothesis, (6.7) holds for v''. Thus, we obtain the following.

$$W_{T,R}(v') \geq \begin{cases} 1 + |\{u' \mid u' \in (\mathit{des}_{T'}(v'') \backslash R)\}| & \text{if } v'' \notin R; \\ |\{u' \mid u' \in (\mathit{des}_{T'}(v'') \backslash R)\}| & \text{otherwise.} \end{cases} \tag{6.8}$$

Since v' has exactly one child $v'' \in V'$, we derive

$$\{v''\} \cup \{d \mid d \in \mathit{des}_{T'}(v'')\} = |\{u \mid u \in \mathit{des}_{T'}(v')\}|. \tag{6.9}$$

Then, due to (6.8) and (6.9),

$$W_{T,R}(v') \geq |\{u \mid u \in (\mathit{des}_{T'}(v') \backslash R)\}|. \tag{6.10}$$

Then, to conclude the proof, we consider two cases.

Case (1) Suppose that $v \notin R$. Then, we can derive the following.

$$\begin{aligned} W_{T,R}(v) = & \quad 1 + \textstyle\sum_{v' \in \mathit{child}_E(v)} W_{T,R}(v') && \text{(by Definition 25)} \\ \geq & \quad 1 + \textstyle\sum_{v' \in \mathit{child}_E(v)} |\{u \mid u \in (\mathit{des}_{T'}(v') \backslash R)\}| && \text{(by (6.10))} \\ = & \quad 1 + |\{u' \mid u' \in (\mathit{des}_{T'}(v) \backslash R)\}|. && \text{(as every child of } v \text{ is an atom vertex)} \end{aligned}$$

Case (2) Suppose that $v \in R$. Then, we can derive the following.

$$\begin{aligned} W_{T,R}(v) = & \quad \textstyle\sum_{v' \in \mathit{child}_E(v)} W_{T,R}(v') && \text{(by Definition 25)} \\ \geq & \quad \textstyle\sum_{v' \in \mathit{child}_E(v)} |\{u \mid u \in (\mathit{des}_{T'}(v') \backslash R)\}| && \text{(by (6.10))} \\ = & \quad |\{u' \mid u' \in (\mathit{des}_{T'}(v) \backslash R\}|. && \text{(as every child of } v \text{ is an atom vertex)} \end{aligned}$$

□

Lemma 16. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, $T = \langle V, E, l, \Pi, X \rangle$ be the and-or explanation tree for p with respect to Π and X, v be the root of T, T' be an explanation tree (with vertices V') in T, and R be a set of rule vertices in V. Then,*

$$W_{T,R}(v) \geq |\{u \mid u \in (\mathit{RVertices}(T') \backslash R)\}|.$$

Proof of Lemma 16. We want to show that $W_{T,R}(v)$ is equal to at least the number of rule vertices in V' but R. Recall that v is the root of T' and there exists exactly one vertex $v' \in V'$ such that $v' \in \mathit{child}_E(v)$ (due to Condition (iii) in Definition 10). Then, we consider two cases.

Case (1) Assume that $v' \notin R$. Then, we can derive the following.

$$\begin{aligned} W_{T,R}(v) = &\ \mathit{max}\{W_{T,R}(c) \mid c \in \mathit{child}_E(v)\} && \text{(by Definition 25)} \\ \geq &\ W_{T,R}(v') && \text{(as } v' \in \mathit{child}_E(v)\text{)} \\ \geq &\ 1 + |\{v'' \mid v'' \in (\mathit{des}_{T'}(v') \backslash R)\}| && \text{(by Lemma 15)} \\ = &\ |\{u \mid u \in (\mathit{RVertices}(T') \backslash R)\}|. && \text{(as } v' \text{ is the only child of } v \text{ in } T'\text{)} \end{aligned}$$

Case (2) Assume that $v' \in R$. Then, we can derive the following.

$$\begin{aligned} W_{T,R}(v) = &\ \mathit{max}\{W_{T,R}(c) \mid c \in \mathit{child}_E(v)\} && \text{(by Definition 25)} \\ \geq &\ W_{T,R}(v') && \text{(as } v' \in \mathit{child}_E(v)\text{)} \\ \geq &\ |\{v'' \mid v'' \in (\mathit{des}_{T'}(v') \backslash R)\}| && \text{(by Lemma 15)} \\ = &\ |\{u \mid u \in (\mathit{RVertices}(T') \backslash R)\}|. && \text{(as } v' \text{ is the only child of } v \text{ in } T'\text{)} \end{aligned}$$

□

Lemma 17. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, T be the and-or explanation tree (with vertices V) for p with respect to Π and X, v be the root of T, $T' = \langle V', E', l, \Pi, X \rangle$ be an explanation for p with respect to Π and X, and R be a set of rule vertices in V. Then, $W_{T,R}(v) \geq |\mathit{RVertices}(T') \backslash R|$.*

Proof of Lemma 17. We show that $W_{T,R}(v)$ is equal to at least $|\mathit{RVertices}(T') \backslash R|$. By Definition 11, there exists an explanation tree T'' (with vertices V'') in T such that $V' = \{v' \mid v' \text{ is a rule vertex in } V''\}$. That is, $\mathit{RVertices}(T') = \mathit{RVertices}(T'')$. By Lemma 16, we know that $W_{T,R}(v) \geq |\{u \mid u \in (\mathit{RVertices}(T'') \backslash R)\}|$. Thus, we conclude that $W_{T,R}(v) \geq |\mathit{RVertices}(T') \backslash R|$.

□

We can now prove our main result which simply indicates that at each iteration i of the loop in Algorithm 6 the distance $\Delta_D(R_{i-1}, K_i)$ is maximized.

Proposition 7. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and k be a positive integer. Let n be the number of explanations for p with respect to Π and X. Then, at the end of each iteration i ($1 \leq i \leq \mathit{min}\{n, k\}$) of the loop in Algorithm 6, $\Delta_D(R_{i-1}, \mathit{RVertices}(K_i))$ is maximized, i.e., there is no other explanation K' such that $\Delta_D(R_{i-1}, \mathit{RVertices}(K_i)) < \Delta_D(R_{i-1}, \mathit{RVertices}(K'))$.*

Proof of Proposition 7. The proof is by contradiction. Assume that there exists an explanation K' such that $\Delta_D(R_{i-1}, \mathit{RVertices}(K_i)) < \Delta_D(R_{i-1}, \mathit{RVertices}(K'))$. That is, $|\mathit{RVertices}(K') \backslash R_{i-1}| > |\mathit{RVertices}(K_i) \backslash R_{i-1}|$. Let v_r be the root of T. Then, by

Lemma 17, $W_{T,R_{i-1}}(v_r) \geq |\mathit{RVertices}(K')\backslash R_{i-1}|$. Also, by Lemma 14, we know that $W_{T,R_{i-1}}(v_r) = |\mathit{RVertices}(K_i)\backslash R_{i-1}|$. Therefore, we obtain that $|\mathit{RVertices}(K_i)\backslash R_{i-1}| \geq |\mathit{RVertices}(K')\backslash R_{i-1}|$. But, that contradicts the assumption $|\mathit{RVertices}(K')\backslash R_{i-1}| > |\mathit{RVertices}(K_i)\backslash R_{i-1}|$. Thus, there is no K' such that $\Delta_D(R_{i-1}, \mathit{RVertices}(K_i)) < \Delta_D(R_{i-1}, \mathit{RVertices}(K'))$, i.e., $\Delta_D(R_{i-1}, \mathit{RVertices}(K_i))$ is maximized.

□

Now, we provide the proof of the corollary that shows how to compute longest explanations.

Corollary 1. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and $k = 1$. Then, Algorithm 6 computes a longest explanation for p with respect to Π and X.*

Proof of Corollary 1. Since $k = 1$, the loop in Algorithm 6 iterates once. At that iteration, by Proposition 6, we know that an explanation K_1 for p with respect to Π and X is computed. By Proposition 7, we also know that $\Delta_D(R_0, \mathit{RVertices}(K_1))$ is maximized. That is, there exists no explanation K' for p with respect to Π and X such that $|\mathit{RVertices}(K')\backslash R_0| > |\mathit{RVertices}(K_1)\backslash R_0|$. Note that R_0 is an empty set. Therefore, there exists no explanation K' for p with respect to Π and X such that $|\mathit{RVertices}(K')| > |\mathit{RVertices}(K_1)|$. As every vertex of an explanation is a rule vertex, we conclude that K_1 is a longest explanation for p with respect to Π and X.

□

Next, we indicate the proof of the corollary that shows Algorithm 6 computes $\mathit{min}\{n, k\}$ different explanations such that for every i ($1 \leq i \leq \mathit{min}\{n, k\}$) the i^{th} explanation is the most distant explanation from the previously computed $i - 1$ explanations.

Corollary 2. *Let Π be a ground ASP program, X be an answer set for Π, p be an atom in X, and k be a positive integer. Let n be the number of explanations for p with respect to Π and X. Then, Algorithm 6 computes $\mathit{min}\{n, k\}$ different explanations $K_1, \ldots, K_{\mathit{min}\{n,k\}}$ for p with respect to Π and X such that for every j ($2 \leq j \leq \mathit{min}\{n, k\}$) $\Delta_D(\bigcup_{z=1}^{j-1} \mathit{RVertices}(K_z), K_j)$ is maximized.*

Proof of Corollary 2. By Proposition 6, we know that $K_1, \ldots, K_{\mathit{min}\{n,k\}}$ are $\mathit{min}\{n, k\}$ different explanations for p with respect to Π and X. Also, by Proposition 7, for every i ($2 \leq i \leq \mathit{min}\{n, k\}$), we obtain that $\Delta_D(R_{i-1}, K_i)$ is maximized. Due to Lines 1 and 10 of Algorithm 6, $R_{i-1} = \bigcup_{j=0}^{i-1} \mathit{RVertices}(K_j)$. Since R_0 is an empty set, we conclude that $\Delta_D(\bigcup_{j=1}^{i-1} \mathit{RVertices}(K_j), K_i)$ is maximized.

□

6.2.4 Proof of Proposition 8 – Complexity of Algorithm 6

We prove Proposition 8 which shows that the time complexity of Algorithm 6 is exponential in the size of the given answer set.

Proposition 8. *Given a ground ASP program Π, an answer set X for Π, an atom p in X and a positive integer k, the time complexity of Algorithm 6 is $O(k \times |\Pi|^{|X|+1} \times |\mathcal{B}_\Pi|)$.*

Proof of Proposition 8. Algorithm 6 calls Algorithm 3 at Line 2. In the proof of Proposition 4, it is shown that the worst case time complexity of Algorithm 3 is $O(|\Pi|^{|X|} \times \mathcal{B}_\Pi)$. Moreover, Algorithm 6 has a loop between Lines 4–10, which iterates at most k times. In every iteration of the loop, Algorithm 7 and Algorithm 5 are called at Lines 5 and 7. Algorithm 7 simply traverses the and-or explanation tree T recursively (cf. Lines $2, 3, 8$ and 9 in Algorithm 7). The height of T is $O(|X|)$ and the branching factor of a vertex in T is $O(|\Pi|)$. Also, at Line 8 in Algorithm 7, we check whether a rule vertex in V is in R. As R is a subset of the rule vertices in V, this check can be done in $O(|\Pi| \times |\mathcal{B}_\Pi|)$ time. Thus, the time complexity of Algorithm 7, in the worst case, is $O(|\Pi|^{|X|} \times |\Pi| \times |\mathcal{B}_\Pi|)$. Similar to Algorithm 7, Algorithm 5 just traverses a portion of T (cf. Lines $3, 5, 9$ and 10 in Algorithm 5). Thus, the time complexity of every iteration of the loop in Algorithm 6 is $O(|\Pi|^{|X|+1} \times |\mathcal{B}_\Pi|)$. As the loop iterates at most k times, the time complexity of Algorithm 6, in the worst case, is $O(k \times |\Pi|^{|X|+1} \times |\mathcal{B}_\Pi|)$.

□

6.3 Relations between Explanations and Justifications

In Chapter 5, we have related explanations to justifications, resulting in Propositions 10 and 11. In the following, we show the proofs of these results.

6.3.1 Proof of Proposition 10 – Soundness of Algorithm 8

In the proof of Proposition 10, the idea is to show that Algorithm 8 returns at Line 17 an explanation tree T' in the and-or explanation tree for p with respect to Π and X. That is, T' satisfies Conditions (i)–(iv) in Definition 10. Before providing the proof of Proposition 10, we consider some useful lemmas and corollaries.

Lemma 18. *Let Π be a ground normal ASP program, X be an answer set for Π, p be an atom in X, U be an assumption in $Assumptions(\Pi, X)$, $G = (V, E)$ be an offline justification of p^+ with respect to X and U, $T = \langle V', E', l, \Pi, X\rangle$ be an output of Algorithm 8 and $P = \langle v_1, \ldots, v_n\rangle$ be a path in T. Take any three consecutive elements v_i, v_{i+1}*

and v_{i+2} in P such that v_i and v_{i+2} are atom vertices and v_{i+1} is a rule vertex. Then, $(l(v_i)^+, l(v_{i+2})^+, +) \in E$ holds.

Proof of Lemma 18. As v_i is an atom vertex, its out-going edges are formed at Line 15 of Algorithm 8. Then, due to Lines 13 and 14, v_{i+1} is a rule vertex such that $B(l(v_{i+1})) = \mathit{support}(l(v_i)^+, G)$. As v_{i+1} is a rule vertex, its out-going edges are formed at Line 10. Then, due to Lines 8 and 9, v_{i+2} is an atom vertex where $l(v_{i+2}) \in B^+(l(v_{i+1}))$. Since $B(l(v_{i+1})) = \mathit{support}(l(v_i)^+, G)$ and $l(v_{i+2}) \in B^+(l(v_{i+1}))$, by Definition 17, $(l(v_i)^+, l(v_{i+2})^+, +) \in E$ holds.

□

Corollary 3. *Let Π be a ground normal ASP program, X be an answer set for Π, p be an atom in X, U be an assumption in $\mathit{Assumptions}(\Pi, X)$, $G = (N, E)$ be an offline justification of p^+ with respect to X and U, $T = \langle V', E', l, \Pi, X\rangle$ be an output of Algorithm 8 and $P = \langle v_1, \ldots, v_n\rangle$ be a path in T where v_1 and v_n are atom vertices. Then, $l(v_n)^+$ is reachable from $l(v_1)^+$ by a positive path in G.*

Proof of Corollary 3. Note that an edge in E' is either defined from an atom vertex to a rule vertex or vice versa due to Lines 10 and 15 of Algorithm 8. By this observation, in P, as v_1 is an atom vertex, v_i is an atom vertex (resp., rule vertex) if i is an odd number (resp., a even number). Then, by Lemma 18, for $1 \leq i \leq n-2$ and $i \mod 2 \neq 0$ (i.e., i is an odd number), $(l(v_i)^+, l(v_{i+2})^+, +) \in E$. Thus, there exists a positive path $\langle l(v_1)^+, l(v_3)^+, l(v_5)^+, \ldots, l(v_{n-2}^+), l(v_n)^+\rangle$ in G. That is, $l(v_n)^+$ is reachable from $l(v_1)^+$ by a positive path in G.

□

Lemma 19. *Let Π be a ground normal ASP program, X be an answer set for Π, p be an atom in X, U be an assumption in $\mathit{Assumptions}(\Pi, X)$, $G = (N, E)$ be an offline justification of p^+ with respect to X and U, $T = \langle V', E', l, \Pi, X\rangle$ be an output of Algorithm 8 and v be an atom vertex in V' such that v is the first element added into the queue Q in Algorithm 8. Consider a sequence $S = \langle v_1 = v, v_2, \ldots, v_n\rangle$ of n elements where $v_i \in V'$ for $1 \leq i \leq n$ such that v_{j+1} is added into Q right after v_j for $1 \leq j < n$. Then, every vertex $v_i \in V'$ ($1 < i \leq n$) is reachable from v by a path in T.*

Proof of Lemma 19. The proof is by induction on the length of S.

Base case: Assume that S has two elements, i.e., $S = \langle v_1 = v, v_2\rangle$. Since v is the first vertex added into Q by definition, it is the first vertex dequeued from Q at Line 5. Then, since v is an atom vertex by definition, the second vertex is added into Q at Line 16. By definition of S, v_2 is the second vertex added into Q. Thus, by Line 15, $(v, v_2) \in E'$, i.e., v_2 is reachable from v by a path in T.

Induction step: As an induction hypothesis, assume that for a sequence $S' = \langle v_1 = v, v_2, \ldots, v_k \rangle$ of k elements, where $v_i \in V'$ for $1 \leq i \leq k$ such that v_{j+1} is added into Q right after v_j for $1 \leq j < k$, every vertex $v_l \in V$ $(1 < l \leq k)$ is reachable from v by a path in T. Let $S'' = \langle v_1 = v, v_2, \ldots, v_{k+1} \rangle$ be a sequence of $k+1$ elements, where $v_i \in V'$ for $1 \leq i \leq k+1$ such that v_{j+1} is added into Q right after v_j for $1 \leq j < k+1$. We show that v reaches every vertex in S'' by using edges in E'. Consider the subsequence S''_{sub} of S'' that consists of the first k elements of S'', i.e., a prefix of S'' with length k. By the induction hypothesis, every vertex $v_i \in S''_{sub}$ is reachable from v by a path in T. Now, we show that v_{k+1} is reachable from v by a path in T. We consider two cases.

Case (1) Assume that v_{k+1} is an atom vertex. Then, v_{k+1} is added into V' at Line 11. So, by Line 10, there exists a vertex v' such that $(v', v_{k+1}) \in E'$. But, due to Line 5, v' must be added into Q prior to v_{k+1}. That is, $v' \in S''_{sub}$. So, by the induction hypothesis, v reaches v' by a path P in T. Thus, by P and (v', v_{k+1}), v_{k+1} is reachable from v by a path in T.

Case (2) Assume that v_{k+1} is a rule vertex. Then, v_{k+1} is added into V' at Line 16. So, by Line 15, there exists a vertex v' such that $(v', v_{k+1}) \in E'$. But, due to Line 5, v' must be added into Q prior to v_{k+1}. That is, $v' \in S''_{sub}$. So, by the induction hypothesis, v reaches v' by a path P in T. Thus, by P and (v', v_{k+1}), v_{k+1} is reachable from v by a path in T.

□

We now prove Proposition 10 that shows the soundness of Algorithm 8.

Proposition 10. *Given a ground normal ASP program Π, an answer set X for Π, an atom p in X, an assumption U in $Assumption(\Pi, X)$, and an offline justification $G = (V, E)$ of p^+ with respect to X and U, Algorithm 8 returns an explanation tree $\langle V', E', l, \Pi, X \rangle$ in the and-or explanation tree for p with respect to Π and X.*

Proof of Proposition 10. We show what Algorithm 8 returns at Line 17, $T' = \langle V', E', l, \Pi, X \rangle$, is an explanation tree in the and-or explanation tree $T = \langle V_T, E_T, l, \Pi, X \rangle$ for p with respect to Π and X. That is, T' satisfies Conditions (i)–(iv) in Definition 10. In the following, we study each condition separately.

(ii) To show that the root of $\langle V', E' \rangle$ is a vertex whose label is p, we need to show three cases hold; (1) there exists a vertex v in V' with label p, (2) every vertex $v' \in V'$ is reachable from v by a path in T'. (3) v has no in-going edge. In the following, we show that each case holds.

Case (1) Observe that a vertex is in V' if and only if it is added into the queue Q. Then, due to Lines 2 and 3, there exists a vertex $v \in V'$ such that $l(v) = p$.

Case (2) The first element added into Q is an atom vertex v with $l(v) = p$, due to Lines 2 and 3. Note that a vertex is in V' if and only if it is added into Q. Then, as v is added into Q, by the observation, $v \in V'$. Now, pick a vertex $v' \in V'$. By the same observation, v' is also added into Q. Since v is the first element queued in Q, v' is queued in V' later on. Thus, by Lemma 19, v' is reachable from v by a path in T'.

Case (3) Assume otherwise. That is, $(v', v) \in E'$ for some vertex $v' \in V'$. The edges in E' are constructed at Lines 10 and 15. Then, as v is an atom vertex, v' is a rule vertex. Observe that a vertex is in V' if and only if it is added into Q. Then, since v' is a rule vertex in V', it is added into Q at Line 16. So, due to Line 15, there exists an atom vertex $v'' \in V'$ such that $(v'', v') \in E'$. Thus, as $(v'', v'), (v', v) \in E'$, there exists a path $\langle v'', v', v\rangle$ in T'. Then, by Corollary 3, $l(v)^+$ is reachable from $l(v'')^+$ by a positive path P in G. Moreover, as shown in Case (2) above, v'' is reachable from v by a path $\langle v_1 = v, v_2, \ldots, v_n = v''\rangle$ in T'. So, by Corollary 3, $l(v'')^+$ is reachable from $l(v)^+$ by a positive path P' in G. Then, by P and P', a positive cycle exists in G. As every offline justification is a safe offline e-graph due to Definition 24, we reach a contradiction.

(i) By Condition (ii) in Definition 10, we know that the root of $\langle V', E'\rangle$ is a vertex v with $l(v) = p$. Then, if we show that for every vertex $v' \in V'$, $\mathit{out}_{E'}(v')$ is a subset of E_T, then we can conclude that $\langle V', E'\rangle$ is a subtree of $\langle V_T, E_T\rangle$. Now, pick a vertex $v' \in V'$. We consider two cases:

Case (1) Assume that v' is an atom vertex. Take an out-going edge (v', v'') of v' in E'. To show that $(v', v'') \in E_T$, we need to show that $l(v'') \in \Pi_{X,\mathit{anc}_{T'}(v'')}(l(v'))$, due to Condition (ii) in Definition 9. For that, we should show that $H(l(v'')) = l(v')$, $B^+(l(v'')) \subseteq X \backslash \mathit{anc}_{T'}(v'')$, $B^-(l(v'')) \cap X = \emptyset$ and $X \models B_{card}(l(v''))$, due to (4.1) in Section 4.1. As Π is a ground normal ASP program, it does not contain cardinality expressions in its body. Thus, $X \models B_{card}(l(v''))$ trivially. In Algorithm 8, out-going edges of atom vertices are formed at Line 15. Then, due to Lines 13 and 14, $H(l(v'')) = l(v')$ and $B(l(v'')) = \mathit{support}(l(v')^+, G)$. Since G is an offline justification, G is an offline e-graph by Definition 24, Then, by Definition 20, G is an (X, U)-based e-graph. So, by Definition 19, $\mathit{support}(l(v')^+, G)$ is an LCE of $l(v')^+$ with respect to (X, U), so does $B(l(v''))$. According to Definition 18, as $B(l(v'')) \neq \{\mathit{assume}\}$, $B^+(l(v'')) \subseteq X$ and $B^-(l(v'')) \cap X = \emptyset$. To complete this part, it remains to show that $B^+(l(v'')) \subseteq X \backslash \mathit{anc}_{T'}(v'')$. Since $B^+(l(v'')) \subseteq X$, it is enough to show that $B^+(l(v'')) \cap \mathit{anc}_{T'}(v'') = \emptyset$. For that, assume $B^+(l(v'')) \cap \mathit{anc}_{T'}(v'') \neq \emptyset$. Let $a \in B^+(l(v'')) \cap \mathit{anc}_{T'}(v'')$. Since $a \in B^+(l(v''))$ and $B(l(v'')) = \mathit{support}(l(v')^+, G)$, $(l(v')^+, a^+, +) \in E$ holds. As $a \in \mathit{anc}_{T'}(v'')$, there exists a vertex $u \in V'$ where $l(u) = a$ such that a path $P = \langle v_1 = u, \ldots, v_n = v'\rangle$ in T' exists. Then, due to Corollary 3, $l(v')^+$ is reachable from a^+

by a positive path in G. But, as $(l(v')^+, a^+, +) \in E$, $(a^+, a^+) \in E^{*,+}$ holds, i.e., there is a positive cycle in the offline justification, which is a contradiction, due to Definition 24.

Case (2) Assume that v' is a rule vertex. Take an out-going edge (v', v'') of v' in E'. To show that $(v', v'') \in E_T$, we need to show that $l(v'') \in B^+(l(v'))$, due to Condition (iii) in Definition 9. Out-going edges of v' are formed at Line 10 which is reached only by satisfying the condition at Line 8. Then, due to Line 9, $l(v'') \in B^+(l(v'))$.

(iii) Observe that an element is added into Q once. Thus, we dequeue each atom vertex only once. Then, due to Lines 12–16, every atom vertex $v' \in V'$ has exactly one out-going edge, i.e., $deg_{E'}(v') = 1$.

(iv) Take a rule vertex $v' \in V'$. Its edges are formed at Line 10. Then, due to the condition at Line 8 and the statement at Line 9, for every $a \in B^+(l(v'))$, there exists a vertex v'' such that $l(v'') = a$. Then, the condition holds.

□

Also, we can show that the and-or explanation tree exists for every atom in an answer set.

Proposition 1. *Let Π be a ground ASP program and X be an answer set for Π. For every p be in X, the and-or explanation tree for p with respect to Π and X is not empty.*

Proof of Proposition 1. By Proposition 9, we know that there exists an offline justification of p^+ with respect to X and $X^- \backslash WF_\Pi^-$. Then, by Proposition 10, there is an explanation tree in the and-or explanation tree for p with respect to Π and X. That is, the and-or explanation tree for p with respect to Π and X is not empty.

□

6.3.2 Proof of Proposition 11 – Soundness of Algorithm 9

Before proving Proposition 11, we provide the necessary lemmas.

Lemma 20. *Let Π be a ground normal ASP program, X be an answer set for Π, p be an atom in X, $T = \langle V', E', l, \Pi, X\rangle$ be an explanation tree in the and-or explanation tree for p with respect to Π and X such that for every $v, v' \in V'$, $l(v) = l(v')$ if and only if $v = v'$, (V, E) be the output of Algorithm 9 called with inputs Π, X, p and T, and $\langle v_1, v_2, v_3\rangle$ be a path in T such that $l(v_1)^+ \in V$. Then, $(l(v_1)^+, l(v_3)^+, +) \in E$ and $l(v_3)^+ \in V$.*

Proof of Lemma 20. All of the nodes, except the one with label $\top$, are added to V at Line 5 of Algorithm 9. As $l(v_1)^+ \in V$, v_1 is extracted from the queue Q at Line 4. Then, since $\langle v_1, v_2, v_3\rangle$ is a path in T and every atom vertex in V' has exactly one child due to

Condition (iii) in Definition 10, v_2 is obtained at Line 6. By the condition at Line 9, every child of v_2 is considered in the loop. Accordingly, at Line 10, the edge $(l(v_1)^+, l(v_3)^+, +)$ is added into E. Also, v_3 is added into Q at Line 11. Then, due to the condition at Line 3 and the statements at Lines 4 and 5, $l(v_3)^+ \in V$.

□

Corollary 4. *Let Π be a ground normal ASP program, X be an answer set for Π, p be an atom in X, $T = \langle V', E', l, \Pi, X\rangle$ be an explanation tree in the and-or explanation tree for p with respect to Π and X such that for every $v, v' \in V'$, $l(v) = l(v')$ if and only if $v = v'$, (V, E) be the output of Algorithm 9 called with inputs Π, X, p and T, and $\langle v_1, v_2, \ldots, v_n\rangle$ be a path in T such that $l(v_1)^+ \in V$. Then, $\langle l(v_1)^+, l(v_3)^+, l(v_5)^+, \ldots, l(v_n)^+\rangle$ is a path in (V, E).*

Proof of Corollary 4. Since $\langle v_1, v_2, v_3\rangle$ is a path in T and $l(v_1)^+ \in V$, by Lemma 20, $(l(v_1)^+, l(v_3)^+, +) \in E$ and $l(v_3)^+ \in V$. Similarly, $(l(v_3)^+, l(v_5)^+, +) \in E$ and $l(v_5)^+ \in V$. Then, this incremental application of Lemma 20 leads that $\langle l(v_1)^+, l(v_3)^+, l(v_5)^+, \ldots, l(v_n)^+\rangle$ is a path in (V, E).

□

We now prove Proposition 11 which shows that Algorithm 9 creates an offline justification of the given atom in the reduct of the given ASP program with respect to the given answer set, provided that labels of the vertices of the given explanation tree are unique labels, i.e., no two different vertices labels the same entity.

Proposition 11. *Given a ground normal ASP program Π , an answer set X for Π, an atom p in X, and an explanation tree $\langle V', E', l, \Pi, X\rangle$ in the and-or explanation tree for p with respect to Π and X such that for every $v, v' \in V'$, $l(v) = l(v')$ if and only if $v = v'$, Algorithm 9 returns an offline justification of p^+ in Π^X with respect to X and $\emptyset$.*

Proof of Proposition 11. To show that the output (V, E) of Algorithm 9 at Line 13 is an offline justification of p^+ in Π^X with respect to X and $\emptyset$, one needs to show that the following conditions hold.

(1) $\emptyset \in Assumptions(\Pi^X, X)$;

(2) (V, E) is an e-graph for Π^X;

(3) (V, E) is a $(X, \emptyset)$-based e-graph of p^+;

(4) (V, E) is an offline e-graph of p^+ with respect to X and $\emptyset$.

Condition (1) We show that $\emptyset \in Assumptions(\Pi^X, X)$. As Π^X is a positive program, i.e., it does not contain any negative atoms, by Definition 21, $\mathcal{TA}_{\Pi^X}(X) = \emptyset$. Also, due to Definition 22, $NR(\Pi^X, X) = \Pi^X$. Then, by Definition 23, $\emptyset \in Assumptions(\Pi^X, X)$.

Condition (2) We show that (V, E) is an e-graph for Π^X. For that, (V, E) should satisfy Conditions $(i) - (iv)$ in Definition 16.

(i) Consider two cases.

Case 1. Take a node $b^+ \in V \backslash \{\top\}$. It is added to V at Line 5. Then, there exists an atom vertex $v \in V'$ such that $l(v) = b$. By Condition (iii) in Definition 10, v has exactly one child v' in $\langle V', E' \rangle$, which is a rule vertex. Then, depending on whether $l(v')$ is a fact or not, an out-going edge of b^+ in E is formed at Line 8 or 10. So, b^+ is not a sink.

Case 2. Let l be $\top$. Then, it is added to V at Line 12. The edges in E are formed at Lines 8 and 10. Accordingly, l has no out-going edge, i.e., it is a sink.

Therefore, (V, E) is an e-graph for Π^X.

(ii) Due to Lines 8 and 10, clearly, for every $b \in V$ there is no edges in the form of $(b, assume, -)$ and $(b, \bot, -)$ in E.

(iii) Similar to (ii), for every $b \in V$ there is no edges in the form of $(b, assume, +)$ and $(b, \top, +)$ in E.

(iv) Let $b^+ \in V$ such that $(b^+, \top, +) \in E$. As out-going edges are created at Lines 8 and 10, $(b, \top, +)$ must be formed at Line 8. Then, there exists an atom vertex $v \in V'$ such that $l(v) = b$ and the label of the child v' of v is a fact (Line 7). Then, since the condition at Line 9 is not satisfied, it is not possible that b^+ has another out-going edge.

Condition (3) We show that (V, E) is an $(X, \emptyset)$-based e-graph of p^+. By Condition (1) above, we know that (V, E) is an e-graph. To show that (V, E) is an $(X, \emptyset)$-based e-graph of p^+, we need to show that Conditions (i) and (ii) in Definition 19 hold.

(i) Take a node $c^+ \in V$. It is added to V at Line 5. Then, there exists an atom vertex $v' \in V'$ such that $l(v') = c$. By Condition (ii) in Definition 10, we know that the root of $\langle V', E' \rangle$ is a vertex v where $l(v) = p$. So, v' is reachable from v by a path $\langle v_1 = v, v_2, v_3, \ldots, v_n = v' \rangle$ in $\langle V', E' \rangle$. Also, as v is added into Q at Line 2, we know that $l(v)^+ \in V$ by Line 5. Then, by Corollary 4, $\langle l(v)^+, l(v_3)^+, \ldots, l(v')^+ \rangle$ is a path in (V, E). That is, c^+ is reachable from p^+.

(ii) Let $G = (V, E)$. Take a node $c^+ \in V \backslash \{\top\}$. Then, $support(c^+, G) = \{a \mid (c^+, a^+, +) \in E\}$ or $support(c^+, G) = \{\top\}$.

Assume that $support(c^+, G) = \{a \mid (c^+, a^+, +) \in E\}$. As $c^+ \in V \backslash \{\top\}$, there exists an atom vertex $v \in V'$ such that $l(v) = c$. By Condition (iii) in Definition 10, there exists exactly one child v' of v in $\langle V', E' \rangle$. Then, due to the condition at Line 9, for each $a \in support(c^+, G)$, $(v', v'') \in E'$ where $v'' \in V'$ with $l(v'') = a$. Thus, $B^+(l(v')) = support(c^+, G)$. By Condition (ii) in Definition 9, $H(l(v')) = c$ and $B^+(l(v')) \subseteq X$. Hence, $support(c^+G)$ is an LCE of $(X, \emptyset)$.

Condition (4) We show that (V, E) is an offline e-graph of p^+ with respect to X and $\emptyset$. By Condition (3) above, we know that (V, E) is an $(X, \emptyset)$-based e-graph of p^+. Then, to show that (V, E) is an offline e-graph of p^+ with respect to X and $\emptyset$, we need to show that (V, E) satisfies Conditions (i) and (ii) in Definition 20. As edges in E are formed at Lines 8 and 10, for every $b \in V$ there are no edges in the form of $(b, assume, +)$ and $(b, assume, -)$ in E. Then, conditions are trivially satisfied.

□

CHAPTER 7

DISCUSSION AND CONCLUSION

Other related work In [20], the authors integrate and annotate biomedical data from public knowledge resources by using semantic methods, from which they build a heterogeneous semantic linked network in relation to drug-target pairs. Then, they develop a statistical model called Semantic Linked Association Prediction (SLAP) which assess the association of a drug-target pair by an association score that is obtained through the relation of drug-target pairs with other linked objects in the network.[1] Based on that, they identify 157 drugs from 10 disease areas and calculate the association score of every drug against 1683 human targets. This results in a drug similarity network where the similarity of drugs are measured by using a 157×1683 score matrix. Through the web interface of SLAP, one can query the similar drugs to a given drug.

We performed experiments to compare the results of SLAP and our greedy method on finding closest drugs to a given drug. We considered 6 drugs as shown in Table 7.1. The second (resp., third) column contains the number of closest drugs found by the SLAP (resp., by the greedy approach). When SLAP found less than 5 drugs for a query, we used the greedy method to find 5 closest drugs for the same query. Otherwise, we used the greedy method to find the same number of drugs as SLAP found. The last column indicates the number of the same closest drugs found by the two methods. Note that the common drugs are quite few, due to different distance measures. In that sense, both approaches are complementary to each other.

The most recent work related to explanation generation in ASP are [14, 90, 47, 78, 74, 75], in the context of debugging ASP programs. Among those, [90] studies why a program does not have an answer set, and [47, 74] studies why a set of atoms is not an answer set. As we study the problem of explaining the reasons why atoms are in the answer set, our work differs from those two work. In [14], similar to our work, the question "why is an atom p in an answer set X for an ASP program Π" is studied. As

[1] `http://chem2bio2rdf.org/slap`

Table 7.1: Comparison of SLAP and our greedy method for finding closest drugs.

Drugs	**# of drugs (SLAP)**	**# of drugs (the greedy approach)**	**# of common drugs**
Diazoxide	3	5	0
Diclofenac	15	15	0
Epinephrine	2	5	0
Methazolamide	26	26	2
Piroxicam	14	14	4
Tolmetin	15	15	4

an answer to this question, the authors of [14] provide the rule in Π that supports X with respect to Π; whereas we compute shortest or k different explanations (as a tree whose vertices are labeled by rules). The authors of [78] introduce the notion of an online justification that aims to justify the truth values of atoms during the computation of an answer set. In [75], a framework where the users can construct interpretations through an interactive stepping process is introduced. As a result, [78] and [75] can be used together to provide the users with justifications of the truth values of atoms during the construction of interpretations interactively through stepping.

Contributions In this thesis, we have extended BIOQUERY-ASP to answer new types of queries and studied explanation generation in the context of ASP. The main contributions can be summarized as follows.

- We have extended the grammar of BIOQUERY-CNL to construct new types of queries, such as negative queries, queries about symptoms of diseases and similarity/diversity queries about drugs. For similarity/diversity queries about drugs, we have considered the distance measure based on the side effects of drugs.

- We have then modified the software system BIOQUERY-ASP to answer such queries. We have also extended BIOQUERY-ASP to most recent biomedical knowledge resources about drugs, genes, and diseases, such as PHARMGKB, DRUGBANK, BIOGRID, CTD, SIDER, DISEASEONTOLOGY and ORPHADATA.

- We have also studied queries that ask for the closest or most distant drugs to a given drug. We have introduced naive/greedy methods to answer such queries using ASP. We have also related the problem to the problem similar/diverse solutions studied in [29].

- We have formally defined explanations in the context of ASP. We have also introduced variations of explanations, such as "shortest explanations" and "k different explanations".

- We have proposed generic algorithms to generate explanations for biomedical queries. In particular, we have presented algorithms to compute shortest or k different explanations. We have analyzed termination, soundness and complexity of these algorithms. In particular, the complexity of generating a shortest explanation for an answer (in an answer set X) is $O(|\Pi|^{|X|} \times |\mathcal{B}_\Pi|)$ where $|\Pi|$ is the number of ASP rules representing the query, the knowledge extracted from biomedical resources and the rule layer, and $|\mathcal{B}_\Pi|$ is the number of atoms in Π. The complexity of generating k different explanations is $O(k \times |\Pi|^{|X|+1} \times |\mathcal{B}_\Pi|)$. For k different explanations, we have defined a distance measure based on the number of different ASP rules between explanations.

- We have developed a computational tool EXPGEN-ASP which implements these algorithms.

- We have embedded EXPGEN-ASP into BIOQUERY-ASP to generate explanations regarding the answers of complex biomedical queries. We have proposed a method to present explanations in a natural language.

- No existing biomedical query answering system is capable of generating explanations; our methods have fulfilled this need in biomedical query answering.

- We have illustrated the applicability of our methods to answer complex biomedical queries over large biomedical knowledge resources about drugs, genes, and diseases, such as PHARMGKB, DRUGBANK, BIOGRID, CTD, SIDER, DISEASEONTOLOGY and ORPHADATA. The total number of the facts extracted from these resources to answer queries is approximately 10.3 million.

It is important to emphasize here that our definitions and methods for explanation generation are general, so they can be applied to other applications (e.g., debugging, query answering in other domains).

Parts of our studies are summarized in [40, 35, 77, 36].

Future work One line of future work is to further extend the grammar of BIOQUERY-CNL*, and BIOQUERY-ASP to new types of queries. For instance, we may extend it to answer queries in the following format:

What are the n closest drugs to the drug d ?

where n is a positive integer and d is a drug name.

Another line of future work is to investigate the use of another type of formalism for representing and answering biomedical queries, like Description Logics (DL) [4, 56],

and extend BIOQUERY-ASP to answer queries using relevant DL reasoners, such as FACT++ [94], HERMIT [84] and PELLET [86].

Also it will be good to generalize the notion of an explanation to generate explanations for programs that contain choice expressions. In this way, explanations can be generated for a wider range of queries.

APPENDIX A

SHORTEST EXPLANATIONS FOR BIOMEDICAL QUERIES

Query: What are the drugs that treat the disease Asthma and that target the gene ADRB1?
Answer: `Epinephrine`

A shortest explanation:

```
what_be_drugs("Epinephrine") :-
      drug_disease("Epinephrine","Asthma"),
      drug_gene("Epinephrine","ADRB1").

   drug_disease("Epinephrine","Asthma") :-
         drug_disease_ctd("Epinephrine","Asthma").

      drug_disease_ctd("Epinephrine","Asthma").

   drug_gene("Epinephrine","ADRB1") :-
         drug_gene_ctd("Epinephrine","ADRB1").

      drug_gene_ctd("Epinephrine","ADRB1").
```

Explanation in natural language:

The disease Asthma is treated by the drug Epinephrine according to CTD.

The drug Epinephrine targets the gene ADRB1 according to CTD.

Figure A.1: A shortest explanation for Q1

Query: What are the side effects of the drugs that treat the disease Asthma and that target the gene ADRB1?
Answer: `hypertension`

A shortest explanation:

```
what_be_sideeffects("hypertension") :-
      drug_disease("Desipramine","Asthma"),
      drug_sideeffect("Desipramine","hypertension"),
      drug_gene("Desipramine","ADRB1").

   drug_disease("Desipramine","Asthma") :-
         drug_disease_ctd("Desipramine","Asthma").

      drug_disease_ctd("Desipramine","Asthma").

   drug_sideeffect("Desipramine","hypertension") :-
         drug_sideeffect_sider("Desipramine","hypertension").

      drug_sideeffect_sider("Desipramine","hypertension").

   drug_gene("Desipramine","ADRB1") :-
         drug_gene_pharmgkb("Desipramine","ADRB1").

      drug_gene_pharmgkb("Desipramine","ADRB1").
```

Explanation in natural language:

The disease Asthma is treated by the drug Desipramine according to CTD.

The drug Desipramine has the side effect hypertension according to SIDER.

The drug Desipramine targets the gene ADRB1 according to PHARMGKB.

Figure A.2: A shortest explanation for Q2

Query: What are the genes that are targeted by the drug Epinephrine and that interact with the gene DLG4?
Answer: ADRB1

A shortest explanation:

```
what_be_genes("ADRB1") :-
      drug_gene("Epinephrine","ADRB1"),
      gene_gene("ADRB1","DLG4").

   drug_gene("Epinephrine","ADRB1") :-
         drug_gene_ctd("Epinephrine","ADRB1").

      drug_gene_ctd("Epinephrine","ADRB1").

   gene_gene("ADRB1","DLG4") :- gene_gene("DLG4","ADRB1").

      gene_gene("DLG4","ADRB1") :-
            gene_gene_biogrid("DLG4","ADRB1").

         gene_gene_biogrid("DLG4","ADRB1").
```

Explanation in natural language:

The drug Epinephrine targets the gene ADRB1 according to CTD.

The gene DLG4 interacts with the gene ADRB1 according to BIOGRID.

Figure A.3: A shortest explanation for Q3

Query: What are the genes that interact with at least 3 genes and that are targeted by the drug Epinephrine?
Answer: GNAS

A shortest explanation:

```
what_be_genes("GNAS") :-
      3#count{gene_gene("GNAS","VIPR1"), ... },
      drug_gene("Epinephrine","GNAS"),
      gene_name("GNAS").

   drug_gene("Epinephrine","GNAS") :-
         drug_gene_pharmgkb("Epinephrine","GNAS").

      drug_gene_pharmgkb("Epinephrine","GNAS").

   gene_name("GNAS") :- gene_gene("GNAS","FSCN1").

      gene_gene("GNAS","FSCN1") :-
            gene_gene_biogrid("GNAS","FSCN1").

         gene_gene_biogrid("GNAS","FSCN1").
```

Explanation in natural language:

The drug Epinephrine targets the gene GNAS according to PHARMGKB.

The gene GNAS interacts with the gene FSCN1 according to BIOGRID.

Figure A.4: A shortest explanation for Q4

Query: What are the drugs that treat the disease Asthma or that react with the drug Epinephrine?
Answer: `Azacitidine`

A shortest explanation:

```
what_be_drugs("Azacitidine") :- drug_disease("Azacitidine","Asthma").

   drug_disease("Azacitidine","Asthma") :-
         drug_disease_ctd("Azacitidine","Asthma").

      drug_disease_ctd("Azacitidine","Asthma").
```

Explanation in natural language:

The disease Asthma is treated by the drug Doxepin according to CTD.

Figure A.5: A shortest explanation for Q5

Query: What are the genes that are related to the gene ADRB1 via a gene-gene interaction chain of length at most 3?
Answer: `CD53`

A shortest explanation:

```
what_be_genes("CD53") :- gene_reachable_from("CD53",3).

   gene_reachable_from("CD53",3) :-
                  gene_gene("CD53","PRKCA"),
                  gene_reachable_from("PRKCA",2), ...

      gene_gene("CD53","PRKCA") :- gene_gene_biogrid("CD53","PRKCA").

         gene_gene_biogrid("CD53","PRKCA").

      gene_reachable_from("PRKCA",2) :-
                     gene_gene("PRKCA","DLG4"),
                     gene_reachable_from("DLG4",1), ...

         gene_gene("PRKCA","DLG4") :- gene_gene("DLG4","PRKCA").

            gene_gene("DLG4","PRKCA") :- gene_gene_biogrid("DLG4","PRKCA").

               gene_gene_biogrid("DLG4","PRKCA").

         gene_reachable_from("DLG4",1) :-
               gene_gene("DLG4","ADRB1"),
               start_gene("ADRB1").

            gene_gene("DLG4","ADRB1") :- gene_gene_biogrid("DLG4","ADRB1").

               gene_gene_biogrid("DLG4","ADRB1").

            start_gene("ADRB1").
```

Explanation in natural language:

The gene CD53 interacts with the gene PRKCA according to BIOGRID.

The gene DLG4 interacts with the gene PRKCA according to BIOGRID.

The gene DLG4 interacts with the gene ADRB1 according to BIOGRID.

The gene ADRB1 is the start gene.

The distance of the gene DLG4 from the start gene is 1.

The distance of the gene PRKCA from the start gene is 2.

The distance of the gene CD53 from the start gene is 3.

Figure A.6: A shortest explanation for Q8

Query: What are the genes that are related to the gene DLG4 via a gene-gene interaction chain of length at most 3 and that are targeted by the drugs that belong to the category Hmg-coa reductase inhibitors?
Answer: ABCA1

A shortest explanation:

```
what_be_genes("ABCA1") :-
      gene_reachable_from("ABCA1",2),
      drug_gene("Pravastatin","ABCA1"),
      drug_category("Pravastatin","Hmg-coa reductase inhibitors").

   gene_reachable_from("ABCA1",2) :-
         gene_gene("ABCA1","HGS"),
         gene_reachable_from("HGS",1), ...

      gene_gene("ABCA1","HGS") :- gene_gene_biogrid("ABCA1","HGS").

         gene_gene_biogrid("ABCA1","HGS").

      gene_reachable_from("HGS",1) :-
            gene_gene("HGS","DLG4"),
            start_gene("DLG4").

         gene_gene("HGS","DLG4") :- gene_gene("DLG4","HGS").

            gene_gene("DLG4","HGS") :- gene_gene_biogrid("DLG4","HGS").

               gene_gene_biogrid("DLG4","HGS").

         start_gene("DLG4").

   drug_gene("Pravastatin","ABCA1") :- drug_gene_ctd("Pravastatin","ABCA1").

      drug_gene_ctd("Pravastatin","ABCA1").

   drug_category("Pravastatin","Hmg-coa reductase inhibitors"):-
         drug_category_drugbank("Pravastatin","Hmg-coa reductase inhibitors").

      drug_category_drugbank("Pravastatin","Hmg-coa reductase inhibitors").
```

Explanation in natural language:

The gene ABCA1 interacts with the gene HGS according to BIOGRID.

The gene DLG4 interacts with the gene HGS according to BIOGRID.

The gene DLG4 is the start gene.

The distance of the gene HGS from the start gene is 1.

The distance of the gene ABCA1 from the start gene is 2.

The drug Pravastatin targets the gene ABCA1 according to CTD.

The drug Pravastatin belongs to the category Hmg-coa reductase inhibitors according to DRUGBANK.

Figure A.7: A shortest explanation for Q10

Query: What are the drugs that treat the disease Depression and that do not target the gene ACYP1?
Answer: `Fluoxetine`

A shortest explanation:

```
what_be_drugs("Fluoxetine") :-
      drug_disease("Fluoxetine","Depression"),
      drug_name("Fluoxetine"),
      not drug_gene("Fluoxetine","ACYP1").

   drug_disease("Fluoxetine","Depression") :-
         drug_disease_pharmgkb("Fluoxetine","Depression").

      drug_disease_pharmgkb("Fluoxetine","Depression").

   drug_name("Fluoxetine") :-
         drug_sideeffect("Fluoxetine","abdominal pain").

      drug_sideeffect("Fluoxetine","abdominal pain") :-
            drug_sideeffect_sider("Fluoxetine","abdominal pain").

         drug_sideeffect_sider("Fluoxetine","abdominal pain").
```

Explanation in natural language:

The disease Depression is treated by the drug Fluoxetine according to PHARMGKB.

The drug Fluoxetine has the side effect abdominal pain according to SIDER.

Figure A.8: A shortest explanation for Q11

Query: What are the symptoms of diseases that are treated by the drug Triadimefon?

Answer: `chest tightness`

A shortest explanation:

```
what_be_symptoms("chest tightness") :-
      disease_symptom("Asthma","chest tightness"),
      drug_disease("Triadimefon","Asthma").

   disease_symptom("Asthma","chest tightness") :-
         disease_symptom_do("Asthma","chest tightness").

      disease_symptom_do("Asthma","chest tightness").

   drug_disease("Triadimefon","Asthma") :-
         drug_disease_ctd("Triadimefon","Asthma").

      drug_disease_ctd("Triadimefon","Asthma").
```

Explanation in natural language:

The disease Asthma has the symptom chest tightness according to DISEASEONTOLOGY.

The disease Asthma is treated by the drug Triadimefon according to CTD.

Figure A.9: A shortest explanation for Q12

BIBLIOGRAPHY

[1] M. Alviano and W. Faber. Dynamic magic sets and super-coherent answer set programs. *AI Commun.*, 24(2):125–145, 2011.

[2] M. Alviano, W. Faber, and N. Leone. Disjunctive ASP with functions: Decidable queries and effective computation. *TPLP*, 10(4-6):497–512, 2010.

[3] K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *J. Log. Program.*, 19/20:9–71, 1994.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2007.

[5] M. Balduccini. Industrial-size scheduling with ASP + CP. In *Proc. of LPNMR*, pages 284–296, 2011.

[6] M. Balduccini and M. Gelfond. Diagnostic reasoning with A-Prolog. *TPLP*, 3(4–5):425–461, 2003.

[7] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[8] C. Baral, M. Gelfond, and J. N. Rushton. Probabilistic reasoning with answer sets. *TPLP*, 9(1):57–144, 2009.

[9] C. Baral and C. Uyan. Declarative specification and solution of combinatorial auctions using logic programming. In *Proc. of LPNMR*, pages 186–199, 2001.

[10] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS press, 2009.

[11] O. Bodenreider, Z. H. Çoban, M. C. Doğanay, E. Erdem, and H. Koşucu. A preliminary report on answering complex queries related to drug discovery using answer set programming. In *Proc. of ALPSWS*, 2008.

[12] G. Boenn, M. Brain, M. D. Vos, and J. Fitch. Anton: Composing logic and logic composing. In *Proc. of LPNMR*, pages 542–547, 2009.

[13] G. Boenn, M. Brain, M. D. Vos, and J. Fitch. Automatic music composition using answer set programming. *CoRR*, abs/1006.4948, 2010.

[14] M. Brain and M. D. Vos. Debugging logic programs under the answer set semantics. In *Proc. of ASP*, 2005.

[15] G. Brewka. Preferences, contexts and answer sets. In *Proc. of ICLP*, page 22, 2007.

[16] G. Brewka, T. Eiter, and M. Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[17] D. Calvanese, T. Eiter, and M. Ortiz. Regular path queries in expressive description logics with nominals. In *Proc. of IJCAI*, pages 714–720, 2009.

[18] M. Campillos, M. Kuhn, A. C. Gavin, L.J. Jensen, and P. Bork. Drug target identification using side-effect similarity. *Science*, 321:263–266, 2009.

[19] D. Çakmak, E. Erdem, and H. Erdoğan. Computing weighted solutions in ASP: Representation-based method vs. search-based method. *Ann. Math. Artif. Intell.*, 62(3-4):219–258, 2011.

[20] B. Chen, Y. Ding, and D.J. Wild. Assessing drug target association using semantic linked data. *PLoS Comput Biol*, 8(7):e1002574, 07 2012.

[21] C. R. Chong and D. J. Sullivan. New uses for old drugs. *Nature*, 448:645–646, 2007.

[22] A.P. Davis, B.L. King, S. Mockus, C.G. Murphy, C. Saraceni-richards, M. Rosenstein, T. Wiegers, and C.J. Mattingly. The comparative toxicogenomics database: update 2011. *Nucleic Acids Res.*, 2011.

[23] J. P. Delgrande, T. Grote, and A. Hunter. A general approach to the verification of cryptographic protocols using answer set programming. In *Proc. of LPNMR*, pages 355–367, 2009.

[24] J. Dix, T. Eiter, M. Fink, A. Polleres, and Y. Zhang. Monitoring agents using declarative planning. *Fundam. Inform.*, 57(2-4):345–370, 2003.

[25] C. Dodaro, M. Alviano, W. Faber, N. Leone, F. Ricca, and M. Sirianni. The birth of a WASP: Preliminary report on a new ASP solver. In *Proc. of CILC*, pages 99–113, 2011.

[26] D. East and M. Truszczynski. More on wire routing with ASP. In *Proc. of ASP*, 2001.

[27] T. Eiter, G. Brewka, M. Dao-Tran, M. Fink, G. Ianni, and T. Krennwallner. Combining nonmonotonic knowledge bases with external sources. In *Proc. of FroCos*, pages 18–42, 2009.

[28] T. Eiter, E. Erdem, H. Erdoğan, and M. Fink. Finding similar or diverse solutions in answer set programming. In *Proc. of ICLP*, pages 342–356, 2009.

[29] T. Eiter, E. Erdem, H. Erdoğan, and M. Fink. Finding similar/diverse solutions in answer set programming. *Theory and Practice of Logic Programming (TPLP)*, 2011. To appear.

[30] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.

[31] T. Eiter, G.Ianni, R.Schindlauer, and H.Tompits. Effective integration of declarative rules with external evaluations for Semantic-Web reasoning. In *Proc. of ESWC*, 2006.

[32] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-founded semantics for description logic programs in the semantic web. In *Proc. of RuleML*, pages 81–97, 2004.

[33] E. Erdem. PHYLO-ASP: Phylogenetic systematics with answer set programming. In *Proc. of LPNMR*, pages 567–572, 2009.

[34] E. Erdem, O. Erdem, and F. Türe. HAPLO-ASP: Haplotype inference using answer set programming. In *Proc. of LPNMR*, pages 573–578, 2009.

[35] E. Erdem, Y. Erdem, H. Erdoğan, and U. Öztok. Finding answers and generating explanations for complex biomedical queries. In *AAAI*, 2011.

[36] E. Erdem, H. Erdoğan, and U. Öztok. BIOQUERY-ASP: Querying biomedical ontologies using answer set programming. In *Proc. of RuleML2011@BRF Challenge*, 2011.

[37] E. Erdem, V. Lifschitz, and D. Ringe. Temporal phylogenetic networks and logic programming. *Theory and Practice of Logic Programming*, 6(5):539–558, 2006.

[38] E. Erdem, V. Lifschitz, and M. F. Wong. Wire routing and satisfiability planning. In *In Proceedings CL-2000*, pages 822–836. Springer-Verlag. LNCS, 2000.

[39] E. Erdem and R. Yeniterzi. Transforming controlled natural language biomedical queries into answer set programs. In *Proc. of the Workshop on BioNLP*, pages 117–124, 2009.

[40] H. Erdoğan, U. Öztok, Y. Erdem, and E. Erdem. Querying biomedical ontologies in natural language using answer set programming. In *Proc. of the Third International Workshop on Semantic Web Applications and Tools for Life Sciences (SWAT4LS'10)*, 2010.

[41] P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5:45–74, 2005.

[42] R. Finkel, V. W. Marek, and M. Truszczynski. Constraint Lingo: A program for solving logic puzzles and other tabular constraint problems, 2002.

[43] M. Gebser, C. Guziolowski, M. Ivanchev, T. Schaub, A. Siegel, S. Thiele, and P. Veber. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In *Proc. of KR*, 2010.

[44] M. Gebser, R. Kaminski, A. König, and T. Schaub. Advances in *gringo* series 3. In *Proc of. LPNMR*, volume 6645, pages 345–351, 2011.

[45] M. Gebser, R. Kaminski, and T. Schaub. aspcud: A Linux package configuration tool based on answer set programming. In *Proc. of LoCoCo*, volume 65, pages 12–25, 2011.

[46] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In *Proc. of LPNMR*, pages 260–265, 2007.

[47] M. Gebser, J. Pührer, T. Schaub, and H. Tompits. A meta-programming technique for debugging answer-set programs. In *Proc. of AAAI*, 2008.

[48] M. Gebser, T. Schaub, S. Thiele, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. *Theory and Practice of Logic Programming*, 11(2):1–38, 2011.

[49] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991.

[50] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP*, pages 1070–1080. MIT Press, 1988.

[51] E. Giunchiglia, Y. Lierler, and M. Maratea. Answer set programming based on propositional satisfiability. *Automated Reasoning*, 36:345–377, 2006.

[52] T. Gower. Born again. *Proto Magazine*, Summer:14–19, 2009.

[53] K. Heljanko and I. Niemela. Bounded LTL model checking with stable models. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 200–212. Springer-Verlag, 2003.

[54] K. Inoue and C. Sakama. Abductive framework for nonmonotonic theory change. In *IJCAI*, pages 204–210, 1995.

[55] C. Knox, V. Law, T. Jewison, P. Liu, S. Ly, A. Frolkis, A. Pon, K. Banco, C. Mak, V. Neveu, Y. Djoumbou, R. Elsner, A.C. Guo, and D.S. Wishart. Drugbank 3.0: a comprehensive resource for 'omics' research on drugs. *Nucleic Acids Res.*, 2011.

[56] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. A description logic primer. *CoRR*, abs/1201.4089, 2012.

[57] M. Kuhn, M. Campillos, I. Letunic, L.J. Jensen, and P. Bork. A side effect resource to capture phenotypic effects of drugs. *Mol Syst Biol.*, 2010.

[58] C. Lefèvre and P. Nicolas. The first version of a new ASP solver : ASPeRiX. In *Proc. of LPNMR*, pages 522–527, 2009.

[59] N. Leone, G. Greco, G. Ianni, V. Lio, G. Terracina, T. Eiter, W. Faber, M. Fink, G. Gottlob, R. Rosati, D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowicki, and W. Staniszkis. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *Proc. of SIGMOD*, pages 915–917, 2005.

[60] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7:499–562, 2006.

[61] Y. Lierler. Abstract answer set solvers with backjumping and learning. *TPLP*, 11(2-3):135–169, 2011.

[62] V. Lifschitz. Answer set programming and plan generation. *Artif. Intell.*, 138(1-2):39–54, 2002.

[63] V. Lifschitz. What is answer set programming? In *Proc. of AAAI*, pages 1594–1597. MIT Press, 2008.

[64] F. Lin and Y. Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artif. Intell.*, 157:115–137, 2004.

[65] L. Liu and M. Truszczynski. Pbmodels - software to compute stable models by pseudoboolean solvers. In *Proc. of LPNMR*, pages 410–415, 2005.

[66] D. V. Marina and D Vermeir. Logic programming agents and game theory, 2001.

[67] E.M. McDonagh, M. Whirl-Carrillo, Y. Garten, Altman R.B., and T.E. Klein. From pharmacogenomic knowledge acquisition to clinical applications: the PharmGKB as a clinical pharmacogenomic biomarker resource. *Biomark Med.*, pages 795–806, 2011.

[68] A. Mileo, D. Merico, and R. Bisiani. Wireless sensor networks supporting context-aware reasoning in assisted living. In *Proc. of PETRA*, pages 1–2, 2008.

[69] A. Mileo, D. Merico, and R. Bisiani. Non-monotonic reasoning supporting wireless sensor networks for intelligent monitoring: The SINDI system. In *Proc. of LPNMR*, pages 585–590, 2009.

[70] A. Mileo, T. Schaub, D. Merico, and R. Bisiani. Knowledge-based multi-criteria optimization to support indoor positioning. *AMAI*, 62(3-4):345–370, 2011.

[71] M.Manna, E. Oro, M. Ruffolo, M. Alviano, and N. Leone. The HiLeX system for semantic information extraction. *T. Large-Scale Data- and Knowledge-Centered Systems*, 5:91–125, 2012.

[72] I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal lp. In *Proc. of LPNMR*, pages 421–430, 1997.

[73] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An A-Prolog decision support system for the space shuttle. In *Proc. of PADL*, pages 169–183. Springer, 2001.

[74] J. Oetsch, J. Pührer, and H. Tompits. Catching the ouroboros: On debugging non-ground answer-set programs. *TPLP*, 10(4-6):513–529, 2010.

[75] J. Oetsch, J. Pührer, and H. Tompits. Stepping through an answer-set program. In *LPNMR*, pages 134–147, 2011.

[76] M. Ostrowski, G. Flouris, T. Schaub, and G. Antoniou. Evolution of ontologies using ASP. In *Proc. of ICLP*, pages 16–27, 2011.

[77] U. Öztok and E. Erdem. Generating explanations for complex biomedical queries. In *AAAI*, 2011.

[78] E. Pontelli, T. C. Son, and O. El-Khatib. Justifications for logic programs under answer set semantics. *Theory and Practice of Logic Programming*, pages 1–56, 2009.

[79] F. Ricca, A. Dimasi, G. Grasso, S. M. Ielpa, S. Iiritano, M. Manna, and N. Leone. A logic-based system for e-tourism. *Fundam. Inform.*, 105(1-2):35–55, 2010.

[80] F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, and N. Leone. Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practice of Logic Programming*, 12, 2012.

[81] C. Sakama. Learning by answer sets. In *Proc. of AAAI Spring Symposium:Answer Set Programming*, 2001.

[82] T. Schaub and S. Thiele. Metabolic network expansion with answer set programming. In *Proc. of ICLP*, pages 312–326, 2009.

[83] T. Schaub and K. Wang. A comparative study of logic programs with preference. In *Proc. of IJCAI*, pages 597–602, 2001.

[84] R. Shearer, B. Motik, and I. Horrocks. HermiT: A highly-efficient OWL reasoner. In *OWLED*, 2008.

[85] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.

[86] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.

[87] T. C. Son, E. Pontelli, and C. Sakama. Logic programming for multiagent planning with negotiation. In *Proc. of ICLP*, pages 99–114, 2009.

[88] T. C. Son and C. Sakama. Reasoning and planning with cooperative actions for multiagents using answer set programming. In *Proc. of DALT*, pages 208–227, 2009.

[89] C. Stark, B.J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: a general repository for interaction datasets. *Nucleic Acids Res.*, 2011.

[90] T. Syrjanen. Debugging inconsistent answer set programs. In *Proc. of NMR*, 2006.

[91] S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M.C. Rousset, and R. A. Schmidt, editors. *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 -*

September 4, 2009, Tutorial Lectures, volume 5689 of *Lecture Notes in Computer Science*. Springer, 2009.

[92] J. Tiihonen, T. Soininen, and R. Sulonen. A practical tool for mass-customising configurable products. In *Proc. of the International Conference on Engineering Design*, pages 1290–1299, 2003.

[93] N. Tran and C. Baral. Reasoning about triggered actions in AnsProlog and its application to molecular interactions in cells. In *Proc. of KR*, pages 554–564, 2004.

[94] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *IJCAR*, pages 292–297, 2006.

[95] F. Türe and E. Erdem. Efficient haplotype inference with answer set programming. In *Proc. of AAAI*, pages 1834–1835, 2008.

[96] M. D. Vos, T. Crick, J. Padget, M. Brain, O. Cliffe, and J. Needham. A multi-agent platform using ordered choice logic programming. In *In Declarative Agent Languages and Technologies (DALT'05)*, pages 72–88, 2005.

[97] M. D. Vos and D. Vermeir. Extending answer sets for logic programming agents. *Ann. Math. Artif. Intell.*, 42(1-3):103–139, 2004.