THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale I2S

Unité de recherche LIRMM

DEFEASIBLE REASONING FOR EXISTENTIAL RULES

Présentée par Abdelraouf HECHAM
Le 9 Juillet 2018

Sous la direction de Madalina CROITORU et Pierre BISQUERT

Devant le jury composé de

Madalina CROITORU, MCF HDR, Université de Montpellier, Montpellier, France	Directeur
Pierre BISQUERT, CDR 2, INRA, Montpellier, France	Co-encadrant
Guillermo Ricardo SIMARI, PU, Universidad Nacional del Sur, Bueno Aires, Argentine	Rapporteur
Nicolas MAUDET, PU, Université Pierre et Marie Curie, Paris, France	Rapporteur
Francesca TONI, PU, Imperial College of London, London, Royaume-Uni	Examinatrice
Odile PAPINI, PU, Université Aix-Marseille, Marseille, France	Examinatrice (Présidente du jury)

UNIVERSITÉ DE MONTPELLIER

Acknowledgment

To the people that brought me into existence, my parents, without which I would be enjoying the bliss of the void and would not have to write this thesis. To my brother and to my family, I could have never achieved so much without you. I love you dearly.

I was lucky twice in my life. First, when I unfortunately won the birth lottery, and second, when I had the absolute pleasure to have Madalina and Pierre as my supervisors. I remember all the insightful discussions about interesting philosophical, personal, and career matters. I can never thank them enough. I would also like to thank Clement for always being available and helping me deconstruct morality and various human reasoning topics.

A special thanks to the members of my thesis defense jury: Guillermo SIMARI, Nicolas MAUDET, Francesca TONI, and Odile PAPINI for accepting to review my work and providing me with key insights and advices.

As Albert Camus said: *"One must imagine Sisyphus happy"*, and it is quite easy to imagine me happy when one knows that I had the chance to work with GraphIK and IATE research teams, and all the people at LIRMM, they made me feel at home during our lunches and cigarette/coffee breaks.

To my friends, I am grateful for their support and help throughout the thesis, especially with the PhD defense party. I would also like to thank all the people who helped me during my thesis and all those that I have met in the conferences, workshops and social events.

Finally, last but by no means least, I would like to thank my sister for existing and for giving me a good reason to continue pushing the boulder up the mountain.

Abstract

Knowledge representation and reasoning on the Semantic Web has recently focused, due to practical rationale, on the subset of first order logic called existential rules. Reasoning amounts to answering queries over a data layer of factual knowledge and an ontological layer of deductive rules and negative constraints. These negative constraints express contradictions that may arise either from the data layer (erroneous facts that lead to inconsistency) or from the ontological layer (defeasible rules that lead to incoherence). Classical query answering in presence of contradictions is trivial since from falsehood everything follows (ex falso quodlibet). In this thesis we investigate reasoning with existential rules in presence of conflicting information and introduce defeasible existential rules reasoning. We provide three main salient results as follows. First we show that classical defeasible reasoning techniques need to be revisited for existential rules and study their theoretical and implementation related challenges. Second, we provide a new combinatorial structure that allows for diverse variants of defeasible reasoning to be captured together and study its expressivity and versatility. Third we evaluate our work with respect to the state of the art in inconsistency handling in existential rules and investigate the human appeal of such reasoning techniques.

Resumé (version courte)

La représentation des connaissances et le raisonnement sur le Web sémantique se sont récemment concentrés, pour des raisons pratiques, sur le sous-ensemble de la logique du premier ordre appelé règles existentielles. Dans cette thèse, nous étudions le raisonnement avec des règles existentielles en présence d'informations contradictoires et introduisons un raisonnement existentiel défaisible. Nous proposons trois résultats principaux: Premièrement, nous montrons que les techniques de raisonnement défaisibles classiques doivent être revisitées pour les règles existentielles et étudions leurs défis théoriques et de mise en œuvre. Deuxièmement, nous fournissons une nouvelle structure combinatoire qui permet de capturer diverses variantes du raisonnement défaisable et étudions son expressivité et sa polyvalence. Troisièmement, nous évaluons notre travail par rapport à l'état de l'art dans le traitement des incohérences et des inconsistances dans les règles existentielles et étudions l'intérêt humain de telles techniques de raisonnement.

Resumé (version longue)

Cette thèse présente une recherche originale dans le domaine de la représentation de la connaissance et du raisonnement, l'un des principaux sous-domaines de l'intelligence artificielle. Le langage pour la représentation des connaissances que nous considérons est Datalog$^{\pm}$ (règles existentielles) [Cali et al., 2010a], une famille de langages logiques pour représenter les ontologies au cœur des domaines de Systèmes de Base De Données et Web sémantique. Ce langage est largement utilisé dans le paradigme Ontology-Based Data Access (OBDA) [Poggi et al., 2008] où une ontologie (ensemble de règles) au-dessus d'une couche de données (connaissance factuelle) est utilisée pour enrichir le processus de raisonnement. Parmi les principales difficultés de cette représentation, nous nous concentrons dans cette thèse sur deux: *assurer un raisonnement fini* et *maintenir la capacité de raisonner en présence de conflits*. Les conflits proviennent de deux sources possibles, soit la connaissance factuelle est incorrecte (connue sous le nom d'inconsistance), ou les règles elles-mêmes sont contradictoires (connu sous le nom d'incohérence). La contribution de la thèse est la proposition d'un formalisme unificateur pour le raisonnement tolérant aux conflits avec des règles existentielles qui prennent en compte les mécanismes utilisés pour assurer un raisonnement fini. L'objectif est de fournir des outils qui fournissent aux ingénieurs de données une variété d'options pour raisonner avec des règles existentielles d'une manière tolérante aux conflits.

Compte tenu de la croissance rapide du Web sémantique et du grand nombre d'ontologies différentes décrivant des points de vue potentiellement

conflictuels pour le même domaine, les problèmes d'incohérence et d'inconsistance risquent de se poser. Par conséquent, la question de recherche à laquelle nous voulons répondre dans cette thèse est:

Question de Recherche

Comment peut-on raisonner avec une base de connaissances incohérente ou inconsistante exprimée à l'aide de règles existentielles?

Pour résoudre ce problème, on peut être tenté de réutiliser les techniques de Raisonnement Défaisable pour les règles existentielles. Malheureusement, cette solution n'est pas simple car elle entraîne plusieurs problèmes techniques. Les règles existentielles viennent avec un ensemble de mécanismes de raisonnement complexes afin d'assurer la décidabilité. Ces mécanismes peuvent induire une perte de chemins de raisonnement (dérivations) en fonction de l'ordre des applications de règles. Cela rend l'application directe des techniques de raisonnement défaisable à des règles existentielles peu solides. Notre problème de recherche peut alors être reformulé en un sous-ensemble de questions de recherche plus précises comme suit:

Questions de Recherche

- *Comment pouvons-nous appliquer les techniques du raisonnement défaisable aux règles existentielles?*
- *Pouvons-nous fournir un ensemble de formalismes et d'outils qui permettent le Raisonnement Défaisable avec différentes intuitions dans le contexte des règles existentielles?*
- *Peut-on comparer le raisonnement défaisable à la sémantique de réparation? Si oui, pouvons-nous fournir un formalisme unificateur pour le raisonnement tolérant aux conflits avec des règles existentielles?*

Voyons maintenant comment les contributions de la thèse abordent les trois questions de recherche mentionnées précédemment.

Contribution 1: Perte de Derivations. Le raisonnement défaisable en général est basé sur la notion de *derivation*, qui est une séquence d'applications de règles qui permet de générer de nouvelles informations. Afin de savoir si cette information tient en présence de conflits, toutes ses dérivations possibles doivent être évaluées. Le besoin d'extraire toutes les dérivations est ce qui empêche l'application directe des techniques de raisonnement défaisable aux règles existentielles, car certaines dérivations peuvent être perdues. Plus précisément, les mécanismes d'application des règles effectuent un ensem-

ble de vérifications qui assurent la décidabilité en supprimant ce qui est considéré comme une information redondante. Cela signifie que certains chemins de raisonnement peuvent être perdus et cela peut être problématique en raisonnant d'une manière défaisable. Notre contribution peut se résumer en ces points principaux:

- *Les techniques de raisonnement défaisibles ne peuvent pas être directement appliquées au langage des règles existentielles en raison du problème de perte de dérivation (certaines applications de règles peuvent être supprimées par le réducteur de dérivation de chasse). Nous définissons formellement quand et pourquoi cela se produit (la perte de dérivation peut survenir dans certains cas en fonction de la poursuite utilisée et de l'ordre dans lequel les règles sont appliquées cf. Propositions 3.1, 3.2, 3.3, et 3.4).*

- *Nous définissons le Graph de la Dépendance d'Atom et montrons comment sa construction est affectée par le chase et comment elle peut être utilisée pour extraire toutes les dérivations.*

- *Nous présentons le premier outil de Raisonnement Defaisable pour les règles existentielles (appelé DEFT) basé sur les Arbres Dialectiques et en nous appuyant sur le Graph de Dépendance d'Atom pour assurer un raisonnement correct et complet.*

- *Nous définissons le premier benchmark pour l'analyse et la classification des outils de Raisonnement Défaisif logique de premier ordre. En plus de montrer que DEFT a des performances satisfaisantes, ce benchmark donne une vision claire de ce que permettent les outils existants pour le Raisonnement Défaisable, quel est le meilleur outil à utiliser en fonction des données et des besoins, et quelles sont les lacunes actuelles des outil existant.*

Après avoir traité le problème de perte de dérivation, nous pouvons soit étendre et implémenter les différentes techniques de Raisonnement Défaisable aux règles existentielles, ou définir un nouveau formalisme capable de représenter la plupart des variantes du Raisonnement Défaisable dans une structure combinatoire unique qui prend en compte le spécificités des règles existentielles.

Contribution 2: Représentation du Raisonnement Défaisable. Étant donné la variété des intuitions de Raisonnement Défaisable et les différents degrés d'expressivité qu'un outil de Raisonnement Défaisible peut avoir, avoir un formalisme flexible qui couvre la plupart des intuitions est d'une grande valeur théorique et pratique. Notre deuxième contribution se concentre sur l'utilisation de notions d'argumentation pour définir une structure

combinatoire appelée *Statement Graphs* qui peut représenter diverses techniques de Raisonnement Défaisif utilisant des fonctions d'étiquetage flexibles. Nous utilisons ce formalisme pour mettre en œuvre un outil appelé ELDR (Existential Rules Language for Defeasible Reasoning) qui couvre la plupart des lacunes identifiées à l'aide de notre indice de référence. Cette contribution peut être résumée dans les points suivants:

- *Nous définissons des Graphiques de Déclaration qui sont un formalisme capable de représenter les Logiques Défaisibles via des fonctions d'étiquetage flexibles (blocage ou propagation d'ambiguïté, avec ou sans défaite en équipe, et échec par boucle). Ils peuvent être définis en utilisant un langage propositionnel, un langage de premier ordre (FOL) sans le quantificateur existentiel, ou des règles existentielles.*

- *Pour les règles existentielles, les Graphes Statement sont construits en utilisant une poursuite de frontière et prennent en compte la perte de dérivation pour représenter la plupart des fonctionnalités discutées pour le Raisonnement Defaisible.*

- *Nous présentons ELDR, une implémentation de Statement Graphs, qui est le premier outil permettant le blocage ou la propagation d'ambiguïtés, avec ou sans défaite d'équipe, et l'échec par bouclage pour les différentes langues considérées (propositionnel, FOL sans quantificateur existentiel et règles existentielles).*

Après avoir présenté le formalisme et l'outil du Raisonnement Défaisable avec des règles existentielles capables de capturer la plupart des intuitions, l'étape suivante consiste à utiliser ce formalisme unifiant pour essayer de comparer et éventuellement combiner la Sémantique de Réparation tolérante à l'inconsistance avec le Raisonnement Défaisable tolérant à la incohérent.

Contribution 3: Sémantiques de Réparation et Raisonnement Défaisable. Le raisonnement défaisable provient de la nécessité de raisonner avec des connaissances incomplètes par " *combler les lacunes dans les informations disponibles en faisant une sorte d'hypothèse plausible (ou souhaitable)* " [Billington et al., 2010]. D'autre part, la sémantique de réparation [Lembo et al., 2010] provient, entre autres, de la nécessité de gérer les incohérences dues à la fusion de différentes sources de données appliquées à "Ontology-Based Data Access" [Poggi et al., 2008] où une ontologie est utilisé pour accéder à un ensemble de sources de données.

Le raisonnement défaisable et la sémantique de réparation sont généralement considérés comme deux approches intrinsèquement distinctes qui répondent à des problèmes différents et sont étudiées par des communautés éloignées et, à notre connaissance, n'ont jamais été explicitement rassemblées. L'objectif de notre troisième contribution est de montrer qu'ils peuvent

être comparés sous la restriction de bases de connaissances inconsistantes mais cohérentes, et de combiner leurs intuitions et de créer une nouvelle sémantique "entre-deux". Cette contribution peut être résumée dans les points suivants:

- *Nous montrons que la sémantique de raisonnement et de réparation défaisable peut être comparée sous les restrictions de règles strictes, de faits irrécupérables et sans préférences (voir propositions 5.4, 5.5 et Figure 5.9 qui affiche le lien de productivité entre les différentes techniques).*

- *Statement Les graphes peuvent être vus comme une représentation unificatrice pour obtenir une implication équivalente à certaines techniques de Sémantique de Raisonnement et de Réparation. Cela nous permet de combiner les intuitions des deux approches. Plus précisément, les sémantiques de réparation IAR et ICAR peuvent être combinées avec l'intuition bloquante d'ambiguïté du raisonnement défaisable. La sémantique résultante semble coïncider avec le raisonnement humain dans des situations abstraites, comme le prouvent les résultats empiriques de l'expérience de la section 5.2.3.*

- *Enfin, nous montrons que les Graphes d'Énoncé peuvent être appliqués à d'autres formes de raisonnement humain que le Raisonnement Défaisable ou la Sémantique de Réparation ne peuvent représenter, à savoir la tâche de suppression où les conclusions logiques valides sont supprimées.*

L'intégration de certaines techniques de Sémantique de Réparation et de Raisonnement Défaisable dans un seul outil de raisonnement tolérant aux conflits avec des règles existentielles fournit aux ingénieurs de données une grande variété de sémantique qu'ils peuvent appliquer en fonction de leurs objectifs.

Contents

List of Figures

List of Tables

1 Introduction

This thesis presents an original research in the field of Knowledge Representation and Reasoning, one of the main sub-domains in Artificial Intelligence. The language for knowledge representation we consider is the existential rules framework (Datalog$^{\pm}$) [Cali et al., 2010a], a family of logical languages for representing ontologies at the heart of Database Systems and Semantic Web domains. This language is widely-used in the Ontology-Based Data Access (OBDA) paradigm [Poggi et al., 2008] where an ontology (set of rules) on top of a data layer (factual knowledge) is used to enrich the reasoning process. Among the main difficulties of this representation, in this thesis we focus on two: *ensuring a finite reasoning* and *maintaining the ability to reason in presence of conflicts*. Conflicts arise from two possible sources, either the factual knowledge is incorrect (known as inconsistence), or the rules themselves are contradictory (known as incoherence). The contribution of the thesis is the proposal of a unifying formalism for conflict-tolerant reasoning with existential rules that takes into account the mechanisms of ensuring a finite reasoning. The goal is to provide tools that supply data engineers with a variety of options to reason with existential rules in a conflict-tolerant manner.

This chapter is structured as follows. In Section 1.1 we introduce the general context of the thesis. Then in Section 1.2 we discuss the problem of conflicts and how they can be handled within the existential rules framework, this allows us to present the research problem alongside our contributions on this regard in Section 1.3. Finally, we conclude this chapter by highlighting the structure of the thesis.

1.1 Knowledge Representation and Reasoning

Knowledge Representation and Reasoning deals with the quest of representing real world knowledge in order to achieve human-level intelligence and reasoning faculties. This requires a trade-off between expressiveness and computation tractability as the difficulty of reasoning increases proportionally with the expressive power of the underlying logical language [Levesque

and Brachman, 1987]. With the rapid growth of data and ontologies for the Semantic Web in the last two decades, an emergent need for tractable yet expressive logical languages has been raised. At first, Description Logics [Baader et al., 2005] where introduced, then with the combination of Relational Databases with Logic Programming, another language has been proposed [Cali et al., 2010a] based on Datalog [Ceri et al., 1989] where data is represented as facts alongside rules written in first order language. Datalog can be seen as the adaptation of "*... Prolog, which has a 'small-data world view to a 'large-data' world*" [Ramakrishnan and Ullman, 1995]. The extension of Datalog to Datalog$^{\pm}$ by the addition of the existential quantifier to account for unknown individuals (the person X has a parent whose name is unknown) has made Datalog$^{\pm}$ general enough to capture a variety of Description Logics families [Cali et al., 2010a]. This generality promotes Datalog$^{\pm}$ as an adequate language for representing ontologies in the Semantic Web [Calì et al., 2012]. Due to historical reasons Datalog$^{\pm}$ is also called **existential rules framework**, and as such we may use the two names interchangeably.

The addition of the existential quantifier makes the reasoning process undecidable since applying rules might not stop as it generates new individuals at each rule application. Nevertheless, different classes of rules based on their syntactic structure can be defined to ensure that the forward chaining mechanism of applying rule becomes finite. The restriction to decidable classes of rules solves the first problem of *ensuring a finite reasoning*. However, the second problem of *maintaining the ability to reason in presence of conflicts* remains.

1.2 Incoherence and Inconsistence Handling

Conflicts arise in Knowledge Representation from two possible sources, either the factual knowledge is *inconsistent*, in the sense that applying the rules on that specific set of facts generates $\bot$ (falsum), or the set of rules is *incoherent* in the sense that applying the set of rules on any set of facts will always lead to falsehood. Inconsistency can be seen as a special case of incoherence [Flouris et al., 2006]. The following Examples 1.1 and 1.2 describe inconsistence and incoherence.

Example 1.1 (Inconsistence). *Consider the following legal situation where we want to know if the defendant Alice is guilty or not of a crime. Suppose there are no means by which we can verify the truthfulness and reliability of the factual knowledge.*

- *Factual knowledge: There is a piece of evidence "e1" incriminating the defendant Alice, there is another piece of evidence "e2" absolving Alice, and Alice has an alibi.*

- *Rules: If there is an incriminating evidence against a defendant then he is responsible of the crime. If there is an absolving evidence for a defendant then he is not responsible of the crime. If a defendant is proven responsible of the crime then he is guilty, and if a defendant has an alibi then he is innocent (not guilty).*

The set of factual knowledge is inconsistent because we can generate the fact that Alice is responsible and not responsible of the crime (given evidence "e1" and "e2"), we can also generate the fact that she is guilty (given that she is responsible of the crime) and that she is innocent (given her alibi) which are contradictions. However, the set of rules in itself is coherent. Indeed, we can a find a set of facts such that all rules can be applied and no contradiction can be generated. For example, there is an evidence incriminating Bob, there is an evidence absolving Alice, and Alice has an alibi.

Example 1.2 (Incoherence). *Consider the following set of rules: penguins are birds, birds fly, penguins do not fly, and one cannot fly and not fly at the same time. Any set of factual knowledge on which these rules are applicable will always lead to contradiction (fly and not fly).*

Inconsistence and Incoherence management is a well-established research discipline in Knowledge Representation. In fact, it dates since the pre-Socratic era with the concept of logical contradiction "*The great Parmenides from beginning to end testified.. 'Never shall this be proved - that things that are not are'* " (Plato, Sophist, 237A). The principle of explosion states that given an inconsistent knowledge base in a logical language $\mathcal{L}$, one can derive that any formula of the language is true. This problem of inconsistence is addressed within the context of existential rules using *Repair Semantics* [Lembo and Ruzzi, 2007] where the set of facts is split into maximally (with respect to inclusion) consistent sets of facts called repairs, these can be seen as possible consistent worlds. However, for Repair Semantics to properly yield results, the set of rules must be coherent [Deagustini et al., 2015]. This leaves *unsolved* the problem of incoherence for existential rules.

In order to solve this problem, we can turn to incoherence-tolerant techniques such as *Defeasible Reasoning* [Pollock, 1987] which is a formalism for non-monotonic reasoning with low computational cost [Nute, 1988]. The idea behind Defeasible Reasoning is to detect conflicts and solve them by choosing the "preferred" outcome. Unfortunately, there is no universally valid way to reason defeasibly. An inherent characteristic of Defeasible Reasoning is its systematic reliance on a set of intuitions and rules of thumb, which have been longly debated between logicians [Horty et al., 1987, Makinson and Schlechta, 1991, Prakken, 2002, Antoniou, 2006]. For instance, one of these intuitions is *ambiguity handling* also known as *zombie paths* described in the following example.

Example 1.3 (Ambiguity Handling). *Consider the knowledge in the previous Example 1.1: there is a doubt about the responsibility of Alice in the crime (given evidence "e1" and "e2"). Some argue that since the guiltiness of Alice is based on a doubtful fact (her responsibility of the crime), her innocence remains unchallenged, and subsequently conclude that Alice is innocent of the crime. However others argue that there is evidence for her responsibility of the crime and therefore we should not be able to say if whether or not she is innocent.*

The domain of Defeasible Reasoning is vivid and full of approaches and intuitions, especially with the various argumentation-based techniques. We limit the scope of this thesis to the following intuitions: Ambiguity Handling, Team Defeat, Floating Conclusions, and Handling of Strict Rules. One of the main approaches we consider in this thesis are *Defeasible Logics* [Antoniou et al., 2000a], *Dialectical Trees* [García and Simari, 2004], and *Argumentation Grounded Semantics* [Dung, 1995]. The problem of incoherence for existential rule can be solved by extending and applying these techniques to Datalog$^{\pm}$.

1.3 Research Problem and Contributions

Given the rapid growth of the Semantic Web and the large number of different ontologies describing potentially conflicting points of view for the same domain, the problems of inconsistence and incoherence are likely to arise. Therefore, the research question we want to answer in this thesis is:

Research Question

How can we reason with an inconsistent or incoherent knowledge base expressed using existential rules?

To address this problem one can be tempted to reuse Defeasible Reasoning techniques for existential rules. Unfortunately this is not a straightforward solution as it leads to several technical challenges. Existential rules come with a set of intricate reasoning mechanisms in order to ensure decidability. These mechanisms might induce a loss of reasoning paths (derivations) depending on the order of rule applications. This makes the direct application of Defeasible Reasoning techniques to existential rules unsound. Our research problem can then be reformulated into a subset of more precise research questions as follows:

Research Questions

- *How can we apply Defeasible Reasoning techniques to Existential Rules?*
- *Can we provide a set of formalisms and tools that allow for Defeasible Reasoning with different intuitions within the context of existential rules?*
- *Can Defeasible Reasoning be compared to Repair Semantics? If so, can we provide a unifying formalism for conflict-tolerant reasoning with existential rules?*

Let us now see how the contributions of the thesis address all three of the previously mentioned research questions.

1.3.1 Contribution 1: Preventing Derivation Loss

Defeasible reasoning in general is based on the notion of *derivation*, which is a sequence of rule applications that allows to generate new information. In order to know if this information holds in presence of conflicts, all its possible derivations must be evaluated. The need to extract all derivations is what prevents the direct application of Defeasible Reasoning techniques to existential rules as some derivations can be lost. More precisely, the rule application mechanisms perform a set of verifications that ensure decidability by removing what is considered redundant new information. This means that certain paths of reasoning might be lost and this can be problematic when reasoning in a defeasible manner. Our contribution can be summed up in these main points:

- *Defeasible reasoning techniques cannot be directly applied to the existential rule language due to the derivation loss problem (some rule applications might be removed by the chase derivation reducer). We formally define when and why this happens (derivation loss can occur in certain cases depending on the used chase and the order in which rules are applied cf. Propositions 3.1, 3.2, 3.3, and 3.4).*
- *We define the Graph of Atom Dependency and show how its construction is affected by the chase and how it can be used to extract all derivations.*
- *We present the first Defeasible Reasoning tool for existential rules (called DEFT) based on Dialectical Trees and relying on the Graph of Atom Dependency to ensure a sound and complete reasoning.*
- *We define the first benchmark for first order logic Defeasible Reasoning tools analysis and classification. Beside showing that DEFT has*

satisfactory performance, this benchmark provides a clear view of what existing tools for Defeasible Reasoning allow for, what is the best tool to use depending on the data and requirements at hand, and what are the current gaps that are not covered yet by any tool.

After dealing with the derivation loss problem, we can either extend and implement the different Defeasible Reasoning techniques to existential rules, or we can define a new formalism that is able to represent most variants of Defeasible Reasoning in a single combinatorial structure that takes into account the specificities of existential rules.

1.3.2 Contribution 2: Representing Defeasible Reasoning

Given the variety of Defeasible Reasoning intuitions and the different degrees of expressiveness a Defeasible Reasoning tool can have, having one flexible formalism that covers most intuitions is of great theoretical and practical value. Our second contribution focuses on using argumentation notions to define a combinatorial structure called *Statement Graphs* that can represent various Defeasible Reasoning techniques using flexible labeling functions. We use this formalism to implement a tool called ELDR (Existential rules Language for Defeasible Reasoning) that covers most gaps identified using our benchmark. This contribution can be summed up in the following points:

- *We define Statement Graphs which are a formalism able to represent Defeasible Logics via flexible labeling functions (ambiguity blocking or propagation, with or without team defeat, and with failure-by-looping). They can be defined using a propositional language, a first order language (FOL) without the existential quantifier, or existential rules.*

- *For existential rules, Statement Graphs are constructed using a frontier chase and account for derivation loss to represent most of the discussed features for Defeasible Reasoning.*

- *We present ELDR, an implementation of Statement Graphs, which is the first tool that allows for ambiguity blocking or propagation, with or without team defeat, and failure-by-looping for the different considered languages (propositional, FOL without existential quantifier, and existential rules).*

After presenting the formalism and the tool for Defeasible Reasoning with existential rules that can capture most intuitions, the next step is to make use of this unifying formalism to try to compare and possibly combine the inconsistence-tolerant Repair Semantics with the incoherence-tolerant Defeasible Reasoning.

1.3.3 Contribution 3: Unifying Repair Semantics and Defeasible Reasoning

Defeasible reasoning originates from the need to reason with incomplete knowledge by "*filling the gaps in the available information by making some kind of plausible (or desirable) assumptions*" [Billington et al., 2010]. Repair semantics [Lembo et al., 2010] on the other hand originate, among others, from the need to handle inconsistency that arises due to merging of different data sources applied to "Ontology-Based Data Access" [Poggi et al., 2008] where an ontology is used to access a set of data sources.

Like apples and oranges, Defeasible Reasoning and Repair Semantics are generally seen as two inherently distinct approaches that answer different problems and are studied by distant communities and, to the best of our knowledge, have never been explicitly put together. The objective behind our third contribution is to show that they can be compared under the restriction of inconsistent but coherent knowledge bases, and to combine their intuitions and to create new "in-between" semantics. This contribution can be summed up in the following points:

- *We show that Defeasible Reasoning and Repair Semantics can be compared under the restrictions of strict rules, defeasible facts and no preferences (cf. propositions 5.4, 5.5 and Figure 5.9 that displays the productivity link between the different techniques).*
- *Statement Graphs can be seen as a unifying representation to obtain equivalent entailment as some Defeasible Reasoning and Repair Semantics techniques. This allows us to combine the intuitions of both approaches. Specifically, IAR and ICAR Repair Semantics can be combined with the ambiguity blocking intuition of Defeasible Reasoning. The resulting semantics seems to coincide with human reasoning under abstract situations as supported by the empirical results of the experiment in Section 5.2.3.*
- *Lastly, we show that Statement Graphs can be applied to other forms of human reasoning that neither Defeasible Reasoning nor Repair Semantics can represent, namely, the suppression task where valid logical conclusions are suppressed.*

Integrating some Repair Semantics and Defeasible Reasoning techniques into a single tool for conflict-tolerant reasoning with existential rules provides data engineers with a wide variety of semantics that they can apply depending on their objectives.

1.4 Thesis Structure

The structure of this thesis is as follows:

Chapter 2. This chapter introduces necessary preliminaries on how to represent knowledge using existential rules and how to reason with this knowledge (chase, decidable classes of rules, and the query entailment problem). It also defines the different types of conflict (inconsistency and incoherence) and presents Defeasible Reasoning as a way to maintain the ability to reason in presence of conflicts. We discuss the different Defeasible Reasoning intuitions that we consider throughout the remainder of the thesis, present Defeasible Logics, Dialectical Trees, and some Argumentation Semantics, study how they can be applied to different logical languages and when they might coincide.

Chapter 3. In this chapter we present the obstacle that prevents the direct application of Defeasible Reasoning techniques to existential rules, namely *derivation loss*. We define the derivation loss problem and when and why it might happen depending on the chosen chase and the order of rules applications. We then introduce the Graph of Atom Dependency as a solution to this problem, we study its construction for the different chases and how it can be used to extract all derivations. We then present the first tool (called DEFT) for Defeasible Reasoning with existential rules based on Dialectical Trees and define a benchmark for first order logic Defeasible Reasoning tools analysis and classification. We use this benchmark to (1) make sure DEFT has satisfactory performance, (2) provide a clear view of what existing tools for Defeasible Reasoning allow for, (3) help data engineers chose the best tool to use, and (4) identify the current gaps that are not covered yet by any tool. This chapter builds upon the work published in [Hecham et al., 2017b, **?**].

Chapter 4. We introduce in this chapter "Statement Graph" which is a formalism able to represent Defeasible Logics via flexible labeling functions. This formalism can be constructed using a propositional language, a first order language without the existential quantifier, or existential rules. For existential rules we use the frontier chase and account for derivation loss. We implement this structure in a tool called ELDR which is the first tool that allows for Defeasible Reasoning with ambiguity blocking or propagation, with or without team defeat, and failure-by-looping for the different considered languages. We run the previously defined benchmark to make sure the performance is satisfactory compared to the existing tools. This chapter build upon the work published in [Hecham et al., 2018].

Chapter 5. In this chapter, we present the Statement Graph's labellings for the IAR and ICAR Repair Semantics. We investigate the links (productivity and complexity) between Defeasible Reasoning and Repair Semantics and when they can be compared. We then show that the intuitions stemming

from the different domains can be combined to produce new semantics; these resulting semantics seem to coincide with human reasoning under abstract situations as supported by the results of an empirical experiment. Lastly we discuss other forms of human reasoning that neither Defeasible Reasoning nor Repair Semantics can represent, namely, the *suppression task* and show how Statement Graphs can be used to represent such reasoning. This chapter builds upon the work published in [Hecham et al., 2017a, Hecham et al., 2018].

Chapter 6. This chapter concludes, summarizes our contributions, and presents a number of interesting future research problems based on possible extensions of both this work and our work published in [Hecham et al., 2016, Bisquert et al., 2016, Bisquert et al., 2017] (but not included in this thesis).

2
Preliminaries

In this chapter we discuss the two main difficulties of knowledge representation, namely, *ensuring a finite reasoning* and *maintaining the ability to reason in presence of conflicts*. Representing knowledge requires a trade-off between expressiveness and computational tractability, since higher expressiveness might lead to infinite reasoning. We start by providing an introduction to knowledge representation with the existential rules logical fragment along with its forward chaining inference mechanism called "chase". We present the reasoning problem of *query entailment* and introduce the notion of finite classes of existential rules for which the chase is guaranteed to halt. Since knowledge representation might contain conflicts, we define the different types of conflicts (inconsistence and incoherence) and discuss the conflict-tolerant Defeasible Reasoning approach along with its intuitions and different formalisms. We study their complexity and when they might

coincide in order to prepare the stage for applying these conflict-tolerant techniques to existential rules.

Research Questions in this Chapter

- *How is knowledge represented using existential rules and how can we reason with this knowledge (query entailment problem)?*
- *What are the different types of conflict (inconsistence and incoherence) and how can we maintain the ability to reason in presence of conflicts?*
- *What are the different intuitions and formalisms for the conflict-tolerant Defeasible Reasoning approach?*

2.1 Existential Rules Framework

The aim of knowledge representation and reasoning is to achieve human-level intelligence and reasoning faculties. The biggest dilemma in this case is the trade-off between expressiveness and computational tractability of a given logical language [Levesque and Brachman, 1987]. Existential rules are a first order logical language that emerged from the intersection of Knowledge Representation, Database Systems, and Semantic Web. It has the ability to express knowledge about "unknown" individuals (e.g. "every human has a parent" this parent might be unknown but its existence still holds). This level of expressiveness comes at the high cost of undecidability (the reasoning mechanism can be infinite), that is why different decidable fragments of existential rules have been defined under the name of $Datalog^{\pm}$ [Calì et al., 2012] which is a generalization of Datalog [Ceri et al., 1989] and certain fragments of Description Logics [Baader et al., 2005].

2.1.1 Logical Language

We consider a first-order logical (FOL) language $\mathcal{L}$ with no function symbols (except for constants) built with the existential and universal quantifiers $(\exists, \forall)$ and the implication and conjunction connectives $(\rightarrow, \wedge)$ on a vocabulary $\mathcal{V}oc = (\mathcal{P}, \mathcal{C})$ composed of a finite set of predicates $\mathcal{P}$ and a potentially infinite set of constants $\mathcal{C}$. Each predicate $p \in \mathcal{P}$ is associated with a positive integer which is called the *arity* of p. We are also given an infinite set of variables $\mathcal{V}$, and an infinite set of existential "fresh" variables $\mathcal{N}$ (called "nulls", which act as placeholders for unknown constants, and can thus be seen as variables). We denote variables by uppercase letters $X, Y, Z, etc.$, constants by lowercase letters $a, b, c, etc.$, and fresh variables (nulls) by $Null_1, Null_2, etc.$

A logical language is a symbolic representation of some knowledge about the world. For these symbols to have meaning, they need to be 'mapped' to elements of the world. This is done using an *interpretation* function which maps predicates and constants symbols to elements of the domain of interpretation.

Definition 2.1 (Interpretation). *An interpretation of a logical language $\mathcal{L}$ built on a vocabulary $\mathcal{V}oc = (\mathcal{P}, \mathcal{C})$ is a pair $(\mathcal{D}, \mathcal{I})$ where $\mathcal{D}$ is a non-empty set called the interpretation domain and $\mathcal{I}$ is an interpretation function of the symbols of $\mathcal{L}$ such that:*

1. *for each constant $c \in \mathcal{C}$, $\mathcal{I}(c) \in \mathcal{D}$.*
2. *for each predicate $p \in \mathcal{P}$ of arity k, $\mathcal{I}(p) \subseteq \mathcal{D}^k$.*
3. *for each pair (c, c') of distinct constants in $\mathcal{C}$, $\mathcal{I}(c) \neq \mathcal{I}(c')$.*

The third condition in the above definition corresponds to the *unique name assumption* and indicates that different constants should be interpreted by different elements of the interpretation domain. This assumption is often made in knowledge representation, however note that as long as equality between constants is not considered (which is the case in this thesis), adopting the unique name assumption or not does not make any difference in the considered reasoning tasks [Baget et al., 2011a].

Knowledge about the world is expressed using formulas built from the logical language. The basic building blocks are called atomic formulas (or atoms).

Definition 2.2 (Atom). *An atom over $\mathcal{V}oc$ is of the form $p(t_1, \ldots, t_k)$, where $p \in \mathcal{P}$ is a predicate of arity k and $t_i \in \mathcal{V} \cup \mathcal{C} \cup \mathcal{N}$ is either a variable, a constant, or a null.*

Given a formula Φ built on a language $\mathcal{L}$, we note $terms(\Phi)$ and $vars(\Phi)$ respectively the terms and variables (including nulls) occurring in Φ. $\top$ (tautology) and $\bot$ (falsity) are allowed and considered themselves atoms. A *ground atom* contains only constants.

Example 2.1 (Atoms, conjunctions, and interpretations). *Consider the following vocabulary $\mathcal{P} = \{p, q\}$, $\mathcal{C} = \{a, b\}$, then "$\exists X\, p(a, X)$" is an atom, "$p(a, b)$" is a ground atom, and "$\exists X\, (p(a, X) \wedge q(X))$" is a conjunction of atoms. An interpretation might map "p" to the concept of parenthood, "$p(a, b)$" might be interpreted as the individual "a" is a parent of "b" (e.g. Alice is a parent of Bob). "$\exists X\, p(a, X)$" might be interpreted as there exists an individual such that "a" is its parent.*

A basic form of knowledge is factual knowledge which is represented using *facts*. Usually a fact is a ground atom, however to account for knowledge expressing the existence of unknown constants (nulls), the definition of

fact is generalized to an atom that contains constants or nulls (existentially quantified variables).

Definition 2.3 (Fact). *A fact on a language $\mathcal{L}$ is an existentially closed atom on $\mathcal{L}$. It is of the form $\exists \vec{X}\, p(\vec{a}, \vec{X})$ where $p \in \mathcal{P}$ is a predicate, $\vec{a}$ is a finite (potentially empty) set of constants, and $\vec{X}$ is a finite (potentially empty) set of existentially quantified variables.*

Please note that for the purposes of this thesis, *a fact is not a conjunction.* To be able to manipulate conjunctions as sets of facts, existential variables are represented using nulls.

Notation 2.1 (From existential variables to nulls). *An existential variable can be represented as a "fresh" Skolem term by removing the existential quantifier and replacing the variable with a null. This null has to be "fresh" (or "safe") meaning that it has not been used before. For example, $\exists X\, p(a, X)$ can be represented as $p(a, Null_1)$ as long as $Null_1$ is fresh (has not been used before).*

Notation 2.2 (From conjunctions to sets of facts). *A conjunction of facts can be represented as a set of facts by removing the existential quantifier and replacing the variables with nulls. For example, $\exists X\, (p(a, X) \wedge q(X))$ can be represented as $\{p(a, Null_1), q(Null_1)\}$ assuming $Null_1$ is fresh.*

A **model** of a formula built on $\mathcal{L}$ is an interpretation of $\mathcal{L}$ that makes this formula true by considering the classical interpretation of logical connectives and quantifiers.

Definition 2.4 (Logical Consequence and Equivalence). *Given a language $\mathcal{L}$ and two formulas Φ_1 and Φ_2 on $\mathcal{L}$, Φ_2 is a (logical) consequence of Φ_1 (denoted $\Phi_1 \models \Phi_2$) if all models of Φ_1 are models of Φ_2. Φ_1 and Φ_2 are said to be logically equivalent (denoted $\Phi_1 \equiv \Phi_2$) if $\Phi_1 \models \Phi_2$ and $\Phi_2 \models \Phi_1$.*

One of the relevant problems in knowledge representation is the *entailment* problem, which is asking whether a formula is a consequence of another formula. This can be expressed on facts as follows: *given two facts f_1 and f_2, is it true that f_2 is a consequence of f_1 (i.e. $f_1 \models f_2$)?* It is well known that $f_1 \models f_2$ if and only if there exists a *homomorphism* from f_2 to f_1 [Baget et al., 2011a].

Definition 2.5 (Substitution and Homomorphism). *Let $\vec{X}$ be a set of variables and $\vec{T}$ a set of terms. a substitution of $\vec{X}$ to $\vec{T}$ is a mapping form $\vec{X}$ to $\vec{T}$ (notation $\vec{X} \to \vec{T}$). A homomorphism π from an atom a_1 to an atom a_2 is a substitution of each occurrence of $X \in vars(a_1)$ by an element in $terms(a_2)$. The resulting atom after the substitution is denoted $\pi(a_1)$. A homomorphism π from a set of atoms S to a set of atoms S' is a substitution of $vars(S)$ to $terms(S')$ such that $\pi(S) \subseteq S'$.*

Example 2.2 (Homomorphism). *The atom $p(a, Null_1)$ can be mapped to the atom $p(a, b)$ by the homomorphism $\pi = \{Null_1 \rightarrow b\}$ that substitutes $Null_1$ by b. Therefore $p(a, b) \vDash p(a, Null_1)$.*

Notation 2.3 (Homomorphism restriction $\pi|_{\vec{X}}$). *Given a homomorphism π, we denote by $dom(\pi)$ the domain of π. Given a set of variables $\vec{X}$, we denoted the restriction of π to $\vec{X}$ by $\pi|_{\vec{X}} = \{(X, \pi(X)) \mid X \in dom(\pi) \cap \vec{X}\}$.*

The entailment problem is generally expressed using *queries* (query answering problem), specifically conjunctive queries which are an existentially closed conjunctions of atoms. These can be seen as asking if there is a set of constants and nulls that make an existentially closed conjunction of atoms a consequence of the set of facts.

Definition 2.6 (Query). *A Conjunctive Query (CQ) is an existentially closed conjunction of atoms of the form $Q(\vec{X}) = \exists \vec{Y}\, \Phi(\vec{X}, \vec{Y})$, where $\vec{X}$ is a set of variables, $\vec{Y}$ is a set of existential variables (possibly with constants) and Φ is a conjunction of atoms. A Boolean Conjunctive Query (BCQ) is a conjunctive query of the form $Q() = \exists \vec{Y}\, \Phi(\vec{Y})$.*

The answers to a conjunctive query $Q(\vec{X}) = \exists \vec{Y}\, \Phi(\vec{X}, \vec{Y})$ over a set of formulas $\mathcal{F}$ is the set of all tuples (constants and nulls) that if, substituted with $\vec{X}$ and $\vec{Y}$, make Φ a consequence of $\mathcal{F}$. The answer to a boolean conjunctive query is either *true* or *false*, and it is *true* over a set of facts $\mathcal{F}$ if and only if it is a consequence of $\mathcal{F}$, otherwise it is *false*.

Example 2.3 (Conjunctive and boolean queries). *Consider the query $Q(X) = \exists Y\, p(X, Y)$, the answers to this query over the set of facts $\mathcal{F} = \{p(a, b), p(c, Null_1)\}$ are $\{a, c\}$ because there is a homomorphism $\pi_1 = \{X \rightarrow a, Y \rightarrow b\}$ from Q to $p(a, b)$, and there is a homomorphism $\pi_2 = \{X \rightarrow c, Y \rightarrow Null_1\}$ to $p(c, Null_1)$. The answer to the BCQ $Q() = \exists X, Y\, p(X, Y)$ is true (because it can be mapped to $\mathcal{F}$ using π_1 or π_2).*

2.1.2 Rules and Reasoning

Rules are formulas that allow the enrichment of a set of facts with new deduced knowledge. These rules generally encode domain-specific implications, for example "if X is a cat then X is an animal". Existential rules [Baget et al., 2011a] are general rules that account for unknown individuals, they are also known as Tuple Generating Dependencies (TGD) [Abiteboul et al., 1995], Conceptual Graphs rules [Salvat and Mugnier, 1996], Datalog$^{\exists}$ rules [Cali et al., 2013], etc.

Definition 2.7 (Existential Rule). *An existential rule (or simply a rule) r is a formula of the form $\forall \vec{X}, \vec{Y}\ \left(\mathcal{B}(\vec{X}, \vec{Y}) \rightarrow \exists \vec{Z}\ \mathcal{H}(\vec{X}, \vec{Z})\right)$ where $\vec{X}, \vec{Y}$ are tuples of variables, $\vec{Z}$ is a tuple of existential variables, and $\mathcal{B}$, $\mathcal{H}$ are*

finite non-empty conjunctions of atoms respectively called body and head of r and denoted $Body(r)$ and $Head(r)$. The frontier of r (denoted $fr(r)$) is the set of variables occurring in both the body and the head of r i.e. $fr(r) = vars(Body(r)) \cap vars(Head(r))$.

Rules are used to *infer* new knowledge starting from an initial set of facts based on the notion of *rule application*.

Definition 2.8 (Rule Application). *A rule r is said to be applicable to a set of facts $\mathcal{F}$ if there is a homomorphism π from $Body(r)$ to $\mathcal{F}$. In that case, the application of r to $\mathcal{F}$ according to π (denoted $\alpha(\mathcal{F}, r, \pi)$) adds to $\mathcal{F}$ a set of facts $\pi^{safe}(Head(r))$ where π^{safe} ensures that existential variables are replaced with fresh nulls.*

Example 2.4 (Rule application). *Consider the rule r stating that two siblings have the same parent: $\forall X, Y\ siblingOf(X, Y) \rightarrow \exists Z\ parentOf(Z, X) \wedge parentOf(Z, Y)$. This rule is applicable to the set $\mathcal{F} = \{siblingOf(alice, bob)\}$ using the homomorphism $\pi = \{X \rightarrow alice, Y \rightarrow bob\}$. Therefore $\alpha(\mathcal{F}, r, \pi) = \mathcal{F} \cup \{parentOf(Null_1, alice), parentOf(Null_1, bob)\}$ assuming $Null_1$ is safe.*

Notation 2.4 (Rules with Atomic Head). *In general, rules might have a conjunction of atoms in their head, however for the purposes of this thesis, we only consider rules with one atom in their head. Any set of rules can be transformed to a set of rules with atomic head [Baget et al., 2011a]. For example, the rule $\forall X, Y\ siblingOf(X, Y) \rightarrow \exists Z\ parentOf(Z, X) \wedge parentOf(Z, Y)$ can be transformed to a set of three rules with atomic heads:*

1. $\forall X, Y\ siblingOf(X, Y) \rightarrow \exists Z\ p(X, Y, Z)$
2. $\forall X, Y, Z\ p(X, Y, Z) \rightarrow parentOf(Z, X)$
3. $\forall X, Y, Z\ p(X, Y, Z) \rightarrow parentOf(Z, Y)$

A rule is applicable on a set of facts if there is a homomorphism from the body of the rule to this set of facts, furthermore, a rule might not be applicable right away but could become applicable after some new knowledge is generated by another rule, which might make another rule applicable and so on. This sequence of rule applications is called a *derivation*. Normally, a derivation is a sequence of facts generated at each rule application, however, we generalize this notion to include the rule and the homomorphism used at each step.

Definition 2.9 (Derivation). *Given a set of facts $\mathcal{F}$ and a set of rules $\mathcal{R}$, a derivation of $\mathcal{F}$ with respect to $\mathcal{R}$ is a (potentially infinite) sequence δ of D_i s.t. D_i is a tuple $(\mathcal{F}_i, r_i, \pi_i)$ composed of a set of facts $\mathcal{F}_i$, a rule r_i and a homomorphism π_i from $Body(r_i)$ to $\mathcal{F}_i$ where: $D_0 = (\mathcal{F}_0 = \mathcal{F}, \emptyset, \emptyset)$, and $\mathcal{F}_i = \alpha(\mathcal{F}_{i-1}, r_i, \pi_i)$. We denote by $Facts(D_i)$, $Rule(D_i)$, and $Homo(D_i)$ the set of facts, rule and homomorphism of a tuple D_i.*

A derivation can be infinite as a rule can be applied again and again without restrictions as shown in the following Example 2.5.

Example 2.5 (Derivation). *Consider the set of facts $\mathcal{F}$ stating that bob is a male human, and the rules $\mathcal{R}$ stating that any human has a parent and that a male human is a man.*

- $\mathcal{F} = \{human(bob),\ male(bob)\}$.
- $\mathcal{R} = \{r_1 : \forall X\, human(X) \rightarrow \exists Y\, parentOf(Y, X),$
  $r_2 : \forall X\, male(X) \wedge human(X) \rightarrow man(X)\}$.

A possible derivation of $\mathcal{F}$ w.r.t $\mathcal{R}$ is:

$$\begin{aligned}\delta = \langle &(\mathcal{F}, \emptyset, \emptyset), (\mathcal{F}_1 = \mathcal{F}_0 \cup \{man(bob)\}, r_2, \pi_1 = \{X \rightarrow bob\}),\\ &(\mathcal{F}_2 = \mathcal{F}_1 \cup \{parentOf(Null_1, bob)\}, r_1, \pi_2 = \{X \rightarrow bob\}),\\ &(\mathcal{F}_3 = \mathcal{F}_2 \cup \{man(bob)\}, r_2, \pi_3 = \{X \rightarrow bob\}),\\ &(\mathcal{F}_4 = \mathcal{F}_3 \cup \{parentOf(Null_2, bob)\}, r_1, \pi_4 = \{X \rightarrow bob\}),\\ &\ldots\rangle.\end{aligned}$$

A derivation for a specific fact f is a finite minimal sequence of rule applications starting from a set of facts $\mathcal{F}$ and ending with a rule application that generates f.

Definition 2.10 (Derivation for a Fact). *Given some sets of facts $\mathcal{F}$ and rules $\mathcal{R}$, a derivation for a fact f is a finite derivation $\delta = \langle D_0, \ldots, D_n\rangle$ of $\mathcal{F}' \subseteq \mathcal{F}$ w.r.t. $\mathcal{R}$ such that:*

1. *$f \in Facts(D_n)$ (i.e. the last rule application contains f).*
2. *δ is minimal i.e. there does not exist another derivation $\delta' = \langle D'_0, \ldots, D'_m\rangle$ for f such that:*
   - *$Facts(D'_0) \subset Facts(D_0)$ and*
   - *$\bigcup_{D' \in \delta'}(Rule(D'), Homo(D')) \subset \bigcup_{D \in \delta}(Rule(D), Homo(D))$.*

Example 2.6 (Derivation for a Fact). *Consider the previous Example 2.5, a derivation from $\mathcal{F}$ to man(bob) is the sequence:*

$$\delta_1 = \langle(\mathcal{F}_0 = \{human(bob), male(bob)\}, \emptyset, \emptyset), (\mathcal{F}_1 = \mathcal{F}_0 \cup \{man(bob)\}, r_2, \pi)\rangle.$$

Query answering over a set of facts and rules can be done by generating all possible knowledge then finding homomorphisms from the queries to this "saturated" set of facts. In order to generate this knowledge, rules are applied in a breadth first manner. A breadth-first derivation is obtained by considering at each "breadth-first" step all possible rule applications on the current set of facts and applying them all before moving to the next step.

Definition 2.11 (Breadth-First Derivation). *Given a set of facts $\mathcal{F}$ and a set of rules $\mathcal{R}$, a breadth-first derivation of $\mathcal{F}$ w.r.t. $\mathcal{R}$ is a derivation $\delta = \langle(\mathcal{F}_0 = \mathcal{F}, \emptyset, \emptyset), \ldots, (\mathcal{F}_i, r_i, \pi_i), \ldots\rangle$ such that for all $i < j$, if $(\mathcal{F}_{i+1} \backslash \mathcal{F}_i) \cap \pi_j(Body(r_j)) \neq \emptyset$ then for all $k > j$, $\pi_k(Body(r_k)) \not\subseteq \mathcal{F}_i$.*

The above definition ensures that if a rule is applied on some atoms generated by a rule application $i+1$ then no rule application afterwards can use *only* the atoms in $\mathcal{F}_i$. Intuitively, once we go to the next breadth-first step, we cannot apply a rule that could have been applied in a previous step according to the same homomorphism.

An exhaustive breadth-first derivation ensures that all rules have been applied according to all possible homomorphisms. An exhaustive derivation may be infinite and might contain "redundant" rule applications, however removing these "redundant" rule applications might make the exhaustive derivation finite. The role of a *chase* is to remove rule applications that it considers redundant.

2.1.3 Chase and Finite Expansion Set

In order to answer queries over a set of facts and rules, the exhaustive derivation has to be finite. A chase is a mechanism that takes an exhaustive derivation and removes what it considers "redundant" rule applications using a *derivation reducer*. We use the formalization of [Rocher, 2016] for its simplicity to define a derivation reducer and a chase.

Definition 2.12 (Derivation Reducer). *Given a set of facts $\mathcal{F}$ and a set of rules $\mathcal{R}$, a derivation reducer σ is a function that takes a rule application tuple $D_i = (\mathcal{F}_i, r_i, \pi_i)$ in a derivation $\delta = \langle D_0, \ldots, D_i, \ldots\rangle$ of $\mathcal{F}$ w.r.t. $\mathcal{R}$ and returns a rule applications tuple $\sigma(D_i) = (\mathcal{F}'_i, r_i, \pi_i)$ such that $\mathcal{F}'_i \equiv \mathcal{F}_i$.*

Definition 2.13 (σ-Chase). *Given a set of facts $\mathcal{F}$, a set of rules $\mathcal{R}$, a derivation reducer σ, and an exhaustive breadth first derivation $\delta = \langle D_0, \ldots, D_i, \ldots\rangle$ of $\mathcal{F}$ w.r.t. $\mathcal{R}$: $\sigma\text{-}chase(\mathcal{F}, \mathcal{R}) = \langle\sigma(D_0), \ldots, \sigma(D_i), \ldots\rangle$ and $\sigma(D_i) \in \sigma\text{-}chase(\mathcal{F}, \mathcal{R})$ if and only if $Facts(\sigma(D_i)) \neq Facts(\sigma(D_{i-1}))$.*

The above definition ensures that only non redundant "meaningful" rule applications are kept (i.e. rule applications that generate something new according to the derivation reducer). A chase is finite if there is a breadth-first rule application step k such that for all D_j at step k, no new facts are generated [Baget et al., 2014b].

Applying a chase on a set of facts $\mathcal{F}$ and a set of rules $\mathcal{R}$ generates the saturated set of facts $\mathcal{F}^*$ that contains all initial and generated facts.

Definition 2.14 (Saturated Set of Facts). *Given a set of facts $\mathcal{F}$ and a set of rules $\mathcal{R}$, the saturated set of facts is: $\mathcal{F}^* = \bigcup_{D \in \sigma\text{-}chase(\mathcal{F}, \mathcal{R})} Facts(D)$*

Saturating a set of facts $\mathcal{F}$ with a set of rules $\mathcal{R}$ until no new rule application is possible allows us to obtain the *universal model*. The particularity of this model is that it is representative of all models of $(\mathcal{F} \cup \mathcal{R})$ (we denote the set of models of $(\mathcal{F} \cup \mathcal{R})$ by $models(\mathcal{F}, \mathcal{R})$).

Definition 2.15 (Universal Model). *Given a set of facts $\mathcal{F}$ and a set of rules $\mathcal{R}$, a universal model M of $(\mathcal{F} \cup \mathcal{R})$ is a model of $(\mathcal{F} \cup \mathcal{R})$ such that for all models M' of $(\mathcal{F} \cup \mathcal{R})$, there is a homomorphism from M to M'.*

It is not always possible to obtain the universal model (the saturated set of facts might be infinite), however if the chase is finite then the model of the saturated set of facts is a universal model [Baget et al., 2011a]. Therefore query entailment can be expressed using the notion of chase.

Theorem 2.1 (Query entailment and Chase [Baget et al., 2011a]). *Given a set of facts $\mathcal{F}$, a set of rules $\mathcal{R}$, and a Boolean conjunctive query Q: if $\sigma\text{-}chase(\mathcal{F}, \mathcal{R})$ is finite then $(\mathcal{F} \cup \mathcal{R}) \models Q$ if and only if $Facts(\sigma\text{-}chase) \models Q$.*

Different kinds of chases can be defined using different derivation reducers. Each derivation reducer ensures a universal model if its chase is finite. The most common chase is the *Frontier chase* [Baget et al., 2011a], it yields equivalent results as the well-known *Skolem chase* [Marnette, 2009] that relies on a "skolemisation" of the rules by replacing each occurrence of an existential variable Y with a functional term $f_Y^r(\vec{X})$, where $\vec{X} = fr(r)$ are the frontier variables of r. Frontier chase and skolem chase yield isomorphic results [Baget et al., 2014a], in the sense that they generate exactly the same atoms, up to a bijective renaming of nulls by skolem terms.

The frontier chase considers two rule applications redundant if their mapping of the frontier variables are the same for the same rule.

Definition 2.16 (Frontier/Skolem Chase). *The frontier chase σ_{fr}-chase (equivalent to the Skolem chase [Marnette, 2009]) relies on the frontier derivation reducer (denoted by σ_{fr}) defined as follows: for any derivation δ, $\sigma_{fr}(D_0) = D_0$ and $\forall D_i = (\mathcal{F}_i, r_i, \pi_i) \in \delta$:*

$$Facts(\sigma_{fr}(D_i)) = \begin{cases} \mathcal{F}_{i-1} \cup \pi_i^{safe}(Head(r_i)) & \text{if } \forall j < i \text{ with } r_i = r_j, \\ & \pi_j|_{fr(r_j)}(Body(r_j)) \neq \pi_i|_{fr(r_i)}(Body(r_i)) \\ \mathcal{F}_{i-1} & \text{otherwise} \end{cases}$$

Example 2.7 (Frontier chase). *Consider the following set of facts $\mathcal{F}$ and set of rules $\mathcal{R}$ stating that an **animal shelter** would keep a dog found alone if it has an owner. If it has a collar or a microchip then it has an owner. A dog named "Jack" with a collar and a microchip is found alone.*

- $\mathcal{F} = \{alone(jack),\ hasCollar(jack),\ hasMicrochip(jack)\}$

- $\mathcal{R} = \{r_1 : \forall X, Y\ \mathit{hasOwner}(X, Y) \rightarrow \mathit{keep}(X),$
  $r_2 : \forall X\ \mathit{alone}(X) \wedge \mathit{hasCollar}(X) \rightarrow \exists Y\ \mathit{hasOwner}(X, Y),$
  $r_3 : \forall X\ \mathit{alone}(X) \wedge \mathit{hasMicrochip}(X) \rightarrow \exists Y\ \mathit{hasOwner}(X, Y)\}$

  A possible frontier chase of $\mathcal{F}$ and $\mathcal{R}$ is:

$$\sigma_{fr}\text{-}\mathit{chase}(\mathcal{F}, \mathcal{R}) = \langle(\mathcal{F}, \emptyset, \emptyset), (\mathcal{F}_1 = \mathcal{F} \cup \{\mathit{hasOwner}(\mathit{jack}, \mathit{Null}_1)\}, r_2, \pi_1 = \{X \rightarrow \mathit{jack}\}),$$
$$(\mathcal{F}_2 = \mathcal{F}_1 \cup \{\mathit{hasOwner}(\mathit{jack}, \mathit{Null}_2)\}, r_3, \pi_2 = \{X \rightarrow \mathit{jack}\}),$$
$$(\mathcal{F}_3 = \mathcal{F}_2 \cup \{\mathit{keep}(\mathit{jack})\}, r_1, \pi_3 = \{X \rightarrow \mathit{jack}, Y \rightarrow \mathit{Null}_1\})\rangle.$$

First, r_2 is applied on $\{\mathit{alone}(\mathit{jack}), \mathit{hasCollar}(\mathit{jack})\}$ and generates $\exists Y\ \mathit{hasOwner}(\mathit{jack}, Y)$ which is not redundant since r_2 has never been applied before, therefore $\mathcal{F}_1 = \mathcal{F}_0 \cup \{\mathit{hasOwner}(\mathit{jack}, \mathit{Null}_1)\}$. Then r_3 is applied on $\{\mathit{alone}(\mathit{jack}), \mathit{hasMicrochip}(\mathit{jack})\}$ and generates $\exists Y\ \mathit{hasOwner}(\mathit{jack}, Y)$ which is also not redundant because r_3 has never been applied before (even if it generates the same atom as r_2), therefore $\mathcal{F}_2 = \mathcal{F}_1 \cup \{\mathit{hasOwner}(\mathit{jack}, \mathit{Null}_2)\}$.

Afterwards, r_1 is applied on $\{\mathit{hasOwner}(\mathit{jack}, \mathit{Null}_1)\}$ and generates $\{\mathit{keep}(\mathit{jack})\}$ which is not redundant as r_1 has never been applied before, therefore $\mathcal{F}_3 = \mathcal{F}_2 \cup \{\mathit{keep}(\mathit{jack})\}$. Finally, r_1 is applied on $\{\mathit{hasOwner}(\mathit{jack}, \mathit{Null}_2)\}$ with the homomorphism $\pi_4 = \{X \rightarrow \mathit{jack}, Y \rightarrow \mathit{Null}_2\}$ and generates $\{\mathit{keep}(\mathit{jack})\}$ which is redundant since this rule application reuses the same rule and frontier mapping as the rule application on $\{\mathit{hasOwner}(\mathit{jack}, \mathit{Null}_1)\}$ (i.e. $\pi_4|_{fr(r_1)} = \pi_3|_{fr(r_1)} = \{X \rightarrow \mathit{jack}\}$). Since any additional rule application would be redundant (all rules have been applied with all possible homomorphisms) the frontier chase stops.

Even if the frontier reducer removes some redundant rule applications, the frontier chase might be infinite as shown in the following Example 2.8.

Example 2.8 (Infinite Frontier Chase). *Consider the set of fact $\mathcal{F}$ and the set of rules $\mathcal{R}$ containing one fact and one rule.*

- $\mathcal{F} = \{p(a)\}$
- $\mathcal{R} = \{r_1 : \forall X\ p(X) \rightarrow \exists Y\ p(Y)\}$

  A possible frontier chase of $\mathcal{F}$ and $\mathcal{R}$ is:

$$\sigma_{fr}\text{-}\mathit{chase}(\mathcal{F}, \mathcal{R}) = \langle(\mathcal{F}, \emptyset, \emptyset), (\mathcal{F}_1 = \mathcal{F} \cup \{p(\mathit{Null}_1)\}, r_1, \pi_1 = \{X \rightarrow a\}),$$
$$(\mathcal{F}_2 = \mathcal{F}_1 \cup \{p(\mathit{Null}_2)\}, r_1, \pi_2 = \{X \rightarrow \mathit{Null}_1\}),$$
$$(\mathcal{F}_3 = \mathcal{F}_2 \cup \{p(\mathit{Null}_3)\}, r_1, \pi_2 = \{X \rightarrow \mathit{Null}_2\}),$$
$$\ldots\rangle.$$

First, r_1 is applied using π_1 and generates $\exists Y\, p(Y)$ which is not redundant since r_1 has never been applied before, therefore $\mathcal{F}_1 = \mathcal{F}_0 \cup \{p(\mathit{Null}_1)\}$. Then r_1 is applied on $\{p(\mathit{Null}_1)\}$ using π_2 and generates $\exists Y\, p(Y)$ which is not redundant since $\pi_2|_{fr(r_1)} = \{X \rightarrow \mathit{Null}_1\} \neq \pi_1|_{fr(r_1)} = \{X \rightarrow a\}$, therefore $\mathcal{F}_2 = \mathcal{F}_1 \cup \{p(\mathit{Null}_2)\}$, and so on infinitely.

Some derivation reducers are "stronger" than others, this implies that their chase might stop in cases where others do not. This is known as the reducer order relation.

Definition 2.17 (Reducer Order Relation [Rocher, 2016]). *Given two derivation reducers σ_1 and σ_2, we say that σ_1 is weaker than σ_2 (denoted by $\sigma_1 \leq \sigma_2$) if for any set of rules $\mathcal{R}$ and set of facts $\mathcal{F}$, if σ_1-chase is finite then σ_2-chase is also finite. Furthermore, we say that σ_1 is strictly weaker than σ_2 if $\sigma_1 \leq \sigma_2$ and $\sigma_2 \not\leq \sigma_1$.*

In the literature, there are four well-known types of chase: the Oblivious chase (σ_{obl}-*chase*) [Cali et al., 2013], the Skolem/Frontier chase (σ_{fr}-*chase*) [Marnette, 2009, Baget et al., 2011a], the Restricted chase (σ_{res}-*chase*) [Fagin et al., 2005], and the Core chase (σ_{core}-*chase*) [Deutsch et al., 2008].

Proposition 2.1 (Chases Finiteness Order [Onet, 2013, Rocher, 2016]). *The following relations hold: $\sigma_{obl} \leq \sigma_{fr} \leq \sigma_{res} \leq \sigma_{core}$.*

It is well-known that query entailment using a chase is undecidable (the chase might be infinite) [Beeri and Vardi, 1981] even under strong restrictions such as using a single rule or restricting to binary predicates with no constants. However, some restrictions on the set of rules can ensure decidability for a specific type of chase. These restrictions are classified into three big categories known as "*abstract classes*". The first one is "*Finite Expansion Set*" (FES) [Baget et al., 2014b] that ensures that a finite universal model of the knowledge base exists and can be generated using a chase. For each chase we can define its FES class: *oblivious-FES*, *skolem-FES*, *restricted-FES*, *and core-FES*. The second class is called "*Finite Unification Set*" (FUS) [Baget et al., 2011a] which guarantees that some backward chaining method halts. Finally, the class called "*Greedy Bounded Treewidth Set*" (GBTS) [Baget et al., 2011b] ensures that the potentially infinite universal model of a knowledge base has a bounded treewidth. Each abstract class has a set of "*concrete classes*" that classifies rules based on their syntactic properties e.g. the concrete class *Datalog* describes rules that do not contain existentially quantified variables. The following Figure 2.1 shows the most studied concrete classes in the literature and the relation between them: an upward edge going from a class C to a class C' means that any set of rules in class C is also in Class C'.

In this thesis we rely mainly on the frontier chase to reason with existential rules, for simplicity we will only give examples and intuitions about concrete classes of skolem-FES [1]. Restricting ourselves to the frontier chase

[1] For more information about these concrete classes see [Baget et al., 2011a]. The online tool Kiabora `http://graphik-team.github.io/graal/downloads/kiabora-online` checks automatically if a set of rules is skolem-FES.

and subsequently to the skolem-FES classes of rules is not a very restrictive constraint since most studied concrete FES classes are skolem-FES (cf. Figure 2.2).

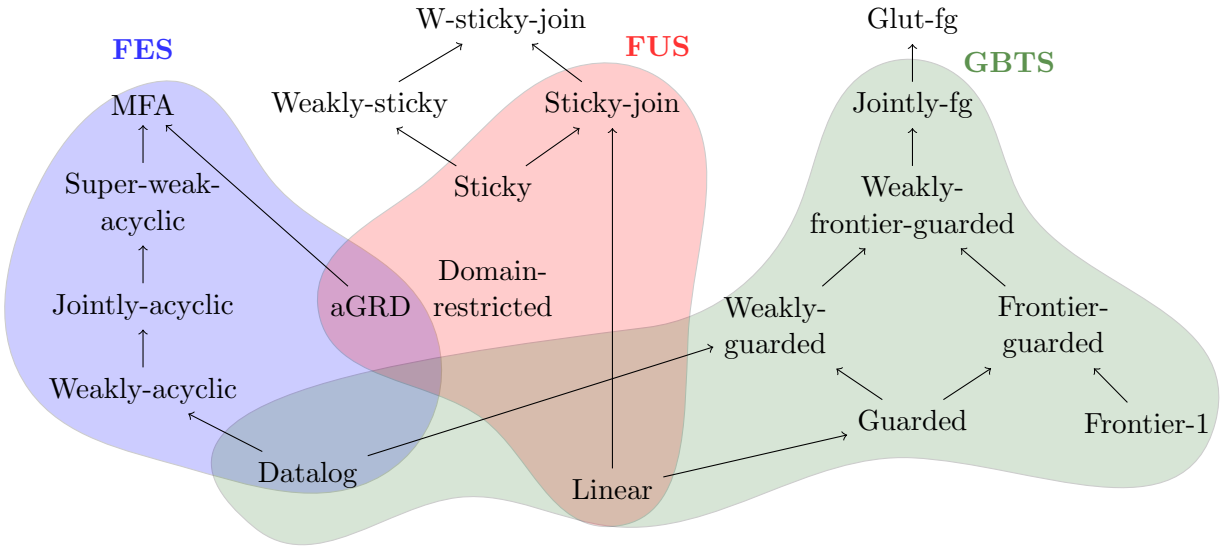

Figure 2.1: Abstract and known concrete classes of existential rules [Baget et al., 2011a, Rocher, 2016]

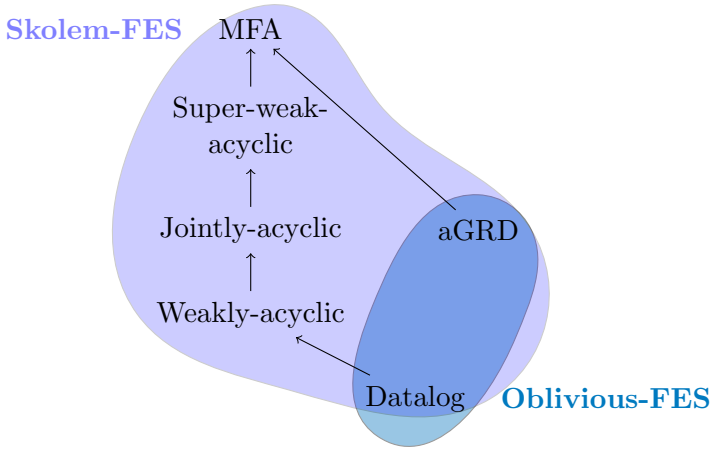

Figure 2.2: Known concrete FES classes and chases finiteness (all skolem-FES concrete classes are restricted-FES and core-FES cf. Proposition 3.4.)

A concrete class is simply a syntactic distinction of rules. The most basic skolem-FES concrete class is the **Datalog class** (also known as Range

Restricted [Abiteboul et al., 1995]) which are rules without the existential quantifier. Another simple class is the **aGRD class** (Acyclic Graph of Rule Dependency) [Baget et al., 2014a]. A Graph of Rule Dependency is a directed graph that encodes possible interactions between rules: the nodes represent the rules and there is an edge from a node r_1 to r_2 iff an application of the rule r_1 may create a new application of the rule r_2. A GRD is acyclic when it has no circuit. The notions of "*weak acyclicity*" [Marnette, 2009] and "*joint acyclicity*" [Krötzsch and Rudolph, 2011] are based on the position of the predicate and the existential and frontier variables. The **MFA** class (Model Faithful Acyclicity) [Cuenca Grau et al., 2013] relies on detecting a specific set of facts called *critical instance*. The following Example 2.9 provides some rules that are skolem-FES.

Example 2.9 (Skolem-FES rules). *Consider the following sets of rules:*

- $\mathcal{R}_1 = \{\forall X, Y, Z\, p(X,Z) \wedge p(Z,Y) \rightarrow p(X,Y)\}$ *is range restricted (Datalog).*
- $\mathcal{R}_2 = \{\forall X, Y\, siblingOf(X,Y) \rightarrow \exists Z\, parentOf(Z,X,Y)\}$ *is aGRD.*
- $\mathcal{R}_3 = \{r_1 : \forall X, Y\, p(X,Y) \rightarrow \exists Z\, r(Y,Z)$,
  $r_2 : \forall X, Y\, r(X,Y) \rightarrow p(Y,X)\}$. $\{r_1\}$ *is aGRD and* $\{r_2\}$ *is range restricted, however* $\mathcal{R}_3$ *is weakly-acyclic and is neither aGRD nor range-restricted.*
- $\mathcal{R}_4 = \{r_1 : \forall X, Y\, p(X,Y) \rightarrow \exists Z\, r(Y,Z)$,
  $r_2 : \forall X, Y\, r(X,Y) \wedge r(Y,X) \rightarrow p(X,Y)\}$. $\{r_1\}$ *is aGRD and* $\{r_2\}$ *is range restricted, however* $\mathcal{R}_4$ *is Jointly-acyclic.*
- $\mathcal{R}_5 = \{r_1 : \forall X q(X) \rightarrow \exists Y\, p(X,Y) \wedge p(Y,X) \wedge p(X,X)$,
  $r_2 : \forall X\, p(X,X) \rightarrow r(X)$,
  $r_3 : \forall X\, r(X) \rightarrow q(X)\}$. $\{r_1\}$ *alone is aGRD,* $\{r_2, r_3\}$ *is range restricted, however* $\mathcal{R}_5$ *is super-weakly-acyclic.*
- $\mathcal{R}_6 = \{\forall X, Y\, p(X,Y) \rightarrow \exists Z, T\, q(Y,Z) \wedge p(Z,T)\}$ *is model-faithful-acyclic.*

Not all concrete classes are created equal, some might have higher complexity for query answering, and applying a chase on these classes would require more time. In the next section we recall the definitions for some complexity classes and describe the complexity of the skolem-FES concrete classes.

2.1.4 Complexity Classes

Complexity is an indication of a computational problem inherent difficulty. We briefly recall the definitions of the complexity classes by increasing complexity. For more details about complexity theory, the reader is referred to [Papadimitriou, 2003].

Definition 2.18 (AC_0). *A problem is in* AC_0 *if it can be solved by a boolean circuit of bounded depth with a polynomial number of AND and OR gates.*

Definition 2.19 (Polynomial Time (PTime)). *A problem is in* PTime *if it can be solved by a deterministic Turing machine running in polynomial time in the input.*

Definition 2.20 (NP). *A problem is in* NP *if it can be solved by a non-deterministic Turing machine running in polynomial time in the input.*

Definition 2.21 (coNP). *A problem is in* coNP *if its complement is in the class* NP*, meaning that there is a polynomial-time algorithm that can verify no instances (counterexamples) using a non-deterministic Turing machine.*

Definition 2.22 (Exponential Time (ExpTime)). *A problem is in* ExpTime *if it can be solved by a deterministic Turing machine running in simple exponential time ($2^{p(n)}$) in the input.* 2ExpTime *is running in exponential time* $2^{2^{p(n)}}$ *while* 3ExpTime *is* $2^{2^{2^{p(n)}}}$.

Furthermore, a problem P is **hard** for a given complexity class C if any instance of a problem from C can be reduced to an instance of P through a reduction (in most cases, this reduction has to be in polynomial time, but for lower classes (PTime and below), logarithmic space reductions must be used). A problem P is **complete** for a given complexity class C if it belongs to C and is hard for C. For the query entailment problem, two different measures of complexity are considered:

- **Combined complexity:** the input contains the set of rules, the set of facts and the query.
- **Data complexity:** the input contains only the set of facts while the set of rules and the query are assumed to be fixed.

Data complexity is sometimes considered more relevant [Lembo et al., 2010] because the query and the rules are usually far smaller than the set of facts in practical applications, however both complexities can help understand where the difficulties lie. Indeed, for instance, query answering over skolem-FES rules using a frontier chase has in the worst case 2ExpTime-complete combined complexity and PTime-complete data complexity. The following Table 2.1 describes the combined and data complexity of query answering for each studied concrete class of Skolem-FES.

2.1.5 Incoherence and Inconsistence

To represent knowledge about the world one should account for "*negative knowledge*", i.e. information that dictates how things ought not to be, especially since generating new knowledge from seemingly correct information

Rule Class	Combined Complexity	Data Complexity
Datalog	ExpTime-complete [Chandra et al., 1981]	PTime-complete [Dantsin et al., 2001]
aGRD	ExpTime-complete [Cali et al., 2010b]	PTime-complete [Cali et al., 2010b]
Jointly-acyclic	2ExpTime-complete [Krötzsch and Rudolph, 2011]	PTime-complete [Krötzsch and Rudolph, 2011]
Weakly-acyclic	2ExpTime-complete [Fagin et al., 2005]	PTime-complete [Fagin et al., 2005]
Super-weakly-acyclic	2ExpTime-complete [Marnette, 2009]	PTime-complete [Marnette, 2009]
MFA	2ExpTime-complete [Zhang et al., 2015]	PTime-complete [Marnette, 2009]

Table 2.1: Complexity of CQ entailment for studied Skolem-FES concrete classes

might lead to a contradiction down the line. A basic form of "negative knowledge" is stating that a fact and its negation (or complement) should not be both asserted at the same time. While the existential rules language $\mathcal{L}$ is negation-free, the notion of integrity constraint from the database domain can be used to express negative knowledge.

Definition 2.23 (Negative Constraint). *A negative constraint (or simply a constraint) is a rule of the form* $\forall \vec{X}\, \mathcal{B}(\vec{X}) \rightarrow \bot$ *where* $\vec{X}$ *is a tuples of variables, and* $\mathcal{B}$ *is a finite non-empty conjunction of atoms.*

In this thesis, we only consider "binary" negative constraints (a.k.a. denial constraints) that express a conflict between two atoms. This restriction simplifies subsequent definitions and does not imply a loss of generality since any negative constraint can be transformed into a set of rules and binary negative constraints [Calì et al., 2012].

Definition 2.24 (Conflicting Facts). *Two facts* f_1 *and* f_2 *are in conflict if the body of a negative constraint can be mapped to* $\{f_1, f_2\}$.

Example 2.10 (Negative Constraint and Conflicting Facts). *Consider the negative constraint stating that it is impossible that a person is alive and dead at the same time:* $\forall X\, \mathit{alive}(X) \wedge \mathit{dead}(X) \rightarrow \bot$. *The fact* $\mathit{alive}(\mathit{bob})$ *is in conflict with* $\mathit{dead}(\mathit{bob})$ *(and vice-versa) because the body of the negative constraint can be mapped to these facts.*

Negative constraints are used to ensure that a set of facts is consistent (i.e. contains no contradiction). This is especially important since in presence of conflict, query answering becomes trivial due to the principle of explosion (ex falso quodlibet) "*from falsehood anything follows*".

In the various domains of knowledge representation, conflicts might be inherent to the represented domain or may arise from an incorrect description of the world. When a set of factual knowledge contains no conflicts it is said to be *consistent*, otherwise it is *inconsistent*.

Definition 2.25 (Inconsistence). *A set of facts $\mathcal{F}$ is inconsistent with respect to a set of negative constraints $\mathcal{N}$ if and only if $(\mathcal{F} \cup \mathcal{N})$ has no possible model ($models(\mathcal{F}, \mathcal{N}) = \emptyset$) i.e. $(\mathcal{F} \cup \mathcal{N}) \models \bot$. In practice, $\mathcal{F}$ is inconsistent if a negative constraint can be applied i.e. $\exists r \in \mathcal{N}$ such that $\mathcal{F} \models Body(r)$.*

An inconsistent set of facts does not necessarily mean an incorrect representation of the factual knowledge of the world. In some cases, the inconsistency of the generated set of facts is unavoidable (i.e. the representation has no model) even with a correct description of factual knowledge. This is due to an *incoherent* set of rules.

Definition 2.26 (Incoherence). *A set of rules $\mathcal{R}$ is incoherent with respect to a set of negative constraints $\mathcal{N}$ if and only if $\mathcal{R} \cup \mathcal{N}$ is unsatisfiable i.e. for any set of facts S such that all rules in $\mathcal{R}$ are applicable, $models(S, \mathcal{R} \cup \mathcal{N}) = \emptyset$. The application of $\mathcal{R}$ on any set of facts S will inevitably lead to an inconsistent saturated set of facts S^* with respect to $\mathcal{N}$.*

Clearly, the notions of incoherence and inconsistence are highly related. In fact, an incoherent set of rules $\mathcal{R}$ will always lead to an inconsistent set of facts $\mathcal{F}^*$ if all rules in $\mathcal{R}$ are applied on $\mathcal{F}$ [Flouris et al., 2006]. The following Examples 2.11 and 2.12 describe the key difference between incoherence and inconsistence.

Example 2.11 (Incoherence). *Consider the following sets of facts $\mathcal{F}$, rules $\mathcal{R}$, and negative constraints $\mathcal{N}$ representing the knowledge that birds fly, penguins are birds, and penguins cannot fly. "Kowalski" is a penguin, does it fly (i.e. $Q_1() = fly(kowalski)$)? Does it not fly (i.e. $Q_2() = notFly(kowalski)$)?*

- $\mathcal{F} = \{penguin(kowalski)\}$
- $\mathcal{R} = \{\ r_1 : \forall X\, penguin(X) \rightarrow bird(X),$
  $r_2 : \forall X\, bird(X) \rightarrow fly(X),$
  $r_3 : \forall X\, penguin(X) \rightarrow notFly(X)\}$
- $\mathcal{N} = \{\forall X\, fly(X) \wedge notFly(X) \rightarrow \bot\}$

The saturated set of facts resulting from a frontier chase is

- $\mathcal{F}* = \{penguin(kowalski),\ bird(kowalski),\ fly(kowalski),\ notFly(kowalski)\}$.

*The set of rules $\mathcal{R}$ is **incoherent** because no set of facts (even outside $\mathcal{F}$) that makes all rules in $\mathcal{R}$ applicable prevents the application of the negative constraint, therefore $models(\mathcal{F}, \mathcal{R} \cup \mathcal{N}) = \emptyset$. The answer to the boolean queries Q_1 and Q_2 is "true" (principle of explosion) i.e. Kowalski flies and does not fly at the same time. The saturated set of facts $\mathcal{F}*$ is **inconsistent** because $models(\mathcal{F}, \mathcal{R} \cup \mathcal{N}) = \emptyset$.*

Example 2.12 (Inconsistence vs Incoherence). *Consider the following sets of facts $\mathcal{F}$, rules $\mathcal{R}$, and negative constraints $\mathcal{N}$ describing a simplified legal situation: If there is a scientific evidence incriminating a defendant then he is responsible for the crime, if there is a scientific evidence absolving a defendant then he is not responsible for the crime. A defendant is guilty if responsibility is proven. If a defendant has an alibi then he is innocent. There is a scientific evidence "e1" incriminating a defendant "alice", while another scientific evidence "e2" is absolving her of the crime. She also has an alibi. Is Alice innocent (i.e. $Q_1() = \mathit{innocent}(\mathit{alice})$)? Is she guilty (i.e. $Q_2() = \mathit{guilty}(\mathit{alice})$)?*

- $\mathcal{F} = \{\mathit{incrim}(e1, \mathit{alice}),\ \mathit{absolv}(e2, \mathit{alice}),\ \mathit{alibi}(\mathit{alice})\}$

- $\mathcal{R} = \{r_1 : \forall X, Y\ \mathit{incrim}(X, Y) \rightarrow \mathit{resp}(Y),$
  $r_2 : \forall X, Y\ \mathit{absolv}(X, Y) \rightarrow \mathit{notResp}(Y),$
  $r_3 : \forall X\ \mathit{resp}(X) \rightarrow \mathit{guilty}(X),$
  $r_4 : \forall X\ \mathit{alibi}(X) \rightarrow \mathit{innocent}(X)\}$

- $\mathcal{N} = \{\forall X\ \mathit{resp}(X) \wedge \mathit{notResp}(X) \rightarrow \bot,$
  $\forall X\ \mathit{guilty}(X) \wedge \mathit{innocent}(X) \rightarrow \bot\}$

The saturated set of facts resulting from a frontier chase is

- $\mathcal{F}^* = \{\mathit{incrim}(e1, \mathit{alice}), \mathit{absolv}(e2, \mathit{alice}), \mathit{alibi}(\mathit{alice}), \mathit{resp}(\mathit{alice}), \mathit{notResp}(\mathit{alice}),$
  $\mathit{guilty}(\mathit{alice}), \mathit{innocent}(\mathit{alice})\}$.

The set of rules $\mathcal{R}$ is ***coherent*** *because $\mathcal{R} \cup \mathcal{N}$ is satisfiable i.e. there exists a possible set of facts $S = \{\mathit{incrim}(e1, \mathbf{bob}), \mathit{absolv}(e2, \mathit{alice}), \mathit{alibi}(\mathit{alice})\}$ such that all rules in $\mathcal{R}$ are applicable and $\mathit{models}(S, \mathcal{R} \cup \mathcal{N}) \neq \emptyset$, the set $S^* = \{\mathit{incrim}(e1, \mathbf{bob}), \mathit{absolv}(e2, \mathit{alice}), \mathit{aibli}(\mathit{alice}), \mathit{resp}(\mathbf{bob}), \mathit{notResp}(\mathit{alice}),$ $\mathit{guilty}(\mathbf{bob}), \mathit{innocent}(\mathit{alice})\}$ is* ***consistent*** *as no negative constraint is applicable on it.*

However the saturated set of facts $\mathcal{F}^*$ is ***inconsistent*** *because a negative constraint is applicable, thus $\mathit{models}(\mathcal{F}, \mathcal{R} \cup \mathcal{N}) = \emptyset$. Since the set of rules is coherent, the inconsistence of $\mathcal{F}^*$ is due to an erroneous set of intial facts (either one of the evidences, the alibi, or all of them are not valid).*

The classical answer to the boolean queries Q_1 and Q_2 is "true" (i.e. Alice is guilty and innocent), because from falsehood, anything follows.

Inconsistence and incoherence are problematic for classical query answering. Nevertheless, the ability to reason in presence of conflicts can be preserved by relying on "*conflict-tolerant*" techniques such as Defeasible Reasoning.

2.2 Defeasible Reasoning

Defeasible Reasoning is a form of non-monotonic reasoning that stems from the need to reason with *incomplete knowledge* by "*filling the gaps in the available information by making some kind of plausible (or desirable) assumptions*" [Pollock, 1987]. It can be seen as "*jumping to conclusions that may prove to be wrong when additional information becomes available. It is the kind of reasoning in which a quick application of a simple rule of thumb is considered to be of more value than a logically correct inference*" [Vreeswijk, 1995].

2.2.1 Defeasible Knowledge Bases and Representation

Making plausible "leaps" to generate new knowledge might ultimately lead to an inconsistent or incoherent representation of the world. This also requires a distinction between definite "*strict*" implications and plausible "*defeasible*" implications. Furthermore, one might want to "block" a plausible implications in a certain context using "*defeater*" implications. We extend the existential rule language $\mathcal{L}$ to the defeasible existential rules language denoted $\mathcal{L}_{\forall\exists}$ (a.k.a. Defeasible Datalog$^{\pm}$ [Martinez et al., 2014]) to allow for the different types of implications.

Definition 2.27 (Strict, Defeasible, and Defeater Rules). *Defeasible reasoning distinguishes three types of rules:*

- ***Strict rules*** *($\rightarrow$) of the form* $r : \forall\vec{X},\vec{Y}\ \left(\mathcal{B}(\vec{X},\vec{Y}) \rightarrow \exists\vec{Z}\ \mathcal{H}(\vec{X},\vec{Z})\right)$ *expressing undeniable implications i.e. if* $\mathit{Body}(r)$ *then definitely* $\mathit{Head}(r)$.
- ***Defeasible rules*** *($\Rightarrow$) of the form* $r : \forall\vec{X},\vec{Y}\ \left(\mathcal{B}(\vec{X},\vec{Y}) \Rightarrow \exists\vec{Z}\ \mathcal{H}(\vec{X},\vec{Z})\right)$ *expressing weaker (plausible) implications i.e. if* $\mathit{Body}(r)$ *then generally* $\mathit{Head}(r)$.
- ***Defeater rules*** *($\rightsquigarrow$) of the form* $r : \forall\vec{X},\vec{Y}\ \left(\mathcal{B}(\vec{X},\vec{Y}) \rightsquigarrow \exists\vec{Z}\ \mathcal{H}(\vec{X},\vec{Z})\right)$ *used to prevent defeasible implications by producing evidence of the contrary i.e. stating that if* $\mathit{Body}(r)$ *then any conflicting atom with* $\mathit{Head}(r)$ *should not be derived. This does not however imply that* $\mathit{Head}(r)$ *is derived.*

We denote the set of strict rules by $\mathcal{R}_{\rightarrow}$*, the set of defeasible rules by* $\mathcal{R}_{\Rightarrow}$*, and the set of defeater rules by* $\mathcal{R}_{\rightsquigarrow}$.

While the intuition behind strict and defeasible rules is easy to grasp, one might find it difficult to intuitively understand the notion of defeater rules. A defeater rule r, when it is applicable, states that given some premises $\mathit{Body}(r)$, there is sufficient evidence to "defeat" (i.e. prevent) the conclusion of any atom conflicting with $\mathit{Head}(r)$. At the same time, these premises are not sufficient evidence to conclude $\mathit{Head}(r)$.

Example 2.13 (Defeasible and Defeater Rules). *Consider the following sets of rules $\mathcal{R}$ and negative constraints $\mathcal{N}$ stating that birds generally fly, and having a broken wing is sufficient evidence against flying.*

- $\mathcal{R} = \{r_1 : \forall X\, bird(X) \Rightarrow fly(X),$
  $r_2 : \forall X\, bird(X) \wedge brokenWings(X) \rightsquigarrow notFly(X)\}$
- $\mathcal{N} = \{\forall X\, fly(X) \wedge notFly(X) \rightarrow \bot\}$

The defeater rule r_2 expresses that having broken wings for a bird is a special context that prevents one from concluding that this bird flies. In other words, one does not wish to conclude that a bird with broken wings does not fly, however it is sufficient evidence to prevent the conclusion that it flies.

Much like strict and defeasible implications, factual knowledge can be strict or defeasible. To express this distinction we transform the set of facts $\mathcal{F}$ to a set of fact rules with a $\top$ body.

Notation 2.5 (Fact Rules). *A set of facts $\mathcal{F}$ can be seen as a set of "fact rules" of the form $\top \Rrightarrow f$ such that f is a fact and $\Rrightarrow \in \{\rightarrow, \Rightarrow\}$. A strict fact is of the form $\top \rightarrow f$ and a defeasible fact is of the form $\top \Rightarrow f$.*

Given the different types of rules and the fact that from a contradiction anything follows, the notion of derivation for a fact has to be adapted to be "*conflict-free*" and to not rely on defeater rules to generate intermediate knowledge. However, there is an interesting debate between logicians on what constitutes a "conflict-free" derivation. Some, mostly from the Defeasible Logic community, consider a derivation "conflict-free" if it does not contain conflicting atoms [Billington et al., 2010]. This is what we will call a *directly consistent derivation.*

Definition 2.28 (Directly Consistent Derivation for a Fact). *Given a set of facts $\mathcal{F}$, a set of rules $\mathcal{R} = \mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow} \cup \mathcal{R}_{\rightsquigarrow}$, a set of negative constraints $\mathcal{N}$, and a derivation $\delta = \langle D_0, \ldots, D_n \rangle$ for a fact f, we say that δ is a directly consistent derivation if and only if:*

1. *$\forall i$ s.t. $0 \leq i < n$, $Rule(D_i) \in \mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow}$ i.e. a derivation can only rely on strict and defeasible rules to generate knowledge.*
2. *$\forall r \in \mathcal{N}$ $Facts(D_n) \not\models Body(r)$ i.e. no negative constraint is directly applicable on the set of facts contained in the derivation.*

On the other hand, other logicians such as [García and Simari, 2004] argue that for a derivation to be "conflict-free" it has to be consistent with regards to the set of strict rules. This is what we call an *indirectly consistent derivation.*

Definition 2.29 (Indirectly Consistent Derivation for a Fact). *Given a set of facts $\mathcal{F}$, a set of rules $\mathcal{R} = \mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow} \cup \mathcal{R}_{\rightsquigarrow}$, a set of negative constraints $\mathcal{N}$, and a derivation $\delta = \langle D_0, \ldots, D_n \rangle$ for a fact f, we say that δ is an indirectly consistent derivation if and only if:*

1. *$\forall i$ s.t. $0 \leq i < n$, $Rule(D_i) \in \mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow}$ i.e. a derivation can only rely on strict and defeasible rules to generate knowledge.*
2. *$models(Facts(D_n), \mathcal{R}_{\rightarrow}) \neq \emptyset$ i.e. the set of facts contained in the derivation are consistent w.r.t. the set of strict rules.*

We can see here that if the set of strict rules is incoherent, no derivation can be indirectly consistent. That is why most conflict-tolerant techniques make the assumption that the set of strict rules is coherent, this assumption is a safe one to make since strict rules describe definite undeniable implications. In the remainder of this thesis we will suppose that the set of strict rule is coherent unless specified otherwise.

Derivations[2] can be seen as arguments in favor of a conclusion. The exact definition of an *argument* varies from a formalism to another [Antoniou et al., 2000a, García and Simari, 2004, Modgil and Prakken, 2014], for simplicity we consider for now that an argument is a tuple $arg = \langle \delta, f \rangle$ where δ is a derivation for f. Depending on the used facts and rules, we can distinguish three types of arguments: strict, defeasible and defeater ones.

Definition 2.30 (Strict, Defeasible, and Defeater Arguments). *Given a set of facts $\mathcal{F}$, a set of rules $\mathcal{R} = \mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow} \cup \mathcal{R}_{\rightsquigarrow}$, a set of negative constraints $\mathcal{N}$, and a derivation $\delta = \langle D_0, \ldots, D_n \rangle$ for a fact f, an argument for f is a tuple $arg = \langle \delta, f \rangle$.*

- *arg is a strict argument if $\forall D_i \in \delta$, $Rule(D_i) \in \mathcal{R}_{\rightarrow}$.*
- *arg is a defeasible argument if it is not a strict argument and $\forall D_i \in \delta$, $Rule(D_i) \in \mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow}$.*
- *arg is a defeater argument if $\forall D_i \in \delta$ such that $i < n$, $Rule(D_i) \in \mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow}$ and $Rule(D_n) \in \mathcal{R}_{\rightsquigarrow}$.*

Defeasible reasoning is basically a formalism that handles the attack between rules applications or arguments. We say that two rule applications attack each other if they generate conflicting atoms and we say that an argument attacks another argument if the conclusion of the first is in conflict with a fact in the derivation of the second.

Definition 2.31 (Attack between Rule applications). *Given a set of negative constraints $\mathcal{N}$, a rule application $\alpha(S_1 \subseteq \mathcal{F}, r_1, \pi_1)$ attacks another rule application $\alpha(S_2 \subseteq \mathcal{F}, r_2, \pi_2)$ (and vice-versa) if $\exists r \in \mathcal{N}$ such that*

[2] "Derivation" refers to either a directly or indirectly consistent derivation in general, we will make a clear distinction when we discuss Defeasible Reasoning formalisms.

$\pi_1(Head(r_1)) \wedge \pi_2(Head(r_2)) \models Body(r)$ *i.e. the body of a negative constraint can be mapped to the generated facts.*

Definition 2.32 (Attack between Arguments). *An argument* $arg_1 = \langle \delta_1, f_1 \rangle$ *for a fact* f_1 *attacks an other argument* $arg_2 = \langle \delta_2, f_2 \rangle$ *if* $\exists f_3 \in Facts(\delta_2)$ *such that* f_1 *and* f_3 *are in conflict. If two argument are for conflicting facts then they attack each other.*

Not all attacks are created equal, Defeasible Reasoning relies on the notion of *preference relation* (a.k.a. priority relation) to handle these attacks and resolve conflicts. Different techniques of Defeasible Reasoning apply a preference relation on different levels, either at the rule level (a rule may override another rule) or at the argument level (an argument may be superior to another argument). Defeasible Reasoning makes the assumption that preferences at the rule level can only be between non strict attacking rules since strict rules are by definition undeniable implications.

Definition 2.33 (Preference Relation). *A preference relation* $\succ$ *(also called priority relation) is a binary relation between rules or between arguments. An element (rule or argument)* e_1 *is superior to another element* e_2 *if and only if* $e_1 \succ e_2$ *and* $e_2 \not\succ e_1$ *(*e_2 *is said to be inferior to* e_1*).*

Defeasible Reasoning focuses mainly on acyclic preference relation i.e. there cannot be two elements e_1 and e_2 such that $e_1 \succ e_2$ and $e_2 \succ e_1$.

Example 2.14 (Preference Relation). *Consider the following set of rules* $\mathcal{R}$*, negative constraints* $\mathcal{N}$*, and preference relation* $\succ$ *describing the fact that a customer would not buy a product if it is detrimental to the environment unless it is cheap:*

- $\mathcal{R} = \{r_1 : \forall X\, eco(detrimental, X) \Rightarrow notBuy(X),$
  $r_2 : \forall X\, cheap(X) \Rightarrow buy(X)\}$
- $\mathcal{N} = \{\forall X\, buy(X) \wedge notBuy(X) \rightarrow \bot\}$
- $\succ = \{(r_2, r_1)\}$ *(can be denoted as* $r_2 \succ r_1$*).*

Suppose the customer was offered a cheaply priced phone that is detrimental to the environment ($\mathcal{F} = \{eco(detrimental, phone),\ cheap(phone)\}$*). Given the preference relation* $\succ$*,* r_2 *can override* r_1 *and the conflict between* $buy(phone)$ *and* $notBuy(phone)$ *is resolved in favor of* $buy(phone)$*.*

Let us now define the concept of "*knowledge base*" which is a structure that regroups all the key notions of defeasible knowledge representation.

Definition 2.34 (Knowledge Base). *A knowledge base is a tuple* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ *where* $\mathcal{F}$ *is a set of fact rules,* $\mathcal{R}$ *is a set of rules,* $\mathcal{N}$ *is a set of negative constraints, and* $\succ$ *is an acyclic preference relation.*

In order to analyze and better understand a defeasible knowledge base, we will make use of schematic inference graphs called Inheritance Networks [Pollock, 1987] with the following conventions: dashed arrows signify inferences (i.e. supports) and simple arrows signify that the complement of the pointed fact is implied (i.e. attacks). For instance, Example 2.11 is illustrated in Figure 2.3.

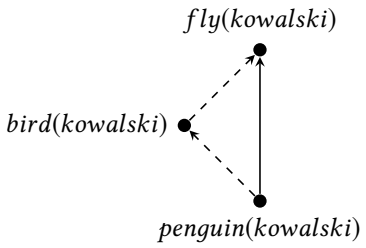

Figure 2.3: Inheritance Network of Example 2.11

Representing defeasible knowledge is only the first step, the next step would be defining the reasoning mechanism for resolving conflicts. The problem however, is that it appears that no single way of Defeasible Reasoning is appropriate in all situations or for all purposes, one technique may achieve desired outcomes in some situations and not in others. In the next section we define most studied Defeasible Reasoning intuitions in the literature.

2.2.2 Defeasible Reasoning Intuitions

Defeasible reasoning techniques generally agree that a fact is entailed if it is not derived from challenged facts and is either not in conflict or is the preferred outcome of its conflicts. However, they disagree on what constitutes a "*valid challenge*" or a "*preferred outcome*". Unfortunately, there is no universally valid way to reason defeasibly. An inherent characteristic of Defeasible Reasoning is its systematic reliance on a set of intuitions and rules of thumb, which have been longly debated between logicians [Horty et al., 1987, Makinson and Schlechta, 1991, Prakken, 2002, Antoniou, 2006].

Discussed Defeasible Reasoning intuitions

- *Ambiguity Handling (what constitutes a valid challenge)?*
- *Team Defeat (what is a preferred outcome)?*
- *Floating Conclusions?*
- *Are arguments evaluated on construction or after construction?*
- *How are strict rules handled (direct or indirect consistency)?*

The aim of Defeasible Reasoning is to resolve conflicts, which might be an easy task when the preference relation indicates the preferred outcome. However, when the preference relation does not indicate which rule or argument is superior, the conflicting facts become ambiguous.

Definition 2.35 (Ambiguous Facts). *A fact f is ambiguous if there is an argument (or rule application) for f that is neither inferior to any argument (or rule application) for a fact in conflict with f nor superior to an argument (or rule application) for a fact in conflict with f.*

Most Defeasible Reasoning techniques agree that ambiguous facts should not be skeptically entailed since one cannot choose between the conflicting outcomes, these ambiguous facts are considered "dead" or unjustified. However, some argue that facts derived from ambiguous ones should be allowed to attack other facts and to "*propagate*" their ambiguity to others. These attacks from ambiguous facts are called "zombie attacks" because the fact itself is "dead" but its attack is "alive". This first intuition is called *ambiguity handling* also known as *zombie arguments* [Stein, 1992].

1. **Ambiguity Handling:** A fact f that is derived from ambiguous facts is ambiguous (i.e. contested). The intuition behind ambiguity handling is whether to "*block*" or "*propagate*" the ambiguity of f to other facts that are in conflict with it.

   Example 2.15 (Ambiguity Handling). *Consider the knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ describing the legal situation of Example 2.12:*

   - $\mathcal{F} = \{\top \Rightarrow \mathit{incrim}(e1, \mathit{alice}),\ \top \Rightarrow \mathit{absolv}(e2, \mathit{alice}),\ \top \Rightarrow \mathit{alibi}(\mathit{alice})\}$
   - $\mathcal{R} = \{r_1 : \forall X, Y\ \mathit{incrim}(X, Y) \rightarrow \mathit{resp}(Y),\ r_2 : \forall X, Y\ \mathit{absolv}(X, Y) \rightarrow \mathit{notResp}(X),\ r_3 : \forall X\ \mathit{resp}(X) \rightarrow \mathit{guilty}(X),\ r_4 : \forall X\ \mathit{alibi}(X) \rightarrow \mathit{innocent}(X)\}$
   - $\mathcal{N} = \{\forall X\ \mathit{resp}(X) \wedge \mathit{notResp}(X) \rightarrow \bot,\ \forall X\ \mathit{guilty}(X) \wedge \mathit{innocent}(X) \rightarrow \bot\}$

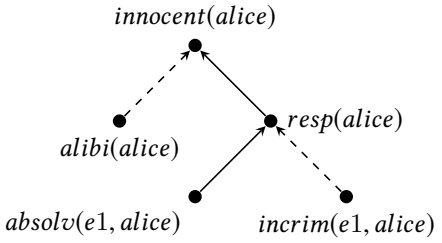

Figure 2.4: Inheritance Network of Example 2.15 for *innocent*(*alice*)

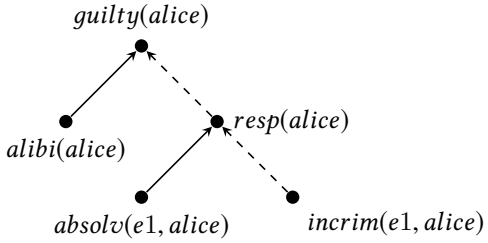

Figure 2.5: Inheritance Network of Example 2.15 for *guilty*(*alice*)

The inheritance networks for innocent(alice) *and* guilty(alice) *are shown in Figures 2.4 and 2.5 respectively. "*incrim(e1, alice)*" is not ambiguous*

because there is no chain of rule applications for a fact directly contradicting it. On the other hand, "resp(alice)" is ambiguous because it can be derived and there is a derivation for its conflicting fact "notResp(alice)". "guilty(alice)" is also ambiguous because it relies on an ambiguous premise (i.e. "resp(alice)").

In an ***ambiguity propagating*** *setting,* $\mathit{innocent}(\mathit{alice})$ *is ambiguous because* $\mathit{guilty}(\mathit{alice})$ *can be derived (even from ambiguous facts), the ambiguity of* $\mathit{guilty}(\mathit{alice})$ *is propagated to* $\mathit{innocent}(\mathit{alice})$*, thus* $\mathcal{KB} \nvdash_{prop} \mathit{innocent}(\mathit{alice})$ *and* $\mathcal{KB} \nvdash_{prop} \mathit{guilty}(\mathit{alice})$ *($\vdash_{prop}$ denotes entailment in ambiguity propagating).*

On the other hand, in an ***ambiguity blocking*** *setting, the ambiguity of* $\mathit{resp}(\mathit{alice})$ *blocks any ambiguity derived from it, meaning that* $\mathit{guilty}(\mathit{alice})$ *cannot be used to attack* $\mathit{innocent}(\mathit{alice})$*. Therefore* $\mathit{innocent}(\mathit{alice})$ *is not ambiguous, thus* $\mathcal{KB} \vdash_{block} \mathit{innocent}(\mathit{alice})$ *and* $\mathcal{KB} \nvdash_{block} \mathit{guilty}(\mathit{alice})$ *($\vdash_{block}$ denotes entailment in ambiguity blocking).*

Ambiguity propagation results in fewer conclusions (since more ambiguities are allowed), which might make it preferable when the cost of an incorrect conclusion is high, whereas *ambiguity blocking* might be more intuitive in situations where contested claims cannot be used to contest other claims (e.g. in the legal domain) [Horty et al., 1987]. The ambiguity handling intuition is simply a disagreement on what constitutes a valid challenge (attack), another intuition is team defeat which defines what is regarded as a preferred outcome.

2. **Team Defeat.** The intuition behind team defeat (also known as direct reinstatement) [Horty et al., 1987, Prakken, 2002] is whether an argument (or a rule application) has to be superior to all its attacking arguments in order to be accepted, or if other arguments can be used to override attacks that this argument is not superior to.

   Example 2.16 (Team Defeat). *Consider the following* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ *that describes the process of deciding whether to buy a product or not: An individual will not buy a product if it is detrimental to the environment unless it is cheap. He will also not buy it if it has slow delivery unless it has good reviews. Suppose we have a cheap smart-phone with good reviews that is detrimental to the environment and with slow delivery. Should the individual buy it (i.e* $Q_1 = \mathit{buy}(\mathit{phone})$*)? Should he not buy it (i.e* $Q_2 = \mathit{notBuy}(\mathit{phone})$*)? The inheritance network for* $\mathit{buy}(\mathit{phone})$ *is shown in Figure 2.6*

   - $\mathcal{F} = \{\top \rightarrow \mathit{price}(\mathit{phone}, \mathit{cheap}),\ \top \rightarrow \mathit{reviews}(\mathit{phone}, \mathit{good}),$
     $\top \rightarrow \mathit{eco}(\mathit{phone}, \mathit{detrimental}),\ \top \rightarrow \mathit{delivery}(\mathit{phone}, \mathit{slow})\}$

- $\mathcal{R} = \{r_1 : \forall X\, price(X, cheap) \Rightarrow buy(X),$
  $r_2 : \forall X\, reviews(X, good) \Rightarrow buy(X),$
  $r_3 : \forall X\, eco(X, detrimental) \Rightarrow notBuy(phone),$
  $r_4 : \forall X, Y\, derlivery(X, slow) \Rightarrow notBuy(phone)\}$
- $\mathcal{N} = \{\forall X\, buy(X) \wedge notBuy(X) \rightarrow \bot\}$
- $(r_1 \succ r_3), (r_2 \succ r_4)$

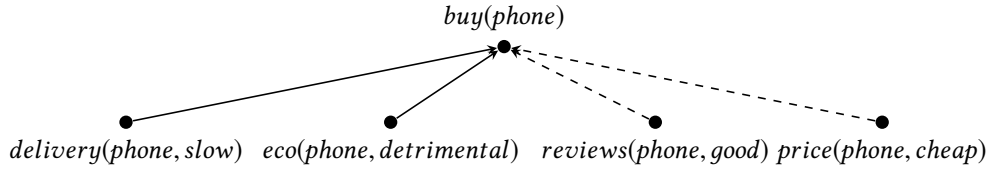

Figure 2.6: Inheritance Network for *buy*(*phone*) of Example 2.16

***In the absence of team defeat**, a fact is accepted if it has a rule application (or argument) for it that single-handedly defeats (i.e. superior to) all its attacking rule applications. In this context, the answer to Q_1 and Q_2 is "false" (i.e. $\mathcal{KB} \nvdash^{noTD}$ buy(phone) and $\mathcal{KB} \nvdash^{noTD}$ notBuy(phone) where $\vdash^{noTD}$ denotes entailment in Defeasible Reasoning without team defeat) because there is no rule application for buy(phone) that is superior to all its attacking rule applications: even if r_1 overrides r_3 ($r_1 \succ r_3$) it is not superior to r_4 ($r_1 \not\succ r_4$) i.e. the cheap price is a valid reason for buying the phone and overrides the fact that the phone is eco-detrimental, however it is not superior to slow delivery. The same applies for r_2: it overrides r_4 ($r_2 \not\succ r_4$) but not r_3 ($r_2 \not\succ r_3$) i.e. good reviews is a valid reason to buy the phone and overrides slow delivery, however it is not strong enough to prevail against the phone being eco-detrimental.*

***In the presence of team defeat**, a fact is accepted if all its attacking rule applications are defeated (overridden). In this context, Q_1 is "true" (i.e. $\mathcal{KB} \vdash^{TD}$ buy(phone) where $\vdash^{TD}$ denotes entailment in Defeasible Reasoning with team defeat) because all attacking rules are overridden: the application of r_1 overrides r_3 and the application of r_2 overrides r_4. The answer to Q_2 is false (i.e. $\mathcal{KB} \nvdash^{TD}$ notBuy(phone)) because all rule applications for notBuy(phone) are overridden.*

The intuition of team defeat can be combined with ambiguity handling to create different semantics for Defeasible Reasoning. Allowing team defeat is less skeptical than forbidding it [Prakken, 2002]. Ambiguity propagating without team defeat is considered the most skeptical approach and is adopted by most argumentation-based techniques [Prakken, 2002, Antoniou

et al., 2000a]. Another intuition that logicians disagree about is "*floating conclusion*" that describes conclusions that would always be reached no matter how conflicts are resolved.

3. **Floating Conclusions:** Sometimes two conflicting and equally strong rule applications (or arguments) might lead to the same conclusion down the line, these conclusions are called "floating conclusions" [Makinson and Schlechta, 1991].

   Example 2.17. *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *describing a criminal case where a first witness says that "Jack" killed "John" by stabbing him while a second witness says that he shot him. Both testimonies are of equal strength and both imply that "Jack" killed "John", however they are conflicting on the modus operandi. Did jack kill John (i.e.* $Q = killed(jack, john)$*)? The inheritance network for* $killed(jack, john)$ *is shown in Figure 2.7.*

   - $\mathcal{F} = \{\top \Rightarrow stabbed(jack, john),\ \top \Rightarrow shot(jack, john).$
   - $\mathcal{R} = \{r_1 : \forall X, Y\ stabbed(X, Y) \Rightarrow killed(X, Y),$ $r_2 : \forall X, Y\ shot(X, Y) \Rightarrow killed(X, Y),\ r_3 : \forall X, Y\ stabbed(X, Y) \Rightarrow notShot(X, Y),\ r_4 : \forall X, Y\ shot(X, Y) \Rightarrow notStabbed(X, Y)\}.$
   - $\mathcal{N} = \{\forall X, Y\ stabbed(X, Y) \wedge notStabbed(X, Y) \rightarrow \bot, \forall X, Y\ shot(X, Y) \wedge notShot(X, Y) \rightarrow \bot\}$

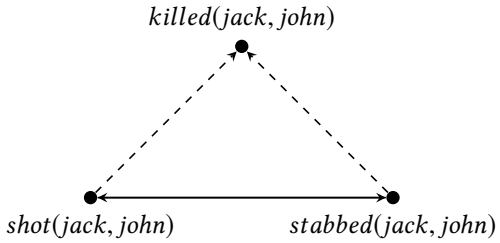

Figure 2.7: Inheritance Network for $killed(jack, john)$ of Example 2.17

"$killed(jack, john)$*" is a floating conclusion. One might argue that regardless of whether the witnesses disagree on the details, the conclusion is the same and therefore the answer to* Q_1 *is* ***'true'*** *(i.e.* $\mathcal{KB} \models^{FC} killed(jack, john)$ *where* $\models^{FC}$ *denotes entailment in Defeasible Reasoning with floating conclusions). However, one can also argue that the two witnesses undermine each other's credibility, and therefore the answer to the query* Q_1 *should be* ***false*** *(i.e.* $\mathcal{KB} \not\models^{noFC} killed(jack, john)$ *where* $\models^{noFC}$ *denotes entailment in Defeasible Reasoning without floating conclusions).*

Defeasible Reasoning intuitions can be combined to create different skeptical semantics, a single example combining these Defeasible Reasoning intuitions can be found in Chapter 3 Example 3.6 on page 92.

Another intuition that indirectly affects the semantics is whether arguments should be evaluated on construction (bottom-up) or after their construction (top-down). Some proponents of *evaluation on construction* such as Horty prefer an approach in which "*arguments are constructed step-by-step and are evaluated in each step of their construction: those that are indefensible (ambiguous) (...) are discarded at once, and so cannot influence the status of others*" [Horty, 2002]. This approach is used as a justification for ambiguity blocking and team defeat, however it is prone to *support and attack cycles* and might lead to *circular reasoning* which is generally seen as a logical fallacy.

4. **Evaluation on Construction:** Most argumentation based techniques construct arguments first while most non-monotonic logics evaluate arguments on construction. This latter approach starts with a conclusion then evaluates its premises and its attacking rule applications which makes the formalism sensible to support and attack cycles.

   Support Cycle: is when a rule application generates one of the premises it relies on. This might lead to an infinite cycle of evaluating the conclusion then the premise then the conclusion and so on as shown in the following example.

   Example 2.18 (Support Cycle). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *for representing legal contracts. A person is generally an individual and an individual is a person. "bob" is a person. Is "bob" an individual* $Q = \mathit{individual}(\mathit{bob})$*? The inheritance network for* $\mathit{person}(\mathit{bob})$ *is shown in Figure 2.8.*

   - $\mathcal{F} = \{\top \rightarrow \mathit{person}(\mathit{bob})\}$
   - $\mathcal{R} = \{r_1 : \forall X\, \mathit{person}(X) \Rightarrow \mathit{individual}(X),\ r_2 : \forall X\, \mathit{individual}(X) \Rightarrow \mathit{person}(X)\}$

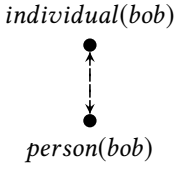

Figure 2.8: Inheritance Network for $\mathit{person}(\mathit{bob})$ of Example 2.18

If arguments are evaluated on construction, evaluating Q would require evaluating the rule application of r_1 *generating* $\mathit{individual}(\mathit{bob})$*,*

which would require evaluating $\mathit{person}(\mathit{bob})$*, this in turn can either stop by noticing that* $\mathit{person}(\mathit{bob})$ *is a fact or it might require evaluating* $\mathit{individual}(\mathit{bob})$ *given* r_2 *and continue on and on in an infinite support cycle.*

In its extreme form, a support cycle might lead to circular reasoning. Most non-monotonic formalisms have now incorporated a *failure-by-looping* mechanism that avoids circular reasoning which is generally seen as a logical fallacy [Brewka, 2001, Maier and Nute, 2010a].

Example 2.19 (Circular Reasoning). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *and the query* $Q = p(a)$*?*

- $\mathcal{F} = \{\top \Rightarrow p(a)\}$
- $\mathcal{R} = \{r_1 : q(a) \rightarrow q(a)\}$
- $\mathcal{N} = \{\forall X\, p(X) \wedge q(X) \rightarrow \bot\}$

Although one could easily see that r_1 *would never be applicable as there is no* $q(a)$*, evaluating* $p(a)$ *on construction would require evaluating* $q(a)$ *as an attacker which in turn would require evaluating* $q(a)$ *and so on. Non-monotonic logics without looping-as-failure would not be able to prove* $p(a)$*.*

Attack Cycle: is when two arguments "undermine" each other, meaning that arg_1 attacks a premise in arg_2 and arg_2 attacks a premise in arg_1 as shown in the following example.

Example 2.20 (Attack Cycle). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *stating that an animal with fur is generally a mammal. An animal that lays eggs is generally a bird. Birds do not have fur and mammals generally do not lay eggs. Suppose we have an animal called "Platy" that appears to have fur and lay eggs. Is it a bird* $(Q = \mathit{bird}(\mathit{platy}))$*? The inheritance network for* $\mathit{bird}(\mathit{platy})$ *is shown in Figure 2.9.*

- $\mathcal{F} = \{\top \Rightarrow \mathit{fur}(\mathit{platy}),\ \top \Rightarrow \mathit{layEggs}(\mathit{platy})\}$
- $\mathcal{R} = \{r_1 : \forall X\, \mathit{fur}(X) \Rightarrow \mathit{mammal}(X),\ r_2 : \forall X\, \mathit{layEggs}(X) \Rightarrow \mathit{bird}(X),\ r_3 : \forall X\, \mathit{mammal}(X) \Rightarrow \mathit{notLayEggs}(X),\ r_4 : \forall X\, \mathit{bird}(X) \rightarrow \mathit{notFur}(X)\}$
- $\mathcal{N} = \{\forall X\, \mathit{layEggs}(X) \wedge \mathit{notLayEggs}(X) \rightarrow \bot, \forall X\, \mathit{fur}(X) \wedge \mathit{notFur}(X) \rightarrow \bot\}$

If arguments are evaluated on construction, evaluating $Q = \mathit{bird}(\mathit{platy})$ *means evaluating* $\mathit{layEggs}(\mathit{platy})$ *which would require evaluating the rule application of* r_3 *as it generates the conflicting atom* $\mathit{notLayEggs}(\mathit{platy})$*, which would require evaluating* $\mathit{mammal}(\mathit{platy})$ *then* $\mathit{fur}(\mathit{platy})$ *that requires evaluating its conflicting atom* $\mathit{notFur}(\mathit{platy})$*, this in turn would require evaluating* $\mathit{bird}(\mathit{platy})$ *and so on in an infinite attack cycle.*

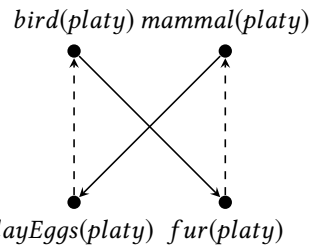

Figure 2.9: Inheritance Network of Example 2.20

One might find it difficult to understand the problem of support and attack cycles without a concrete formalism of a non-monotonic logic, in the next section we will present defeasible logics and clearly show the problems related to support and attack cycles.

The definition of what constitutes an argument changes from a formalism to another. Some, such as Defeasible Logics, consider an argument as a *directly consistent* derivation for a conclusion. Others such as Defeasible Logic Programming consider an argument as an *indirectly consistent* derivation for a conclusion. This difference has a direct effect on the semantics.

5. **How strict rule are handled:** some logicians argue that facts that are derivable and not in direct conflict should be accepted, others point out that since strict rules are definite implications, facts that would lead to a conflict if strict rules are applied should not be accepted as shown in the following example:

   Example 2.21. *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *stating that a person with a wedding ring is generally married. A person that is married has a wife. A bachelor does not have a wife. A person that is happy is generally a bachelor. John is a happy person with a wedding ring. Is John married* ($Q_1 = married(john)$)*? Is he a bachelor* ($Q_2 = bachelor(john)$)*? The inheritance network is shown in Figure 2.10.*

   - $\mathcal{F} = \{\top \rightarrow weddingRing(john), \top \rightarrow happy(john)\}$
   - $\mathcal{R} = \{r_1 : \forall X\, weddingRing(X) \Rightarrow married(X),\ r_2 : \forall X\, married(X) \rightarrow hasWife(X),\ r_3 : \forall X\, bachelor(X) \rightarrow noWife(X),\ r_4 : \forall X\, happy(X) \Rightarrow bachelor(X),$
   - $\mathcal{N} = \{\forall X\, hasWife(X) \wedge noWife(X) \rightarrow \bot\}$

   Some formalisms such as Defeasible Logics consider $Q_1 = married(john)$ *and* $Q_2 = bachelor(john)$ *"true" because there is a directly consistent derivation for* $bachelor(john)$ *and* $married(john)$ *that is not attacked by any other directly consistent derivation, the entailed facts in this case*

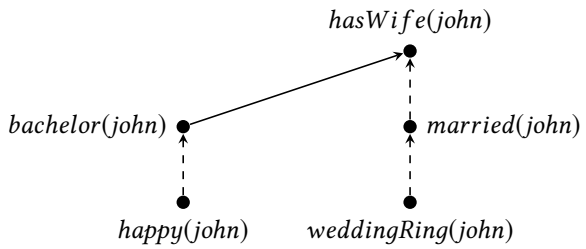

Figure 2.10: Inheritance Network of Example 2.21

are $\{happy(john), weddingRing(john), married(john), bachelor(john)\}$. *However, if we consider strict rules, we can see that* $married(john)$ *and* $bachelor(john)$ *are conflicting because they generate* $hasWife(john)$ *and* $noWife(john)$ *when strict rules are applied. Some formalisms such as Defeasible Logic Programing discard any fact that can generate conflicts when strict rules are applied.*

The previously discussed intuitions are, of course, not the only intuitions in Defeasible Reasoning, "*implicit preferences*", "*consistent answers*", and "*default negation*" among others are also important points of discussion between logicians. Given that our main objective is to handle conflicts in existential rules, we will mainly focus on ambiguity handling, team defeat, floating conclusions, consistent derivations, and cycles. Incorporating other intuitions will be the subject of future work.

The main Defeasible Reasoning techniques that will be discussed in this thesis are variants of non-monotonic logics called Defeasible Logics and the argumentation based techniques: Dialectical Trees and Grounded Semantics.

2.2.3 Defeasible Logics

Defeasible Logics are a simple rule-based skeptical form of non-monotonic reasoning originally proposed by Nute [Nute, 1988]. Their appeal resides in their low computational complexity and high flexibility [Antoniou et al., 2000a]. They have been applied in various domains such as modeling regulations and business rules [Antoniou et al., 1999], legal reasoning [Governatori and Rotolo, 2010], agent negotiations [Governatori and Rotolo, 2008, Governatori and Rotolo, 2004], modeling of contracts [Governatori, 2005], planning [Dastani et al., 2005], and Semantic Web [Bassiliades et al., 2006, Kravari et al., 2010]. Defeasible Logics are defined using a propositional defeasible language and extended to a first order defeasible language without the existential quantifier or function symbols.

2.2.3.1 Propositional Defeasible Language

Consider a propositional language $\mathcal{L}_p$ with the implication and conjunction connectives $(\rightarrow, \Rightarrow, \rightsquigarrow, \wedge)$ and strong negation $(\neg)$ on a finite set of literals (facts) where a literal is either an atomic proposition or its negation. Given a literal f, $\overline{f}$ denotes its complement, that is, for an atomic proposition p, $\overline{p} = \neg p$ and $\overline{\neg p} = p$.

Notation 2.6 (Negative Constraint and $\neg$). *In the context of Defeasible Reasoning, strong negation $(\neg)$ can be seen as a succinct representation of a negative constraint [Antoniou, 2006]. For example $\neg p$ can be translated to a literal np with the negative constraint $p \wedge np \rightarrow \bot$. We omit the negative constraint when using $\neg$.*

A fact in $\mathcal{L}_p$ is simply a literal (so is a ground atomic query) while the body and head of a rule are finite conjunction of literals. A rule expressed in $\mathcal{L}_p$ is applicable on a set of literals if its body is included in this set.

Example 2.22 (Propositional $\mathcal{KB}$). *Consider the knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ of a penguin with broken wings expressed in $\mathcal{L}_p$:*

- $\mathcal{F} = \{\top \rightarrow penguin\}$
- $\mathcal{R} = \{penguin \rightarrow bird, penguin \rightarrow \neg fly, bird \Rightarrow fly, brokenwings \rightsquigarrow \neg fly\}$

In Defeasible Logics, arguments are simply a *locally consistent derivation* for a fact. They are evaluated on construction using proofs which are linear sequences $Proof = \langle Proof(1), ..., Proof(n)\rangle$ of labeled literals. A literal f in a $\mathcal{KB}$ expressed in $\mathcal{L}_p$ has one of four possible labels describing its entailment (provability):

- $+\Delta f$ means that f is strictly (definitely) provable.
- $-\Delta f$ means that it is proved that f is not strictly provable.
- $+\delta f$ means that f is defeasibly provable.
- $-\delta f$ means it is proved that f is not defeasibly provable.

To define the labeling of literals we use the formalism of [Billington, 1993]. The conditions for each label are essentially inference rules phrased as conditions on proofs. $Proof(1..i)$ denotes the initial part of the sequence $Proof$ of length i. $\mathcal{F}_{\rightarrow}$ denotes strict fact rules.

- $+\Delta$: If $Proof(i+1) = +\Delta f$ then $\exists r \in \mathcal{R}_{\rightarrow} \cup \mathcal{F}_{\rightarrow}$ s.t. $Head(r) = f$ and $\forall \phi \in Body(r) : +\Delta\phi \in Proof(1..i)$.
- $-\Delta$: If $Proof(i+1) = -\Delta f$ then $\forall r \in \mathcal{R}_{\rightarrow} \cup \mathcal{F}_{\rightarrow}$ s.t. $Head(r) = f$, $\exists \phi \in Body(r) : -\Delta\phi \in Proof(1..i)$.

In order to show that a literal f is strictly provable, we need to show that there is a strict derivation for f, specifically: there is a strict rule for f and all literals in its body are deducible with strict rules and so on recursively until we reach a fact rule (*since* $\top$ *is always strictly provable*). On the other hand, in order to show that it is proved that f is not strictly provable, we need to show that there is no strict derivation for f (i.e. no strict rule for f can be applied by chaining strict rules starting form $\mathcal{F}$). For instance, in Example 2.22 $+\Delta penguin$, $+\Delta bird$, $+\Delta\neg fly$, and $-\Delta fly$. *Defeasible provability* relies on the notions of supported and defeated rules:

- A rule is *supported* if and only if all its premises are defeasibly provable. A rule is applicable if and only if it is supported.
- A rule for f is *defeated* if and only if there is an applicable rule for $\overline{f}$ that is superior to it.

Defeasible Logics come in different "flavors" of defeasible provability that are "tunable" to the desired set of intuitions. Nevertheless they all follow the same structure of proof in three phases:

1. In the first phase we put forward a rule for the literal we want to prove.
2. In the second phase we consider all possible rule applications against the literal.
3. In the third phase we rebut the attacks with two possible options:
   (a) we either show that the attack is unsupported, i.e. some of the premises do not hold, or
   (b) we can defeat the attacking rule by providing a stronger applicable rule for the literal.

2.2.3.2 Ambiguity Blocking with Team Defeat

Defeasible Logic for ambiguity blocking with team defeat is closely related to the "directly skeptical" semantics of non-monotonic inheritance networks [Antoniou et al., 2000b] "arguments are constructed step-by-step and are evaluated in each step of their construction: those that are indefensible [ambiguous] (...) are discarded at once, and so cannot influence the status of others." [Horty, 2002].

All Defeasible Logics follow the same definition for strict (non)provability. Defeasible provability on the other hand requires considering the arguments for conflicting literals and possible resolution of these conflicts using the preference relation (Defeasible Logics only consider preference relation between rules). Defeasible provability for ambiguity blocking with team defeat is denoted $+\delta^{TD}_{block}$ and defined as follows:

- $+\delta^{TD}_{block}$: If $Proof(i+1) = +\delta^{TD}_{block} f$ then either f is strictly provable or the following three conditions hold:
  1. there is an applicable strict or defeasible rule for f and
  2. $\overline{f}$ is not strictly provable and
  3. every rule for $\overline{f}$ is either:
     3.1. unsupported (one of the premises is not provable) or
     3.2. defeated (overridden by a superior applicable rule for f).

Formally:

- $+\delta^{TD}_{block}$: If $Proof(i+1) = +\delta^{TD}_{block} f$ then either $+\Delta f \in Proof(1..i)$ or the following three conditions hold:
  1. $\exists r \in (\mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow} \cup \mathcal{F})$ s.t. $Head(r) = f$ and $\forall \phi \in Body(r) : +\delta^{TD}_{block} \phi \in Proof(1..i)$ and
  2. $-\Delta\overline{f} \in Proof(1..i)$ and
  3. $\forall r' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r') = \overline{f}$, either:
     3.1. $\exists \phi \in Body(r') : -\delta^{TD}_{block} \phi \in Proof(1..i)$, or
     3.2. $\exists r'' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r'') = f$ and $\forall \phi \in Body(r'') : +\delta^{TD}_{block} \phi \in Proof(1..i)$ and $r'' \succ r'$.

Defeasible non-provability (denoted $-\delta^{TD}_{block}$) is simply the negation of defeasible provability and is defined as follows:

- $-\delta^{TD}_{block}$: If $Proof(i+1) = -\delta^{TD}_{block} f$ then f is not strictly provable and one of the following three conditions holds:
  1. either there is no applicable strict or defeasible rule for f or
  2. $\overline{f}$ is strictly provable or
  3. there is a rule for $\overline{f}$ such that
     3.1. the rule is supported and
     3.2. the rule is not defeated.

Formally:

- $-\delta^{TD}_{block}$: If $Proof(i+1) = -\delta^{TD}_{block} f$ then $-\Delta f \in Proof(1..i)$ and one of the following three conditions holds:
  1. $\forall r \in (\mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow} \cup \mathcal{F})$ s.t. $Head(r) = f$, $\exists \phi \in Body(r) : -\delta^{TD}_{block} \phi \in Proof(1..i)$ or
  2. $+\Delta\overline{f} \in Proof(1..i)$ or
  3. $\exists r' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r') = \overline{f}$ and:

3.1. $\forall \phi \in Body(r') : +\delta^{TD}_{block}\phi \in Proof(1..i)$ and
3.2. $\forall r'' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r'') = f$, $\exists \phi \in Body(r'') : -\delta^{TD}_{block}\phi \in Proof(1..i)$ or $r'' \not> r'$.

Example 2.23. *Consider the knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ of Example 2.15 expressed in the propositional language $\mathcal{L}_p$:*

- $\mathcal{F} = \{\top \Rightarrow e1, \top \Rightarrow e2, \top \Rightarrow alibi\}$
- $\mathcal{R} = \{r_1 : e1 \rightarrow responsible, r_2 : e2 \rightarrow \neg responsible,$
  $r_3 : responsible \rightarrow guilty, r_4 : alibi \rightarrow \neg guilty\}$

Applying the Defeasible Logic with ambiguity blocking would result in:

- $+\delta^{TD}_{block}e1$, $+\delta^{TD}_{block}e2$, *and* $+\delta^{TD}_{block}alibi$ *because there is a defeasible unattacked rule for these literals.*
- $-\delta^{TD}_{block}responsible$ *and* $-\delta^{TD}_{block}\neg responsible$ *because there is a rule application for their conflicting atoms that relies on accepted premises ($+\delta^{TD}_{block}e2$, $+\delta^{TD}_{block}e1$ respectively) and is not overridden.*
- $-\delta^{TD}_{block}guilty$ *because the only rule application for this literal relies on the rejected ambiguous premise $-\delta^{TD}_{block}responsible$.*
- $+\delta^{TD}_{block}\neg guilty$ *because there is an applicable rule application for it using $+\delta^{TD}_{block}alibi$ and the attacking rule application is blocked since it relies on rejected premises $-\delta^{TD}_{block}responsible$.*

2.2.3.3 Ambiguity Propagating with Team Defeat

Another variant of Defeasible Logics is ambiguity propagating with team defeat [Antoniou et al., 2000a]. Compared to ambiguity blocking, ambiguity propagating considers attacks coming from ambiguous facts. In order to do that, this logic introduces a new label $\sum^{TD}$ indicating if there is a non overridden derivation for a literal by considering team defeat.

- $+\sum^{TD}$: If $Proof(i+1) = +\sum^{TD} f$ then either f is strictly provable or:
  1. there is a strict or defeasible rule for f with derivable premises and
  2. $\overline{f}$ is not strictly provable and
  3. every rule r' for $\overline{f}$ is either:
     3.1. unsupported (w.r.t. defeasible provability with ambiguity propagating and team defeat).
     3.2. or there is a rule for f with derivable premises that is not inferior to r'.

Formally:

- $+\sum^{TD}$: If $Proof(i+1) = +\sum^{TD} f$ then either $+\Delta f \in Proof(1..i)$ or:
  1. $\exists r \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r) = f$ and $\forall \phi \in Body(r) : +\sum^{TD} \phi \in Proof(1..i)$ and
  2. $-\Delta\overline{f} \in Proof(1..i)$ and
  3. $\forall r' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r') = \overline{f}$ either:
     3.1. $\exists \phi \in Body(r') : -\delta^{TD}_{prop}\phi \in Proof(1..i)$, or
     3.2. $\exists r'' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r'') = f$ and $\forall \phi \in Body(r'') : +\sum^{TD} \phi \in Proof(1..i)$ and $r' \not\succ r''$.

Non derivability (denoted $-\sum^{TD}$) is defined as follows:

- $-\sum^{TD}$: If $Proof(i+1) = -\sum^{TD} f$ then f is not strictly provable and:
  1. either there is no strict or defeasible rule for f with derivable premises or
  2. $\overline{f}$ is strictly provable or
  3. there is a rule r' for $\overline{f}$ such that:
     3.1. r' is supported (w.r.t. defeasible provability with ambiguity propagating and team defeat) and
     3.2. r' is superior to all rules for f with derivable premises.

Formally:

- $-\sum^{TD}$: If $Proof(i+1) = -\sum^{TD} f$ then either f is not strictly provable and:
  1. $\forall r \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r) = f$, $\exists \phi \in Body(r) : -\sum^{TD} \phi \in Proof(1..i)$ or
  2. $+\Delta\overline{f} \in Proof(1..i)$ or
  3. $\exists r' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r') = \overline{f}$ and:
     3.1. $\forall \phi \in Body(r') : +\delta^{TD}_{prop}\phi \in Proof(1..i)$ and
     3.2. $\forall r'' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r'') = f$, $\exists \phi \in Body(r'') : -\sum^{TD} \phi \in Proof(1..i)$ or $r' \succ r''$.

The ambiguity propagation defeasible provability ($+\delta^{TD}_{prop}$) variant of Defeasible Logic can be easily achieved by changing the conditions of $+\delta^{TD}_{block}$ to allow for attacks from derivable literals.

- $+\delta^{TD}_{prop}$: If $Proof(i+1) = +\delta^{TD}_{prop} f$ then either $+\Delta f \in Proof(1..i)$ or the following three conditions hold:

1. $\exists r \in (\mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow} \cup \mathcal{F})$ s.t. $Head(r) = f$ and $\forall \phi \in Body(r) : +\delta^{TD}_{prop}\phi \in Proof(1..i)$ and
2. $-\Delta\overline{f} \in Proof(1..i)$ and
3. $\forall r' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r') = \overline{f}$, either:
   3.1. $\exists \phi \in Body(r') : -\sum^{TD} \phi \in Proof(1..i)$, or
   3.2. $\exists r'' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r'') = f$ and $\forall \phi \in Body(r'') : +\delta^{TD}_{prop}\phi \in Proof(1..i)$ and $r'' \succ r'$.

Defeasible provability for a literal f in ambiguity propagation is the same as for ambiguity blocking, the difference however, is that any rule for $\overline{f}$ has either a premise that is not derivable (3.1), or there is a defeasible applicable rule for f that is superior to it. (i.e. an attacking rule only need to rely on derivable premises to be considered). Defeasible nonprovability in ambiguity propagation (denoted $-\delta^{TD}_{prop}$) is as follows:

- $-\delta^{TD}_{prop}$: If $Proof(i+1) = -\delta^{TD}_{prop}f$ then $-\Delta f \in Proof(1..i)$ and one of the following three conditions holds:
  1. $\forall r \in (\mathcal{R}_{\rightarrow} \cup \mathcal{R}_{\Rightarrow} \cup \mathcal{F})$ s.t. $Head(r) = f$, $\exists \phi \in Body(r) : -\delta^{TD}_{block}\phi \in Proof(1..i)$ or
  2. $+\Delta\overline{f} \in Proof(1..i)$ or
  3. $\exists r' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r') = \overline{f}$ and:
     3.1. $\forall \phi \in Body(r') : +\sum^{TD} \phi \in Proof(1..i)$ and
     3.2. $\forall r'' \in \mathcal{R} \cup \mathcal{F}$ s.t. $Head(r'') = f$, $\exists \phi \in Body(r'') : -\delta^{TD}_{prop}\phi \in Proof(1..i)$ or $r'' \not\succ r'$.

A literal f is not defeasibly provable in ambiguity propagation if there is an attacking rule with derivable premises such that no rule for f is applicable on defeasibly provable premises or is superior to it.

Example 2.24. *Consider $\mathcal{KB}$ of the previous Example 2.23:*

- $+\delta^{TD}_{prop}e1$, $+\delta^{TD}_{prop}e2$, *and* $+\delta^{TD}_{prop}alibi$ *because there is a defeasible unattacked rule for these literals.*
- $-\delta^{TD}_{prop}responsible$ *and* $-\delta^{TD}_{prop}\neg responsible$ *because there is a rule application for their conflicting atoms that relies on accepted premises ($+\delta^{TD}_{prop}e2$, $+\delta^{TD}_{prop}e1$ respectively) and is not overridden.*
- $-\delta^{TD}_{prop}guilty$ *because the only rule application for this literal relies on the rejected ambiguous premise $-\delta^{TD}_{prop}responsible$.*
- $-\delta^{TD}_{prop}\neg guilty$ *because there is an attacking rule that relies on derivable premises ($+\sum responsible$) and is not inferior to the rule for $\neg guilty$.*

2.2.3.4 Removing Team Defeat

Defeasible Logics discussed so far incorporate the idea of team defeat. That is, an attack on a rule with head f by a rule for $\overline{f}$ may be defeated by a different rule for f. One might find the idea of team defeat natural, however many other non-monotonic formalism and most argumentation frameworks do not adopt this idea. To remove team defeat we only need to change one condition (3.2) in the definitions of defeasible (non)provability to the following [Billington et al., 2010]:

- for defeasible provability ($+\delta^{noTD}$): 3.2. $r \succ r'$.
- for defeasible non provability ($-\delta^{noTD}$): 3.2. $r' \not\succ r$.
- for derivability ($+\sum^{noTD}$): 3.2. $r' \not\succ r$.
- for non derivability ($-\sum^{noTD}$): 3.2. $r' \succ r$.

This means that a rule that is defeasibly applicable for f has to override, all by itself, any attack it receives.

Example 2.25. *Consider the $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ of Example 2.16 expressed in the propositional language $\mathcal{L}_p$:*

- $\mathcal{F} = \{\top \rightarrow cheap, \top \rightarrow goodReviews, \top \rightarrow detrimental, \top \rightarrow slowDelivery\}$
- $\mathcal{R} = \{r_1 : cheap \Rightarrow buy, goodReviews \Rightarrow buy,$
  $detrimental \Rightarrow \neg buy, slowDelivery \Rightarrow \neg buy\}$
- $(r_1 \succ r_3), (r_2 \succ r_4)$

If team defeat is allowed, we can see that $+\delta^{TD}buy$ because for every attacking rule there is a rule that overrides it. However, if team defeat is removed, then $-\delta^{noTD}buy$ because there is no single rule for buy that is superior to all attacking rules.

The authors in [Antoniou et al., 2000b, Billington et al., 2010] stated that all variants of Defeasible Logics are coherent in the sense that no literal is provable and not provable at the same time.

Theorem 2.2 ([Billington et al., 2010]). *Given a knowledge base $\mathcal{KB}$ with a coherent set of strict rules (i.e. there is no literal f such that $\mathcal{KB} \vdash +\Delta f$ and $\mathcal{KB} \vdash +\Delta\overline{f}$) then there is no literal f such that $\mathcal{KB} \vdash +Lblf$ and $\mathcal{KB} \vdash -Lblf$ where lbl denotes any of the discussed labels (δ^{TD}_{block}, δ^{noTD}_{block}, δ^{TD}_{prop}, δ^{noTD}_{prop}, $\sum^{TD}$, $\sum^{noTD}$).*

Furthermore, the "strength" (degree of cautiousness) of the different conditions of provability can be compared. The most cautious inference is strict inference $+\Delta$, the least cautious one is $+\sum^{noTD}$. Accordingly, the

strongest condition in rejecting a literal is strict non provability $-\Delta$, the weakest one is $-\sum^{noTD}$. A label Lbl_1 is included in another label Lbl_2 if any literal that is labeled Lbl_2 is also labeled Lbl_1.

Theorem 2.3 (Inclusion [Billington et al., 2010]).

1. $+\Delta \subseteq +\delta^{noTD}_{prop} \subseteq +\delta^{TD}_{prop} \subseteq +\delta^{TD}_{block} \subseteq +\sum^{TD} \subseteq +\sum^{noTD}$
2. $+\sum^{noTD} \subseteq +\sum^{TD} \subseteq -\delta^{TD}_{block} \subseteq -\delta^{TD}_{prop} \subseteq -\delta^{noTD}_{prop} \subseteq -\Delta$
3. *For each inclusion relationship there exists a knowledge base $\mathcal{KB}$ such that the inclusion is strict.*

The inclusions (1) and (2) in Theorem 2.3 are to be expected. For example, the relation $+\delta^{noTD}_{prop} \subseteq +\delta^{TD}_{prop}$ appears trivial since the absence of team defeat makes the inference rule weaker. However there is a potential source of complication: when the logic fails to prove a literal f and instead shows its non-provability, then that result may be used by the logic to prove another literal f' that could not be proven if f were provable. That is why $+\delta^{noTD}_{block}$ is not included in $+\delta^{TD}_{block}$ (while $+\delta^{noTD}_{prop}$ is included in $+\delta^{TD}_{prop}$).

2.2.3.5 Support and Attack Cycles

Defeasible Logics evaluate argument on construction which makes them prone to infinite loops in presence of support and attack cycles. Several efforts have been made to incorporate "loop-checking" into the inference mechanism of the logics. Nute among others, in a series of works [Maier and Nute, 2010a, Maier and Nute, 2006], have investigated Defeasible Logic with explicit "*failure-by-looping*" mechanisms and their relationship to well-founded semantics of logic programs. In particular, they have defined two Defeasible Logics, namely: NDL and ADL, and shown to be equivalent to well-founded semantics. Concurrently, Billington [Billington, 2004, Billington, 2008] has developed several logics that involve incorporating loop checking mechanisms. Other work [Governatori et al., 2004, Billington et al., 2010] investigated Defeasible Logics with various inference structures under the original notion of failure detection [Antoniou et al., 2001].

Failure-by-looping provides a mechanism for falsifying ($-\delta$ or $-\sum$) a literal when it is within a loop that cannot be avoided by using rules outside the cycle. If a literal f is within a support cycle that cannot be avoided by considering other rules then it cannot be derived ($-\sum f$, $-\delta^{TD}_{block} f$, and $-\delta^{noTD}_{block}$). If it is within an attack cycle that cannot be avoided by considering other rules and preferences then it cannot be defeasibly proven ($-\delta^{TD}_{block} f$, $-\delta^{noTD}_{block} f$, $-\delta^{TD}_{prop} f$, $-\delta^{noTD}_{prop} f$) [Billington, 2004, Lam, 2012]. Failure-by-looping can be implemented by keeping track of evaluated literals in a "bin" as the proof procedure recursively calls itself, as soon as a literal appears

twice in the same evaluation then its argument is discarded and other rules for the literal are evaluated if there is any [Billington, 2004]. However, this loop checking has a computational cost as we will discuss in the next section.

Example 2.26 (Support Cycles). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *of the support cycle Example 2.18 expressed in* $\mathcal{L}_p$ *and stating that from a legal point of view a person is generally an individual. An individual is a person. A company is not an individual, and a company is a company (circular reasoning). Suppose we have a person and we want to know if it is an individual* $Q = individual$.

- $\mathcal{F} = \{\top \rightarrow person\}$
- $\mathcal{R} = \{r_1 : person \Rightarrow individual,\ r_2 : individual \Rightarrow person,\ r_3 : company \rightarrow company,\ r_4 : company \rightarrow \neg individual\}$

Evaluating Q *requires evaluating the application of* r_1 *that relies on* $person$ *which might lead to a loop if* r_2 *is evaluated next. However, failure-by-looping also considers other rules for* $person$ *and can prove that* $+\Delta person$ *given the fact rule. Then the attacking rule for* $\neg individual$ *is evaluated along with its premise* $company$ *which is generated by the support loop of* r_3. *Given failure-by-looping we can deduce* $-\sum company$, $-\delta company$ *(*$-\delta$ *denotes defeasible non-provability with ambiguity blocking or propagating with or without team defeat), therefore the attacking rule cannot be applied and the query is defeasibly provable* $+\delta individual$.

Example 2.27 (Attack Cycle). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *of Example 2.20 expressed in* $\mathcal{L}_p$.

- $\mathcal{F} = \{\top \Rightarrow fur, \top \Rightarrow layEggs\}$
- $\mathcal{R} = \{r_1 : fur \Rightarrow mammal,\ r_2 : layEggs \Rightarrow bird,$
  $r_3 : mammal \Rightarrow \neg layEggs,\ r_4 : bird \rightarrow \neg fur\}$

Evaluating the query $Q = bird$ *would result in an attack cycle (a.k.a. negative loop [Billington, 2004]) that cannot be avoided. Given failure-by-looping we can deduce* $-\delta bird$, $-\delta mammal$, $-\delta fur$, $-\delta layEggs$.

2.2.3.6 First Order Defeasible Language

Defeasible Logics have been defined for the propositional language $\mathcal{L}_p$. However, they can be applied to a first order language without existential quantifier (denoted $\mathcal{L}_\forall$) built with the universal quantifier $\forall$, the connectives $\rightarrow$, $\Rightarrow, \rightsquigarrow, \wedge$, and atomic negation $\neg$ [Billington et al., 2010].

In this language $\mathcal{L}_\forall$, facts are simply ground atoms and rules are of the form $\forall \vec{X}, \vec{Y}\, \mathcal{B}(\vec{X}, \vec{Y}) \Rrightarrow \mathcal{H}(\vec{Y})$ where $\Rrightarrow \in \{\rightarrow, \Rightarrow, \rightsquigarrow\}$, $\vec{X}$ and $\vec{Y}$ are tuples of universally quantified variables, and $\mathcal{B}$,$\mathcal{H}$ are conjunctions of atom.

Using a grounding procedure, rules in $\mathcal{L}_\forall$ can be transformed into a set of ground rules without variables that can be seen as rules expressed in $\mathcal{L}_p$, this grounding phase however has a computational cost [Chandra et al., 1981].

Defeasible Logics evaluate arguments on construction, another Defeasible Reasoning technique is Dialectical Trees that constructs arguments first then evaluates them in a tree describing the defeat relation between them.

2.2.4 Dialectical Trees

Dialectical Trees [García and Simari, 2004] are an argumentation-based formalism for Defeasible Reasoning with the aim to uncover which information prevails in a conflict (i.e. which piece of information is such that no acceptable reason (argument) can be put forward against it). Dialectical Trees are used as a *warrant* (proof procedure) for accepting a literal in Defeasible Logic Programming (DeLP) [García and Simari, 2004].

Dialectical Trees have been firstly defined for the defeasible propositional language $\mathcal{L}_p$ without defeater rules (only strict and defeasible rules). Using a grounding phase, they can also be applied to the first order defeasible language $\mathcal{L}_\forall$ without the existential quantifier and without defeater rules.

Within the context of Dialectical Trees, a knowledge base is seen as a logic program with strict and defeasible facts and rules. The definition of derivation is the same as the one presented previously except that the rules are not represented.

Definition 2.36 (Derivation [García and Simari, 2004]). *Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ and a literal f. A derivation is a finite sequence of ground atoms $\langle f_1, \ldots, f_n = f \rangle$ such that f_i is either a fact or there is a rule with the head f_i and every literal of its body appears before f_i. We say that $\mathcal{KB} \vdash f$ iff there is a derivation for f in $\mathcal{KB}$.*

An argument for a literal f is a minimal consistent set of defeasible rules that allows (possibly using strict facts and strict rules) to derive f.

Definition 2.37 (Argument [García and Simari, 2004]). *Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$, an argument for a literal f is a tuple $\mathbf{arg} = \langle \mathcal{A}, f \rangle$ where $\mathcal{A} \subseteq \mathcal{F}_\Rightarrow \cup \mathcal{R}_\Rightarrow$ is a set of defeasible facts and rules s.t.*

1. *there is a derivation for f from $\mathcal{A} \cup \mathcal{F}_\rightarrow \cup \mathcal{R}_\rightarrow$*
2. *no conflicting atoms can be derived from $\mathcal{A} \cup \mathcal{F}_\rightarrow \cup \mathcal{R}_\rightarrow$*
3. *$\mathcal{A}$ is minimal: there is no proper subset $\mathcal{A}'$ of $\mathcal{A}$ such that $\mathcal{A}'$ satisfies condition (1) and (2)*

An argument $\mathbf{arg} = \langle \mathcal{A}, f \rangle$ is a sub argument of $\mathbf{arg}' = \langle \mathcal{A}', f' \rangle$ iff $\mathcal{A} \subseteq \mathcal{A}'$.

An argument attacks another argument if its conclusion combined with a literal in the second argument leads to conflicting atoms when strict facts and rules are applied.

Definition 2.38 (Counter-argument). *An argument $arg_1 = \langle \mathcal{A}_1, f_1 \rangle$ attacks (counter-argues) an argument $arg_2 = \langle \mathcal{A}_2, f_2 \rangle$ at fact f of arg_2 if and only if there exists a sub-argument $arg' = \langle \mathcal{A}', f' \rangle$ of arg_1 such that conflicting atoms can be derived from $\{f', f\} \cup \mathcal{F}_{\rightarrow} \cup \mathcal{R}_{\rightarrow}$.*

Example 2.28 (Arguments and Counter-arguments). *Consider the following $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ expressed in $\mathcal{L}_{\forall}$ stating that chickens are birds that generally do not fly, generally a scared chicken flies, penguins are birds that do not fly. Tina is a scared chicken, does Tina fly $Q = fly(tina)$? :*

- $\mathcal{F} = \{\top \rightarrow chicken(tina),\ \top \rightarrow scared(tina)\}$
- $r_1 : \forall X\ chicken(X) \rightarrow bird(X)$,
  $r_4 : \forall X\ bird(X) \Rightarrow fly(X)$,
  $r_5 : \forall X\ chicken(X) \Rightarrow \neg fly(X)$,
  $r_4 : \forall X\ chicken(X) \wedge scared(X) \Rightarrow fly(X)\}$

To evaluate a query, all the arguments for and against it are constructed. For example, there are two arguments for $fly(tina)$: $arg_1 = \langle \{bird(tina) \Rightarrow fly(tina)\},\ fly(tina) \rangle$ and $arg_2 = \langle \{chicken(tina) \wedge scared(tina) \Rightarrow fly(tina)\},\ fly(tina) \rangle$. There is also the counter-argument $arg_3 = \langle \{chicken(tina) \Rightarrow \neg fly(tina)\},\ \neg fly(tina) \rangle$.

It is worth noticing that an argument for a strictly derived literal has an empty $\mathcal{A}$, for example the argument for $bird(tina)$ is $arg_4 = \langle \emptyset,\ bird(tina) \rangle$. Furthermore, this argument cannot have or be a counter-argument for any other argument [García and Simari, 2004].

2.2.4.1 Comparing Arguments

Within the DeLP framework, arguments and their counter-arguments can be compared based on any arbitrary preference between arguments. However, two preference relations have been defined, one that is implicit called "*Generalized Specificity*", and another that relies on explicit preference between defeasible rules and elevates it to a preference between arguments.

Generalized specificity favors two aspects in an argument: it prefers an argument (1) with greater information content (more precise i.e. relies on more facts) or (2) with fewer use of rules (more concise).

Definition 2.39 (Specificity Preference [García and Simari, 2004]). *Let $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ be a knowledge base, $\mathcal{F}^*$ the set of all literals that have a derivation, and $arg_1 = \langle \mathcal{A}_1, f_1 \rangle$ and $arg_2 = \langle \mathcal{A}_2, f_2 \rangle$ be two argument obtained from $\mathcal{KB}$. arg_1 is more specific than arg_2 (denoted $arg_1 \succ arg_2$) if the following conditions hold:*

1. *for all* $\mathcal{F}' \subset \mathcal{F}^*$*: if* $\mathcal{R}_{\rightarrow} \cup \mathcal{F}' \cup \mathcal{A}_1 \vdash f_1$ *and* $\mathcal{R}_{\rightarrow} \cup \mathcal{F}' \nvdash f_1$*, then* $\mathcal{R}_{\rightarrow} \cup \mathcal{F}' \cup \mathcal{A}_2 \vdash f_2$*, and*

2. *there exists* $\mathcal{F}'' \subset \mathcal{F}^*$ *such that* $\mathcal{R}_{\rightarrow} \cup \mathcal{F}'' \cup \mathcal{A}_2 \vdash f_2$ *and* $\mathcal{R}_{\rightarrow} \cup \mathcal{F}'' \nvdash f_2$*, and* $\mathcal{R}_{\rightarrow} \cup \mathcal{F}'' \cup \mathcal{A}_1 \nvdash f_1$

Example 2.29 (Generalized Specificity). *Consider the knowledge base of Example 2.28, the argument* $arg_2 = \langle \{chicken(tina) \wedge scared(tina) \Rightarrow fly(tina)\}$*, is more precise than* $arg_3 = \langle \{chicken(tina) \Rightarrow \neg fly(tina)\}$*, because it relies on more literals* $chicken(tina)$ *plus* $scared(tina)$*. Therefore* $arg_2 \succ arg_3$

However, arg_3 *is more concise than* $arg_1 = \langle \{bird(tina) \Rightarrow fly(tina)\}, fly(tina)\rangle$ *because* arg_1 *relies on the rules* r_1 *and* r_4 *whereas* arg_3 *relies on one rule* r_5*, therefore* $arg_3 \succ arg_1$*.*

DeLP also defines a preference relation based on explicit preference between defeasible rules. Basically, when there is an explicit preference between rules, an argument is preferred if it has a superior rule to another argument, if both argument have a superior rule to the other, then no argument is preferred.

Definition 2.40 (Explicit Preference [García and Simari, 2004]). *An argument* $arg_1 = \langle \mathcal{A}_1, f_1 \rangle$ *is preferred to* $arg_2 = \langle \mathcal{A}_2, f_2 \rangle$ *if:*

- *there exists a rule* $r_1 \in \mathcal{A}_1$ *and a rule* $r_2 \in \mathcal{A}_2$ *such that* r_1 *is superior to* r_2*, and*

- *there is no rules* $r_2' \in \mathcal{A}_2$ *and* $r_1' \in \mathcal{A}_1$ *such that* r_2' *is superior to* r_1'*.*

Given two arguments arg_1 and arg_2 and a sub-argument arg_2' of arg_2 such that arg_1 is a counter-argument to arg_2', if $arg_1 \succ arg_2'$ then arg_1 is said to be a "*proper defeater*" of arg_2. On the other hand if there is no preference between arg_1 and arg_2' (i.e. $arg_1 \not\succ arg_2'$ and $arg_2' \not\succ arg_1$) then arg_1 is said to be a "*blocking defeater*" of arg_2.

2.2.4.2 Warrant Procedure

To establish whether an argument is undefeated (it has no "warranted" defeater), all its defeaters have to be considered along with its "defenders" and so on (an argument arg_1 defends another argument arg_2 if arg_1 defeats a defeater of arg_2). This creates a sequence of argument called "*argumentation line*" where each element of the sequence defeats its predecessor. In order to avoid logical fallacies, an argumentation line has to follow some conditions:

1. An argument $arg_1 = \langle \mathcal{A}_1, f_1 \rangle$ cannot defend another argument $arg_2 = \langle \mathcal{A}_2, f_2 \rangle$ while $\mathcal{A}_1$ and $\mathcal{A}_2$ would lead to conflicting literal when strict rules and facts are added (defending arguments have to be consistent with each other).

2. A sub-argument cannot be reintroduced if it was defeated earlier in the line, this is a form of unwanted circular reasoning [García and Simari, 2004].

3. A blocking defeater can only be defeated by a proper defeater.

Definition 2.41 (Argumentation Line [García and Simari, 2004]). *Given a knowledge base $\mathcal{KB}$ and an argument $arg_1 = \langle \mathcal{A}_1, f_1 \rangle$ obtained from $\mathcal{KB}$. An argumentation line for arg_1 is a finite sequence of arguments from $\mathcal{KB}$ denoted $\Lambda = \langle arg_1, arg_2, ..., arg_n \rangle$ where each argument arg_i ($i > 1$) of the sequence is a defeater of its predecessor arg_{i-1} and:*

1. *$\forall arg_i = \langle \mathcal{A}_i, f_i \rangle$ such that i is an odd number (defending arguments of arg_1), no conflicting literals can be derived from $\mathcal{R}_{\rightarrow} \cup \mathcal{F}_{\rightarrow} \cup \bigcup_{i=1}^{n} \mathcal{A}_i$. The same applies for even i.*

2. *No argument arg_k in Λ is a sub-argument of an argument arg_i appearing earlier in Λ (i.e. $i < k$).*

3. *$\forall i$ such that arg_i is a blocking defeater for arg_{i-1}, if arg_{i+1} exists, then arg_{i+1} is a proper defeater for arg_i.*

Example 2.30 (Argumentation Line [García and Simari, 2004]). *Consider the following knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ stating that generally tigers are dangerous and baby animals and pets are not. Suppose we have a baby pet tiger, is it not dangerous $Q = \neg dangerous$?*

- $\mathcal{F} = \{\top \rightarrow tiger, \top \rightarrow baby, \top \rightarrow pet\}$
- $\mathcal{R} = \{r_1 : tiger \Rightarrow dangerous, r_2 : baby \Rightarrow \neg dangerous, r_3 : pet \Rightarrow \neg dangerous\}$
- $\succ$ *is generalized specificity.*

The arguments for Q are $arg_1 = \langle \{baby \Rightarrow \neg dangerous\}, \neg dangerous \rangle$ and $arg_2 = \langle \{pet \Rightarrow \neg dangerous\}, \neg dangerous \rangle$ and their counter-argument is $arg_3 = \langle \{tiger \Rightarrow dangerous\}, dangerous \rangle$. The sequence $\langle arg_1, arg_3, arg_2 \rangle$ is not a valid argumentation line for arg_1 because arg_3 is a blocking defeater and it is being defeated by another blocking defeater. Therefore the argumentation line for arg_1 is $\Lambda = \langle arg_1, arg_3 \rangle$.

An argument can have different defeater and consequently different argumentation lines, putting them together creates a *Dialectical Tree*.

Definition 2.42 (Dialectical Trees [García and Simari, 2004]). *Given an argument arg_1 obtained from a $\mathcal{KB}$. A dialectical tree for arg_1 (denoted $\mathcal{T}_{arg_1}$) is defined as follows:*

1. *arg_1 is the root node of the tree.*

2. *Given a non root node arg_n of the tree and $\langle arg_1, \ldots, arg_n \rangle$ the sequence of nodes from the root to arg_n, if there is a defeater arg' of arg_n such that $\Lambda = \langle arg_1, \ldots, arg_n, arg' \rangle$ is an argumentation line, then arg' is a child node of arg_n otherwise arg_n is a leaf node.*

In a dialectical tree every node (except the root) represents a defeater (proper or blocking) of its parent, and leaves correspond to undefeated arguments. Each path from the root to a leaf corresponds to a different argumentation line. In order to decide whether the root of a dialectical tree is defeated (not warranted), a labeling process is used: nodes are recursively labeled "*Defeated*" or "*Undefeated*".

Definition 2.43 (Dialectical Tree Marking [García and Simari, 2004]). *Given a Dialectical Tree $\mathcal{T}_{arg_1}$ for an argument arg_1:*

1. *All leaves in $\mathcal{T}_{arg_1}$ are marked as "Undefeated".*
2. *A non-leaf node arg_n is marked as "Undefeated" iff every one of its children are marked "Defeated". It is marked "Defeated" iff at least one of its children is marked "Defeated".*

This marking procedure is a bottom-up process, through which the label (marking) of the root argument is determined. We denote the label of the root argument in $\mathcal{T}_{arg_1}$ by $Label(\mathcal{T}_{arg_1})$.

Example 2.31 (Dialectical Trees). *Consider the $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ of the legal reasoning Example 2.23 and the queries $Q_1 = guilty$ and $Q_2 = \neg guilty$:*

- $\mathcal{F} = \{\top \Rightarrow e1,\ \top \Rightarrow e2,\ \top \Rightarrow alibi\}$
- $\mathcal{R} = \{r_1 : e1 \rightarrow responsible,\ r_2 : e2 \rightarrow \neg responsible,$
  $r_3 : responsible \rightarrow guilty,\ r_4 : alibi \rightarrow \neg guilty\}$
- $\succ$ *is generalized specificity.*

The argument for Q_1 is $arg_1 = \langle \{\top \Rightarrow e1\}, guilty \rangle$ and its counter-arguments are $arg_2 = \langle \{\top \Rightarrow e1\}, \neg responsible \rangle$ and $arg_3 = \langle \{\top \Rightarrow alibi\}, \neg guilty \rangle$. Given the preference relation, no argument is superior i.e. $arg_1 \not\succ arg_2$, $arg_2 \not\succ arg_1$, $arg_1 \not\succ arg_3$ and $arg_3 \not\succ arg_1$. The Dialectical Trees $\mathcal{T}_{arg_1}$ for arg_1 and $\mathcal{T}_{arg_3}$ for arg_3 are shown in Figure 2.11 and 2.12 respectively.

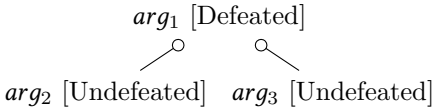

Figure 2.11: Dialectical Tree for arg_1

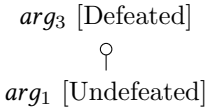

Figure 2.12: Dialectical Tree for arg_3

As we will show next, since neither Q_1 nor Q_2 have an argument that is labeled undefeated by its dialectical tree, the answer to Q_1 and Q_2 is UNDECIDED.

Dialectical Trees are used to determine if a literal is warranted (justified) or not. A literal f is *warranted* if and only if there is an argument arg for f such that $label(\mathcal{T}_{arg})$ is *Undefeated*, otherwise it is not warranted. Answers to queries can then be defined based on the warrant procedure.

Definition 2.44 (Query Answering in Dialectical Trees [García and Simari, 2004]). *Given a knowledge base $\mathcal{KB}$ expressed in $\mathcal{L}_p$, or $\mathcal{L}_\forall$ without defeater rules. A ground query $Q = f$ has four possible answers:*

- *Q is answered YES iff f is warranted.*
- *Q is answered NO iff $\overline{f}$ is warranted.*
- *Q is answered UNDECIDED iff neither f nor $\overline{f}$ are warranted.*
- *Q is answered UNKNOWN iff f does not appear in $\mathcal{KB}$.*

Notation 2.7 (Dialectical Trees Entailment). *For simplicity, we denote then answers to queries in Dialectical Trees using the entailment notation. $\models_{DT}$ denotes entailment in Dialectical Trees.*

- *$\mathcal{KB} \models_{DT} f$ iff $Q = f$ is answered YES.*
- *$\mathcal{KB} \not\models_{DT} f$ iff $Q = f$ is answered NO, UNDECIDED, or UNKNOWN.*

Dialectical Trees can be extended to the first order language $\mathcal{L}_\forall$ without defeater rules using a grounding phase. In recent work, the notion of Dialectical Trees has been applied to the first order defeasible language with existential rules $\mathcal{L}_{\forall\exists}$ (without defeater rules) [Martinez et al., 2014, Deagustini et al., 2015]. The definitions of DeLP for $\mathcal{L}_p$ are extended to existential rules using a Skolem chase, the nulls are considered as skolem terms and the generated atoms are considered ground. All Dialectical Trees notions can therefore directly be applied to $\mathcal{L}_{\forall\exists}$ [Martinez et al., 2014]. However, in the following Chapter 3, we will show that this direct application of Dialectical Trees to $\mathcal{L}_{\forall\exists}$ can be unsound as reasoning with existential rules might induce a possible loss of derivations due to the derivation reducer (as we will show in Chapter 3).

Dialectical Trees provide a link between argumentation and logic programming. Argumentation has always been seen as a natural application for non-monotonic reasoning [Governatori et al., 2004], especially with *argumentation semantics* proposed by Dung in his seminal paper on abstract argumentation [Dung, 1995].

2.2.5 Argumentation Semantics

Argumentation has long been used to study Defeasible Reasoning [Chesñevar et al., 2000]. Abstract argumentation frameworks [Dung, 1995] have been

developed to support the characterization of non-monotonic reasoning in argumentation-theoretic terms, this characterization is abstract because it is independent of what an *argument* means from a logical point of view. The basic element of these frameworks is the notion of "*acceptability*" of an argument. Briefly, an argument is acceptable if we can show that it is not possible to rebut it with other arguments. Several well-known non-monotonic reasoning systems can be seen as concrete instances of abstract argumentation frameworks [Governatori et al., 2004].

2.2.5.1 Abstract Argumentation Semantics

An abstract argumentation framework as defined by [Dung, 1995] takes as input a set of arguments and a pre-constructed binary relation that represents attacks between arguments.

Definition 2.45 (Argumentation Framework [Dung, 1995]). *An argumentation framework is a pair $\mathfrak{F} = (\mathscr{A}, \mathscr{R})$ where $\mathscr{A}$ is a set of arguments and $\mathscr{R}$ is a binary relation over $\mathscr{A}$. Given two arguments $a, b \in \mathscr{A}$, we say that a attacks b if $(a, b) \in \mathscr{R}$.*

An argumentation framework can be seen as a directed graph where vertices represent arguments and edges represent attack between argument.

Example 2.32 (Argumentation Framework). *Suppose we have three arguments a, b, and c such that a and b attack each other (i.e. $(a, b), (b, a) \in \mathscr{R}$), and c attacks b (i.e. $(c, b) \in \mathscr{R}$). This argumentation framework is shown in Figure 2.13.*

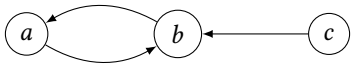

Figure 2.13: Argumentation Framework of Example 2.32

Definition 2.46 (Set Attack and Defense). *A set of argument S attacks an argument b if there exists an argument $c \in S$ such that $(c, b) \in \mathscr{R}$. If there is an argument $a \in S$ such that $(b, a) \in \mathscr{R}$ and S attacks b then S defends a.*

Example 2.33 (Cont'd Example 2.32). *The set $\{c\}$ attacks b and defends a.*

Argumentation is based on the notion of *acceptability of an argument* in the sense that a rational agent accepts only arguments which she can defend from all possible attacks.

Definition 2.47 (Acceptability of an Argument). *Given an argumentation framework $\mathfrak{F} = (\mathscr{A}, \mathscr{R})$. An arguments $a \in \mathscr{A}$ is acceptable with respect to a set of argument $S \subseteq \mathscr{A}$ if and only if S defends a from all its attacks, that is $\forall b \in \mathscr{A}$ such that $(b, a) \in \mathscr{R}$, $\exists c \in S$ such that $(c, b) \in \mathscr{R}$.*

Example 2.34 (Cont'd Example 2.32). *a is acceptable w.r.t. $\{c\}$.*

Acceptability of argument is used to define argumentation semantics. Two different methods are proposed to define semantics: *extension-based* [Dung, 1995] and *labeling-based* [Caminada, 2006]. The latter labels arguments with **in**, **out**, and **undec** to represent that an argument is accepted, rejected and undecided respectively. We start by the extension-based approach which defines what an acceptable argument means under some specific semantics. Examples of these semantics, the admissible, complete, grounded, preferred and stable [Dung, 1995]. Other semantics such as prudent, recursive, semi-stable and ideal (among others) can be found in [Baroni and Giacomin, 2009]. We limit the scope of the thesis to grounded semantics as it can be shown to coincide with Defeasible Logics.

Extension-based semantics are based on the principle of conflict-freeness which translates the idea the arguments in an extension should be able to "stand together", that is the arguments of the same extension do not attack each other.

Definition 2.48 (Conflict-freeness). *Given an argumentation framework $\mathfrak{F} = (\mathscr{A}, \mathscr{R})$. A set of arguments $S \subseteq \mathscr{A}$ is conflict-free if and only if there are no $a, b \in S$ such that $(a, b) \in \mathscr{R}$).*

Example 2.35 (Cont'd Example 2.32). *$\{a, c\}$ is conflict-free.*

A set of non-conflicting arguments can be seen as an agent's position in a debate, for this position to hold she has to defend all its argument. This corresponds to the notion of admissibility [Dung, 1995].

Definition 2.49 (Admissibility of a Set). *Given an argumentation framework $\mathfrak{F} = (\mathscr{A}, \mathscr{R})$. A conflict-free set of arguments $S \subseteq \mathscr{A}$ is admissible if and only if every argument $a \in S$ is acceptable with respect to S.*

An admissible set of arguments is a set of non-conflicting arguments that defends all its elements, such set is called an *admissible extension*. Every argumentation framework has at least one admissible set: the empty set.

Example 2.36 (Cont'd Example 2.32). *The admissible extensions are: $\emptyset$, $\{a\}$, $\{c\}$, and $\{a, c\}$. Note that $\{b\}$ is not an admissible set since it does not defend itself from c.*

Grounded semantics is the most skeptical of argumentation semantics, it is defined based on the notion of complete extension. That is the admissible extension that includes all the arguments it can defend from all attacks.

Definition 2.50 (Complete Semantics). *Given an argumentation framework $\mathfrak{F} = (\mathscr{A}, \mathscr{R})$. An admissible set of arguments $S \subseteq \mathscr{A}$ is a complete extension if and only if $\forall a \in \mathscr{A}$ if S defends a then $a \in S$.*

Definition 2.51 (Grounded Semantics). *The grounded extension of an argumentation framework is the least (w.r.t. set-inclusion) complete extension.*

Example 2.37. *Consider the following argumentation framework*

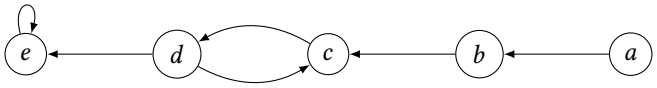

Figure 2.14: Argumentation Framework of Example 2.32

The admissible extensions are $\{d\}$, $\{a\}$, $\{a, d\}$, $\{a, c\}$. The complete extensions are $\{a\}$, $\{a, c\}$, and $\{a, d\}$. The least complete extension is $\{a\}$ which is the ground extension.

Grounded semantics (along with all Dung semantics) can also be obtained using labeling-based semantics of [Caminada, 2006]. A labeling is function that maps an argument to a set of labels $\{in, out, undec\}$. A reinstatement labeling is a function that maps an argument to "in" if and only if all its defeaters are labeled "out", an argument to "out" if it has a defeater that is "in", and an argument to "undec" if it is neither "in" nor "out".

Definition 2.52 (Reinstatement Labeling [Caminada, 2006]). *Given an argumentation framework $\mathfrak{F} = (\mathscr{A}, \mathscr{R})$ and a (total) function $L : \mathscr{A} \rightarrow \{in, out, undec\}$. L is a reinstatement labeling if and only if it satisfies the following:*

- $\forall a \in \mathscr{A} : L(a) = out$ *iff* $\exists b \in \mathscr{A}$ *s.t.* $(b, a) \in \mathscr{R}$ *and* $L(b) = in$,
- *and* $\forall a \in \mathscr{A} : L(a) = in$ *iff* $\forall b \in \mathscr{A}$ *s.t.* $(b, a) \in \mathscr{R}$, $L(b) = out$.

An argumentation framework can have different reinstatement labellings. These labeling can be seen as an extension: "in" arguments are elements of the extension, "out" arguments are attacked by the extension, and "undec" arguments are not part of the extension and not attacked by it. All possible reinstatement labellings correspond to complete extensions and the reinstatement labeling with the most arguments labeled "undec" coincides with the grounded extension [Caminada, 2006].

Example 2.38 (Grounded Labeling). *The argumentation framework in Example 2.37 can have different reinstatement labellings. Figure 2.15 corresponds to the complete extension $\{a, c\}$ and Figure 2.16 corresponds to the grounded extension $\{a\}$.*

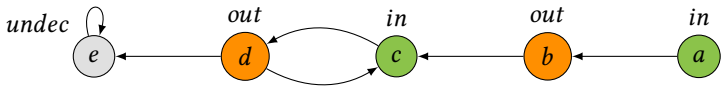

Figure 2.15: Labeling that corresponds to $\{a, c\}$ of Example 2.32

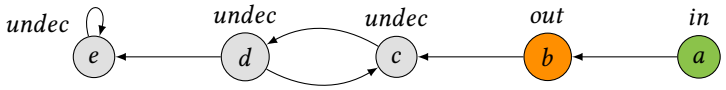

Figure 2.16: Labeling that corresponds to $\{a\}$ of Example 2.32

In order to relate these argumentation semantics to knowledge representation, abstract argumentation frameworks need to be instantiated using a logical language. Different approaches have been used such as [Besnard and Hunter, 2001, Bondarenko et al., 1993, Wyner et al., 2013, Modgil and Prakken, 2014], each instantiation has a specific objective in mind and postulates that it adheres to. Since we are interested in a specific set of intuitions and languages, we will not delve deeply into each approach. The reader is referred to [Rahwan and Simari, 2009] for more details about different methods of instantiating argumentation frameworks.

2.2.5.2 Logic-based Argumentation

Instantiating an argumentation framework from a knowledge base consists in the following four steps [Caminada and Amgoud, 2007]:

1. Constructing arguments (in favor or against a literal).
2. Determining the different conflicts among arguments.
3. Evaluating the acceptability of the arguments.
4. Defining the justified conclusions.

Most instantiations of argumentation frameworks do not consider defeater rules. In order to fully represent the language $\mathcal{L}_p$ with defeater rules we consider the instantiation in [Governatori et al., 2004]. An argument in this instantiation is a directly consistent derivation for a literal where each step of rule application is itself an argument. An argument arg_1 attacks another argument arg_2 if arg_2 is not a strict argument and the conclusion of arg_1 is in conflict with the conclusion or a premise in arg_2 that is not part of a strict sub-argument. An argument arg_1 defeats another argument arg_2 if and only if arg_1 attacks arg_2 and arg_2 is not preferred to arg_1 ($arg_2 \not\succ arg_1$) [Lam, 2012, Lam et al., 2016]. Handling preferences has always been tricky, the first problem is to elevate preferences on rules to preferences on arguments, the second one is that in most cases removing attacks based on preferences will lead to counter-intuitive results [Amgoud and Vesic, 2011]. The

instantiation in [Governatori et al., 2004] replaces preferences on rules by new rules based on rule labels [Antoniou et al., 2001], for example the rules $r_1 : bird \Rightarrow fly$ and $r_2 : brokenWings \Rightarrow \neg fly$ with $r_2 \succ r_1$ are transformed to:

$$bird \Rightarrow \neg inf(r) \qquad \neg inf(r_2) \Rightarrow \neg fly$$

$$\neg inf(r) \Rightarrow fly$$

$$brokenWing \Rightarrow \neg inf(r_1) \qquad \neg inf(r_2) \Rightarrow inf(r_1)$$

This transformation has been shown that it might lead to non-intuitive results [García and Simari, 2004]. However, it allows to obtain equivalent results with Defeasible Logics [Governatori et al., 2004, Lam et al., 2016].

Under these definitions, an argumentation framework can be built based on the defeat relation between arguments. Grounded semantics is then applied to identify justified arguments [Governatori et al., 2004]. A conclusion f is entailed using grounded semantics if and only if there is an argument for f that is an element of the grounded extension [Governatori et al., 2004].

Proposition 2.2 (Grounded Semantics entailment [Governatori et al., 2004]). *Given a knowledge base $\mathcal{KB}$ expressed in $\mathcal{L}_p$ and a literal f. We denote entailment under grounded semantics by $\vdash_{GS}$.*

- *$\mathcal{KB} \vdash_{GS} f$ iff there is an argument for f that is an element of the grounded extension built on $\mathcal{KB}$.*
- *$\mathcal{KB} \nvdash_{GS} f$ iff there is no argument for f that is an element of the grounded extension built on $\mathcal{KB}$.*

Another instantiation of abstract argumentation frameworks is the one defined in [Modgil and Prakken, 2014, Amgoud et al., 2004] where arguments are defined using $\mathcal{L}_p$ without defeater rules. If no preference relation is considered (with the condition that defeasible arguments cannot attack strict arguments), the grounded semantics obtained in this definition and constructed using a knowledge base expressed in $\mathcal{L}_p$ without defeater rules is equivalent to the one obtained by the instantiation of [Governatori et al., 2004] as shown in [Lam et al., 2016].

There are many ways to instantiate an argumentation framework, some define argument as directly consistent derivation for a literal [Amgoud et al., 2004], other define them as indirectly consistent derivation for a conjunction of literals [Croitoru and Vesic, 2013], some remove attacks from an inferior argument to a superior one [Modgil and Prakken, 2014], others apply preferences after constructing extensions [Amgoud and Vesic, 2011]. Considering all these instantiations is beyond the scope of this thesis given our primary objective of obtaining Defeasible Reasoning for existential rules. However,

the work presented in the following chapters can be extended to various instantiations of argumentation framework as discussed in Chapter 6.

2.2.6 Comparing Defeasible Reasoning Techniques

The discussed Defeasible Reasoning techniques can be compared, in fact, under certain conditions, they can be shown to yield equivalent results. Grounded Semantics under the instantiation of [Governatori et al., 2004] or ASPIC+ [Modgil and Prakken, 2014] without implicit preferences give equivalent results to Defeasible Logic with ambiguity propagation without team defeat [Lam et al., 2016, Governatori et al., 2004].

Proposition 2.3 (Grounded Semantics and Defeasible Logic [Governatori et al., 2004]). *Given a knowledge base $\mathcal{KB}$ expressed in $\mathcal{L}_p$ or $\mathcal{L}_\forall$ and a literal f:*

- $\mathcal{KB} \vdash +\delta^{noTD}_{prop} f$ *iff* $\mathcal{KB} \models_{GS} f$
- $\mathcal{KB} \vdash -\delta^{noTD}_{prop} f$ *iff* $\mathcal{KB} \nvdash_{GS} f$

It can be shown that Defeasible Logic Programming (DeLP) with its Dialectical Trees *follows the intuition of ambiguity propagating when no preference is used.* We recall that a literal f is ambiguous if there is an argument for f that is neither inferior to any argument for $\overline{f}$ nor superior to an argument for $\overline{f}$. In Dialectical Trees terms this entails that an argument for f has a blocking defeater and no proper defeater. Given that no preference is used, all attacks become blocking defeats, meaning that the argument for f will be at the end of the argumentation line of the arguments it attacks and cannot be "blocked" (since a blocking defeater cannot be defeated with another blocking defeater), therefore no literal that is attacked can have a warranted argument and ambiguous literal are allowed to propagate their ambiguity to other literals as shown in Example 2.31. However this only holds when no preference relation is used. If a preference relation is used then DeLP can yield unintuitive results that do not correspond with ambiguity propagating as shown in the following Example 2.39.

Example 2.39. *Consider the following knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$:*

- $\mathcal{F} = \{\top \rightarrow a, \top \rightarrow b, \top \rightarrow c\}$
- $\mathcal{R} = \{r_1 : a \Rightarrow f,\ r_2 : b \Rightarrow f,\ a \wedge c \Rightarrow \neg f\}$
- $\succ$ *is generalized specificity.*

Suppose we want to evaluate the query $Q = f$. The arguments that can be constructed from this knowledge base are:

- $arg_1 = \langle \{a \Rightarrow f\}, f \rangle$

- $arg_2 = \langle \{b \Rightarrow f\}, f \rangle$
- $arg_3 = \langle \{a \wedge c \Rightarrow \neg f\}, \neg f \rangle$

Given generalized specificity, arg_3 is preferred to arg_1 because arg_3 is more precise (relies on more strict facts), but it is not preferred to arg_2 (they rely on different strict facts) i.e. $arg_3 \succ arg_1$, $arg_3 \not\succ arg_2$ and $arg_2 \not\succ arg_3$.

One can see that the query $Q = f$ should be "false" because the literal f is ambiguous (there is a derivation for f (arg_2) that is not inferior to any derivation attacking it and there is a derivation for $\overline{f}$ (arg_3) such that neither derivations is superior). However, having $arg_3 \succ arg_1$ makes arg_3 a proper defeater of arg_1, this allows us to use arg_2 as a blocking defeater to arg_3 making arg_1 undefeated as shown in Figure 2.17, therefore $\mathcal{KB} \models_{DT} f$.

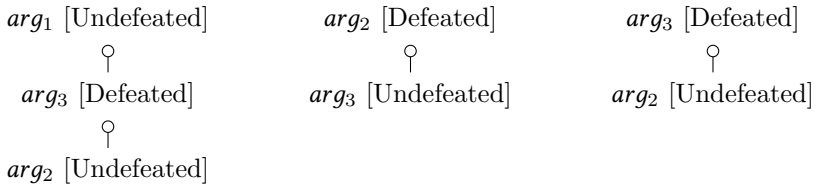

Figure 2.17: Dialectical Trees for arg_1, arg_2, and arg_3

The following proposition defines the link between Dialectical Trees and Defeasible Logic with ambiguity propagation.

Proposition 2.4 (Dialectical Trees and Defeasible Logic). *Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ expressed in $\mathcal{L}_p$ or $\mathcal{L}_\forall$ without defeater rules, strict rules nor a preference relation:*

- $\mathcal{KB} \vdash +\delta_{prop} f$ *iff* $\mathcal{KB} \models_{DT} f$.
- $\mathcal{KB} \vdash -\delta_{prop} f$ *iff* $\mathcal{KB} \not\models_{DT} f$.

Proof Sketch. No preference relation means that all defeats are blocking defeats, thus if an argument is attacked it becomes unwarranted, meaning that a literal can only have a warranted arguments iff it is not attacked, this means that its derivation does not rely on ambiguous literals which exactly corresponds to Defeasible Logic with ambiguity propagation (cf. detailed Proof 7.2.1 on page iv). □

It can also be shown that Dialectical Trees follow the intuition of team defeat as described in the following Example 2.40.

Example 2.40. *Suppose that there are two arguments arg_1, arg_2 for a literal f and two arguments arg_3, arg_4 for $\overline{f}$ such that $arg_1 \succ arg_3$ and $arg_2 \succ arg_4$. The Dialectical Trees for arg_1 and arg_2 are shown in Figures 2.18 and 2.19. Even if arg_4 is a blocking defeater of arg_1, arg_2 is a proper defeater of arg_3 which makes arg_1 undefeated, and the same applied for arg_2.*

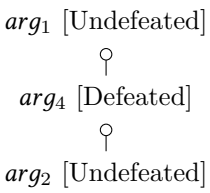

Figure 2.18: Dialectical Tree for arg_1

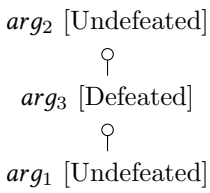

Figure 2.19: Dialectical Tree for arg_2

However, given Example 2.39 it can be shown that Dialectical Trees and Defeasible Logic with ambiguity propagating and team defeat do not coincide even under defeasible rules only. Table 2.2 describes the intuitions of Defeasible Reasoning techniques. Please note that the fact that two techniques follow the same intuition *does not imply* they yield the same results, this serves as an indicator on the set of intuitions a certain formalism adopts.

Feature		Defeasible Logics	Grounded Semantics	Dialectical Trees
Ambiguity	Prop.	✓	✓	✓*
	Block	✓	-	-
Team Defeat	TD	✓	-	✓
	noTD	✓	✓	-
Floating Conclusions	FC	-	-	-
	noFC	✓	✓	✓
Consistent Derivation	Direct	✓	✓	-
	Indirect	-	-	✓

Table 2.2: Intuitions of Defeasible Reasoning Techniques ($*$ in the absence of preferences)

Finally, all the discussed Defeasible Reasoning techniques do not respect the rationality postulate of indirect consistency [Caminada and Amgoud, 2007], in the sense that applying strict rules over the set of all accepted literals might lead to a conflict.

Example 2.41 (Consistent Answers). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$

- $\mathcal{F} = \{\top \Rightarrow a, \top \Rightarrow b, \top \Rightarrow c, \top \rightarrow d\}$
- $\mathcal{R} = \{a \wedge b \wedge c \rightarrow \neg d\}$

- *in Defeasible Logics (and Grounded Semantics), the set of accepted literals is* $\{a, b, c, d\}$ *since* $+\delta a$, $+\delta b$, $+\delta c$, *and* $+\Delta d$. *However, this set of accepted literals is not consistent w.r.t. strict rules as* d *and* $\neg d$ *can be derived.*

- *In DeLP, the argument* $\langle\{\top \Rightarrow a\}, a\rangle$ *has no attacker (since there is no fact* f *such that* $\{f, a\} \cup \mathcal{F}_{\rightarrow} \cup \mathcal{R}_{\rightarrow}$ *is inconsistent), therefore* a *is warranted (same for* b*,* c*, and* d*). The set of all warranted literals* $\{a, b, c, d\}$ *is not consistent w.r.t. strict rules.*

Having a consistent set of answers is a desirable feature, even a critical one for some [Caminada and Amgoud, 2007]. However, the restriction of the logical languages (no disjunction) makes satisfying rationality postulate hard [Caminada and Amgoud, 2007]. Given our initial aim of applying Defeasible Reasoning techniques to the already problematic existential rules, we will not consider rationality postulates in this thesis.

2.2.7 Defeasible Reasoning Tools

A number of Defeasible Reasoning tools and systems have been proposed in recent years to cover different techniques as well as other intuitions of non-monotonic reasoning:

- **ASPIC** [Prakken, 2010] is a framework for specifying systems in structured argumentation. It is used for Defeasible Reasoning thanks to its grounded semantics. An official *prototype* JAVA implementation available online[3] (denoted here by *ASPIC**) uses a Prolog-like syntax to express the first order language $\mathcal{L}_{\forall}$ without defeater rules and allows for preference over rules using decimal numbers between 0 and 1. It implements grounded semantics instantiated with the ASPIC theoretical framework and provides explanation for query entailment. Other implementations of ASPIC either use a propositional language or are not publicly available.

- **DeLP** [García and Simari, 2004] (Defeasible Logic Programming) is a formalism that combines results of Logic Programming and Argumentation to obtain Defeasible Reasoning thanks to the Dialectical Trees warranting procedure. A *prototype* JAVA implementation of DeLP provided in Tweety1.7 libraries [Thimm, 2014] (that we denote by *DeLP**) uses the first order language $\mathcal{L}_{\forall}$ without defeater rules and defines a set of preference relations including generalized specificity. An online tool called DeLPclient[4] is also available and provides a dialectical explanation of query entailment.

- **Flora-2** [Yang et al., 2003, Wan et al., 2015] is a rule-based knowledge base system designed for a variety of automated tasks on the Semantic Web, ranging from meta-data management to intelligent agents. It is based on Defeasible Logics and uses a prolog-like syntax to express

[3] `http://aspic.cossac.org`
[4] `http://lidia.cs.uns.edu.ar/delp_client`

the first order language $\mathcal{L}_\forall$. It allows for explicit preferences on rules. Flora-2 has a commercial version called Ergo[5] with additional functionalities including explanation of query entailments and the ability to use relational databases and OWL ontologies to represent defeasible knowledge base.

- **SPINdle** [Lam, 2012] is a reasoning tool based on Defeasible Logics that allows for ambiguity blocking or propagating with team defeat for the first order language $\mathcal{L}_\forall$. It is a bottom-up approach which is capable to perform efficient and scalable reasoning with defeasible knowledge bases. It is implemented with performance in mind and has been used for most applications of Defeasible Logics. We use its official implementation[6].

To the best of our knowledge, these are the only still functioning, publicly available tools for first order Defeasible Reasoning as of the time of writing of this thesis. Other implementations such as DR-Device and DR-Prolog [Bikakis and Antoniou, 2005] are no longer maintained. The interested reader is referred to [Bryant and Krause, 2008] for a discussion on the evolution of Defeasible Reasoning tools.

2.3 Summary

In this chapter we presented the existential rule logical fragment along with the *frontier chase* forward chaining inference mechanism. We showed that allowing the existential quantifier might lead to infinite rule applications, that is why a derivation reducer is needed to *remove redundant rule applications*. We presented the *frontier derivation reducer* and showed the types of rules (*Skolem-FES*) for which it is guaranteed to stop. Then we defined the different types of conflicts, namely, *inconsistence* when a negative constraint is applicable, and *incoherence* when the set of rules is unsatisfiable.

Afterwards, we presented *Defeasible Reasoning* which is a conflict-tolerant non-monotonic form of reasoning and discussed the different intuitions a formalism can adopt, namely, ambiguity blocking or propagating, team defeat, floating conclusions, and how strict rules are handled (directly or indirectly consistent derivation). We presented various Defeasible Reasoning techniques that we will consider throughout this thesis and showed when they might coincide:

1. *Defeasible Logics* is a top-down approach (arguments are evaluated on construction), it has different variants for ambiguity blocking or propagation with or without team defeat along with failure-by-looping to remove cycles.

[5] `http://coherentknowledge.com/`
[6] `http://spindle.data61.csiro.au/spindle`

2. *Grounded Semantics* is a bottom-up argumentation approach (arguments are evaluated after construction) that can be instantiated in various ways. We presented the instantiation of [Governatori et al., 2004] that coincides with Defeasible Logic ambiguity propagation without team defeat (Proposition 2.3).

3. *Dialectical Trees* is a bottom-up approach, it relies on indirectly consistent derivations. We showed that it coincides with Defeasible Logic ambiguity propagation under the restriction of no defeater rules or preferences (Proposition 2.4), and follows the intuition of team defeat (Example 2.40).

We discussed that Dialectical Trees are the only Defeasible Reasoning technique that has been applied to the existential rule language $\mathcal{L}_{\forall\exists}$ without defeater rules. However, as we will see in the next chapter, directly applying Defeasible Reasoning techniques to existential rules is not as straightforward as it might seem. Finally, we presented the different first order Defeasible Reasoning tools that are still functioning and publicly available as of the time of writing of this thesis.

Chapter 2 in a Nutshell

- *Reasoning with existential rules requires a derivation reducer to become decidable. Frontier/Skolem chase is the most used forward chaining inference mechanism and has decidable classes (types) of rules called Skolem-FES.*
- *There are two types of conflicts: inconsistency when a negative constraint is applicable, and incoherence when the set of rules is unsatisfiable.*
- *There is no universal way of reasoning in presence of conflict. However, a set of intuitions can be adopted: ambiguity blocking or propagating, team defeat, floating conclusions, and directly or indirectly consistent derivations (among others).*
- *Defeasible reasoning is a conflict-tolerant reasoning approach, some of its techniques are: Defeasible Logics, Dialectical Trees, and Grounded Argumentation Semantics. These techniques have different semantics and coincide under certain conditions.*
- *The discussed Defeasible Reasoning techniques have been defined for the propositional language $\mathcal{L}_p$ (and by grounding can be applied to the first order language without existential quantifier $\mathcal{L}_\forall$), not all of them allow for defeater rules.*
- *Aside from Dialectical Trees, there is no Defeasible Reasoning technique for the defeasible existential rules language $\mathcal{L}_{\forall\exists}$.*

3

Applying Defeasible Reasoning to Existential Rules

Intuitively, the ideas behind defeasible reasoning techniques such as defeasible logics, dialectical trees, and argumentation semantics are general enough to be directly applied to the existential rules framework as theoretically demonstrated by [Martinez et al., 2014] using dialectical trees for Datalog$^{\pm}$. One might be tempted to assume that transitioning from the propositional language of these techniques to the existential rules logical fragment would be straightforward. However, the particularities of existential rule make this transition not as clear-cut.

In this chapter, we explain the problem of "Derivation Loss" when reasoning with existential rules, we provide a combinatorial structure to prevent this problem and present the first tool for defeasible reasoning with existential rules using Dialectical Trees. Finally we provide the first benchmark for the classification of first order logic defeasible reasoning tools based on the *intuitions* discussed in Section 2.2.2 (ambiguity handling, etc), *expressiveness* (logical language, preferences, etc.), and basic *performance*.

Research Questions in this Chapter

- *Can defeasible reasoning techniques (for example Dialectical Trees) be directly applied to the existential rule language $\mathcal{L}_{\forall\exists}$? If not, what is needed to apply these techniques to $\mathcal{L}_{\forall\exists}$?*
- *Given the variety of tools and formalisms for defeasible reasoning, how can we help a Data Engineer chose the best defeasible reasoning tool depending on the data and the requirements at hand?*

3.1 Derivation Loss Problem

Most of the defeasible reasoning techniques are based on the notion of *derivation for a fact*. Therefore, it is essential to have a mechanism for extracting these derivations. Existing work of derivation extraction for existential rules focuses on obtaining one derivation for a specific fact as it is enough for classical query entailment. However, for certain practical applications such as explanation [Frawley et al., 1992], abduction [Kakas et al., 1998], debugging [Caballero et al., 2008], as well as for our purposes in this thesis (as show in Section 3.1.1) we need to extract all possible derivations (a.k.a. provenance paths) for that fact. Without a sound and complete derivation extraction mechanism, defeasible reasoning with dialectal trees for existential rules [Martinez et al., 2014] becomes unsound. This unexpected behavior is due to *the order in which rules are applied* and to the *type of chase* used as demonstrated in Section 3.1.2.

3.1.1 Derivation Loss and the Frontier Chase

The most common chase is the frontier/skolem chase, however, there are other chases that generate a universal model with more or less strong derivation reducers such as the Oblivious chase [Cali et al., 2013] and the Restricted chase [Fagin et al., 2005]. While these derivation reducers are crucial for the chase to stop, they might induce a loss of rule applications depending on the order in which the rules are applied. This order of rule applications (as long as it is breadth-first) does not impact the generation of a universal model (and thus does not affect entailment in Datalog$^{\pm}$), however, it might induce a derivation loss not picked up by existing work on derivation extraction which focuses on obtaining one derivation. Obtaining all derivations was implicitly assumed to be not a difficult task but a mere enumeration of the first. Unfortunately this is not the case as shown in the following example.

Example 3.1 (Animal Shelter Extended). *Let $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ be a knowledge base expressed in $\mathcal{L}_{\forall\exists}$ of an **animal shelter** describing the process of deciding if a found dog should be kept or put for adoption. Generally, an animal found alone is a stray and is put for adoption. If it has an owner then it is kept. If it has a collar or a microchip then it generally has an owner. An animal cannot be kept and put for adoption at the same time. A dog named "Jack" with a collar and a microchip is found alone, should it be kept ($Q_1 = keep(jack)$)? Or put for adoption ($Q_2 = adoption(jack)$)?*

- $\mathcal{F} = \{\top \rightarrow alone(jack),\ \top \rightarrow hasCollar(jack),\ \top \rightarrow hasMicrochip(jack)\}$
- $\mathcal{R} = \{r_1 : \forall X, Y\ hasOwner(X, Y) \rightarrow keep(X),$
  $r_2 : \forall X\ hasCollar(X) \Rightarrow \exists Y\ hasOwner(X, Y),$
  $r_3 : \forall X\ hasMicrochip(X) \Rightarrow \exists Y\ hasOwner(X, Y),$
  $r_4 : \forall X\ alone(X) \Rightarrow stray(X),$
  $r_5 : \forall X\ stray(X) \rightarrow adoption(X)\}$
- $\mathcal{N} = \{\forall X\ adoption(X) \wedge keep(X) \rightarrow \bot\}$

A possible frontier chase of $\mathcal{KB}$ is:

$$\begin{aligned}\sigma_{fr}\text{-}chase(\mathcal{F}, \mathcal{R}) = \langle &(\mathcal{F}, \emptyset, \emptyset), (\mathcal{F}_1 = \mathcal{F} \cup \{hasOwner(jack, Null_1)\}, r_2, \pi_1 = \{X \rightarrow jack\}),\\ &(\mathcal{F}_2 = \mathcal{F}_1 \cup \{hasOwner(jack, Null_2)\}, r_3, \pi_2 = \{X \rightarrow jack\}),\\ &(\mathcal{F}_3 = \mathcal{F}_2 \cup \{stray(jack)\}, r_4, \pi_3 = \{X \rightarrow jack\}),\\ &(\mathcal{F}_4 = \mathcal{F}_3 \cup \{adoption(jack)\}, r_5, \pi_4 = \{X \rightarrow jack\}),\\ &(\mathcal{F}_5 = \mathcal{F}_4 \cup \{keep(jack)\}, r_1, \pi_5 = \{X \rightarrow jack, Y \rightarrow Null_1\})\rangle.\end{aligned}$$

To extract derivations using forward chaining, the state of the art uses a chase graph [Calì et al., 2012] (also called derivation trees [Arora et al., 1993]). A chase graph is a directed graph representing the rule applications in a chase, the set of nodes represents the facts and there is an edge from a fact u to v if and only if v is obtained from u (possibly with other atoms) by a rule application in the chase. The chase graph of $\sigma_{fr}\text{-}chase(\mathcal{F}, \mathcal{R})$ is represented in Figure 3.1.

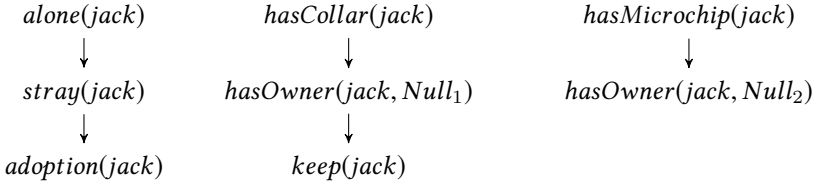

Figure 3.1: chase graph for $\sigma_{fr}\text{-}chase(\mathcal{F}, \mathcal{R})$ of Example 3.1

From the chase graph we can find that there is only one derivation δ_1 for $keep(jack)$ which is applying r_2 on $hasCollar(jack)$ then r_1 on the resulting

fact $hasOwner(jack, Null_1)$:

$$\delta_1 = \langle(\mathcal{F}_0 = \{hasCollar(jack)\}, \emptyset, \emptyset), (\mathcal{F}_1 = \mathcal{F}_0 \cup \{hasOwner(jack, Null_1)\}, r_2, \pi_1),$$
$$(\mathcal{F}_2 = \mathcal{F}_1 \cup \{keep(jack)\}, r_1, \pi_5)\rangle.$$

However, we can see that by applying r_3 on the atom $hasOwner(jack, Null_2)$ we also get $keep(jack)$, which gives us another derivation that does not show in the chase graph. This rule application $(\mathcal{F}_6 = \mathcal{F}_5 \cup \{keep(jack)\}, r_1, \pi_6 = \{X \rightarrow jack, Y \rightarrow Null_2\})$ is not part of the chase sequence because the frontier derivation reducer considers it redundant as it uses the same rule r_1 with $\pi_5|_{fr(r_1)} = \pi_6|_{fr(r_1)} = \{X \rightarrow jack\}$.

Moreover if r_1 is applied on $hasOwner(jack, Null_2)$ first, then its application on $hasOwner(jack, Null_1)$ would be redundant and therefore dismissed as shown in Figure 3.2.

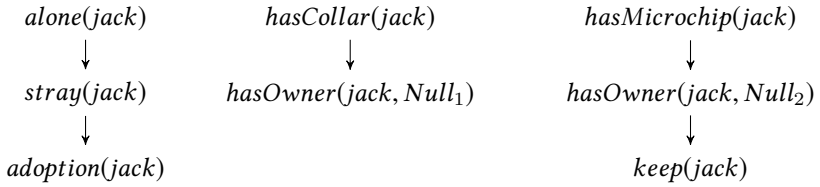

Figure 3.2: Chase graph for σ_{fr}-$chase(\mathcal{F}, \mathcal{R})$ if r_1 is applied on $hasOwner(jack, Null_2)$ first

In this case, there is also only one derivation δ_1' for $keep(jack)$:

$$\delta_1' = \langle(\mathcal{F}_0 = \{hasMicrochip(jack)\}, \emptyset, \emptyset),$$
$$(\mathcal{F}_1 = \mathcal{F}_0 \cup \{hasOwner(jack, Null_2)\}, r_3, \pi_2),$$
$$(\mathcal{F}_2 = \mathcal{F}_1 \cup \{keep(jack)\}, r_1, \pi_6)\rangle.$$

This loss of derivation has no effect on classical entailment, however it has critical consequences for defeasible reasoning techniques. To answer the queries $Q_1 = keep(jack)$ and $Q_2 = adoption(jack)$ we need to build the arguments for and against it. Given Figure 3.1 there is only one argument for $keep(jack)$ $(arg_1 = \langle\delta_1, keep(jack)\rangle)$ and one argument for $adoption(jack)$ $(arg_2 = \langle\delta_2, adoption(jack)\rangle)$ such that:

$$\delta_2 = \langle(\mathcal{F}_0 = \{alone(jack)\}, \emptyset, \emptyset),$$
$$(\mathcal{F}_1 = \mathcal{F}_0 \cup \{stray(jack)\}, r_4, \pi_3),$$
$$(\mathcal{F}_2 = \mathcal{F}_1 \cup \{adoption(jack)\}, r_5, \pi_4)\rangle.$$

Suppose the preference relation $\succ$ used in this $\mathcal{KB}$ considers that having a collar is a good reason to keep the animal and not put it for adoption (i.e. $arg_1 \succ arg_2$). Therefore arg_1 has no defeater and it defeats arg_2. The dialectical trees that evaluate arg_1 and arg_2 are shown in Figure 3.3 and 3.4 respectively.

$arg_1 = \langle \delta_1, keep(jack) \rangle$ [Undefeated]

Figure 3.3: Dialectical Tree for arg_1

$arg_2 = \langle \delta_2, adoption(jack) \rangle$ [Defeated]

$arg_1 = \langle \delta_1, keep(jack) \rangle$ [Undefeated]

Figure 3.4: Dialectical Tree for arg_2

From Figures 3.3 and 3.4 arg_1 is undefeated while arg_2 is defeated, therefore the answer to the query $Q_1 = keep(jack)$ is *true* (i.e. $\mathcal{KB} \models_{DT} Q_1$) and to the query $Q_2 = adoption(jack)$ is *false* (i.e. $\mathcal{KB} \not\models_{DT} Q_2$). However if we extract derivations from the chase graph in Figure 3.2 we get one argument for $keep(jack)$ ($arg'_1 = \langle \delta'_1, keep(jack) \rangle$) and one argument for $adoption(jack)$ ($arg_2 = \langle \delta_2, adoption(jack) \rangle$), and if we suppose that the preference relation $\succ$ considers having a microchip not a strong reason for having a current owner because people might abandon their animals without removing the microchip (i.e. $arg'_1 \not\succ arg_2$ and $arg_2 \not\succ arg'_1$), then arg'_1 is a blocking defeater for arg_2 and vise-versa. The dialectical trees that evaluate arg'_1 and arg_2 are shown in Figures 3.5 and 3.6 respectively.

$arg'_1 = \langle \delta'_1, keep(jack) \rangle$ [Defeated]

$arg_2 = \langle \delta_2, adoption(jack) \rangle$ [Undefeated]

Figure 3.5: Dialectical Tree for arg'_1

$arg_2 = \langle \delta_2, adoption(jack) \rangle$ [Defeated]

$arg'_1 = \langle \delta'_1, keep(jack) \rangle$ [Undefeated]

Figure 3.6: Dialectical Tree for arg_2

From Figures 3.5 and 3.6 arg'_1 and arg_2 are defeated, therefore the answer to the queries $Q_1 = keep(jack)$ and $Q_2 = adoption(jack)$ is *false* (i.e. $\mathcal{KB} \not\models_{DT} Q_1$ and $\mathcal{KB} \not\models_{DT} Q_2$). The loss of derivation affects query entailment in defeasible Datalog$^{\pm}$, in one case Q_1 is entailed and in the other it is not depending on the order in which rules are applied. This order is non-deterministic as we have no control over which rules are applied when two or more can be applied at the same breadth-first step.

The above example showcases a simple case of derivation loss, but depending on the chase, losing paths might become more intricate. The loss of derivations in forward chaining depends on two main factors: the *order of rule applications* and the *derivation reducer* of the used chase. Motivated by the above example, the first research question we answer in this chapter is the following: "*In existential rules, how does the chase choice affect derivation extraction and how can we prevent it?*".

3.1.2 Derivation Loss and the Different Kinds of Chase

As shown in the previous example, some derivations might not show in the chase graph built with a frontier chase.

Proposition 3.1 (Frontier Chase Graph Derivations Incompleteness). *Extracting derivations for a fact from a frontier chase using chase graph is incomplete.*

Proof. By contradiction (cf. Example 3.1) □

There are different kinds of chases defined in the literature (Oblivious, Skolem/Frontier, Restricted, and Core chases) each one with a "stronger" derivation reducer (by "stronger" we mean that they might remove rule applications in cases where others do not). A first possible solution to the derivation loss problem that might come to mind is to use a derivation reducer that is less strong than the frontier one (for example, the *Oblivious derivation reducer*).

The oblivious chase (also called naive chase [Cali et al., 2013]) removes rule applications that reuse the same rule with the exact same homomorphism. Contrary to the frontier chase, it compares the whole homomorphism and not only the frontier part.

Definition 3.1 (Oblivious Chase [Cali et al., 2013]). *The oblivious chase σ_{obl}-chase relies on the oblivious derivation reducer (denoted σ_{obl}) defined as follows: for any derivation δ, $\sigma_{obl}(D_0) = D_0$ and $\forall D_i = (\mathcal{F}_i, r_i, \pi_i) \in \delta$:*

$$Facts(\sigma_{obl}(D_i)) = \begin{cases} \mathcal{F}_{i-1} \cup \pi_i^{safe}(Head(r_i)) & \text{if } \forall j < i, \pi_j \neq \pi_j \text{ or } r_j \neq r_i \\ \mathcal{F}_{i-1} & \text{otherwise} \end{cases}$$

Essentially, the oblivious chase ensures that a rule r is applied according to a homomorphism π only if it has not already been applied according to that same homomorphism. A possible application of the oblivious chase on the animal shelter example is:

$$\begin{aligned} \sigma_{obl}\text{-}chase(\mathcal{F}, \mathcal{R}) = \langle &(\mathcal{F}, \emptyset, \emptyset), (\mathcal{F}_1 = \mathcal{F} \cup \{hasOwner(jack, Null_1)\}, r_2, \pi_1 = \{X \to jack\}), \\ &(\mathcal{F}_2 = \mathcal{F}_1 \cup \{hasOwner(jack, Null_2)\}, r_3, \pi_2 = \{X \to jack\}), \\ &(\mathcal{F}_3 = \mathcal{F}_2 \cup \{stray(jack)\}, r_4, \pi_3 = \{X \to jack\}), \\ &(\mathcal{F}_4 = \mathcal{F}_3 \cup \{adoption(jack)\}, r_5, \pi_4 = \{X \to jack\}), \\ &(\mathcal{F}_5 = \mathcal{F}_4 \cup \{keep(jack)\}, r_1, \pi_5 = \{X \to jack, Y \to Null_1\}), \\ &(\mathcal{F}_6 = \mathcal{F}_5 \cup \{keep(jack)\}, r_1, \pi_6 = \{X \to jack, Y \to Null_2\})\rangle \end{aligned}$$

From the chase graph of σ_{obl}-$chase(\mathcal{F}, \mathcal{R})$ in Figure 3.7, we can see that no derivation for $keep(jack)$ is lost as the chase considers that applying r_1 using π_5 and π_6 is not redundant since $\pi_5 \neq \pi_6$. Therefore there are two arguments for $keep(jack)$ ($arg_1 = \langle \delta_1, keep(jack) \rangle$ and $arg'_1 = \langle \delta'_1, keep(jack) \rangle$) and one argument for $adoption(jack)$ ($arg_2 = \langle \delta_2, adoption(jack) \rangle$).

To evaluate the queries Q_1 and Q_2 we just need to build the dialectical trees for arg'_1 (or arg_1) and arg_2 respectively. From Figures 3.8 and 3.9 arg'_1 is undefeated while arg_2 is defeated, therefore $\mathcal{KB} \models_{DT} Q_1$ and $\mathcal{KB} \not\models_{DT} Q_2$.

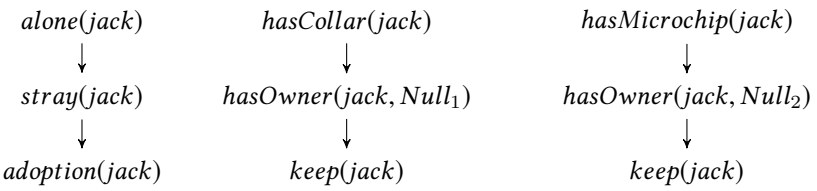

Figure 3.7: Chase graph for σ_{obl}-chase$(\mathcal{F}, \mathcal{R})$ of Example 3.1

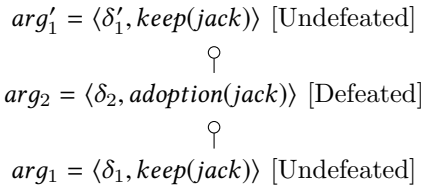

Figure 3.8: Dialectical Tree for arg'_1

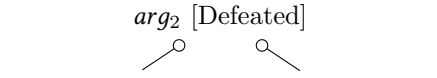

Figure 3.9: Dialectical Tree for arg_2

As shown above, the oblivious chase does not lose derivations, however the "weakness" of its derivation reducer compared to the frontier one removes the guarantee that the chase will stop for some concrete classes of Finite Expansion Set [Onet, 2013] (as explained in Chapter 2 Figure 2.2).

Proposition 3.2 (Oblivious Chase Graph Derivations Completeness). *Extracting derivations for a fact from an oblivious chase using chase graph is complete.*

Proof. We prove this by construction, a rule application is removed only if the rule has already been applied with the exact same homomorphism, therefore all possible rule applications with all possible homomorphisms are kept once, thus no derivation is lost. □

Intuitively, the stronger the derivation reducer, the bigger the risk of losing derivations. The restricted chase is stronger than the frontier one, it relies on the notion of "useful" rule application. A rule application is "useful" if it generates facts that cannot be mapped to previously generated ones, meaning that there is no homomorphism from the generated facts to the existing set of facts.

Definition 3.2 (Restricted Chase [Fagin et al., 2005]). *The restricted chase σ_{res}-chase (also called standard chase [Fagin et al., 2005]) uses the restricted derivation reducer denoted by σ_{res} and defined as follows: for any derivation δ, $\sigma_{res}(D_0) = D_0$ and $\forall D_i = (\mathcal{F}_i, r_i, \pi_i) \in \delta$:*

$$Facts(\sigma_{res}(D_i)) = \begin{cases} \mathcal{F}_{i-1} \cup \pi_i^{safe}(Head(r_i)) & \text{if } \mathcal{F}_{i-1} \not\models \pi_i^{safe}(Head(r_i)) \\ \mathcal{F}_{i-1} & \text{otherwise} \end{cases}$$

The restricted chase removes any rule application that generates facts that can be mapped to existing ones. A possible application of the restricted chase on the animal shelter example is:

$$\begin{aligned}\sigma_{res}\text{-}chase(\mathcal{F},\mathcal{R}) = \langle &(\mathcal{F},\emptyset,\emptyset),(\mathcal{F}_1 = \mathcal{F}\cup\{hasOwner(jack,Null_1)\},r_2,\pi_1 = \{X\to jack\}),\\ &(\mathcal{F}_2 = \mathcal{F}_1\cup\{stray(jack)\},r_4,\pi_2 = \{X\to jack\}),\\ &(\mathcal{F}_3 = \mathcal{F}_2\cup\{adoption(jack)\},r_5,\pi_3 = \{X\to jack\}),\\ &(\mathcal{F}_4 = \mathcal{F}_3\cup\{keep(jack)\},r_1,\pi_4 = \{X\to jack, Y\to Null_1\})\rangle.\end{aligned}$$

Applying the rule r_2 would generate $\{hasOwner(jack, Null_1)\}$, which is considered new since it is not directly entailed by $\mathcal{F}_0$ ($\mathcal{F}_0 \nvDash \{hasOwner(jack, Null_1)\}$). Hence $\mathcal{F}_1 = \mathcal{F}_0 \cup \{hasOwner(jack, Null_1)\}$. Afterwards applying r_3 would generate $\{hasOwner(jack, Null_2)\}$, which is redundant since it can be mapped to $\{hasOwner(jack, Null_1)\}$ ($Null_2$ is mapped to $Null_1$), this rule application is therefore discarded, then $keep(jack)$ is generated by applying r_1 on $hasOwner(jack, Null_1)$. The chase graph of this restricted chase is depicted in Figure 3.10.

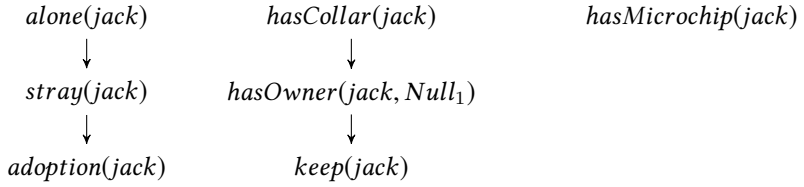

Figure 3.10: Chase graph for σ_{res}-$chase(\mathcal{F},\mathcal{R})$ if r_2 is applied before r_3

However, if r_3 is applied before r_2 then r_3 would generate $\{hasOwner(jack, Null_1)\}$ which is considered new as $F_0 \nvDash \{hasOwner(jack, Null_1)\}$, then applying r_2 would generate $\{hasOwner(jack, Null_2)\}$ which is considered redundant as it can be mapped the previously generated $hasOwner(jack, Null_1)$:

$$\begin{aligned}\sigma_{res}\text{-}chase(\mathcal{F},\mathcal{R}) = \langle &(\mathcal{F},\emptyset,\emptyset),(\mathcal{F}_2 = \mathcal{F}_1\cup\{hasOwner(jack,Null_1)\},r_3,\pi_1 = \{X\to jack\}),\\ &(\mathcal{F}_2 = \mathcal{F}_1\cup\{stray(jack)\},r_4,\pi_2 = \{X\to jack\}),\\ &(\mathcal{F}_3 = \mathcal{F}_2\cup\{adoption(jack)\},r_5,\pi_3 = \{X\to jack\}),\\ &(\mathcal{F}_5 = \mathcal{F}_4\cup\{keep(jack)\},r_1,\pi_4 = \{X\to jack, Y\to Null_1\})\rangle\end{aligned}$$

The chase graph built with the restricted chase when r_3 is applied before r_2 is shown in Figure 3.11.

As shown above, depending on the order of rule applications, chase graph built using the restricted chase is also prone to derivation loss.

Proposition 3.3 (Restricted Chase Graph Derivations Incompleteness). *Extracting derivations for a fact from a restricted chase using chase graph is incomplete.*

Proof. By contradiction (cf. Example 3.1) □

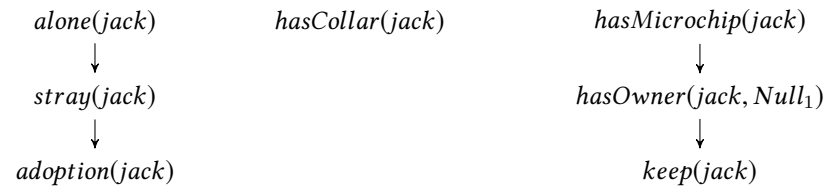

Figure 3.11: Chase graph for σ_{res}-*chase*$(\mathcal{F}, \mathcal{R})$ if r_3 is applied before r_2

The restricted chase checks only for local redundancy (i.e. if the generated facts can be mapped to the existing set of facts), the core chase however checks for local and global redundancy (i.e. if there are also some facts in the existing set of facts that can be mapped to the new generated facts), therefore any rule application removed by the restricted chase, will also be removed by the core chase. In fact, given the strength results of the derivation reducer order relation in Proposition 2.1 on page 21 we can define a derivation loss order as shown in the following Proposition 3.4.

Definition 3.3 (Derivations Loss Chase Order). *We say that a type of σ_1-chase is more prone to derivation loss than another type of chase σ_2-chase (denoted σ_1-chase $\leq$ σ_2-chase) if when a derivation for a fact is lost in σ_1-chase, it is also lost in σ_2-chase.*

Proposition 3.4. *The following holds: σ_{obl}-chase $\leq$ σ_{fr}-chase $\leq$ σ_{res}-chase $\leq$ σ_{core}-chase.*

Proof. Given the reducer order relation $\leq$ of Proposition 2.1, if $\sigma_1 \leq \sigma_2$ then any rule application removed by σ_1 will also be removed by σ_2. Therefore a chase graph built with σ_2-*chase* will at least lose the same derivations as a chase graph built with σ_1-*chase*. □

3.2 Derivation Loss Fix: Graph of Atom Dependency

In order to avoid the derivation loss problem, a more adapted combinatorial structure for derivation extraction is needed. In this section we present the notion of *Graph of Atom Dependency* (GAD), we define the algorithms that can be used to construct it and explain how the different kinds of chase can impact its construction, then we show how this hypergraph structure is used in order to construct all derivations for a fact.

The Graph of Atom Dependency relies on the notions of hypergraphs and hyperpaths [Gallo et al., 1993, Nguyen and Pallottino, 1989]. A hypergraph is a graph where the edges link a set of nodes to another set of nodes.

Definition 3.4 (Edge-labeled Directed Hypergraph [Gallo et al., 1993]). *A directed edge-labeled hypergraph is a tuple $\mathcal{H} = (\mathcal{V}, \mathcal{E}, L)$ where:*

- $\mathcal{V}$ *is a set of vertices (or nodes).*

- $\mathcal{E} \subseteq 2^{\mathcal{V}} \times 2^{\mathcal{V}}$ *is a set of directed hyperedges (or edges). A directed hyperedge* $e \in \mathcal{E}$ *is an ordered pair* $e = (U, W)$ *of non empty disjoint subsets of vertices* $U, W \in 2^{\mathcal{V}}$*; U is the tail of e while W is its head noted* $tail(e)$ *and* $head(e)$ *respectively.*

- $L : \mathcal{E} \rightarrow Labels$ *is a labeling function that maps each edge* $e \in \mathcal{E}$ *with an element of the labeling set Labels.*

Definition 3.5 (Path and Hyperpath [Nguyen and Pallottino, 1989]). *In a hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$*, a path* $P_{s/t}$ *of length k in a hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ *from a node* $s \in \mathcal{V}$ *to a node* $t \in \mathcal{V}$ *is a sequence of hyperedges* $\langle e_1, \ldots, e_k \rangle$ *such that:* $s \in tail(e_1)$*,* $t \in head(e_k)$*, and* $\forall 1 < i \leq k, head(e_{i-1}) \cap tail(e_i) \neq \emptyset$*.*

A hyperpath $\Theta_{S/t}$ *from* $S \subseteq \mathcal{V}$ *to* $t \in \mathcal{V}$ *is a hypergraph* $\mathcal{H}_p = (\mathcal{V}_p, \mathcal{E}_p)$ *satisfying the following conditions:*

1. $\mathcal{E}_p \subseteq \mathcal{E}$*,*

2. $S \cup \{t\} \subseteq \mathcal{V}_p$ *where* $\mathcal{V}_p = \bigcup_{e \in \mathcal{E}_p} (tail(e) \cup head(e))$*,*

3. $\forall v \in \mathcal{V}_p$*, there is a path* $P_{v/t}$ *from v to t.*

In order to clearly define hypergraphs and how we draw them in this chapter, let us consider the following Example 3.2 that illustrates the notions of hypergraph and hyperpath. In Figure 3.13 we give the equivalent bipartite depiction of the hypergraph shown in Figure 3.12. For clarity reasons we will use the bipartite depiction throughout the chapter.

Example 3.2 (Hypergraph and Hyperpath). *Consider a hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E}, L)$ *with*

- $\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5\}$*.*

- $\mathcal{E} = \{\varepsilon_1, \varepsilon_2\}$ *s.t.* $\varepsilon_1 = (\{v_1\}, \{v_3, v_4, v_5\})$ *and* $\varepsilon_2 = (\{v_1, v_2\}, \{v_3\})$*.*

- $L = \{(\varepsilon_1, label_{\varepsilon_1}), (\varepsilon_2, label_{\varepsilon_2})\}$*.*

In this hypergraph we have $tail(e_2) = \{v_1, v_2\}$ *(please note that* $tail(e_2)$ *is depicted in the upper half of the hyperedge* e_2 *in Figure 3.13).*

A path from v_1 *to* v_4 *is a sequence of hyperedges* $P_{v_1/v_4} = \langle \varepsilon_2 \rangle$*. A hyperpath from* $\{v_1\}$ *to* v_4 *is the hypergraph* $\Theta_{\{v_1\}/v_4} = (\mathcal{V}_\Theta, \mathcal{E}_\Theta)$ *s.t.* $\mathcal{V}_\Theta = \{v_1, v_3, v_4, v_5\}$ *and* $\mathcal{E}_\Theta = \{\varepsilon_1\}$*.*

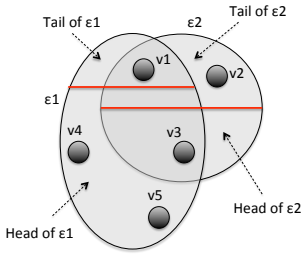

Figure 3.12: Hypergraph in Example 3.2

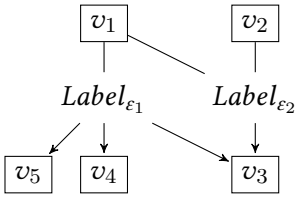

Figure 3.13: Bipartite depiction of the hypergraph in Example 3.2

A Graph of Atom Dependency (GAD) is a hypergraph where the set of nodes corresponds to the set of atoms and the set of labeled edges corresponds to rule applications labeled by the rule and the corresponding homomorphism.

Definition 3.6 (Graph of Atom Dependency). *Given a knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ *and the set of all homomorphisms* Π*, a Graph of Atom Dependency of* $\mathcal{KB}$ *is a directed edge-labeled hypergraph* $GAD_{\mathcal{KB}} = (\mathcal{V}, \mathcal{E}, L)$ *where:*

- $\mathcal{V}$ *is a set of ground atoms s.t.* $\mathcal{F}^* \subseteq \mathcal{V}$ *(*$\mathcal{V}$ *contains* $\mathcal{F}$ *and all generated atoms from* $\mathcal{F}$ *using* $\mathcal{R}$*).*
- $\mathcal{E} \subseteq 2^{\mathcal{V}} \times 2^{\mathcal{V}}$ *is a set of hyperedges.*
- $L : \mathcal{E} \rightarrow \mathcal{R} \times \Pi$ *is a labeling function that maps each edge* $e \in \mathcal{E}$ *to a tuple* (r, π) *where* $r \in \mathcal{R}$ *and* $\pi \in \Pi$*, s.t.* $head(e)$ *is obtained by applying* r *on* $tail(e)$ *using* π*.*

Example 3.3 (GAD of σ_{obl}-chase for Animal Shelter). *Figure 3.14 describes the Graph of Atom Dependency* $GAD_{\mathcal{KB}} = (\mathcal{V}, \mathcal{E}, L)$ *of the oblivious chase in Example 3.1:*

- $\mathcal{V} = \{alone(jack),\ hasCollar(jack),\ hasMicrochip(jack),\ hasOwner(jack,\ Null_1),\ hasOwner(jack, Null_2),\ stray(jack),\ adoption(jack),\ keep(jack)\}$
- $\mathcal{E} = \{e_1 = (\ \{hasCollar(jack)\}, \{hasOwner(jack, Null_1)\}\ ),$
  $e_2 = (\ \{hasMicrochip(jack)\}, \{hasOwner(jack, Null_2)\}\ ),$
  $e_3 = (\ \{alone(jack)\}, \{stray(jack)\}\ ),\ e_4 = (\ \{stray(jack)\}, \{adoption(jack)\}\ ),$
  $e_5 = (\ \{hasOwner(jack, Null_1)\}, \{keep(jack)\}\ ),$
  $e_6 = (\ \{hasOwner(jack, Null_2)\}, \{keep(jack)\}\ )\ \}$
- $L = \{(\ e_1, (r_2, \pi_1)\ ),\ (\ e_2, (r_3, \pi_2)\ ),\ (\ e_3, (r_4, \pi_3)\ ),\ (\ e_4, (r_5, \pi_4)\ ),\ (\ e_5, (r_1, \pi_5)\ ),$
  $(\ e_6, (r_1, \pi_6)\ )\}$

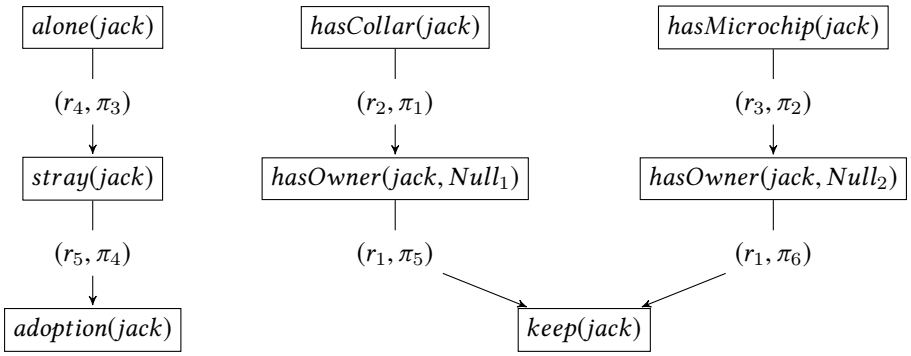

Figure 3.14: Graph of Atom Dependency for the oblivious chase of Example 3.1

3.2.1 GAD Construction and Chases Variants

The Graph of Atom Dependency is constructed using a chase. The chase choice directly affects how the GAD is built. The algorithm to construct a $GAD = (\mathcal{V}, \mathcal{E})$ using a *σ-chase* is run alongside the chase on an exhaustive breadth-first derivation of the knowledge base and takes as input the breadth-first derivation and the derivation reducer of the chase (as described by Algorithm 3.1): for each possible rule application, if it generates new facts (according to the chase derivation reducer), then these new facts are added to the set of nodes. Else, if no new facts are generated for the last breadth-first rule application step (i.e. the chase stops) then the algorithm terminates. Given that each chase considers "new facts" differently, a procedure named *HandleRuleApplication* that handles rule applications is called. This procedure is specific to the type of chase and its aim is to create edges between the used facts and the generated ones of each rule application.

The algorithm is finite if the chase is finite and its complexity depends on the complexity of *HandleRuleApplication* since the tests in lines 3, 5, and 7 of Algorithm 3.1 can be done in polynomial time. The call to the procedure *HandleRuleApplication* is what differentiates a Graph of Atom Dependency from a chase graph.

In what follows, we define the *HandleRuleApplication* procedure for the various kinds of chase: oblivious, frontier, and restricted.

Oblivious Chase. It ensures that a rule is never applied more than once with the same homomorphism, thus two rule applications are considered the same (i.e. redundant) if they used the same rule with the same homomorphism. Due to the simplicity of this test, the *HandleRuleApplication* procedure (defined in Algorithm 3.2) for the oblivious chase simply ensures that for any rule application, if an edge representing it has not already been created, it creates it and adds it to the set of edges. This algorithm is

Algorithm 3.1 GAD construction with chase

Function `ChaseGAD` (δ, σ)
input : δ : An exhaustive breadth-first derivation of a knowledge base,
σ: The derivation reducer
output: $GAD = (\mathcal{V}, \mathcal{E})$: Graph of Atom dependency w.r.t. $\mathcal{F}$ and $\mathcal{R}$

1 $\mathcal{V} \leftarrow \mathcal{F}_0$; $\mathcal{E} \leftarrow \emptyset$; $GAD \leftarrow (\mathcal{V}, \mathcal{E})$;
2 **foreach** $D_i = (\mathcal{F}_i, r_i, \pi_i) \in \delta$ **do**
3 **if** $Facts(\sigma(D_i)) \neq Facts(\sigma(D_{i-1}))$ **then**
4 **foreach** $v \in \pi_i(Head(r_i))$ **do**
5 **if** $v \notin \mathcal{V}$ **then**
6 Add v to $\mathcal{V}$;
7 **else if** *No new facts are generated for the last breadth-first step* **then**
8 break;
9 *HandleRuleApplication*(δ, D_i, GAD);
10 **return** GAD;

polynomial because checking if an edge is part of the set of edges is polynomial. The GAD of the oblivious chase of the animal shelter Example 3.1 is depicted in Figure 3.14 on the preceding page.

Algorithm 3.2 Handle rule applications for oblivious chase

Procedure `HandleRuleApplication` (δ, D_i, GAD)
input : δ: An exhaustive breadth-first derivation of $\mathcal{KB}$,
$D_i = (\mathcal{F}_i, r_i, \pi_i)$: rule application tuple,
$GAD = (\mathcal{V}, \mathcal{E})$: graph of atom dependency

1 **if** $e = (\pi_i(H_i), \pi_i(C_i)) \notin \mathcal{E}$ **then**
2 Add e to $\mathcal{E}$;

The following proposition states that for any rule application in an exhaustive breadth-first derivation of a knowledge base, there exists an edge representing it in the generated GAD using Algorithms 3.1 and 3.2, meaning that no rule application is lost.

Proposition 3.5 (GAD σ_{obl}-*chase* Completeness). *Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ expressed in $\mathcal{L}_{\forall\exists}$ and $GAD = (\mathcal{V}, \mathcal{E})$ generated by a finite oblivious chase, for all (r_i, π_i) possible rule applications in $\mathcal{KB}$, $\exists e \in \mathcal{E}$ such that $e = (\pi_i(Body(r_i)), \pi_i(Head(r_i)))$.*

Proof. We prove this by construction. Given a knowledge base $\mathcal{KB}$ and an exhaustive breadth-first derivation δ of $\mathcal{KB}$, since for any $D_i = (\mathcal{F}_i, r_i, \pi_i) \in \delta$ until the last breadth-first step k in which all possible rule applications have

been applied at least once in δ, *HandleRuleApplication* is called (as per Algorithm 3.1), if $e = (\pi_i(Body(r_i)), \pi_i(Head(r_i))) \notin \mathcal{E}$ then it is added, otherwise it already exists. Therefore all possible rule applications are represented with an edge. □

Frontier/Skolem Chase. For this chase, a rule application is redundant if its rule has already been applied with the same mapping of the frontier variables. The *HandleRuleApplication* for the frontier chase (defined in Algorithm 3.3) ensures that if a rule application tuple $D_i = (\mathcal{F}_i, r_i, \pi_i)$ is applying a rule that has already been applied with a homomorphism π_j having the same mapping of frontier variable as π_i, then an edge starting from the facts used by this rule application D_i to the generated facts of the previous rule application D_j is added to the set of edges (if it does not already exists). Otherwise D_i is not redundant and the edge representing it is added to the set of edges. This algorithm is simply applying the same tests made by the frontier derivation reducer and its complexity is the same as the frontier chase (polynomial data complexity and exponential combined complexity) [Marnette, 2009].

Algorithm 3.3 Handle rule application for Frontier chase

Procedure `HandleRuleApplication` (δ, D_i, GAD)

```
    input : δ: An exhaustive breadth-first derivation of KB,
            D_i = (F_i, r_i, π_i) : rule application tuple,
            GAD = (V, E) : graph of atom dependency
1   if ∃D_j = (F_j, r_j, π_j) ∈ δ s.t. j < i, r_j = r_i, and π_j|_fr(r_j)(Head(r_j)) =
     π_i|_fr(r_i)(Head(r_i)) then
2       if e = (π_i(Body(r_i)), π_j(Head(r_i))) ∉ E then
3           Add e to E;
4   else
5       Add e = (π_i(Body(r_i)), π_i(Head(r_i))) to E;
```

Similarly to Proposition 3.5, Proposition 3.6 states that no possible rule application is lost, even if it does not generate new facts, it is still added to the edges of the GAD.

Proposition 3.6 (GAD σ_{fr}-*chase* Completeness). *Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ expressed in $\mathcal{L}_{\forall\exists}$ and GAD $= (\mathcal{V}, \mathcal{E})$ generated by a frontier chase, $\forall(r_i, \pi_i)$ possible rule applications in $\mathcal{KB}$, $\exists e \in \mathcal{E}$ such that $e = (\pi_i(Body(r_i)), \pi_i(Head(r_i)))$ or $e = (\pi_i(Body(r_i)), \pi_j(Head(r_i)))$ where $\pi_j|_{fr(r_i)} = \pi_i|_{fr(r_i)}$.*

Proof. We prove this by construction. Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ and an exhaustive breadth-first derivation δ of $\mathcal{KB}$, since for

any $D_i = (\mathcal{F}_i, r_i, \pi_i) \in \delta$ until the last breadth-first step k in which all possible rule applications have been applied at least once in δ, *HandleSameAtoms* is called (as per Algorithm 3.1). There are three possible cases: first, if a rule has never been applied with the same homomorphism for the frontier variables then the edge representing it is simply added. Second, if it has been applied with the same frontier mapping then an edge is added with a head pointing to the previously generated facts. Third, if it has been applied with the same homomorphism then the edge representing it already exists. Therefore all possible rule applications are represented with an edge. □

The GAD of the frontier chase of the animal shelter Example 3.1 is the same as the one constructed using the oblivious chase (depicted in Figure 3.14 on page 80). However, since *HandleRuleApplication* for the frontier chase is more general than the one for the oblivious chase, it might result in different GADs for the same knowledge base as shown in the following Example 3.4.

Example 3.4 (Oblivious vs Frontier GAD). *Consider the knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ *where:*

- $\mathcal{F} = \{p(a,b), p(a,c)\}$
- $\mathcal{R} = \{r_1 : \forall X, Y\, p(X,Y) \rightarrow \exists Z\, q(X,Z);\ \ r_2 : \forall X, Y\, q(X,Y) \rightarrow t(X)\}$.

A possible oblivious chase of $\mathcal{KB}$*:*

$$\begin{aligned}\sigma_{obl}\text{-}chase(\mathcal{F},\mathcal{R}) = \langle &(\mathcal{F}, \emptyset, \emptyset), (\mathcal{F}_1 = \mathcal{F} \cup \{q(a, Null_1)\}, r_1, \pi_1 = \{X \rightarrow a,\ Y \rightarrow b\}),\\ &(\mathcal{F}_2 = \mathcal{F}_1 \cup \{q(a, Null_2)\}, r_1, \pi_2 = \{X \rightarrow a,\ Y \rightarrow c\}),\\ &(\mathcal{F}_3 = \mathcal{F}_2 \cup \{t(a)\}, r_2, \pi_3 = \{X \rightarrow a, Y \rightarrow Null_1\}),\\ &(\mathcal{F}_4 = \mathcal{F}_3 \cup \{t(a)\}, r_2, \pi_4 = \{X \rightarrow a, Y \rightarrow Null_2\})\rangle\end{aligned}$$

The rule applications of r_1 *on* $p(a,c)$ *and* r_2 *on* $q(a, Null_2)$ *are not considered redundant by the oblivious chase reducer because the same rules are applied with a different homomorphism. However, for the frontier chase reducer, applying* r_1 *on* $p(a,b)$ *is the same as applying it on* $p(a,c)$ *because they have the same mapping for frontier variable (i.e.* $\pi_1|_{fr(r_1)} = \pi_2|_{fr(r_1)}$*). A possible frontier chase of* $\mathcal{KB}$*:*

$$\begin{aligned}\sigma_{fr}\text{-}chase(\mathcal{F},\mathcal{R}) = \langle &(\mathcal{F}, \emptyset, \emptyset), (\mathcal{F}_1 = \mathcal{F} \cup \{q(a, Null_1)\}, r_1, \pi_1 = \{X \rightarrow a,\ Y \rightarrow b\}),\\ &(\mathcal{F}_2 = \mathcal{F}_1 \cup \{t(a)\}, r_2, \pi_2 = \{X \rightarrow a, Y \rightarrow Null_1\})\rangle\end{aligned}$$

The Graphs of Atom Dependency $GAD_{\sigma_{obl}}$ *and* $GAD_{\sigma_{fr}}$ *generated by* σ_{obl}*-chase*$(\mathcal{F}, \mathcal{R})$ *and* σ_{fr}*-chase*$(\mathcal{F}, \mathcal{R})$ *are shown in Figures 3.15 and 3.16 respectively.*

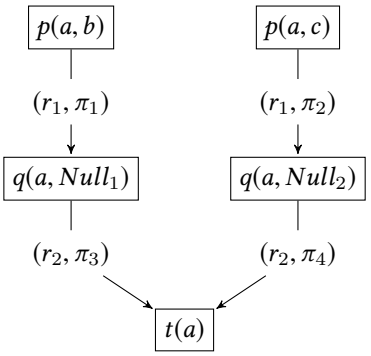

Figure 3.15: $GAD_{\sigma_{obl}}$ from oblivious chase of Example 3.4

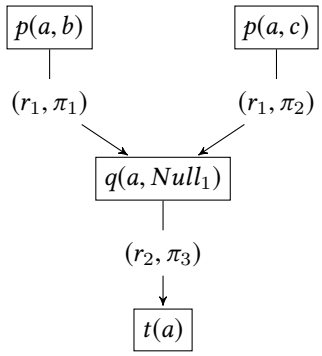

Figure 3.16: $GAD_{\sigma_{fr}}$ from frontier chase of Example 3.4

The structural differences between a GAD constructed using an oblivious chase and a GAD constructed using a frontier chase can be quantified as shown in the following proposition.

Proposition 3.7 (Structural link between GADs obtained by oblivious and frontier chases). *Let* $GAD_{\sigma_{obl}} = (\mathcal{V}_{\sigma_{obl}}, \mathcal{E}_{\sigma_{obl}})$ *and* $GAD_{\sigma_{fr}} = (\mathcal{V}_{\sigma_{fr}}, \mathcal{E}_{\sigma_{fr}})$ *be two Graphs of Atom Dependency for* $(\mathcal{F}, \mathcal{R})$ *generated by an oblivious and a frontier chase respectively. If the oblivious chase is finite, then* $|\mathcal{V}_{\sigma_{fr}}| \leq |\mathcal{V}_{\sigma_{obl}}|$ *and* $|\mathcal{E}_{\sigma_{fr}}| \leq |\mathcal{E}_{\sigma_{obl}}|$.

Proof. Given that frontier is stronger than the oblivious chase (cf. Proposition 2.1 on page 21) more rule applications might be removed, thus some facts containing nulls might not be generated compared to the oblivious chase, therefore $|\mathcal{V}_{\sigma_{fr}}| \leq |\mathcal{V}_{\sigma_{obl}}|$ and $|\mathcal{E}_{\sigma_{fr}}| \leq |\mathcal{E}_{\sigma_{obl}}|$. □

Furthermore, since rule applications are not lost given Propositions 3.5 and 3.6, the generated GADs by an oblivious or a frontier chase would always yield the same number of derivations for the same fact containing the same rule applications. In Example 3.4, the derivations for $t(a)$ using oblivious chase are:

- $\delta_1 = \langle(\mathcal{F}_0 = \{p(a,b)\}, \emptyset, \emptyset)\ ,\ (\mathcal{F}_1 = \mathcal{F}_0 \cup \{q(a, Null_1)\}, r_1, \pi_1),\ (\mathcal{F}_2 = \mathcal{F}_1 \cup \{t(a)\}, r_2, \pi_3)\rangle$
- $\delta_2 = \langle(\mathcal{F}_0 = \{p(a,c)\}, \emptyset, \emptyset)\ ,\ (\mathcal{F}_1 = \mathcal{F}_0 \cup \{q(a, Null_2)\}, r_1, \pi_2),\ (\mathcal{F}_2 = \mathcal{F}_1 \cup \{t(a)\}, r_2, \pi_4)\rangle$.

The derivations for $t(a)$ using frontier chase are:

- $\delta_1 = \langle(\mathcal{F}_0 = \{p(a,b)\}, \emptyset, \emptyset)\ ,\ (\mathcal{F}_1 = \mathcal{F}_0 \cup \{q(a, Null_1)\}, r_1, \pi_1),\ (\mathcal{F}_2 = \mathcal{F}_1 \cup \{t(a)\}, r_2, \pi_3)\rangle$
- $\delta_2 = \langle(\mathcal{F}_0 = \{p(a,c)\}, \emptyset, \emptyset)\ ,\ (\mathcal{F}_1 = \mathcal{F}_0 \cup \{q(a, Null_1)\}, r_1, \pi_2),\ (\mathcal{F}_2 = \mathcal{F}_1 \cup \{t(a)\}, r_2, \pi_3)\rangle$.

Restricted Chase. A rule application is redundant in the restricted chase if the facts it generates can be mapped to existing ones. The *HandleRuleApplication* for the restricted chase (defined in Algorithm 3.4) ensures that if there is a homomorphism π' from the body and the head of the rule to the existing set of facts (meaning that this rule application and its homomorphism are not "useful") then an edge starting from the used facts to the previously generated ones is added (if it does not already exist in the set of edges). Otherwise, this rule application and its homomorphism are useful and an edge representing it is added. This algorithm is simply applying the same tests made by the restricted derivation reducer and its complexity is the same as the restricted chase which is polynomial data complexity and exponential combined complexity [Fagin et al., 2005].

Algorithm 3.4 Handle same atoms for restricted chase

Procedure `HandleRuleApplication` $(\mathcal{F}_{i-1}, D_i, GAD)$

input : δ: An exhaustive breadth-first derivation of $\mathcal{KB}$,
$D_i = (\mathcal{F}_i, r_i, \pi_i)$: rule application tuple,
$GAD = (\mathcal{V}, \mathcal{E})$: graph of atom dependency

1 **if** $\exists \pi'$ *s.t.* $\pi'(Head(r_i)) \subseteq \mathcal{F}_{i-1}$ **then**
2 /* π_i *is not a useful homomorphism* */
3 **if** $e = (\pi(Body(r_i)), \pi'(Head(r_i))) \notin \mathcal{E}$ **then**
4 Add e to $\mathcal{E}$;
5 **else**
6 /* π_i *is a useful homomorphism* */
7 Add $e = (\pi_i(Body(r_i)), \pi_i(Head(r_i)))$ to $\mathcal{E}$;

Similarly to Propositions 3.5 and 3.6, Proposition 3.8 states that no rule application is lost for the restricted chase.

Proposition 3.8 (GAD σ_{res}-*chase*$(\mathcal{F}, \mathcal{R})$ Completeness). *Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ expressed in $\mathcal{L}_{\forall\exists}$ and $GAD = (\mathcal{V}, \mathcal{E})$ generated by a frontier chase, $\forall (r_i, \pi_i)$ possible rule applications in $\mathcal{KB}$, $\exists e \in \mathcal{E}$ such that $e = (\pi_i(Body(r_i)), \pi_i(Head(r_i)))$ or $e = (\pi(Body(r_i)), \pi'(Head(r_i)))$ where π' is a homomorphism such that $\pi'(Head(r_i)) \subseteq \mathcal{F}_{i-1}$.*

Proof. We prove this by construction. Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ and an exhaustive breadth-first derivation δ of $\mathcal{KB}$, since for any $D_i = (\mathcal{F}_i, r_i, \pi_i) \in \delta$ until the last breadth-first step k in which all possible rule applications have been applied at least once in δ, *HandleSameAtoms* is called (as per Algorithm 3.1). There are three possible cases: (1) either the generated facts can be mapped to existing ones (π is not useful) then an edge is added with a head pointing to the previously generated atoms, or (2) the generated facts are new (π is useful) then the edge representing its rule

application is simply added, or (3) the rule application already exists then the edge representing it has been previously added. Therefore all possible rule applications are represented with an edge. □

The GAD of the restricted chase of the animal shelter Example 3.1 is shown in Figure 3.17, it is the same if r_2 is applied before r_3 or not.

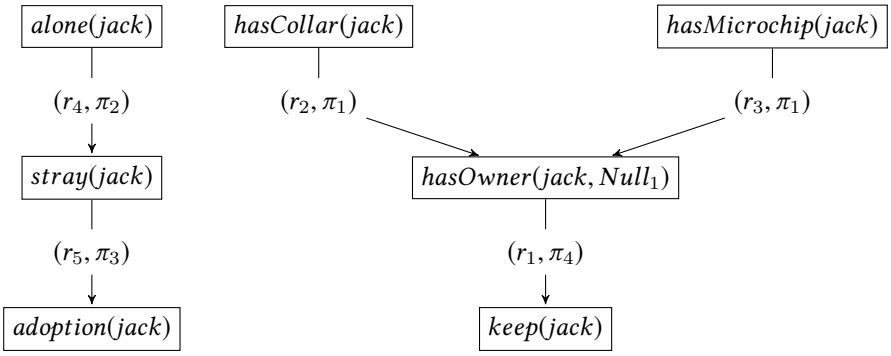

Figure 3.17: Graph of Atom Dependency for the restricted chase of Example 3.1 if r_3 is applied before r_2

Since the restricted derivation reducer is stronger than the frontier one, *HandleRuleApplication* might result in different GADs for the same knowledge base as shown in Figures 3.14 and 3.17. They would yield however, the same number of derivations containing the same rule applications for the same fact since rule applications are not lost given Propositions 3.6 and 3.8. The structural differences between a GAD constructed using a frontier chase and a GAD constructed using a restricted chase is quantified in the following Proposition 3.9.

Proposition 3.9 (Structural link between GADs obtained by frontier and restricted chases). *Let* $GAD_{\sigma_{fr}} = (\mathcal{V}_{\sigma_{fr}}, \mathcal{E}_{\sigma_{fr}})$ *and* $GAD_{\sigma_{res}} = (\mathcal{V}_{\sigma_{res}}, \mathcal{E}_{\sigma_{res}})$ *be two Graphs of Atom Dependency for a knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ *generated by a restricted and a frontier chase respectively. If the frontier chase is finite, then* $|\mathcal{V}_{\sigma_{res}}| \leq |\mathcal{V}_{\sigma_{fr}}|$ *and* $|\mathcal{E}_{\sigma_{res}}| \leq |\mathcal{E}_{\sigma_{fr}}|$.

Proof. Given that the restricted reducer is stronger than the frontier reducer (cf. Proposition 2.1) more rule applications might be removed, thus some facts containing nulls might not be generated compared to the frontier chase, therefore $|\mathcal{V}_{\sigma_{res}}| \leq |\mathcal{V}_{\sigma_{fr}}|$ and $|\mathcal{E}_{\sigma_{res}}| \leq |\mathcal{E}_{\sigma_{fr}}|$. □

3.2.2 Derivation Extraction

The intuition behind the use of the GAD is that, for a given GAD and a given facts f, there is a one-to-one mapping, up to *derivation equivalence*, between the set of hyperpaths to f and the set of derivations to f. Therefore,

once the GAD constructed (by considering the different chase mechanisms) the problem of obtaining all derivations for f can be transformed into the problem of generating all hyperpaths from the starting set of facts to f.

Two derivations for a fact are equivalent if they have the same set of facts and the same set of applied rules (along with their respective homomorphisms).

Definition 3.7 (Derivation Equivalence). *Given two derivations δ and δ' for a fact f, we say that δ and δ' are equivalent (denoted $\delta \simeq \delta'$) iff:*

- $\bigcup_{D\in\delta} fact(D) = \bigcup_{D'\in\delta'} fact(D')$ *and*
- $\bigcup_{D\in\delta}(rule(D), homorph(D)) = \bigcup_{D'\in\delta'}(rule(D'), homorph(D'))$

Derivations are constructed from the hyperpaths of the GAD. The following proposition shows that for every hyperpath of the GAD starting from a subset of the initial set of facts to a certain fact we can construct an equivalent derivation for that fact. This will ensure the soundness of the hyperpath generation with respect to the problem of generating all derivations.

Proposition 3.10 (Hyperpath Soundness w.r.t. a Derivation). *Let GAD be a Graph of Atom Dependency generated by applying a σ-chase$(\mathcal{F}, \mathcal{R})$ on a set of facts $\mathcal{F}$ w.r.t. a set of rules $\mathcal{R}$. If there exists a hyperpath $\Theta_{S/f}$ in GAD from $S \subseteq \mathcal{F}$ to a fact f, then there exists a derivation δ for f.*

Proof. We prove this by construction. Since $\Theta_{S/t}$ starts from a subset of the initial set of facts $\mathcal{F}$ then there is at least one acyclic ordering of its hyperedges [Gallo et al., 1993] $\langle e_0, \ldots, e_k\rangle$. Based on this ordering we can generate a sequence equivalent to the derivation for f as follows: $D_0 = (F_0 = S, \emptyset, \emptyset)$ and $D_i = (\mathcal{F}_i = \mathcal{F}_{i-1} \cup head(e_i), r_i, \pi_i)$. □

The following proposition indicates that for a given GAD and for a given hyperpath $\Theta_{S/f}$ in that GAD, the derivations for f that can be constructed from $\Theta_{S/f}$ are equivalent.

Proposition 3.11 (Hyperpath Soundness). *Given a GAD of a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ and a hyperpath $\Theta_{S/f}$ in GAD from $S \subseteq \mathcal{F}$ to a fact f, all the possible derivations for f generated from $\Theta_{\mathcal{F}/t}$ are equivalent.*

Proof. We prove this by construction. Since $\Theta_{S/t}$ starts from a subset of the initial set of facts $\mathcal{F}$ then there is at least one acyclic ordering of its hyperedges [Gallo et al., 1993]. In fact, $\Theta_{\mathcal{F}/t}$ can have different acyclic orderings of its hyperedges [Gallo et al., 1993]. Derivations generated from these valid orderings contain the same facts and rule applications since all ordering are for the same hyperedges. Thus, the generated derivations are equivalent. □

Let us now show that the completeness holds. More precisely we can show that for a given knowledge base and its GAD, if there is a derivation for a fact, then there is a hyperpath in the GAD from a subset of the initial set of facts to this fact.

Proposition 3.12 (Hyperpath Completeness). *Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$, if there is a derivation δ for f in $\mathcal{KB}$, then there exists a hyperpath $\Theta_{S/f}$ in the GAD of $\mathcal{KB}$ such that $S \subseteq \mathcal{F}$ and any derivation extracted from $\Theta_{S/f}$ is equivalent to δ.*

Proof. We prove this by contradiction. Let us suppose that there exists a derivation δ for a fact f in $\mathcal{KB}$ such that no hyperpath $\Theta_{S/f}$ with $S \subseteq \mathcal{F}$ can be constructed in the associated *GAD*. This means that a rule application in the derivation is not present in the hyperpath. which implies that in the construction of the GAD this rule application has not been considered. This is impossible given the results of the completeness of GAD construction using different chase variants (Propositions 3.5, 3.6, and 3.8). □

In order to construct all non-equivalent derivations from a set of facts $\mathcal{F}$ to a fact f we only need to find all hyperpaths from $\mathcal{F}$ to f and then construct an acyclic ordering of their edges. To compute the paths (sequences of hyperedges) from $S \subseteq \mathcal{F}$ to f we start from the node representing f and we branch backward using the incoming edges. The function that lists the incoming edges is called the backward star [Nguyen and Pallottino, 1989] and is defined as follows: $BS(v) = \{e \in \mathcal{E} | v \in head(e)\}$. The recursive function *FP* defined in Algorithm 3.5 computes all acyclic paths that connect a subset of $\mathcal{F}$ to an atom f using backward branching; we then use these paths in the procedure *FindAllHyperpaths* in order to construct the hyperpaths.

The Algorithm 3.5 is a modification of [Nguyen and Pallottino, 1989] to take into account hyperedges rather than hyperarcs, it extracts all hyperpaths from a source to a target. The function *FP* is called in the worst case $maxdepth \times branching$ times where *maxdepth* is the maximum depth of the GAD and *branching* is the branching factor (maximum number of incoming edges). This function has a cost of $O(branching^2 \times n^3)$ (where n is the number of nodes) since it performs a Cartesian product ($O(n^2)$) and a cycle check ($O(n)$) [Pandy and Chawla, 2003], the over all execution cost is $O(n^{k(maxdepth \times branching)})$ where k is a constant. Therefore the combined complexity of Algorithm 3.5 is exponential while its data complexity is polynomial since data complexity considers the set of rules and subsequently the set of edges fixed which makes *maxdepth* and *branching* constants as they depend on the number and the body size of the rules.

The following Algorithm 3.6 is sound and complete with respect to the problem of all hyperpaths generation. The soundness is straightforward as the output is a list of hyperpaths by construction. The completeness can be proved by contradiction. Let us suppose that there exists a hyperpath that

Algorithm 3.5 Compute Paths

Function FP (S, t)

input : S : source nodes, t: target node
output: paths: set of all paths between S and t

1 paths $\leftarrow \{\}$;
2 **if** $t \in S$ **then**
3 **return** {}; /* t is part of the starting set S; stopping condition */
4 **if** $BS(t) = \emptyset$ **then**
5 **return** null; /* t has no incoming edges, thus there is no path from S to t i.e. no derivation for t */
6 **foreach** $e \in BS(t)$ **do**
7 path $\leftarrow \{e\}$;
8 tmp $\leftarrow \{\}$;
9 **foreach** $v \in tail(e)$ **do**
10 /* Compute all possible paths for all nodes in the tail of e */
tmp $\leftarrow$ FP $(S, v) \times$ tmp; /* Cartesian product */
11 /* Add the edge e to all paths for the nodes in its tail */
If adding e to a path in tmp creates a cycle remove it from tmp;
paths $\leftarrow$ (path $\times$ tmp);
12 **return** paths;

has not been outputted by the algorithm. This means that there exists a path not computed by Algorithm 3.5, implying that there exists an incoming edge to a node that is not part of the backward star of that node which is impossible.

Algorithm 3.6 Construct Hyperpaths

Function FindAllHyperpaths (S, t)

input : S : source nodes, t: target node
output: hyperpaths: set of all hyperpaths between S and t

1 hyperpaths $\leftarrow \{\}$; paths $\leftarrow FP(S, t)$;
2 **foreach** *path* $\in$paths **do**
3 $\mathcal{V} \leftarrow S$;
4 $\mathcal{E} \leftarrow path$;
5 **foreach** $e \in \mathcal{E}$ **do**
6 Add $head(e)$ and $tail(e)$ to $\mathcal{V}$;
7 Add $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ to hyperpaths;

The complexity of constructing the Graph of Atom Dependency and extracting all derivations for a fact f using the previous algorithms is exponential combined complexity and polynomial data complexity.

After addressing the problem of derivation loss has, in the next section we present a tool for defeasible reasoning with existential rules using Dialectical Trees that relies on the Graph of Atom Dependency.

3.2.3 Graph of Atom Dependency at Work: DEFT Tool

In this section we present our chase-based implementation of defeasible Datalog$^{\pm}$ (cf. $\mathcal{L}_{\forall\exists}$ in Section 2.2.1 on page 28) with dialectical trees called DEFT[1] that allows for lossless derivation extraction using the Graph of Atom Dependency.

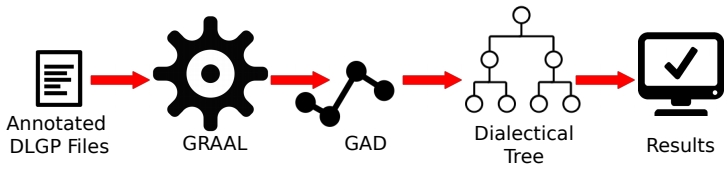

Figure 3.18: Simplified DEFT Architecture.

Implementation. DEFT architecture (depicted in Figure 3.18) relies on the Datalog$^{\pm}$ dedicated inference engine called GRAAL [Baget et al., 2015] that accepts a wide variety of formats (OWL2, RuleML and the Datalog$^{\pm}$ format DLGP [Baget et al., 2015]). To represent defeasible facts and rules we update DLGP to allow for the strict implication "<-" and the defeasible implication "<=" as shown in Example 3.5.

Example 3.5 (DLGP representation of Example 3.1). *The DLGP representation of the animal shelter defeasible knowledge base:*

```
% ------------------------ Rules ------------------------
[r1] keep(X) <- hasOwner(X,Y).
[r2] hasOwner(X,Y) <= hasCollar(X).
[r3] hasOwner(X,Y) <= hasMicrochip(X).
[r4] stray(X) <= alone(X).
[r5] adoption(X) <- stray(X).
% ---------------- Negative Constraints ----------------
! <- keep(X), adoption(X).
% ------------------------ Facts ------------------------
alone(jack) <- .
hasCollar(jack) <- .
hasMicrochip(jack) <- .
```

We run the frontier chase using the GRAAL framework and create the GAD. We then generate the arguments for a query and use a preference criterion to compute their dialectical tree. DEFT can either use explicit preferences based on rule labels or rely on the general specificity criterion

[1] Available at: `https://github.com/hamhec/DEFT`

[García and Simari, 2004, Stolzenburg et al., 2003] which favors two aspects in an argument: it prefers a more precise argument (relying on more atoms) or a more concise argument (relying on fewer rules). DEFT's API also provides hooks to define custom preference functions if needed.

DEFT is the first tool for defeasible reasoning with existential rules (without defeater rules). In order to assess the performance overhead of relying on the Graph of Atom Dependency to avoid derivation loss, we define a benchmark that tests the performance of defeasible reasoning tools and their limitations for each considered feature of defeasible reasoning (cf. Section 2.2.2). This benchmark can also be used to analyze defeasible reasoning implementations and classify them based on their semantics (e.g. ambiguity handling), expressiveness (e.g. existential rules).

3.3 Benchmark for Defeasible Reasoning Tools

An inherent characteristic of defeasible reasoning is its systematic reliance on a set of intuitions and rules of thumb, which have been longly debated between logicians [Horty et al., 1987, Makinson and Schlechta, 1991, Prakken, 2002, Antoniou, 2006]. For example, should an information that is derived from a contested claim be used to contest another claim (i.e. ambiguity handling)? Or, can different 'chains' of reasoning for the same claim be combined to defend against challenging statements (i.e. team defeat)? How strict rules are handled? etc. It appears that no single approach is appropriate in all situations, or for all purposes [Antoniou, 2006].

Several tools in the literature implement defeasible reasoning (cf. Section 2.2.7), each one follows a certain set of intuitions. In this section we are interested in defining a general benchmark that can be used by a data engineer *looking to select what tool to use* to perform defeasible reasoning. It is the first benchmark in the literature for first order logic defeasible reasoning tools classification and analysis based on their *semantics* (e.g. ambiguity handling), logical language (e.g. existential rules) and *expressiveness* (e.g. priorities on rules). We stress that we do *not want to compare* the tools amongst themselves to find the 'best' one. We want to be able to provide *an informative benchmark* that will allow data scientists to better understand the *strengths of available tools and the intuitions they follow and allow for.* This is particularly important as some of the tools are not full implementations of the formalisms behind them and their features and chosen intuitions *are not explicitly stated* in their description or companion papers.

3.3.1 Semantics, Expressiveness, and Performance

Let us now concretely discuss the various **features** concerning *semantics*, *expressiveness* and *performance* based on the different **intuitions** behind defeasible reasoning (cf. Section 2.2.2 on page 32). Please note that we do

not discuss what intuitions are better to adopt, as these often conflict, our aim is to facilitate the task of selecting what defeasible reasoning tool to use based on the reasoning requirements and the data at hand.

Semantics. Different intuitions impact what conclusions are accepted or rejected in a defeasible reasoning setting (cf. Section 2.2.2):

1. *Ambiguity Handling:* The intuition is whether information derived from an ambiguous (i.e. contested) fact should be used to contest another fact (ambiguity blocking / ambiguity propagating) [Stein, 1992].

2. *Team Defeat:* The absence of team defeat means that a rule r_1 (or a derivation) for a fact f attacked by another rule r_2 for a conflicting fact f' can only defend itself (meaning that for r_1 to 'survive' it has to be superior to r_2 i.e. $r_1 \succ r_2$ and $r_2 \not\succ r_1$). However, if we allow team defeat, r_1 can be successfully defended by another rule r_3 for f that is superior to r_2 (even if r_1 is inferior to r_2).

3. *Floating Conclusions:* Sometimes two conflicting and equally strong rules (or derivations) might be used to generate the same fact down the line [Makinson and Schlechta, 1991]. The intuition is whether this fact should be considered entailed given that regardless of how conflicts are handled it can always be concluded.

   The following Example 3.6 explains the different effects of combining ambiguity handling, team defeat, and floating conclusions.

   Example 3.6. *Consider the following* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ *that describes the process of deciding whether to buy a product or go on a vacation: An individual will not go on a vacation if he buys a smart-phone. He will not buy a product if it is detrimental to the environment unless it is cheap. He will also not buy it if it has slow delivery unless it has good reviews. In either case he will take a loan to pay for the phone or the vacation.*

   Suppose we have a cheap smart-phone with good reviews that is detrimental to the environment and with slow delivery. Should the individual buy it (i.e $Q_1 = buy(phone)$*)? Should he go on a vacation (*$Q_2 = go(vacation)$*)? Will he take a loan (*$Q_3 = take(loan)$*)?:*

   - $\mathcal{F} = \{\top \Rightarrow go(vacation), \top \rightarrow price(phone, cheap), \top \rightarrow reviews(phone, good), \top \rightarrow eco(phone, detrimental), \top \rightarrow delivery(phone, slow)\}$
   - $\mathcal{R} = \{r_1 : \forall X\, price(X, cheap) \Rightarrow buy(Y),$
     $r_2 : \forall X\, reviews(X, good) \Rightarrow buy(X),$
     $r_3 : \forall X\, eco(X, detrimental) \Rightarrow notBuy(phone),$
     $r_4 : \forall X, Y\, derlivery(X, slow) \Rightarrow notBuy(phone),$

$r_5 : \forall X\ buy(X) \Rightarrow take(loan)$,
$r_6 : \forall X\ buy(X) \Rightarrow notGo(vacation)$,
$r_6 : go(vacation) \Rightarrow take(loan)\}$

- $\mathcal{N} = \{\forall X\ buy(X) \wedge notBuy(X) \rightarrow \bot$,
  $go(vacation) \wedge notGo(vacation) \rightarrow \bot\}$
- $(r_1 \succ r_3), (r_2 \succ r_4)$

- $Q_1 = buy(phone)$ ***entailment:*** *in presence of team defeat,* $buy(phone)$ *is not ambiguous because for each chain of reasoning using a rule* r *for* $notBuy(phone)$ *there is a chain of reasoning using a rule* r' *for* $buy(phone)$ *such that* $r' \succ r$ *(i.e.* $(r_1 \succ r_3), (r_2 \succ r_4)$*). Therefore* $\mathcal{KB} \vdash^{TD} buy(phone)$ *and* $\mathcal{KB} \nvdash^{TD} notBuy(phone)$.
  In the absence of team defeat, $buy(phone)$ *is ambiguous because there is no chain of reasoning for* $buy(phone)$ *that can defend itself from all attacks: even if* r_1 *defends itself from* r_3 *(because* $r_1 \succ r_3$*), it does not defend against* r_4 *(since* $r_1 \not\succ r_4$ *and* $r_4 \not\succ r_1$*), and the same applies for* r_2*: it defends against* r_4 *but not against* r_3 *because* $r_2 \not\succ r_3$ *and* $r_3 \not\succ r_2$*. Therefore* $\mathcal{KB} \nvdash^{nTD} buy(phone)$ *and* $\mathcal{KB} \nvdash^{nTD} notBuy(phone)$.
- $Q_2 = go(vacation)$ ***entailment:*** *In an ambiguity blocking setting, if* $buy(phone)$ *is ambiguous (absence of team defeat) then* $notGo(vacation)$ *is also ambiguous and cannot be used to contest* $go(vacation)$*. Therefore* $\mathcal{KB} \vdash^{nTD}_{block} go(vacation)$ *and* $\mathcal{KB} \nvdash^{nTD}_{block} notGo(vacation)$*. On the other hand if* $buy(phone)$ *is not ambiguous (presence of team defeat) then* $notGo(vacation)$ *does not rely on an ambiguous fact and can be used to contest* $go(vacation)$*. Therefore* $\mathcal{KB} \nvdash^{TD}_{block} go(vacation)$ *and* $\mathcal{KB} \nvdash^{TD}_{block} notGo(vacation)$.
  In an ambiguity propagating setting, whether $buy(phone)$ *is ambiguous or not, it can always be used to generate* $notGo(vacation)$ *and contest* $go(vacation)$*. Therefore* $\mathcal{KB} \nvdash_{prop} go(vacation)$ *and* $\mathcal{KB} \nvdash_{prop} notGo(vacation)$.
- $Q_3 = take(loan)$ ***entailment:*** *If floating conclusions are allowed then* $take(loan)$ *can be derived, it can also be derived if* $buy(phone)$ *is entailed (presence of team defeat).*

4. *Handling of Strict Rules:* In some defeasible reasoning tools, such as the ones based on Defeasible Logics, facts that are not in direct conflict, and that are defeasibly derived from non ambiguous ones, are accepted (even if the application of strict rules and facts generates conflict). Other formalisms reject any fact that leads to conflict when strict rules and facts are applied.

Example 3.7 (Consistent Answers). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$*:*

- $\mathcal{F} = \{\top \Rightarrow incrim(e1, alice),\ \top \Rightarrow absolv(e2, alice)\}$
- $\mathcal{R} = \{r_1 : \forall X, Y\ incrim(X, Y) \rightarrow resp(Y),$
  $r_2 : \forall X, Y\ absolv(X, Y) \rightarrow notResp(X)\}$
- $\mathcal{N} = \{\forall X\ resp(X) \wedge notResp(X) \rightarrow \bot\}$

The facts $incrim(e1, alice)$ *and* $absolv(e2, alice)$ *are entailed in some formalisms because there is no direct attack on them, however other formalisms consider that these facts are not entailed because they lead to a conflict if strict rules are applied.*

Expressiveness. Reasoning tools can be classified w.r.t. the expressiveness of their underlying language:

1. *Rules with Existential Variables:* This logical fragment is useful in application such as Ontology Based Data Access (OBDA) [Lenzerini, 2002] and Semantic Web [Cali et al., 2010b]. Detecting support for existential rules is tricky since most defeasible reasoning tools omit *quantifiers* which might lead to unwanted results. Variables appearing in the head of rules are sometimes considered existential variables, for example the rule $p(X) \rightarrow q(X, Y)$ can be either interpreted as $\forall X p(X) \rightarrow \exists Y q(X, Y)$ or $\forall X, Y p(X) \rightarrow q(X, Y)$. Example 3.8 shows how this affects reasoning.

   Example 3.8. *Consider the following situation where "jack" is a murderer and "john" is a victim. A murderer is a person who killed someone. This situation is described in* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$*:*

   - $\mathcal{F} = \{\top \Rightarrow murderer(jack),\ \top \Rightarrow victim(john)\}$.
   - $\mathcal{R} = \{murderer(X) \rightarrow killed(X, Y)\}$.

   If we run the query $Q = killed(jack, john)$ *(did jack kill john?) using a tool that does not take into account existential variables the answer would be "**true**" because it assumes that all known constants (persons) are killed by all murderers (i.e.* $\forall X, Y\ murderer(X) \wedge \top(Y) \rightarrow killed(X, Y)$*). In fact, it will also consider that* $killed(jack, jack)$ *is "true" (jack killed himself).*

   *However if we run the query Q using a tool that supports existential variables, the answer would be **"false"** since it is not possible to make the link between the generated* $Null$ *and the constant "john".*

2. *Cycles:* We consider two types of cycles: Support cycles (when the Graph of Rule Dependency is cyclic) and Attack cycles (cf. Section 2.2.2 on page 32). In presence of a *cyclic GRD* some inference mechanisms (such as SLD resolution [Apt and Van Emden, 1982])

might loop infinitely. *Cyclic conflict* happens when two chains of reasoning attack each other at different levels as shown in Example 3.9. Some logics (such as Defeasible Logics) would loop infinitely in such situations.

Example 3.9 (Attack Cycle). *Let* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$*:*

- $\mathcal{F} = \{\top \Rightarrow p(a), \top \Rightarrow q(a)\}$.
- $\mathcal{R} = \{\forall X\, p(X) \Rightarrow nq(X),\ \forall X\, q(X) \Rightarrow np(X)\}$
- $\mathcal{N} = \{\forall X\, p(X) \wedge np(X) \rightarrow \bot,\ \forall X\, q(X) \wedge nq(X) \rightarrow \bot\}$

Evaluating the query $Q = np(a)$ *would result in an infinite cycle if the arguments are evaluated on construction. Otherwise the answer to* Q *is* ***'false'****.*

3. *Rule Application Blocking:* Some situations require that rules are prevented from being applied. For example we might want to express that a fact should not be derived for some reason, and at the same time its conflicting fact is not necessary derived either. This solves some non-intuitive results of certain logics [Prakken, 2002]. Blocking rule applications can be achieved in two ways, either by giving rules labels and considering them as atoms, or by using defeater rules.

   Example 3.10. *Birds generally fly. We cannot say that birds with broken wings can fly or not. Let* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$*:*

   - $\mathcal{F} = \{\top \Rightarrow bird(tweety), \top \Rightarrow brokenWings(tweety)\}$.
   - $\mathcal{R} = \{r_1 : \forall X\, bird(X) \Rightarrow fly(X),$
     $r_2 : \forall X\, brokenWings(X) \rightsquigarrow notFly(X)\}$.
   - $\mathcal{N} = \{\forall X\, fly(X) \wedge notFly(X) \rightarrow \bot\}$

   Or, if we use rule labels:

   - $\mathcal{F} = \{\top \Rightarrow bird(tweety), \top \Rightarrow brokenWings(tweety)\}$.
   - $\mathcal{R} = \{r_1 : \forall X\, bird(X) \Rightarrow fly(X),$
     $r_2 : \forall X\, brokenWings(X) \Rightarrow \neg r_1\}$.

   The answer to the query $Q = fly(tweety)$ *is* ***"false"****.*

4. *Priority Relation:* Priorities between rules are used to resolve ambiguities. These priorities can be expressed in various manners: (1) either by using decimal numbers (between 0.0 and 1.0), or (2) a partial explicit priority relation using the labels of rules, or (3) even not rely on an explicit priority relation but rather to use an implicit one like generalized specificity [García and Simari, 2004].

5. *Type of Queries:* All defeasible tools support at least atomic boolean ground queries, but some of them allow for conjunctive ground and non-ground queries.

Basic Performance. Independently of the semantics or expressiveness of tools, basic performance relates to simple situations where there are long derivations, a large number of short derivations, or many derivations for the same fact. This indicator is not meant to identify the best performing tools, as most of the time performance must be sacrificed for semantics or expressiveness. However it can be used to decide what tool could be used when requirements and data at hand allow the use of two or more tools.

3.3.2 Benchmark Description

The benchmark provides indications on how defeasible reasoning tools are handling the previously described features (their support and subsequent scaling up).

Benchmarking Methodology. We build upon the *propositional* defeasible logic *performance-oriented* benchmark from [Maher et al., 2001] that generates various parameterized knowledge bases (also known as theories). We *adapt existing theories for the first order language* and *extend them with twelve additional theories* to account for features listed in the previous section. These theories serve two purposes: first, to test the tools' ability to handle the features (especially when these features are not explicitly stated in the companion paper of the tools). Second, to test their performance when faced with gradually complex situations requiring these features. For example: does the tool allow for team defeat? How does it perform when there are larger and larger instances requiring team defeat?

Before defining the benchmark, two key notions must be kept in mind:

1. To test the support for a semantics feature, this feature must be "isolated", meaning that the result of the query must only depend on the feature and no other external factor. That is why most theories use only defeasible rules (to avoid the disruptive effect of handling strict rules) and no preferences.

2. While negative results (i.e. situations where tools are not able to give the results required by a certain feature) are definitive, positive results (i.e. situations where tools do provide the intended results of a feature) on the other hand *do not prove the feature is fully supported.*

Benchmark Theories. Figures 3.19 and 3.20 give a representation of the benchmark theories, dashed lines represent conflict, $\rightarrow$, $\Rightarrow$ and $\rightsquigarrow$ are strict,

defeasible, and defeater rules respectively. For simplicity we use strong negation to represent negative constraints:

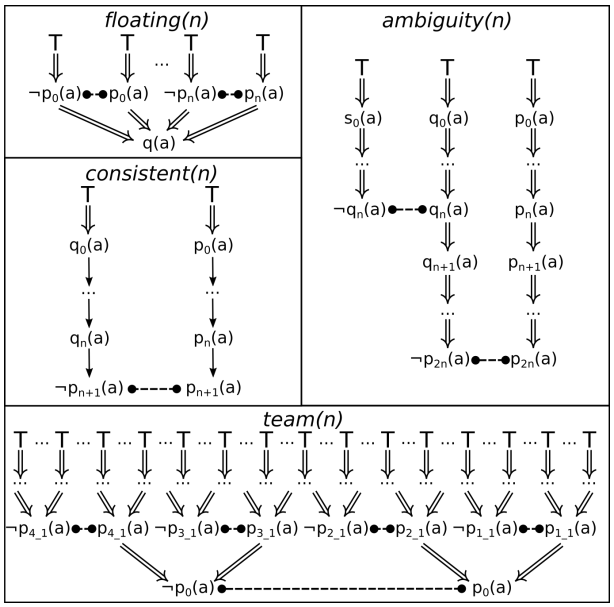

Figure 3.19: Representation of semantics theories

- **Ambiguity:** (denoted *ambiguity*(n)) contains a chain of n rules $s_{i-1}(a) \Rightarrow s_i(a)$, and two chains of $2n$ rules $q_{i-1}(a) \Rightarrow q_i(a)$ and $p_{i-1}(a) \Rightarrow p_i(a)$, plus the rules $s_n(a) \Rightarrow \neg q_n(a)$ and $q_{2n}(a) \Rightarrow \neg p_{2n}(a)$, and the defeasible facts $s_0(a)$, $q_0(a)$, $p_0(a)$. The query $Q = p_{2n}(a)$? is not entailed (false) in ambiguity propagating, but is entailed (true) in ambiguity blocking. The parameter n allows the scaling of the theory to longer and longer chains where conflicts appear further down the line.

- **Team Defeat:** [Maher et al., 2001] (denoted *team*(n)) each conclusion is supported by a team of two defeasible rules and attacked by another team of two defeasible rules. Preferences ensure that each attacking rule is inferior to one of the supporting rules. The antecedents of these rules are in turn supported and attacked by cascades (n levels) of teams of rules. The query $Q = p_0(a)$? is entailed if team defeat is allowed, otherwise, it is not entailed.

- **Floating Conclusions:** (denoted $floating(n)$) contains n couples of conflicting rules $\top \Rightarrow p_i(a)$ and $\top \Rightarrow \neg p_i(a)$, and n rules $p_i(a) \Rightarrow q(a)$. The query $Q = q(a)$? is entailed if floating conclusions are allowed. The parameter n allows the scaling by creating more and more couples of conflicting rules leading to the same conclusion.

- **Consistent Derivation:** (denoted *consistent*(n)) contains two defeasible facts $p_0(a)$, $q_0(a)$ and two chains of n strict rules of the form $p_{i-1}(a) \rightarrow p_i(a)$ and $q_{i-1}(a) \rightarrow q_i(a)$ that lead to a conflict down the line because of the rules $p_n(a) \rightarrow p_{n+1}(a)$ and $q_n(a) \rightarrow \neg p_{n+1}(a)$. The query $Q = p_0(a)$ is not entailed if atoms need to be consistent w.r.t. strict rules (indirectly consistent derivation), otherwise it is entailed.

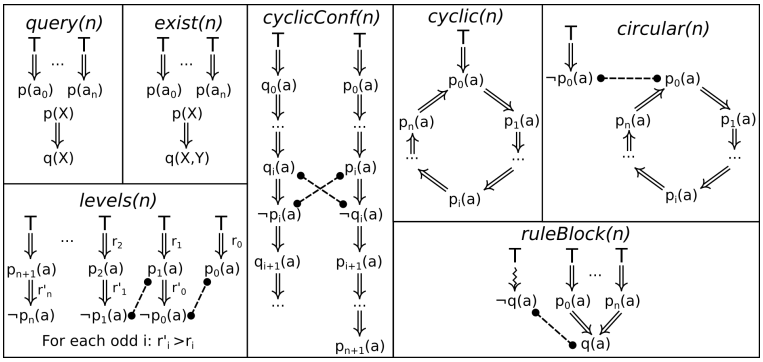

Figure 3.20: Representation of some expressiveness theories

- **Existential Rules:** (denoted *exist*(n)) composed of n rules $\top \Rightarrow p(a_i)$, and the rule without quantifiers $p(X) \Rightarrow q(X, Y)$. The query $q = q(a_0, a_n)$? is not entailed if existential rules are supported.

- **Skolem-FES:** (denoted *skolemFES*()) simply contains the sets of rules of concrete classes of Skolem-FES shown in Example 2.9 chained so that each set fires the next set of rules:

  1. $\top \Rightarrow p_{rr}(a, b)$, $\top \Rightarrow p_{rr}(b, c)$,
  2. $\forall X, Y, Z\, p_{rr}(X, Z) \wedge p_{rr}(Z, Y) \Rightarrow p_{rr}(X, Y) \wedge p_{wa}(X, Y)$ (this rule is range-restricted and prepares the atom $p_{wa}(a, b)$ for the next set of weakly acyclic rules),
  3. $\forall X, Y\, p_{wa}(X, Y) \Rightarrow \exists Z\, r_{wa}(Y, Z)$, $\forall X, Y\, r_{wa}(X, Y) \Rightarrow p_{wa}(Y, X) \wedge p_{ja}(X, Y)$ (these rules are weakly-acyclic and generates the atom $p_{ja}(a, b)$ for the next set of rules),
  4. $\forall X, Y\, p_{ja}(X, Y) \Rightarrow \exists Z\, r_{ja}(Y, Z)$, $\forall X, Y\, r_{ja}(X, Y) \wedge r_{ja}(Y, X) \Rightarrow p_{ja}(X, Y) \wedge q_{swa}(X)$ (these rules are jointly-acyclic and generate the atom $q_{swa}(a)$ for the next set of rules),
  5. $\forall X q_{swa}(X) \Rightarrow \exists Y\, p_{swa}(X, Y) \wedge p_{swa}(Y, X) \wedge p_{swa}(X, X)$, $\forall X\, p_{swa}(X, X) \Rightarrow r_{swa}(X)$, $\forall X\, r_{swa}(X) \Rightarrow \exists Y q_{swa}(X) \wedge p_{mfa}(X, Y)$ (these rules are super-weakly-acyclic and generates the atom $p_{mfa}(a, Null)$ for the next set of rules),

6. $\forall X, Y\, p_{mfa}(X,Y) \Rightarrow \exists Z, T\, q_{mfa}(Y,Z) \wedge p_{mfa}(Z,T) \wedge p(X)$ (this rule is model-faithful-acyclic and generates the atom $p(a)$).

Evaluating the query $Q = p(a)$? would result in an infinite loop if Skolem-FES existential rules are not supported.

- **FUS:** (denoted *linear*(n)) contains the rule $\top \Rightarrow p_0(a,b)$, and a chain of n *linear* rules of the form $\forall X, Y,\ p_{i-1}(X,Y) \Rightarrow \exists Z,\ p_{i-1}(X,Z) \wedge p_{i-1}(Z,Y) \wedge p_i(X,Y)$. The query $Q = p_n(a,b)$? would result in an infinite loop if FUS existential rules are not supported.

- **GBTS:** (denoted *chainFrG*(n)) contains the rule $\top \Rightarrow p_0(a,b) \wedge p_0(b,c)$, and a chain of n *frontier-guarded rules* of the form $\forall X, Y, Z,\ p_{i-1}(X,Y) \wedge p_{i-1}(Y,Z) \Rightarrow \exists W,\ p_{i-1}(X,W) \wedge p_{i-1}(W,Y) \wedge p_i(X,Y) \wedge p_i(Y,W)$. Evaluating the query $Q = p_n(a,b)$? would result in an infinite loop if GBTS existential rules are not supported.

- **Cyclic GRD:** (denoted *cyclicSupp*(n)) contains the rule $\top \Rightarrow p_1(a)$ and a cyclic chain of n defeasible rules of the form $p_i(X) \Rightarrow p_{i \bmod n}(X)$. Evaluating the query $Q = p_0(a)$? might result in an infinite loop due to the support cycle.

- **Circular Reasoning:** (denoted *circular*(n)) consists of a defeasible fact $\neg p_0(a)$ and a cycle of rules of the form $p_i(a) \Rightarrow p_{i \bmod n}(a)$. Evaluating the query $Q = \neg p_0(a)$ might result in an infinite loop due to circular reasoning.

- **Cyclic Conflict:** (denoted *cyclicConf*(n)) composed of the defeasible facts $p_0(a)$, $q_0(a)$ and n cyclic conflict of the form $p_i(a) \Rightarrow \neg q_i(a)$ and $q_i(a) \Rightarrow \neg p_i(a)$. Evaluating the query $Q = p_{n+1}(a)$? might loop infinitely due to the Attack cycle. The parameter n determines how many attack cycles are generated.

- **Rule Block:** (denoted *ruleBlock*(n)) contains n rules $\top \Rightarrow p_i(a)$ and $p_i(a) \Rightarrow q(a)$, and a single defeater rule $\top \rightsquigarrow \neg q(a)$ that blocks all other rules. The queries $Q_1 = q(a)$? and $Q_2 = \neg q(a)$? are not entailed. The parameter n determines how many rules have to be blocked. This theory tests performance with regards to handling rule applications blocking.

- **Preference:** [Maher et al., 2001] (denoted *levels*(n)) is a cascade of n disputed conclusions $p_i(a)$. For each i, there are rules $r_i : \top \Rightarrow p_i(a)$ and $r_i' : p_{i+1} \Rightarrow \neg p_i(a)$. For each *odd* $i \geq 1$ a priority asserts that $r_i' \succ r_i$. A final rule $\top \Rightarrow p_{n+1}(a)$ gives uncontested support for $p_{n+1}(a)$. The query $Q = p_0(a)$? is entailed if explicit preferences are respected. The parameter n allows the scaling to n conflicts with preferences in order to determine performance when faced with more and more preferences between rules.

- **Queries:** (denoted $query(n)$) composed of n rules $\top \Rightarrow p(a_i)$, and the rule $\forall X, p(X) \Rightarrow q(X)$. The query $Q = \exists X q(X)$? is entailed if existentially closed queries are supported (as there are n atoms of the form $q(a_i)$).

- **Chain Theory** [Maher et al., 2001] (denoted $chain(n)$) contains the rule $\top \Rightarrow p_0(a)$ and a chain of n defeasible rules of the form $p_{i-1}(X) \Rightarrow p_i(X)$. Evaluating the query $Q = p_n(a)$? would test performance when faced with a long chain of rules.

- **Tree Theory** [Maher et al., 2001] (denoted $tree(n, k)$) is a k-branching tree of depth n in which every atom occurs once and $p_0(a)$ is its root. The query $Q = p_0(a)$? would test performance w.r.t. a large number of short arguments.

- **Directed Acyclic Graph Theory** [Maher et al., 2001] (denoted $dag(n, k)$) is a k-branching tree of depth n in which every literal occurs k times. The query $Q = p_0(a)$? would test performance when faced with many arguments for the same atom.

The generated knowledge bases have to be adapted to the format of each tool. For example, rules can be transformed into an equivalent set of rules with atomic head, defeater rules can be transformed into rules for negated rule labels, negation can be represented with negative constraints, etc.

We conclude this section by an important remark: while the support for Skolem-FES can be fully known since all concrete Skolem-FES classes are represented, the support for other abstract classes FUS and GBTS is only achieved via one concrete decidable classes. Not satisfying a concrete class (e.g. linear) implies not satisfying the corresponding abstract class (e.g. FUS rules). However, the inverse is not necessarily true.

3.3.3 Running the Benchmark on Tools

To the best of our knowledge, defeasible reasoning tools discussed in Section 2.2.7 (*ASPIC**, *DeLP**, Flora-2, SPINdle, and DEFT) are the only still functioning, publicly available tools for first order defeasible reasoning as of the time of writing of this thesis.

A key notion to keep in mind is that we are not comparing the formalisms themselves, we are comparing the tools based on those formalisms. A formalism might allow for more than what the tool presents. That is one of the reasons that justify having a dedicated benchmark to better analyze and understand the implementation of the tools. Furthermore, some of the tools considered are prototypes, therefore they might not have been developed with performance in mind.

Tools can only be compared on theories where they compute the same results. That is why benchmark theories isolate the tested features. For example, most theories rely on defeasible rules to avoid the effects of strict rules

and were designed to avoid any effect of ambiguity blocking or propagating. Furthermore, team defeat and preferences theories use explicit preferences and different starting facts for each conflicting atoms to avoid the effects of implicit preferences, that is also why we use the explicit preference of DeLP rather than its generalized specificity.

All experiments presented in this section were performed on an Intel core i7 2.60GHz quad core Linux machine with 8GB of RAM and a Java heap of 2GB. To avoid random performance fluctuations each test is performed five times for each tool and we record the average *in CPU* execution time. The experiments are reproducible [2].

Tools Benchmark Results. Table 3.1 presents the time (in CPU seconds) required for each tool to answer the query according to the size and type of the query (the execution time includes the time needed for parsing the knowledge base and answering the query). ∞ denotes a stack overflow, *T.O.* denotes a timeout (set to 5 minutes), and *N.A.* indicates that a test theory is not applicable for that tool.

Results discussion. The main objective of the proposed benchmark is to help with tools analysis and classification according to the type of knowledge base and reasoning they can handle. To this end, from the results in Table 3.1, we can draw the following conclusions (summarised in Table 3.2):

- **Semantics:** The underlying theoretical and practical choices affect the semantics the tools can handle:

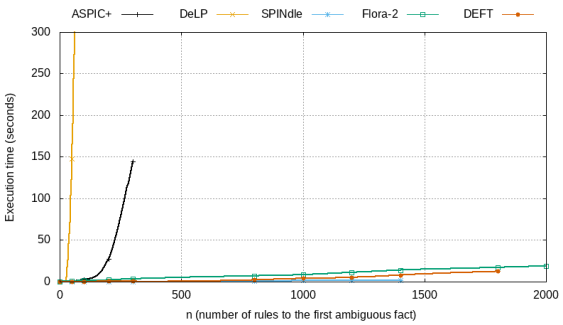

Figure 3.21: Response time for *ambiguity*(n)

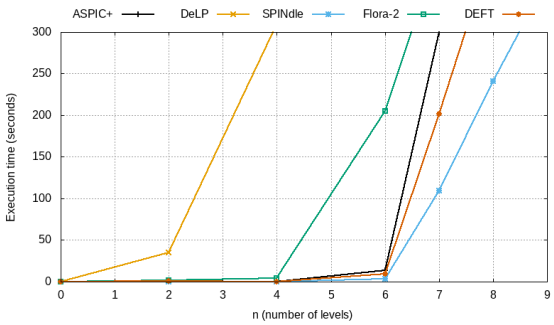

Figure 3.22: Response time for *team*(n)

  - *Ambiguity Handling:* *ASPIC**, DEFT and *DeLP** cannot express ambiguity blocking and correspond to ambiguity propagation due to the underlying formalisms. Flora corresponds only to ambiguity blocking

[2] `https://github.com/anoConf/Benchmark`

Theory		*ASPIC**	*DeLP**	SPINdle	Flora-2	DEFT
ambiguity(n)	n = 50	0.44 (false)	148.17 (false)	**0.11 (false)** | **0.09 (true)**	1.06 **(true)**	**0.11 (false)**
	n = 1800	∞	*T.O.*	∞	**17.237 (true)**	**12.503 (false)**
	n = 2000	∞	*T.O.*	∞	**18.942 (true)**	∞
team(n)	n = 4	**0.227 (false)**	301.19 (true)	0.289 (true)	4.358 (true)	**0.287 (true)**
	n = 7	*T.O.*	*T.O.*	**109.46 (true)**	*T.O.*	201.917(*true*)
floating(n)	n = 100	0.270 (false)	209.45 (false)	**0.332 (false)**	2.143 (false)	1.345 (false)
	n = 5000	198.861 (false)	*T.O.*	**150.144 (false)**	*T.O.*	203.18 (false)
consistent(n)	n = 1000	0.193 (true)	269.984 (false)	**0.703 (true)**	5.292 (true)	**2.969 (false)**
	n = 8000	8.321 (true)	*T.O.*	**8.854 (true)**	36.821 (true)	239.504(*false*)
exist(n)	n = 100	0.09 (true)	0.93	284.39 (true)	1.28 (true)	**0.01 (false)**
SkolemFES(n)	n = 100	*N.A.*	*N.A.*	*N.A.*	*N.A.*	**253.62 (true)**
linear(n)	n = 1	*N.A.*	*N.A.*	*N.A.*	*N.A.*	*T.O.*
chainFrG(n)	n = 1	*N.A.*	*N.A.*	*N.A.*	*N.A.*	*T.O.*
cyclicSupp(n)	n = 1000	∞	291.37 (true)	**0.35 (true)**	5.712 (true)	0.44 (true)
	n = 10000	∞	*T.O.*	**26.61 (true)**	51.72 (true)	288.71 (true)
circular(n)	n = 1000	∞	284.90 (true)	0.31 (true)	4.38 (true)	**0.04 (true)**
cyclicConf(n)	n = 5	0.627 (true)	55.89 (true)	0.903 (true)	0.922 (true)	**0.106 (true)**
	n = 1000	*T.O.*	*T.O.*	*T.O.*	*T.O.*	**79.525 (true)**
ruleBlock(n)	n = 500	19.23 (true)	*N.A.*	**1.41 (true)**	12.99 (true)	*N.A.*
levels(n)	n = 100	**0.20** (true)	4.61 (true)	0.33 (true)	5.17 (true)	0.81 (true)
query(n)	n = 100	*N.A.*	*N.A.*	*N.A.*	**1.25** (true)	*N.A.*
chain(n)	n = 600	108.05 (true)	99.46 (true)	**0.24 (true)**	2.35 (true)	0.33 (true)
	n = 10000	∞	*T.O.*	**16.04 (true)**	45.44 (true)	288.71 (true)
tree(n, 5)	n = 2	0.04 (true)	193.64 (true)	**0.03 (true)**	0.83 (true)	0.022 (true)
	n = 7	∞	*T.O.*	∞	211.94 (true)	**182.83 (true)**
dag(n, 10)	n = 1	∞	239.75 (true)	**7.51 (true)**	18.41 (true)	19.53 (true)
	n = 10	∞	*T.O.*	**60.82 (true)**	113.53 (true)	73.05 (true)

Table 3.1: Execution time in seconds (selected results). 'true' and 'false' indicate query entailment and are used to check support of the feature (the best time is shown in bold)

while SPINdle is the only tool that can handle both blocking and propagating. Performance wise (Figure 3.21), *DeLP** has a timeout at $n = 10$, *ASPIC** stops at $n = 300$, SPINdle at $n = 1400$, and DEFT $n = 1800$ due to stack overflow. Flora-2, DEFT, and SPINdle can scale to longer chain of rules for ambiguous facts with significantly low response time.

– *Team Defeat:* Most tools allow only for team defeat except *ASPIC** that does not allow for it. While Defeasible Logics can represent the presence and absence of team defeat, SPINdle and Flora-2 only implement the presence of team defeat. SPINdle has the best performance, followed by DEFT, *ASPIC**, Flora-2, then *DeLP**.

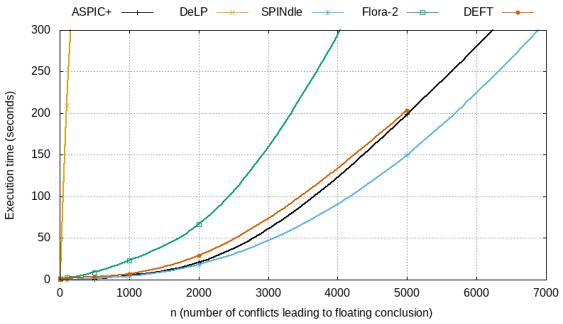

Figure 3.23: Response time for $floating(n)$

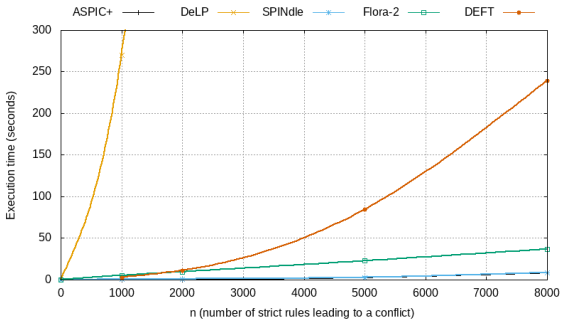

Figure 3.24: Response time for $consistent(n)$

– *Floating Conclusions:* None of the considered tools support floating conclusions due to their underlying formalisms.
– *Handling Strict Rules:* DEFT and *DeLP** use indirectly consistent derivations while *ASPIC**, Flora and SPINdle do not. This directly impacts performance results (as seen in Figure 3.24).

Feature		*ASPIC**	*DeLP**	SPINdle	Flora-2	DEFT
Ambiguity	Prop.	✓	✓	✓	-	✓
	Block.	-	-	✓	✓	-
Team Defeat	TD	-	✓	✓	✓	✓
	noTD	✓	-	-	-	-
Floating Conclusions	FC	-	-	-	-	-
	noFC	✓	✓	✓	✓	✓
Consistent Derivation	Direct	-	✓	-	-	✓
	Indirect	✓	-	✓	✓	-
Existential Rules	S-FES	-	-	-	-	✓
	FUS	-	-	-	-	-
	GBTS	-	-	-	-	-
Cycles	Support	-	✓	✓	✓	✓
	Attack	✓	✓	✓	✓	✓
Rule Block		✓	-	✓	✓	-
Preference	$\succ$	-	✓	✓	✓	✓
	$\mathbb{R}$	✓	-	-	-	-
Non-gound Queries		-	-	-	✓	-

Table 3.2: Classification results (✓indicates the tool supports the feature).

- **Expressiveness:** The choice of the inference mechanism affects the expressiveness the tool can handle.
  - *Existential rules:* *ASPIC**, *DeLP**, SPINdle, and Flora were not designed to account for existential rules. As supported by the results of the Existential theory, FES, FUS and GBTS are not applicable in their context. DEFT can handle existential rules in general, and SkolemFES rules in particular (due to its use of forward chaining), however it loops infinitely in FUS and GBTS fragments.
  - *Cycles:* DEFT, *DeLP**, and Flora can handle cyclic GRDs and circular reasoning (*support cycle*) contrary to ASPIC+. This is due to the fact that *ASPIC** relies on SLD resolution (which loops infinitely in presence of cycles in the GRD), while DEFT uses a chase mechanism (which is guaranteed to stop when no existential rule is used). *DeLP**, SPINdle and Flora rely on resolution with a grounding phase and cycle checks. All considered tools can handle cyclic conflicts (*attack cycles*). However, the attack cycle checks are not needed for DEFT since arguments are evaluated after construction, that is why it outperforms other tools (e.g. $n = 1000$ in Figure 3.26).

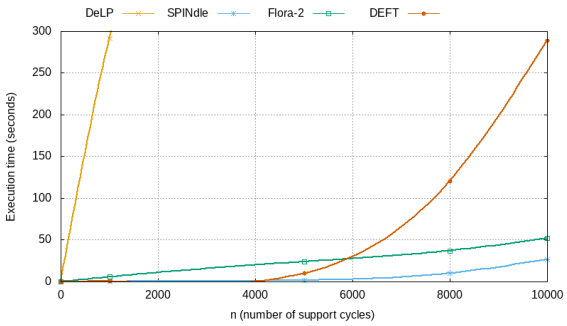

Figure 3.25: Response time for $cyclicSupp(n)$

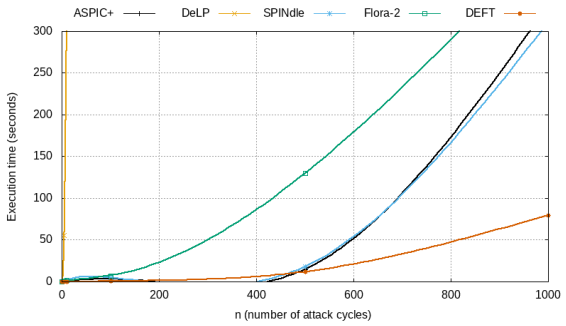

Figure 3.26: Response time for $cyclicConf(n)$

  - *Rule Application Block:* *ASPIC** uses negated labels of rules to block their application, SPINdle uses defeater rules, while Flora uses the predicate '$\backslash cancel(label)$'. DEFT and *DeLP** have no support for such feature. As seen in Table 3.1, SPINdle has the best performance followed by Flora-2 and *ASPIC**.
  - *Preference between rules:* *ASPIC** uses decimal values to express priority on rules (this priority relation is total and might lead to unwanted behavior as it is hard to express incomparability between rules). *DeLP**, SPINdle, Flora, and DEFT use a partial priority relation based on labeled rules.
  - *Non-ground queries:* Only Flora supports non ground queries.

- **Performance:** In case of a tie in expressiveness or semantics, one can use performance to make an informed choice on the tool to be used. From Table 3.1 we can see that each tool makes trade-offs between performance and expressiveness. In general, SPINdle has the best performance compared to other tools, followed by DEFT and Flora-2. *ASPIC** is as fast as SPINdle on small knowledge bases but it does not scale well (cf. Figure 3.27). DEFT has the best performance when there are attack cycles as shown in Figure 3.26. These differences in performance are due to three main factors. First, *grounding phase is costly.* DEFT, for instance, achieves its performance results thanks to its forward chaining algorithms that ground rules on the fly, contrary to the other tools. Second, *handling cycles is costly. ASPIC** is faster than *DeLP** for example because the latter relies on cycle checks to avoid infinite loops. DEFT does not need to check loops contrary to all other tools. Third, *expressiveness is costly*, DEFT and *DeLP** has to perform consistency checks using strict rule. Flora also provides very powerful syntactic features (dynamic rule labels, higher order syntax, etc.) which might affect its performance. Overall, our tool *DEFT has satisfactory performance* given the expressiveness (existential rules) and semantics it allows for (e.g. indirectly consistent derivations).

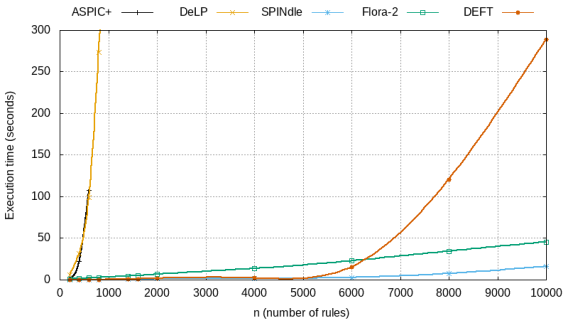

Figure 3.27: Response time for $chain(n)$

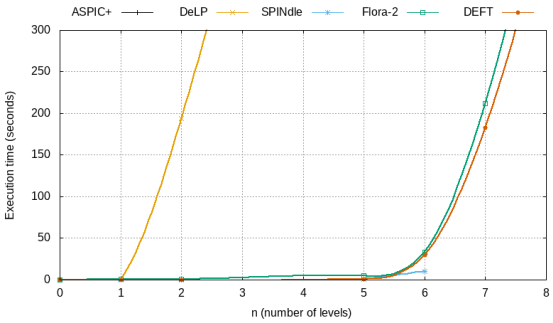

Figure 3.28: Response time for $tree(n, 5)$

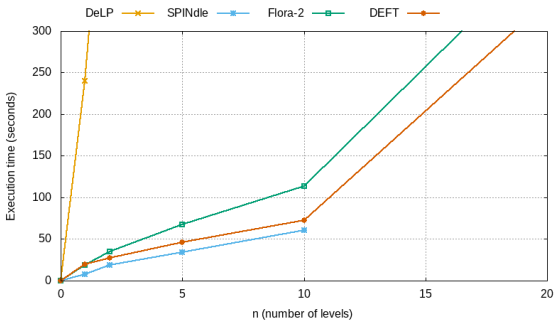

Figure 3.29: Response time for $dag(n, 10)$

Practical Example. To conclude this section, let us show how a data engineer could make practical use of our benchmark in order to chose the appropriate defeasible reasoning tool. We consider the following example:

Example 3.11. *Consider the following decision scenario of an emergency response team that wants to determine if a person victim of an accident is an organ donor. A person being hurt in an accident is considered a victim. Legally any victim is assumed not to be an organ donor. A person that gives her consent is considered an organ donor. A person in a critical condition generally cannot give her consent. The legal tutor of a person can give his consent for her being an organ donor. The following KB describes this use case.* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *where:*

- $\mathcal{F} = \{\top \Rightarrow \mathit{hurt}(\mathit{john})\}$.
- $\mathcal{R} = \{r_1 : \forall X\ \mathit{hurt}(X) \Rightarrow \mathit{victim}(X),$
  $r_2 : \forall X,\ \mathit{victim}(X) \Rightarrow \mathit{notOrganDonor}(X),$
  $r_3 : \forall X,\ \mathit{consentFor}(X, X) \Rightarrow \mathit{organDonor}(X),$
  $r_3 : \forall X, Y,\ \mathit{legalTutor}(X, Y) \wedge \mathit{consentFor}(X, Y) \Rightarrow \mathit{organDonor}(Y),$
  $r_3 : \forall X,\ \mathit{critical}(X) \Rightarrow \mathit{notConsentFor}(X, X)\ \}$.
- $\mathcal{N} = \{\forall X \mathit{organDonor}(X) \wedge \mathit{notOrganDonor}(X) \rightarrow \bot, \forall X, Y\ \mathit{consentFor}(X, Y) \wedge \mathit{notConsentFor}(X, Y) \rightarrow \bot\}$

This knowledge base does not use existential rules and is acyclic. Therefore, given the results of the benchmark, all considered tools can be applied. If the data engineer wants to use ambiguity blocking with team defeat then she can either use SPINdle or Flora-2 (SPINdle is in this case recommended given the performance results), if she does not want to allow for team defeat then ***no tool can be used****. If on the other hand the data engineer wants to use ambiguity propagation then she can either use DeLP* or DEFT if she needs team defeat (DEFT is in this case recommended given the performance results) or ASPIC* if she does not.*

Let us add the rule that a victim is probably someone who is hurt ($\forall X$, $\mathit{victim}(X) \Rightarrow \mathit{hurt}(X)$*). In this case the knowledge base becomes cyclic. Therefore ASPIC* cannot be used (cf. Table 3.2). Let us now add a new existential rule stating that if someone is an organ donor then somebody gave his consent (the person or her tutor i.e.* $\forall X\ \mathit{organDonor}(X) \rightarrow \exists Y\ \mathit{consentFor}(Y, X)$*). In this case, according to the Table 3.2 results,* ***only DEFT can be used****.*

3.4 Summary

In this chapter we demonstrated the significance of the derivation loss problem and showed how it prevents the direct application of defeasible reasoning techniques to existential rules. Derivation loss can occur in certain cases depending on the used chase and the order in which rules are applied. To

solve this problem, we presented the notion of Graph of Atom Dependency and showed how the chase choice impacts its construction and how it can be used to extract all derivations for a given fact. We then presented the first defeasible reasoning tool for existential rules using Dialectical Trees called DEFT that implements the algorithms of the Graph of Atom Dependency to extract all derivations.

In order to evaluate our tool with regard to the state of the art, we defined the first benchmark for first order logic defeasible reasoning tool classification with the aim to shed light on existing tools and their capabilities. Given the wide range of defeasible knowledge base expressiveness and the diversity of needs, selecting a relevant tool for a given application might be difficult. The benchmark provides a set of scalable knowledge bases that tests defeasible reasoning tools against a list of features inspired from different discussions of intuitions [Horty et al., 1987, Makinson and Schlechta, 1991, Prakken, 2002, Antoniou, 2006] and important differences in expressiveness. This benchmark however does not include *implicit priority relations*. Nevertheless most of these implicit priorities can be represented using explicit priorities [Prakken, 2002] (a list of examples regarding implicit priority relations is discussed in [Vreeswijk, 1995]).

Beside showing that DEFT has more than satisfactory performance, the benchmark gives a clearer view of what it is possible to handle with the existing tools and provides insights about current gaps in the state of the art and the limitation of our tool DEFT. For instance, we can observe in Table 3.2 that some features such as ambiguity blocking with or without team defeat for existential rules is not supported by any tool. In order to expand the usability and appeal of Defeasible Reasoning to existential rules, *those gaps need to be filled*. However, rather than creating a tool that implements different defeasible reasoning techniques for each desired feature, in the next chapter we present a new formalism that is able to represent most variants of defeasible reasoning in a single combinatorial structure.

Chapter 3 in a Nutshell

- *Defeasible Reasoning techniques cannot be directly applied to the existential rule language $\mathcal{L}_{\forall\exists}$ due to the derivation loss problem (some rule applications might be removed by the chase derivation reducer).*
- *Derivation loss can occur in certain cases depending on the used chase and the order in which rules are applied (cf. Propositions 3.1, 3.2, 3.3, and 3.4).*
- *To solve this problem, we defined the notion of Graph of Atom Dependency and showed how its construction is affected by the chase and how it can be used to extract all derivations.*
- *We presented the first defeasible reasoning tool for existential rules based on Dialectical Trees (called DEFT) that relies on the Graph of Atom Dependency to extract all derivations.*
- *We defined the first benchmark for first order logic defeasible reasoning tools analysis and classification. Beside showing that DEFT has satisfactory performance, this benchmark provides a clear view of what existing tools for defeasible reasoning allow for, what is the best tool to use depending on the data and reasoning requirements at hand, and what are the current gaps that are not covered yet by any tool.*

4

Statement Graph and Defeasible Logics

Given the variety of defeasible reasoning intuitions and the different degrees of expressiveness a defeasible reasoning tool can have (as described in the previous chapter Table 3.2), extending each defeasible reasoning technique to existential rules seems tedious. A better approach would be to have a unifying formalism that takes into account the specificities of existential rules and can capture most features discussed in Table 3.2. To this end, in this chapter we define a new formalism called "Statement Graph" and show how it can capture defeasible reasoning features via flexible labeling functions. The chapter is organized as follows: we start by defining Statement Graphs using the propositional language $\mathcal{L}_p$ in order to model Defeasible Logics, then we extend it to existential rule language $\mathcal{L}_{\forall\exists}$ and provide the first tool that can handle most discussed features of defeasible reasoning for existential rules.

Research Questions in this Chapter

- *How can we define a formalism that can capture most features of defeasible reasoning discussed in Table 3.2 while taking into account the logical specificities of existential rules?*
- *Can we provide a performant tool based on this formalism that provides defeasible reasoning with existential rules covering most gaps in the literature identified in Table 3.2?*

4.1 Propositional Statement Graph

A Statement Graph (SG) is a representation of the reasoning process happening inside a knowledge base, it can be seen as an updated Inheritance Net [Horty et al., 1990] or an instantiation of Abstract Dialectical Framework [Brewka and Woltran, 2010] with a custom labeling function. An SG is built using logical building blocks (called statements) that describe a situation (premises) and a rule that can be applied on that situation. To define statement graphs we use the propositional language $\mathcal{L}_p$ discussed in Section 2.2.3.1 and consider defeater rules.

Definition 4.1 (Statement). *A statement* $\mathbb{s}$ *is either:*

1. *A 'query statement'* $(Q \rightarrow \emptyset)$ *where* Q *is a query.*
2. *The 'Top statement'* $(\emptyset \rightarrow \top)$.
3. *A 'rule application statement'* $(\mathit{Body}(r) \Rrightarrow \mathit{Head}(r))$ *where* r *is a rule and* $\Rrightarrow \in \{\rightarrow, \Rightarrow, \rightsquigarrow\}$ *if* r *is strict, defeasible, or defeater respectively.*

Given a statement $\mathbb{s} = (\Phi \Rrightarrow \psi)$ *we denote by the premises by* $\mathit{Premise}(\mathbb{s}) = \Phi$ *and the conclusion by* $\mathit{Conc}(\mathbb{s}) = \psi$. *We denote the rule of a rule application statement* $\mathbb{s}$ *by* $\mathit{Rule}(\mathbb{s})$.

Statements can *attack* or *support* each other. Intuitively, a statement $\mathbb{s}_1$ supports another statement $\mathbb{s}_2$ if the conclusion of $\mathbb{s}_1$ is used by the rule application in $\mathbb{s}_2$. Furthermore, a statement $\mathbb{s}_1$ can attack another statement $\mathbb{s}_2$ in two possible ways: either the conclusion of $\mathbb{s}_1$ conflicts with a premise in $\mathbb{s}_2$ (we say that $\mathbb{s}_1$ *undercuts* $\mathbb{s}_2$), or the rule in $\mathbb{s}_1$ is a defeater rule with a conclusion that conflicts with the conclusion of $\mathbb{s}_2$ (we say that $\mathbb{s}_1$ *attacks the rule application of* $\mathbb{s}_2$).

Definition 4.2 (Statements Attack and Support). *Given two statements* $\mathbb{s}_1$ *and* $\mathbb{s}_2$*:*

- $\mathbb{s}_1$ *supports* $\mathbb{s}_2$ *iff* $\mathit{Conc}(\mathbb{s}_1) \neq \emptyset$, $\mathit{Conc}(\mathbb{s}_1) \in \mathit{Premise}(\mathbb{s}_2)$, *and* $\mathit{Rule}(\mathbb{s}_1) \notin \mathcal{R}_{\rightsquigarrow}$. *(we say that* $\mathbb{s}_1$ *supports* $\mathbb{s}_2$ *on* f*).*
- $\mathbb{s}_1$ *attacks* $\mathbb{s}_2$ *iff:*
  1. *Either* $\exists f \in \mathit{Premise}(\mathbb{s}_2)$ *s.t.* f *and* $\mathit{Conc}(\mathbb{s}_1)$ *are in conflict, and* $\mathit{Rule}(\mathbb{s}_1) \notin \mathcal{R}_{\rightsquigarrow}$. *(we say that* $\mathbb{s}_1$ *undercuts* $\mathbb{s}_2$ *on* f*).*
  2. *Or* $\mathit{Rule}(\mathbb{s}_1) \in \mathcal{R}_{\rightsquigarrow}$ *and* $\mathit{Conc}(\mathbb{s}_1)$ *and* $\mathit{Conc}(\mathbb{s}_2)$ *are in conflict (we say that* $\mathbb{s}_1$ *attacks the rule application of* $\mathbb{s}_2$*).*

Please note that statements with defeater rules cannot support other statements. Furthermore, we do not consider "rebut" attacks (i.e. when the conclusion of a statement is in conflict with the conclusion of another

statement) except for statements with defeater rules as a way to express rule application attack. To better understand the notions of support and attack between statements consider the following Example 4.1.

Example 4.1 (Undercut and Rule Application Attack). *Consider the knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *describing a penguin with a broken wing expressed in the propositional language* $\mathcal{L}_p$*:*

- $\mathcal{F} = \{\top \rightarrow penguin,\ \top \rightarrow brokenWings\}$
- $\mathcal{R} = \{penguin \rightarrow \neg fly,\ penguin \rightarrow bird,\ bird \Rightarrow fly,\ brokenWings \rightsquigarrow \neg fly\}$

For example, the query statement $(fly \rightarrow \emptyset)$ *is supported by the statement* $(bird \Rightarrow fly)$ *and is undercut (attacked) by the statement* $(penguin \rightarrow \neg fly)$*. The statement* $(brokenWings \rightsquigarrow \neg fly)$ *that relies on a defeater rule, attacks the rule application of* $(bird \rightarrow fly)$*.*

Statements are generated from a knowledge base, they can be structured in a graph according to their support and attack relations.

Definition 4.3 (Statement Graph). *A Statement Graph of the knowledge base* $\mathcal{KB}$ *is a directed graph* $\mathbb{SG}_{\mathcal{KB}} = (\mathcal{V}, \mathcal{E}_S, \mathcal{E}_A)$*:*

- $\mathcal{V}$ *is the set of statements generated from* $\mathcal{KB}$*.*
- $\mathcal{E}_S \subseteq \mathcal{V} \times \mathcal{V}$ *is the set of support edges. There is a support edge* $e = (\mathbb{s}_1, \mathbb{s}_2) \in \mathcal{E}_S$ *iff* $\mathbb{s}_1$ *supports* $\mathbb{s}_2$*.*
- $\mathcal{E}_A \subseteq \mathcal{V} \times \mathcal{V}$ *is the set of attack edges. There is an attack edge* $e = (\mathbb{s}_1, \mathbb{s}_2) \in \mathcal{E}_A$ *iff the statement* $\mathbb{s}_1$ *attacks* $\mathbb{s}_2$*.*

For an edge $e = (\mathbb{s}_1, \mathbb{s}_2)$*, we denote* $\mathbb{s}_1$ *by* $Source(e)$ *and* $\mathbb{s}_2$ *by* $Target(e)$*. For a statement* $\mathbb{s}$ *we denote its incoming attack edges by* $\mathcal{E}_A^-(\mathbb{s})=\{e \in \mathcal{E}_A | Target(e) = \mathbb{s}\}$ *and its incoming support edges by* $\mathcal{E}_S^-(\mathbb{s})=\{e \in \mathcal{E}_S | Target(e) = \mathbb{s}\}$*. We also denote its outgoing attack edges by* $\mathcal{E}_A^+(\mathbb{s})=\{e \in \mathcal{E}_A | Source(e) = \mathbb{s}\}$ *and outgoing support edges by* $\mathcal{E}_S^+(\mathbb{s})=\{e \in \mathcal{E}_S | Source(e)=\mathbb{s}\}$*.*

We say that an edge e ***is superior to*** *another edge* e' *and that* e' *is inferior to* e *iff* $Rule(Source(e)) \succ Rule(Source(e'))$*.*

A Statement Graph is constructed by generating all possible fact and rule application statements in a knowledge base along with their attack and support edges. Query statements are added when evaluating queries, as shown in the following Example 4.2.

Example 4.2 (Statement Graph of Example 4.1). *Consider the knowledge base in the previous Example 4.1, its Statement Graph* $\mathbb{SG}_{\mathcal{KB}} = (\mathcal{V}, \mathcal{E}_A, \mathcal{E}_S)$ *is shown in the following Figure 4.1 along with the query statement for* fly *and* $\neg fly$ *(support edges are depicted by dashed arrows, the top statement and the starting set of fact rules are colored gray).*

- $\mathcal{V} = \{\mathbb{s}_1 = (\emptyset \rightarrow \top)$, $\mathbb{s}_2 = (\top \rightarrow penguin)$, $\mathbb{s}_3 = (\top \rightarrow brokenWings)$, $\mathbb{s}_4 = (penguin \rightarrow \neg fly)$, $\mathbb{s}_5 = (penguin \rightarrow bird)$, $\mathbb{s}_6 = (bird \Rightarrow fly)$, $\mathbb{s}_7 = (brokenWings \rightsquigarrow \neg fly)$, $\mathbb{s}_8 = (fly \rightarrow \emptyset)$, $\mathbb{s}_9 = (\neg fly \rightarrow \emptyset)\}$
- $\mathcal{E}_S = \{e_1 = (\mathbb{s}_1, \mathbb{s}_2)$, $e_2 = (\mathbb{s}_1, \mathbb{s}_3)$, $e_3 = (\mathbb{s}_2, \mathbb{s}_4)$, $e_4 = (\mathbb{s}_2, \mathbb{s}_5)$, $e_5 = (\mathbb{s}_3, \mathbb{s}_7)$, $e_6 = (\mathbb{s}_4, \mathbb{s}_9)$, $e_7 = (\mathbb{s}_5, \mathbb{s}_6)$, $e_8 = (\mathbb{s}_6, \mathbb{s}_8)\}$
- $\mathcal{E}_A = \{e_9 = (\mathbb{s}_4, \mathbb{s}_8)$ $e_{10} = (\mathbb{s}_6, \mathbb{s}_9)$, $e_{11} = (\mathbb{s}_7, \mathbb{s}_6)\}$

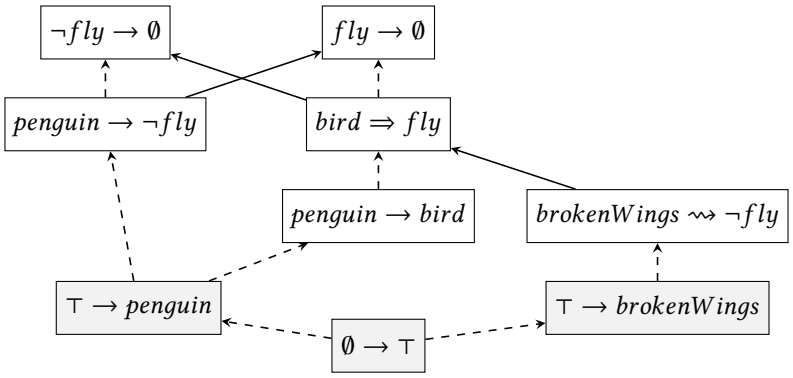

Figure 4.1: SG generated from $\mathcal{KB}$ in Example 2.15.

An SG provides statements and edges with a label using a *labeling function* that starts from the Top statement and propagates labels to the other statements. Query answering can then be determined based on the label of the query statement.

Definition 4.4 (Labeling Function). *A labeling function applied to a statement graph is a function* $Lbl : \mathcal{V} \cup \mathcal{E}_A \cup \mathcal{E}_S \rightarrow Label$ *that takes as input a statement* $\mathbb{s} \in \mathcal{V}$ *or an edge* $e \in \mathcal{E}_A \cup \mathcal{E}_S$ *and returns a label in* $Label = \{INstr, INdef, OUTstr, OUTdef, AMBIG, UNSUP\}$.

The intuition behind these labels is as follows:

- **INstr** indicates that the statement is accepted and its rule can be strictly applied based on strictly accepted premises.
- **INdef** indicates that the statement is accepted and its rule can be defeasibly applied based on strictly or defeasibly accepted premises.
- **OUTstr** and **OUTdef** indicate that the statement is not accepted because its rule or premises have been strictly or defeasibly defeated respectively.
- **AMBIG** indicates that the statement's rule or premises are challenged and the superiority relation cannot be used to determine if it is accepted or not.

- **UNSUP** indicates that at least one of the statement's premises is not supported by undefeated facts.

A statement is given a label based on its incoming edges and their labels. The notion of *complete support* describes the situation where a statement has a support edge for each one of its premises.

Definition 4.5 (Complete Support). *A complete support for a statement* $\mathbb{s}$ *is a set of support edges denoted* $\mathcal{E}_{CS}^{\mathbb{s}}$ *such that:*

- $\forall f \in Premise(\mathbb{s})$, $\exists e \in \mathcal{E}_{CS}^{\mathbb{s}}$ *such that* $Source(e)$ *supports* $\mathbb{s}$ *on* f.
- $\nexists S'$ *such that* $S' \subset \mathcal{E}_{CS}^{\mathbb{s}}$ *and* S' *is a complete support for* $\mathbb{s}$ *(minimality w.r.t. set inclusion).*

Example 4.3 (Complete Support). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$*: A person will stay home if it is cold with a bad weather. It is "bad weather" if it will rain or be cloudy. The weather forecast indicates that today will have low temperature and will be cloudy with a significant chance of rain.*

- $\mathcal{F} = \{\top \Rightarrow lowTemp,\ \top \Rightarrow cloudy,\ \top \Rightarrow rain,\}$
- $\mathcal{R} = \{cold \wedge badWheather \Rightarrow stayHome,\ lowTemp \rightarrow cold,\ cloudy \Rightarrow badWheather,\ rain \Rightarrow badWheather\}$

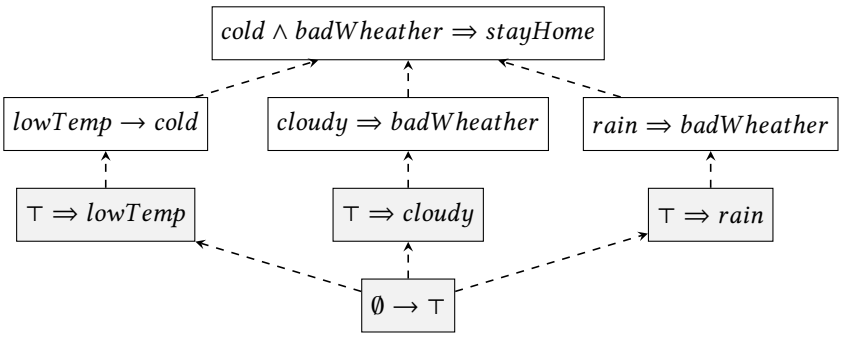

Figure 4.2: SG generated from $\mathcal{KB}$ in Example 4.3.

The Statement Graph $\mathbb{SG}_{\mathcal{KB}} = (\mathcal{V}, \mathcal{E}_A, \mathcal{E}_S)$ *of* $\mathcal{KB}$ *is depicted in Figure 4.2.*

- $\mathcal{V} = \{\mathbb{s}_1 = (\emptyset \Rightarrow \top),\ \mathbb{s}_2 = (\top \Rightarrow lowTemp),\ \mathbb{s}_3 = (\top \Rightarrow cloudy),\ \mathbb{s}_4 = (\top \Rightarrow rain),\ \mathbb{s}_5 = (lowTemp \rightarrow cold),\ \mathbb{s}_6 = (cloudy \Rightarrow badWheather),\ \mathbb{s}_7 = (rain \Rightarrow badWheather),\ \mathbb{s}_8 = (cold \wedge badWheather \Rightarrow stayHome)\}$
- $\mathcal{E}_S = \{e_1 = (\mathbb{s}_1, \mathbb{s}_2),\ e_2 = (\mathbb{s}_1, \mathbb{s}_3),\ e_3 = (\mathbb{s}_1, \mathbb{s}_4),\ e_4 = (\mathbb{s}_2, \mathbb{s}_5),\ e_5 = (\mathbb{s}_3, \mathbb{s}_6),\ e_6 = (\mathbb{s}_4, \mathbb{s}_7),\ e_7 = (\mathbb{s}_5, \mathbb{s}_8),\ e_8 = (\mathbb{s}_6, \mathbb{s}_8),\ e_9 = (\mathbb{s}_7, \mathbb{s}_8)\}$ *and* $\mathcal{E}_A = \emptyset$.

Statement $\mathbb{s}_8$ *has two complete supports* $\mathcal{E}_{CS}^{\mathbb{s}_8} = \{e_7, e_8\}$ *and* $\mathcal{E}_{CS'}^{\mathbb{s}_8} = \{e_7, e_9\}$.

Given a complete support $\mathcal{E}^{\mathbb{s}}_{CS}$:

- $\mathcal{E}^{\mathbb{s}}_{CS}$ is called "**INstr complete support**" iff $\forall e \in \mathcal{E}^{\mathbb{s}}_{CS}$, $Lbl(e)$ = INstr.
- $\mathcal{E}^{\mathbb{s}}_{CS}$ is called "**INdef complete support**" iff it is not a INstr complete support, and $\forall e \in \mathcal{E}^{\mathbb{s}}_{CS}$, $Lbl(e) \in$ {INstr, INdef}.
- $\mathcal{E}^{\mathbb{s}}_{CS}$ is called "**AMBIG complete support**" iff it is not an INstr nor an INdef complete support, and $\forall e \in \mathcal{E}^{\mathbb{s}}_{CS}$, $Lbl(e) \in$ {INstr, INdef, AMBIG}.

4.2 Reasoning with Statement Graphs

Statement Graphs are flexible enough to represent most variants of defeasible reasoning depicted in Table 2.2 on page 63. This flexibility is due to the labeling function that evaluates all supports and attacks for a specific rule application step. We start by explaining how SGs capture basic defeasible reasoning with ambiguity blocking, team defeat, and without cycles.

4.2.1 Labeling for Ambiguity Blocking

The intuition behind ambiguity blocking is to "block" facts that rely on ambiguous premises from being used to contest other facts. Defeasible Logic with ambiguity blocking allows for team defeat by default. From SGs point of view ambiguity blocking means that all ambiguous attack edges are not taken into account while team defeat means that a statement survives as long as each attacking edge is defeated by one of its support edges. We use the labeling function '**BDL**' (Blocking Defeasible Logic) to obtain entailment results equivalent to Billington's defeasible logic [Billington, 1993] (i.e. defeasible reasoning with ambiguity blocking, team defeat and without cycles). BDL is defined as follows: edges are given the same label as their source statements (i.e. given an edge e, BDL(e) = BDL($Source(e)$). Given a statement $\mathbb{s}$, *if* $\mathbb{s}$ *is the Top statement then* $BDL(\mathbb{s}) = INstr$. Otherwise:

(a) $\boldsymbol{BDL}(\mathbb{s}) = \boldsymbol{INstr}$ *if* $\mathbb{s}$ *has an INstr complete support, has a strict rule* $Rule(\mathbb{s})$*, and* $\nexists \boldsymbol{e} \in \mathcal{E}^{-}_{A}(\mathbb{s})$ *s.t.* $BDL(\boldsymbol{e}) = INstr$.

A statement is labeled INstr (i.e. strictly accepted) iff it is the Top statement or if it has a complete strict support (i.e. there is a strict derivation for each of its premises), uses a strict rule, and is not attacked by a strict derivation.

(b) $\boldsymbol{BDL}(\mathbb{s}) = \boldsymbol{OUTstr}$ *iff* $\exists \boldsymbol{e} \in \mathcal{E}^{-}_{A}(\mathbb{s})$ *s.t.* $BDL(\boldsymbol{e}) = INstr$.

A statement is labeled OUTstr (i.e. strictly defeated) iff it is strictly attacked (i.e there is a strict derivation against its premises or its rule application).

(c) $\boldsymbol{BDL}(\mathbb{s}) = \boldsymbol{INdef}$ *iff* $BDL(\mathbb{s}) \notin \{INstr, OUTstr\}$ *and* $\mathbb{s}$ *has a INstr or INdef complete support and*

1. $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *that undercut* $\mathbb{s}$ *on a premise* $f \in \textit{Premise}(\mathbb{s})$ *s.t.* $BDL(\boldsymbol{e}) = INdef$, $\exists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $BDL(\boldsymbol{e_s}) = INstr$ *or* $(BDL(\boldsymbol{e_s}) = INdef$ *and* $\boldsymbol{e_s}$ *is superior to* $\boldsymbol{e})$.
2. *and* $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $BDL(\boldsymbol{e}) = INdef$ *and* $\boldsymbol{e}$ *attacks the rule application of* $\mathbb{s}$, $\textit{Rule}(\mathbb{s})$ *is either a strict rule or* $\textit{Rule}(\mathbb{s}) \succ \textit{Rule}(\textit{Source}(\boldsymbol{e}))$.

A statement is labeled INdef iff it is not strictly accepted nor strictly defeated and it has a strict or defeasibly accepted complete support (i.e. there is a strict or defeasibly accepted derivation for each of its premises) and (c.1.) for any defeasibly accepted attack it receives on a premise, it has a support edge for that premise that is either strictly accepted or is defeasibly accepted and superior to the attacking edge (this condition allows for team defeat since a support edge does not have to defeat all attacks by itself) and (c.2.) the statement rule is either a strict rule or is superior to any defeasibly applicable defeater rule attacking it.

(d) $\boldsymbol{BDL(\mathbb{s}) = OUTdef}$ *iff* $BDL(\mathbb{s}) \neq OUTstr$ *and* $\mathbb{s}$ *has an INstr or INdef complete support and*

1. *either* $\exists f \in \textit{Premise}(\mathbb{s})$ *where* $\nexists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $BDL(\boldsymbol{e_s}) = INstr$ *and* $\forall \boldsymbol{e'_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $BDL(\boldsymbol{e'_s}) \in \{INdef, AMBIG\}$, $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *attacking* $\mathbb{s}$ *on* f *s.t.* $BDL(\boldsymbol{e}) = INdef$ *and* $\boldsymbol{e}$ *is superior to* $\boldsymbol{e'_s}$.
2. *or* $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $BDL(\boldsymbol{e}) = INdef$ *attacking the rule application of* $\mathbb{s}$ *and* $\textit{Rule}(\mathbb{s})$ *is not a strict rule and* $\textit{Rule}(\textit{Source}(\boldsymbol{e})) \succ \textit{Rule}(\mathbb{s})$.

A statement is labeled OUTdef iff it is not strictly defeated and it has a strict or defeasibly accepted complete support and either (d.1.) one of its premises is not strictly supported and for all its defeasibly accepted or ambiguous support edges, there exists a defeasibly accepted attack edge that is superior to it (this condition allows for team defeat as an attack edge does not have to defeat all supports by itself). Or (d.2.) the statement's rule is not strict and is inferior to a defeasibly applicable defeater rule attacking it.

(e) $\boldsymbol{BDL(\mathbb{s}) = AMBIG}$ *if* $BDL(\mathbb{s}) \notin \{INstr, OUTstr, INdef, OUTdef\}$ *and* $\mathbb{s}$ *has an INstr, INdef, or AMBIG complete support*

A statement is labeled AMBIG if it is not strictly or defeasibly accepted or defeated and it has a complete support that is either strict, defeasible or ambiguous.

(f) $\boldsymbol{BDL(\mathbb{s}) = UNSUP}$ *if* $BDL(\mathbb{s}) \neq OUTstr$ *and* $\exists f \in \textit{Premise}(\mathbb{s})$ *s.t.* $\nexists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *where* $\textit{Conc}(\textit{Source}(\boldsymbol{e_s})) = f$ *and* $BDL(\boldsymbol{e_s}) \in \{INstr, INdef, AMBIG\}$.

A statement is labeled UNSUP iff it is not strictly defeated and it has a premise that is not supported by a strictly accepted, defeasibly accepted, or ambiguous edge.

Example 4.4 (BDL labeling function). *Consider the SG of Example 2.15. Applying BDL labeling function results in Figure 4.3. In particular:*

- *BDL*$(\top \rightarrow evidA)$ = *INstr because it has an INstr complete support with a strict rule.*
- *BDL*$(evidA \Rightarrow responsible)$ = *INdef because it has a complete INstr support and a defeasible rule that is not attacked by a defeater rule.*
- *BDL*$(responsible \Rightarrow guilty)$ = *AMBIG because it has a INdef complete support and is attacked by a defeasible accepted edge that is neither superior nor inferior to its support edge.*
- *BDL*$(guilty \rightarrow \emptyset)$ = *AMBIG because it has an AMBIG complete support and no INstr or INdef complete support.*
- *BDL*$(\neg guilty \rightarrow \emptyset)$ = *INdef because it has a INdef complete support and is not attacked by a strictly or defeasibly accepted edge.*

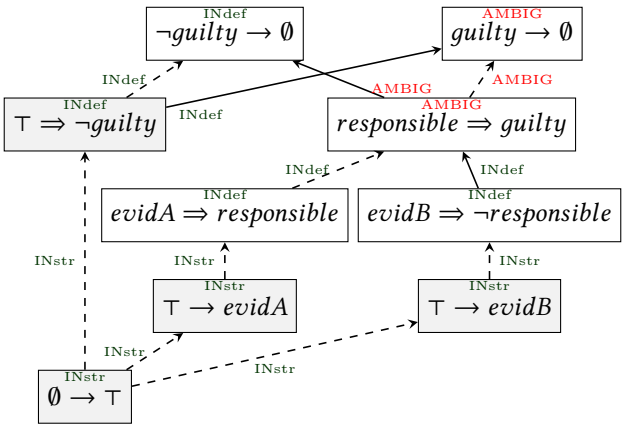

Figure 4.3: $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}$ of Example 2.15.

The labeling process of the BDL function can be expressed as a decision diagram shown in Figure 4.4. $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}$ denotes an SG that uses the BDL labeling function, and $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle \mathbb{s} \rangle$ denotes the label of a statement $\mathbb{s}$.

Lemma 4.1 (BDL is a function). *All statements in a knowledge base $\mathcal{KB}$ have exactly one label in $\mathbb{SG}^{BDL}_{\mathcal{KB}}$.*

Proof. From the definition of BDL (cf. Figure 4.4). □

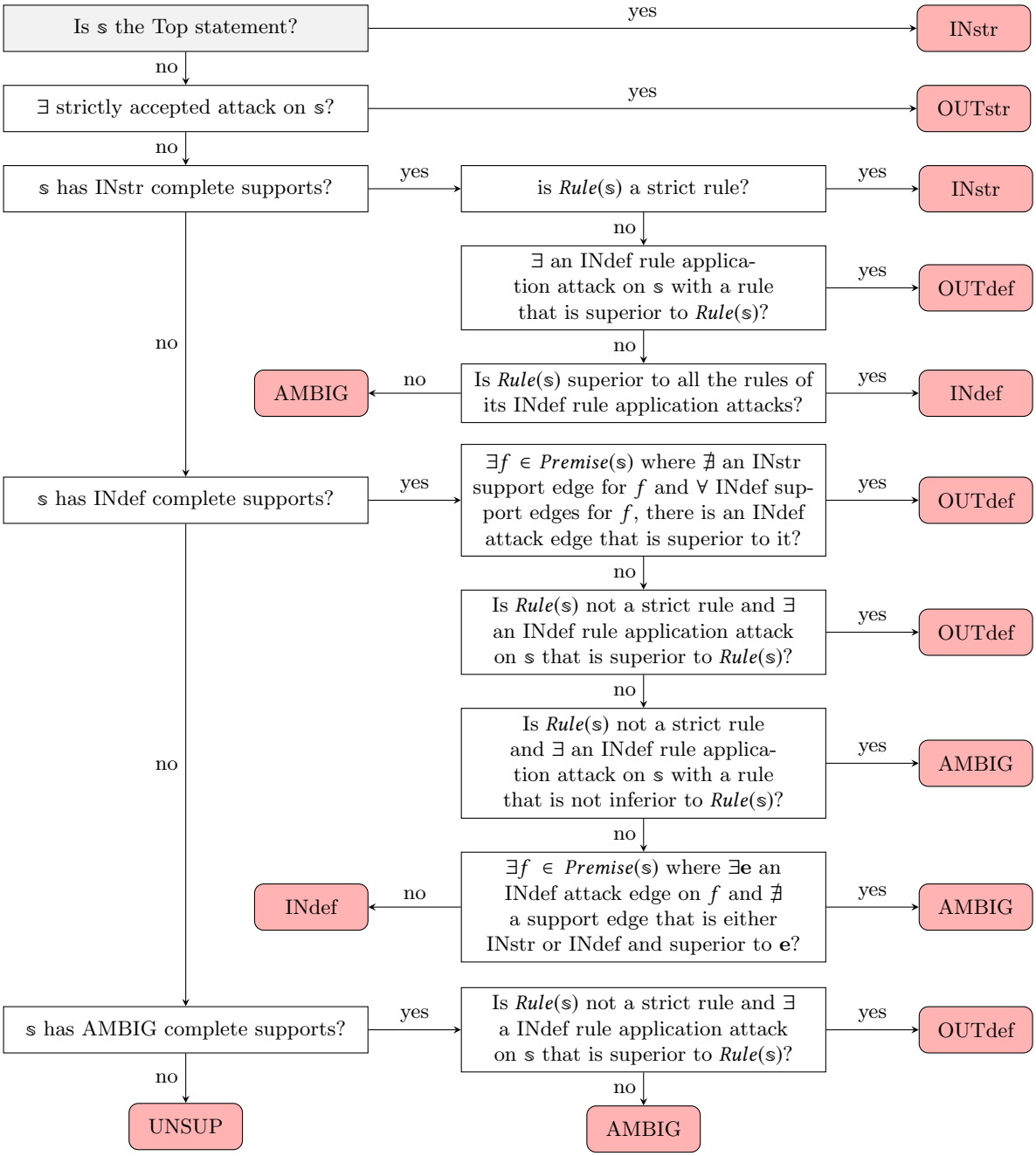

Figure 4.4: BDL function's decision diagram.

The equivalence between BDL and Defeasible Logic with ambiguity blocking and team defeat without attack or support cycles is described in the following Proposition 4.1.

Proposition 4.1. *Let f be a literal in a defeasible $\mathcal{KB}$ expressed in $\mathcal{L}_p$ that*

contains no attack or support cycles:

1. $\mathcal{KB} \vdash +\Delta f$ *iff* $\mathbb{SG}^{BDL}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = INstr$
2. $\mathcal{KB} \vdash -\Delta f$ *iff* $\mathbb{SG}^{BDL}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq INstr$
3. $\mathcal{KB} \vdash +\delta^{TD}_{block} f$ *iff* $\mathbb{SG}^{BDL}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{INstr, INdef\}$
4. $\mathcal{KB} \vdash -\delta^{TD}_{block} f$ *iff* $\mathbb{SG}^{BDL}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{OUTstr, OUTdef, AMBIG, UNSUP\}$

Proof. cf. Proof 4.1 in Section 7.2.2 on page v. □

4.2.2 Labeling for Ambiguity Propagating

The intuition behind ambiguity propagation is to reject a literal if there is a derivation attacking it which is not inferior (whether it relies on ambiguous literals or not). From an SG point of view, ambiguity propagating means that ambiguous attack edges are considered valid attacks that make the statement ambiguous if it cannot defend against them.

We use the labeling function '**PDL**' (Propagating Defeasible Logic) to obtain entailment results equivalent to defeasible reasoning with ambiguity propagating, team defeat and without cycles [Antoniou et al., 2000a]. PDL is defined the same as BDL except for the definition of the INdef label. Edges are given the same label as their source statements (i.e. given an edge $\boldsymbol{e}$, PDL($\boldsymbol{e}$) = PDL(*Source*($\boldsymbol{e}$))). Given a statement $\mathbb{s}$, *if* $\mathbb{s}$ *is the Top statement then BDL*($\mathbb{s}$) = *INstr*. Otherwise:

(a) $\boldsymbol{PDL}(\mathbb{s}) = \boldsymbol{INstr}$ *if* $\mathbb{s}$ *has an INstr complete support, has a strict rule* $Rule(\mathbb{s})$*, and* $\nexists \boldsymbol{e} \in \mathcal{E}^-_A(\mathbb{s})$ *s.t.* $PDL(\boldsymbol{e}) = INstr$.

(b) $\boldsymbol{PDL}(\mathbb{s}) = \boldsymbol{OUTstr}$ *iff* $\exists \boldsymbol{e} \in \mathcal{E}^-_A(\mathbb{s})$ *s.t.* $PDL(\boldsymbol{e}) = INstr$.

(d) $\boldsymbol{PDL}(\mathbb{s}) = \boldsymbol{OUTdef}$ *iff* $PDL(\mathbb{s}) \neq OUTstr$ *and* $\mathbb{s}$ *has an INstr or INdef complete support and*

1. *either* $\exists f \in Premise(\mathbb{s})$ *where* $\nexists \boldsymbol{e_s} \in \mathcal{E}^-_S(\mathbb{s})$ *for* f *s.t.* $PDL(\boldsymbol{e_s}) = INstr$ *and* $\forall \boldsymbol{e'_s} \in \mathcal{E}^-_S(\mathbb{s})$ *for* f *s.t.* $PDL(\boldsymbol{e'_s}) \in \{INdef, AMBIG\}$, $\exists \boldsymbol{e} \in \mathcal{E}^-_A(\mathbb{s})$ *attacking* $\mathbb{s}$ *on* f *s.t.* $PDL(\boldsymbol{e}) = INdef$ *and* $\boldsymbol{e}$ *is superior to* $\boldsymbol{e'_s}$.
2. *or* $\exists \boldsymbol{e} \in \mathcal{E}^-_A(\mathbb{s})$ *s.t.* $PDL(\boldsymbol{e}) = INdef$ *attacking the rule application of* $\mathbb{s}$ *and* $Rule(\mathbb{s})$ *is not a strict rule and* $Rule(Source(\boldsymbol{e})) \succ Rule(\mathbb{s})$.

(e) $\boldsymbol{PDL}(\mathbb{s}) = \boldsymbol{AMBIG}$ *if* $PDL(\mathbb{s}) \notin \{INstr, OUTstr, INdef, OUTdef\}$ *and* $\mathbb{s}$ *has an INstr, INdef, or AMBIG complete support*

(f) $\boldsymbol{PDL}(\mathbb{s}) = \boldsymbol{UNSUP}$ *if* $PDL(\mathbb{s}) \neq OUTstr$ *and* $\exists f \in Premise(\mathbb{s})$ *s.t.* $\nexists \boldsymbol{e_s} \in \mathcal{E}^-_S(\mathbb{s})$ *where* $PDL(\boldsymbol{e_s}) \in \{INstr, INdef, AMBIG\}$.

The only difference between ambiguity blocking (BDL) and ambiguity propagating (PDL) is that in the latter ambiguous attacks are taken into account and can make the statement ambiguous. This change only affects the definition of INdef labeling.

(c) ***PDL***$(\mathbb{s}) =$ ***INdef*** *iff* $PDL(\mathbb{s}) \notin \{INstr, OUTstr\}$ *and* $\mathbb{s}$ *has a INstr or INdef complete support and*

1. $\forall \boldsymbol{e} \in \mathcal{E}^{-}_{A}(\mathbb{s})$ *that undercut* $\mathbb{s}$ *on a premise* $f \in \textit{Premise}(\mathbb{s})$ *s.t.* $PDL(\boldsymbol{e}) \in \{INdef, AMBIG\}$, $\exists \boldsymbol{e_s} \in \mathcal{E}^{-}_{S}(\mathbb{s})$ *for* f *s.t.* $PDL(\boldsymbol{e_s}) = INstr$ *or* $(PDL(\boldsymbol{e_s}) = INdef$ *and* $\boldsymbol{e_s}$ *is superior to* $\boldsymbol{e})$.
2. *and* $\forall \boldsymbol{e} \in \mathcal{E}^{-}_{A}(\mathbb{s})$ *s.t.* $BDL(\boldsymbol{e}) \in \{INdef, AMBIG\}$ *and* $\boldsymbol{e}$ *attacks the rule application of* $\mathbb{s}$, $\textit{Rule}(\mathbb{s})$ *is either a strict rule or* $\textit{Rule}(\mathbb{s}) \succ \textit{Rule}(\textit{Source}(\boldsymbol{e}))$.

In BDL, an INdef statement has to defend itself against defeasibly accepted attacks; in PDL it also has to defend itself against ambiguous ones.

Example 4.5 (PDL labeling function). *Consider the SG of Example 2.15. Applying PDL labeling function results in Figure 4.5. In particular:* $PDL(\neg guilty \Rightarrow \emptyset) = AMBIG$ *because it has an INdef complete support and is attacked by an ambiguous edge.*

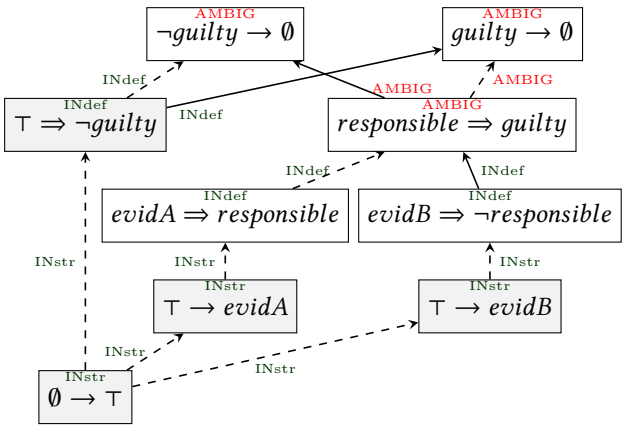

Figure 4.5: $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}$ of Example 2.15.

The equivalence between PDL and Defeasible Logic with ambiguity propagating and team defeat is shown in Proposition 4.2.

Proposition 4.2. *Let* f *be a literal in a defeasible* $\mathcal{KB}$ *expressed in* $\mathcal{L}_p$ *that contains no attack nor support cycles:*

1. $\mathcal{KB} \vdash +\Delta f$ *iff* $\mathbb{SG}^{PDL}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = INstr$

2. $\mathcal{KB} \vdash -\Delta f$ *iff* $\mathbb{SG}^{PDL}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq INstr$

3. $\mathcal{KB} \vdash +\delta^{TD}_{prop} f$ *iff* $\mathbb{SG}^{PDL}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{INstr, INdef\}$

4. $\mathcal{KB} \vdash -\delta^{TD}_{prop} f$ *iff* $\mathbb{SG}^{PDL}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{OUTstr, OUTdef, AMBIG, UNSUP\}$

Proof. cf. Proof 4.2 in Section 7.2.2 on page ix. □

4.2.3 Labeling without Team Defeat

The intuition behind removing team defeat is that for a literal to be accepted it has to have a chain of reasoning that defends itself, alone, against all attacks. From an SG's perspective, forbidding team defeat means that for a statement to be INdef it has to have a support edge that is superior to all attacks on the same premise. For a statement to be OUTdef it has to have an attack edge that is superior to all supports for the same premise, as shown in the following Example 4.6.

Example 4.6 (Labeling without Team Defeat). *Consider the SG of the knowledge base $\mathcal{KB}$ of Example 2.25 with the query statements for $(buy, \emptyset)$ and $(\neg buy, \emptyset)$. Applying a labeling that allows for team defeat would result in Figure 4.6. The statement $(buy \rightarrow \emptyset)$ is labeled INdef because for each INdef attack edge there is an INdef support edge that is superior. On the other hand, applying a labeling that does not allow for team defeat would result in Figure 4.7. The statement $(buy \rightarrow \emptyset)$ is labeled AMBIG because there is no INdef support edge that is superior to all INdef attacks.*

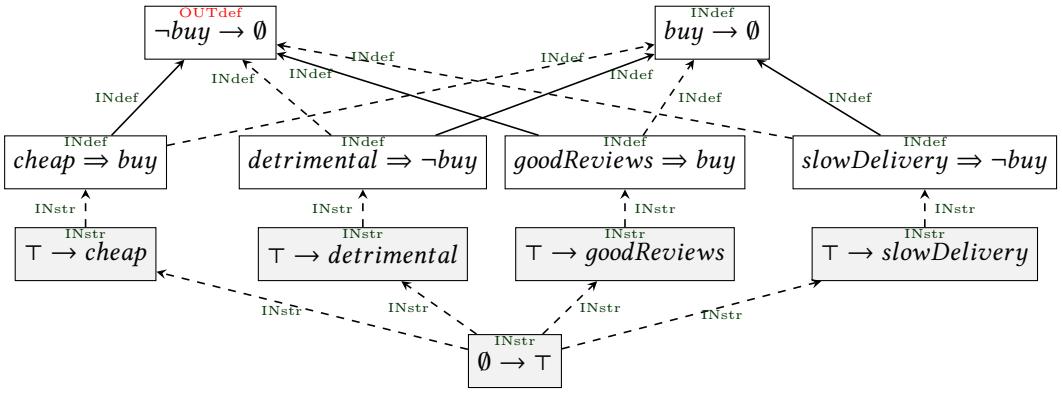

Figure 4.6: $\mathbb{SG}_{\mathcal{KB}}$ of Example 2.25 with a defeasible logic labeling that allows for team defeat.

We denote the labeling function for ambiguity blocking without team defeat by BDL_{noTD} which is almost the same as BDL except for INdef and OUTdef labels.

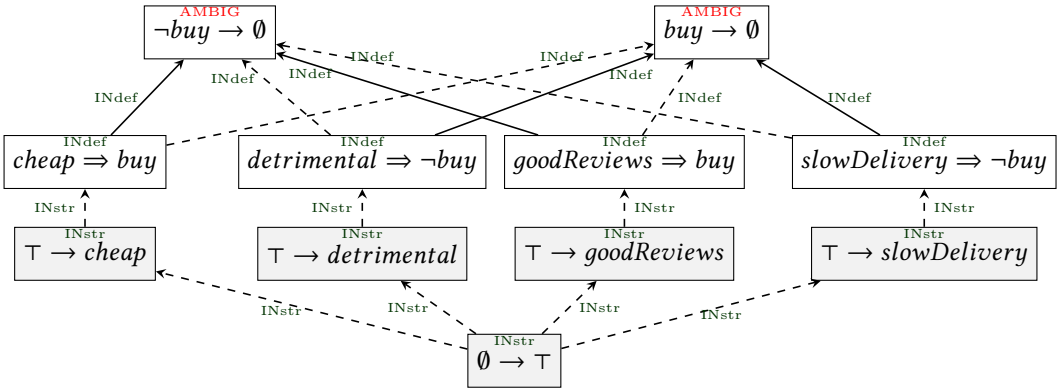

Figure 4.7: $\mathbb{SG}_{\mathcal{KB}}$ of Example 2.25 with a defeasible logic labeling that does not allow team defeat.

(a) $\boldsymbol{BDL_{noTD}(\mathbb{s}) = INstr}$ *iff* $BDL(\mathbb{s}) = INstr$.

(b) $\boldsymbol{BDL_{noTD}(\mathbb{s}) = OUTstr}$ *iff* $BDL(\mathbb{s}) = OUTstr$.

(e) $\boldsymbol{BDL_{noTD}(\mathbb{s}) = AMBIG}$ *iff* $BDL(\mathbb{s}) = AMBIG$.

(f) $\boldsymbol{BDL_{noTD}(\mathbb{s}) = UNSUP}$ *iff* $BDL(\mathbb{s}) = UNSUP$.

For INdef, only the condition (c).1 is changed to state that there must exist a support edge that is superior to all attack edges on the same premise.

(c) $\boldsymbol{BDL_{noTD}(\mathbb{s}) = INdef}$ *iff* $BDL_{noTD}(\mathbb{s}) \notin \{INstr, OUTstr\}$ *and* $\mathbb{s}$ *has a INstr or INdef complete support and*

1. $\exists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *that supports a premise* f *s.t.* $BDL_{noTD}(\boldsymbol{e_s}) = INstr$ *or* $BDL_{noTD}(\boldsymbol{e_s}) = INdef$ *and* $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *that undercut* $\mathbb{s}$ *on* f *s.t.* $BDL_{noTD}(\boldsymbol{e}) = INdef$, $\boldsymbol{e_s}$ *is superior to* $\boldsymbol{e}$.
2. *and* $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $BDL_{noTD}(\boldsymbol{e}) = INdef$ *and* $\boldsymbol{e}$ *attacks the rule application of* $\mathbb{s}$, $Rule(\mathbb{s})$ *is either a strict rule or* $Rule(\mathbb{s}) \succ Rule(Source(\boldsymbol{e}))$.

For OUTdef, only the condition (d).1 is changed to state that there must exist an attack edge that is superior to all support edges for the same premise.

(d) $\boldsymbol{BDL_{noTD}(\mathbb{s}) = OUTdef}$ *iff* $BDL_{noTD}(\mathbb{s}) \neq OUTstr$ *and* $\mathbb{s}$ *has an INstr or INdef complete support and*

1. *either* $\exists f \in Premise(\mathbb{s})$ *where* $\nexists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $BDL_{noTD}(\boldsymbol{e_s}) = INstr$ *and* $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *attacking* $\mathbb{s}$ *on* f *s.t.* $BDL_{noTD}(\boldsymbol{e}) = INdef$ *and* $\forall \boldsymbol{e'_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $BDL_{noTD}(\boldsymbol{e'_s}) \in \{INdef, AMBIG\}$, $\boldsymbol{e}$ *is superior to* $\boldsymbol{e'_s}$.

2. *or* $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $BDL_{noTD}(\boldsymbol{e}) = INdef$ *attacking the rule application of* $\mathbb{s}$ *and* $\mathit{Rule}(\mathbb{s})$ *is not a strict rule and* $\mathit{Rule}(\mathit{Source}(\boldsymbol{e})) \succ \mathit{Rule}(\mathbb{s})$.

The equivalence between BDL_{noTD} and Defeasible Logic with ambiguity blocking without team defeat is shown in Proposition 4.3.

Proposition 4.3. *Let* f *be a literal in a defeasible* $\mathcal{KB}$ *that contains no attack or support cycles:*

1. $\mathcal{KB} \vdash +\Delta f$ *iff* $\mathbb{SG}_{\mathcal{KB}}^{BDL_{noTD}}\langle(f \rightarrow \emptyset)\rangle = INstr$
2. $\mathcal{KB} \vdash -\Delta f$ *iff* $\mathbb{SG}_{\mathcal{KB}}^{BDL_{noTD}}\langle(f \rightarrow \emptyset)\rangle \neq INstr$
3. $\mathcal{KB} \vdash +\delta_{block}^{noTD} f$ *iff* $\mathbb{SG}_{\mathcal{KB}}^{BDL_{noTD}}\langle(f \rightarrow \emptyset)\rangle \in \{INstr, INdef\}$
4. $\mathcal{KB} \vdash -\delta_{block}^{noTD} f$ *iff* $\mathbb{SG}_{\mathcal{KB}}^{BDL_{noTD}}\langle(f \rightarrow \emptyset)\rangle \in \{OUTstr, OUTdef, AMBIG, UNSUP\}$

Proof. cf. Proof 4.3 in Section 7.2.2 on page xi. □

As for ambiguity propagating without team defeat (denoted by PDL_{noTD}) the same changes are done accordingly.

(a) $\boldsymbol{PDL_{noTD}}(\mathbb{s}) = \boldsymbol{INstr}$ *iff* $PDL(\mathbb{s}) = INstr$.

(b) $\boldsymbol{PDL_{noTD}}(\mathbb{s}) = \boldsymbol{OUTstr}$ *iff* $PDL(\mathbb{s}) = OUTstr$.

(c) $\boldsymbol{PDL_{noTD}}(\mathbb{s}) = \boldsymbol{INdef}$ *iff* $PDL_{noTD}(\mathbb{s}) \notin \{INstr, OUTstr\}$ *and* $\mathbb{s}$ *has a INstr or INdef complete support and*

1. $\exists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *that supports a premise* f *s.t.* $PDL_{noTD}(\boldsymbol{e_s}) = INstr$ *or* $PDL_{noTD}(\boldsymbol{e_s}) = INdef$ *and* $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *that undercut* $\mathbb{s}$ *on* f *s.t.* $PDL_{noTD}(\boldsymbol{e}) \in \{INdef\}$, $\boldsymbol{e_s}$ *is superior to* $\boldsymbol{e}$.
2. *and* $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $PDL_{noTD}(\boldsymbol{e}) \in \{INdef, AMBIG\}$ *and* $\boldsymbol{e}$ *attacks the rule application of* $\mathbb{s}$, $\mathit{Rule}(\mathbb{s})$ *is either a strict rule or* $\mathit{Rule}(\mathbb{s}) \succ \mathit{Rule}(\mathit{Source}(\boldsymbol{e}))$.

(d) $\boldsymbol{PDL}(\mathbb{s}) = \boldsymbol{OUTdef}$ *iff* $PDL_{noTD}(\mathbb{s}) \neq OUTstr$ *and* $\mathbb{s}$ *has an INstr or INdef complete support and*

1. *either* $\exists f \in \mathit{Premise}(\mathbb{s})$ *where* $\nexists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $PDL_{noTD}(\boldsymbol{e_s}) = INstr$ *and* $\forall \boldsymbol{e'_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $PDL_{noTD}(\boldsymbol{e'_s}) \in \{INdef, AMBIG\}$, $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *attacking* $\mathbb{s}$ *on* f *s.t.* $PDL_{noTD}(\boldsymbol{e}) = INdef$ *and* $\boldsymbol{e}$ *is superior to* $\boldsymbol{e'_s}$.
2. *or* $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $PDL_{noTD}(\boldsymbol{e}) = INdef$ *attacking the rule application of* $\mathbb{s}$ *and* $\mathit{Rule}(\mathbb{s})$ *is not a strict rule and* $\mathit{Rule}(\mathit{Source}(\boldsymbol{e})) \succ \mathit{Rule}(\mathbb{s})$.

(e) $\boldsymbol{PDL_{noTD}}(\mathbb{s}) = \boldsymbol{AMBIG}$ *iff* $PDL(\mathbb{s}) = AMBIG$.

(f) $\boldsymbol{PDL_{noTD}}(\mathbb{s}) = \boldsymbol{UNSUP}$ *iff* $PDL(\mathbb{s}) = UNSUP$.

The equivalence between PDL_{noTD} and Defeasible Logic with ambiguity propagating without team defeat is shown in Proposition 4.4.

Proposition 4.4. *Let f be a literal in a defeasible $\mathcal{KB}$ that contains no attack or support cycles:*

1. $\mathcal{KB} \vdash +\Delta f$ *iff* $\mathbb{SG}^{PDL_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = INstr$
2. $\mathcal{KB} \vdash -\Delta f$ *iff* $\mathbb{SG}^{PDL_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq INstr$
3. $\mathcal{KB} \vdash +\delta^{noTD}_{prop} f$ *iff* $\mathbb{SG}^{PDL_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{INstr, INdef\}$
4. $\mathcal{KB} \vdash -\delta^{noTD}_{prop} f$ *iff* $\mathbb{SG}^{PDL_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{OUTstr, OUTdef, AMBIG, UNSUP\}$

Proof. Proof 4.3 in Section 7.2.2 on page xi. □

4.2.4 Support and Attack Cycles

The first formalisms of defeasible reasoning [Nute, 1988, Billington, 1993, Antoniou et al., 2000a] did not take attack and support cycles into account and would loop infinitely, thus fail to draw reasonable conclusions in some cases [Maier and Nute, 2010b]. There are two types of cycles, *support cycles* (a.k.a. positive loops [Billington, 2008]) where cycles are due to rule applications (in SGs the cycle would only contain support edges), and attack cycles (a.k.a. negative loops [Billington, 2008]) where the cycles are due to conflicting rules (these cycles contain attack and possibly support edges). *Failure-by-looping* is a mechanism to avoid drawing unreasonable conclusions in presence of cycles [Maier and Nute, 2010b].

Support cycle: a sequence of unlabeled edges $\langle e_0, \ldots, e_n\rangle$ where $e_i \in \mathcal{E}_S$, $\forall i \in [0..n-1]$ $Target(e_i) = Source(e_{i+1})$, and $Source(e_0) = Target(e_n)$. If all statements in the cycle cannot be labeled by taking into account other edges outside this cycle then these statements are labeled UNSUP, as described in the following Example 4.7. Formally, (for all labeling functions, not only BDL):

(g) $\boldsymbol{BDL}(\mathbb{s}) = \boldsymbol{UNSUP}$ *if* $BDL(\mathbb{s}) \notin \{INstr, OUTstr, INdef, OUTdef, AMBIG\}$ *and* $\mathbb{s}$ *is part of a support cycle.*

Example 4.7 (Support Cycle). *Consider the following $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ (and $\mathbb{SG}^{BDL}_{\mathcal{KB}}$ in Figure 4.8) representing the knowledge that if a defendant is guilty then he is responsible. If he is guilty then he is responsible. The defendant is presumed not guilty unless responsibility is proven, and there is no proof for or against his responsibility.*

- $\mathcal{F} = \{\top \Rightarrow \neg guilty\}$
- $\mathcal{R} = \{r_1 : responsible \Rightarrow guilty, r_2 : guilty \Rightarrow responsible\}$.

The query 'is the defendant not guilty?' cannot be answered without failure-by-looping. The original Defeasible Logics would loop infinitely between "responsible" and "guilty". The statement $(responsible \Rightarrow guilty)$ *cannot be labeled without taking into account its support cycle with* $(guilty \Rightarrow responsible)$*, therefore all statements in this cycle are labeled UNSUP. Thus, the defendant is not guilty (i.e.* $\mathbb{SG}^{BDL}_{\mathcal{KB}}\langle(\neg guilty, \emptyset)\rangle = INdef$*).*

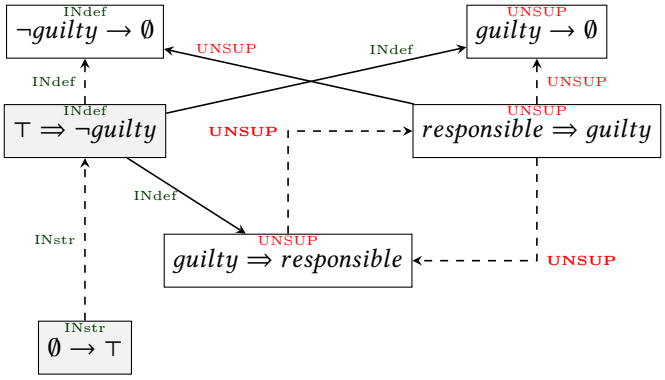

Figure 4.8: $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}$ of Example 4.7.

Support cycles are taken into account only when there is no other choice i.e. the statement cannot be labeled using edges outside the support cycle. In some cases however, the labeling function does not need to check the support cycle as shown in the following Figure 4.9. Suppose we add to the $\mathcal{KB}$ *of this example the fact that there is an evidence "evidA" implicating the responsibility of the defendant. In this case, the statement* $(responsible \Rightarrow guilty)$ *is labeled INdef because it has an INdef complete support and the support cycle has no impact.*

Attack cycle: a sequence of unlabeled edges $\langle e_0, \ldots, e_n\rangle$ where $e_i \in \mathcal{E}_A \cup \mathcal{E}_S$, $\forall i \in [0..n-1]$ $Target(e_i) = Source(e_{i+1})$, $Source(e_0) = Target(e_n)$, and at least one edge is an attack edge. If all statements in the cycle cannot be labeled using edges outside this cycle then these statements are labeled AMBIG, as described in Example 4.8. Formally (for all labeling functions, not only BDL):

(h) $\boldsymbol{BDL(\mathbb{s}) = AMBIG}$ *if* $BDL(\mathbb{s}) \notin \{INstr, OUTstr, INdef, OUTdef, UNSUP\}$ *and* $\mathbb{s}$ *is not part of a support cycle and is part of an attack cycle.*

Example 4.8. *Consider the following* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *of Example 2.27 (and* $\mathbb{SG}^{BDL}_{\mathcal{KB}}$ *in Figure 4.10).*

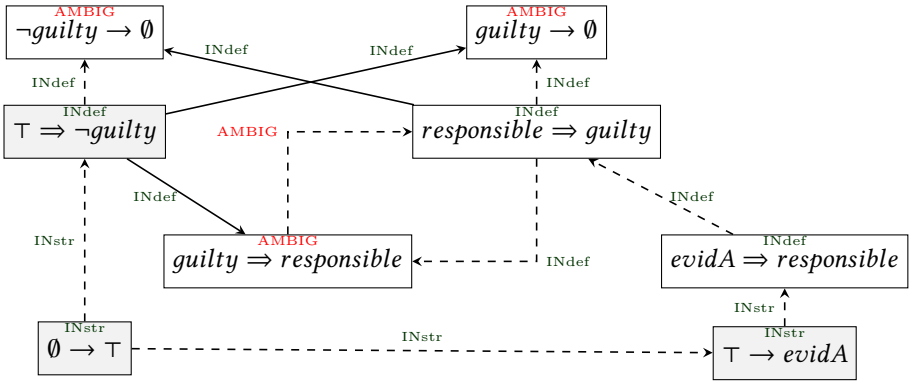

Figure 4.9: $\mathbb{SG}^{\mathrm{BDL}}_{\mathcal{KB}}$ of Example 4.7 with evidence for responsibility.

The query ***"is this animal a bird?"*** *cannot be answered without failure-by-looping. The statement* $(layEggs \Rightarrow bird)$ *cannot be labeled without taking into account the attack cycle with* $(mammal \Rightarrow \neg layEggs)$*, therefore all statements in this cycle are labeled AMBIG. Thus, we cannot say if the animal is a bird or not (i.e.* $\mathbb{SG}^{BDL}_{\mathcal{KB}}\langle(bird, \emptyset)\rangle = AMBIG$*).*

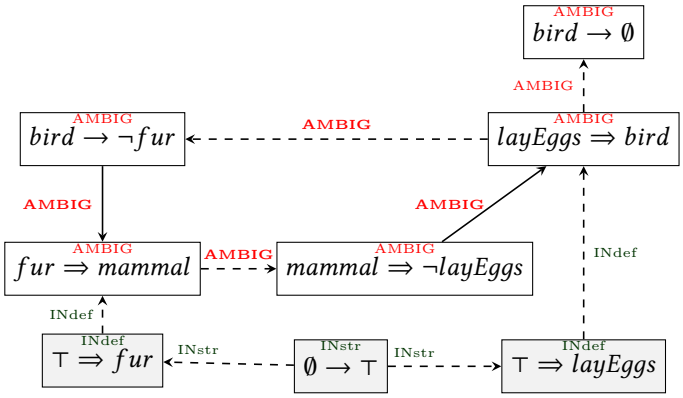

Figure 4.10: $\mathbb{SG}^{\mathrm{BDL}}_{\mathcal{KB}}$ of Example 4.8.

Similar to support cycles, attack cycles are taken into account only when there is no other choice i.e. the statement cannot be labeled using edges outside the attack cycle. In some cases however, the labeling function does not need to check the attack cycle as shown in the following Figure 4.11. Suppose we only observe that the animal lays eggs and does not have wings (i.e. has fur is removed from the $\mathcal{KB}$ *of this example). In this case, the statement* $(mammal \Rightarrow \neg layEggs)$ *is labeled UNSUP because it does not have a complete support and the attack cycle has no impact.*

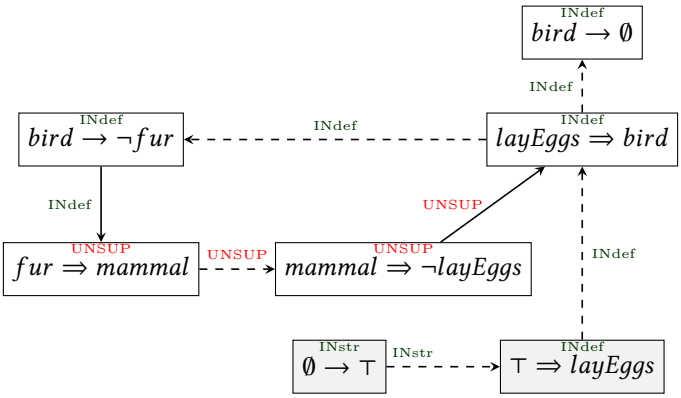

Figure 4.11: $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}$ of Example 4.8.

Proposition 4.5 states that the equivalence between labeling functions and Defeasible Logics with failure-by-looping still holds in presence of attack and support cycles.

Proposition 4.5. *Propositions 4.1, 4.3, 4.2, and 4.4 still hold for failure-by-looping in presence of attack and support cycles.*

Proof. cf. Proof 4.5 in Section 7.2.2 on page xiv. □

Proposition 4.6 makes the link between Statement Graphs and argumentation's Grounded Semantics.

Proposition 4.6. *Given a knowledge base $\mathcal{KB}$ expressed in $\mathcal{L}_p$:*

1. $\mathcal{KB} \vdash_{GS} f$ *iff* $\mathbb{SG}^{PDL_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{INstr, INdef\}$.
2. $\mathcal{KB} \nvdash_{GS} f$ *iff* $\mathbb{SG}^{PDL_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{OUTstr, OUTdef, UNSUP\}$.

Proof. Directly follows from Propositions 2.3 and 4.5. □

Constructing the Statement Graph of a knowledge base $\mathcal{KB}$ expressed in the propositional language $\mathcal{L}_p$ can be done in polynomial time by creating a rule application statement for each rule and fact then generating the support and attack edges. Labeling a statement graph can also be done in polynomial time since detecting if a statement is part of a cycle has polynomial complexity [Tarjan, 1972] along with checking the labels of the incoming edges which amounts to a breadth first graph traversal.

Proposition 4.7 (SG construction and labeling complexity for $\mathcal{L}_p$). *Given a knowledge base $\mathcal{KB}$ expressed in the propositional language $\mathcal{L}_p$, constructing and labeling $\mathbb{SG}^{Lbl}_{\mathcal{KB}}$ where $Lbl \in \{BDL, BDL_{noTD}, PDL, PDL_{noTD}\}$ has polynomial data and combined complexity.*

Proof. Constructing a Statement Graph in $\mathcal{L}_p$ amounts to creating a statement for each rule and fact rule then adding attack and support edges which is done in linear time. Computing the label of a query statement amount to a breadth first traversal of the graph with loop checking which has polynomial complexity [Tarjan, 1972]. □

Since Defeasible Reasoning techniques for the propositional language $\mathcal{L}_p$ can be extended to a first order language without the existential quantifier $\mathcal{L}_\forall$ by grounding the rules using the constants, all the entailment equivalence between Statement Graph labellings and Defeasible Reasoning techniques still hold for $\mathcal{L}_\forall$. However the computational complexity changes as the grounding phase has an exponential combined complexity [Chandra et al., 1981].

Proposition 4.8 (SG construction and labeling complexity for $\mathcal{L}_\forall$). *Given a knowledge base $\mathcal{KB}$ expressed in the first order language without existential quantifier $\mathcal{L}_\forall$, constructing and labeling $\mathbb{SG}^{Lbl}_{\mathcal{KB}}$ where $Lbl \in \{BDL, BDL_{noTD}, PDL, PDL_{noTD}\}$ has polynomial data complexity and exponential combined complexity.*

Proof. Grounding a knowledge base expressed in $\mathcal{L}_\forall$ produces a program in $\mathcal{L}_p$ [Chandra et al., 1981], Statement Graph can then be constructed directly after grounding. This grounding procedure has polynomial data complexity and exponential combined complexity [Chandra et al., 1981, Dantsin et al., 2001]. Given that reasoning with Statement Graph for $\mathcal{L}_p$ is polynomial, computing labels with a Statement Graph in a knowledge base expressed in $\mathcal{L}_\forall$ has polynomial data complexity and exponential combined complexity. □

Statement Graphs can be seen as a logic instantiation of Abstract Dialectical Framework (ADF) or bipolar argumentation frameworks [Cayrol and Lagasquie-Schiex, 2005]. ADF are abstract argumentation frameworks with attack and support relations between arguments, they have been instantiated in [Strass, 2013] with a propositional language without defeater rules. The difference with Statement Graphs is that the semantics for ADFs were designed to coincide with argumentation extensions and labellings (grounded, preferred, etc.), therefore, one cannot represent ambiguity blocking with ADFs semantics for example. Nevertheless, it seems possible to transform Statement Graphs to ADFs or argumentation framework, however this is left for future work.

In this section we showed how Statement Graphs can be used to obtain equivalent entailment results as Defeasible Logics and some argumentation semantics for the propositional language $\mathcal{L}_p$ and the first order language without existential quantifiers $\mathcal{L}_\forall$. In the next section, we define Statement Graphs for existential rules and study their expressiveness and complexity.

4.3 Existential Rules Statement Graph

One of the advantages of using Statement Graphs is that they can be easily extended to the existential rules language $\mathcal{L}_{\forall\exists}$. Since the labeling functions are defined based on statements' edges, they are not affected by the language used to construct the Statement Graph (as long as no rule application is lost). However, the definition of statement has to be adapted to the existential rules logical fragment.

4.3.1 Statement Graph Construction and Labeling

For existential rules, a rule statement is no longer simply the body and the head of a rule, but rather the application of that rule.

Definition 4.6 (Statement Existential Rules). *Given a knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ *expressed in* $\mathcal{L}_{\forall\exists}$ *, a statement* $\mathbb{s}$ *is either:*

1. *A 'query statement'* $(Q \rightarrow \emptyset)$ *where* Q *is a query.*
2. *The 'Top statement'* $(\emptyset \rightarrow \top)$.
3. *A 'rule application statement'* $(\Phi \Rrightarrow \psi)$ *represents a rule application* $\alpha(\mathcal{F}' \subseteq \mathcal{F}, r \in \mathcal{R}, \pi)$ *such that* $\Phi = \pi(Body(r))$, $\psi = \pi(Head(r))$, *and* $\Rrightarrow \in \{\rightarrow, \Rightarrow, \rightsquigarrow\}$ *if* r *is a strict, defeasible, or defeater rule respectively.*

By using the Frontier/Skolem chase to generate rule applications all the generated atoms can be seen as ground atoms, therefore the definitions of attack and support do not change: a statement $\mathbb{s}_1$ supports another statement $\mathbb{s}_2$ iff the conclusion of $\mathbb{s}_1$ is included in the premises of $\mathbb{s}_2$, and a statement $\mathbb{s}_1$ attacks another statement $\mathbb{s}_2$ iff the conclusion of $\mathbb{s}_1$ is in conflict with one of the premises of $\mathbb{s}_2$ or $\mathbb{s}_1$ uses a defeater rule, and the conclusions of $\mathbb{s}_1$ and $\mathbb{s}_2$ are in conflict.

Example 4.9 (SG with Existential Rules). *Consider the knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ)$ *of the animal shelter example (Example 3.1 on page 70). The statement graph of this example is shown in Figure 4.12.*

- $\mathcal{F} = \{\top \rightarrow alone(jack),\ \top \rightarrow hasCollar(jack),\ \top \rightarrow hasMicrochip(jack)\}$
- $\mathcal{R} = \{r_1 : \forall X, Y\ hasOwner(X, Y) \rightarrow keep(X),$
  $r_2 : \forall X\ hasCollar(X) \Rrightarrow \exists Y\ hasOwner(X, Y),$
  $r_3 : \forall X\ hasMicrochip(X) \Rrightarrow \exists Y\ hasOwner(X, Y),$
  $r_4 : \forall X\ alone(X) \Rrightarrow stray(X),\ r_5 : \forall X\ stray(X) \rightarrow adoption(X)\}$
- $\mathcal{N} = \{\forall X\ adoption(X) \wedge keep(X) \rightarrow \bot\}$
- $r_5 \succ r_2,\ r_3 \succ r_5$

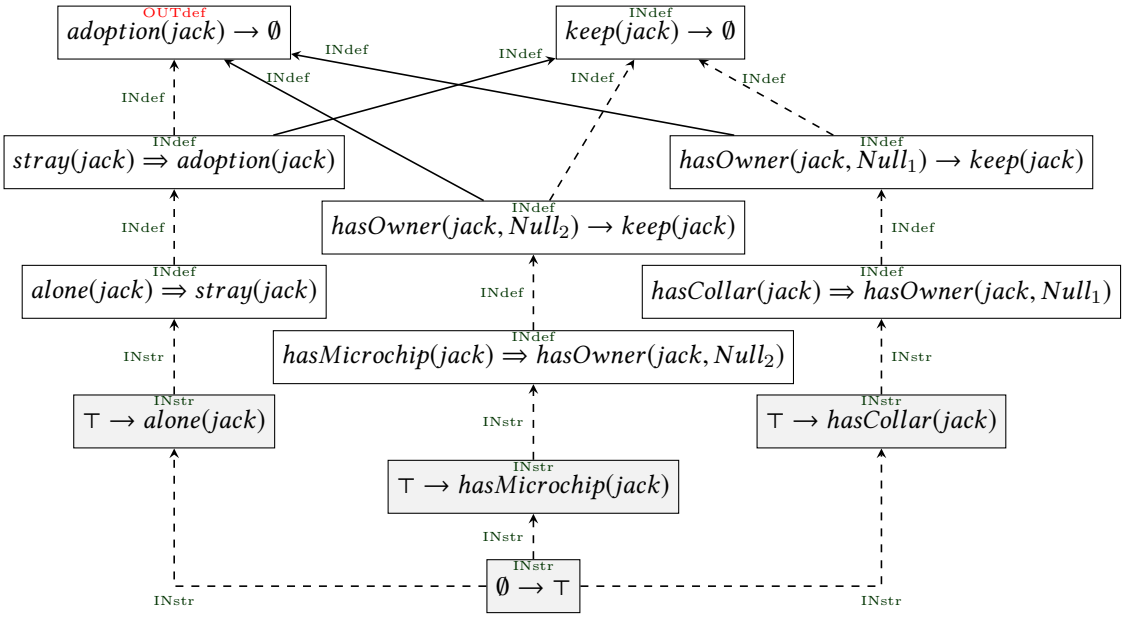

Figure 4.12: $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}$ of animal shelter Example 3.1.

However, relying on the frontier chase to construct the Statement Graph makes it prone to *rule application loss*. As discussed in the previous Chapter 3, the frontier chase derivation reducer might remove some rule applications which leads to the loss of some support edges. The following Algorithm 4.1 for the construction of a statement graph relies on the same tests as the Graph of Atom Dependency construction Algorithm 3.3 for the frontier chase in order to prevent any rule application loss.

The construction of a Statement Graph for a knowledge base $\mathcal{KB}$ expressed in the existential rule language $\mathcal{L}_{\forall\exists}$ has the same complexity as the frontier chase i.e. polynomial data complexity and exponential combined complexity. As previously discussed, the labeling can be done in polynomial time since it amounts to a breadth first graph traversal.

Proposition 4.9 (SG construction and labeling complexity for $\mathcal{L}_{\forall\exists}$). *Given a knowledge base $\mathcal{KB}$ expressed in the existential rule language $\mathcal{L}_{\forall\exists}$, constructing and labeling $\mathbb{SG}^{Lbl}_{\mathcal{KB}}$ where $Lbl \in \{BDL, BDL_{noTD}, PDL, PDL_{noTD}\}$ has polynomial data complexity and exponential combined complexity.*

Proof. The Algorithm 4.1 for constructing a Statement Graph of knowledge base expressed in $\mathcal{L}_{\forall\exists}$ using a Frontier/Skolem chase starts by running the chase to create the statements and at the same time uses the chase derivation reducer checks to prevent derivation loss which has polynomial data complexity and exponential combined complexity (cf Table 2.1 and Algorithm 4.1). Computing the labels afterwards for a query Statement amounts to a breadth first graph traversal which has polynomial complexity. □

Algorithm 4.1 SG construction with a frontier chase

Function `SGConstruction` $(\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \succ), \delta)$

input : $\mathcal{KB}$: A knowledge base expressed with the existential rules language, δ : An exhaustive breadth-first derivation of $\mathcal{KB}$

output: $\mathbb{SG}_{\mathcal{KB}}$: Statement Graph of $\mathcal{KB}$

1 $\mathcal{V} \leftarrow \{(\emptyset \rightarrow \top)\}$; $\mathcal{E}_S \leftarrow \emptyset$; $\mathcal{E}_A \leftarrow \emptyset$; $\mathbb{SG}_{\mathcal{KB}} \leftarrow (\mathcal{V}, \mathcal{E}_S, \mathcal{E}_A)$;
2 **foreach** $r \in \mathcal{F}$ **do**
3 $\mathcal{V} \leftarrow r$; /* *Add fact rules* */
4 /* *Creating Statements* */
5 **foreach** $D_i = (\mathcal{F}_i, r_i, \pi_i) \in \delta$ **do**
6 **if** $\exists D_j = (\mathcal{F}_j, r_j, \pi_j) \in \delta$ *s.t.* $j < i$, $r_j = r_i$, *and* $\pi_j|_{fr(r_j)}(Head(r_j)) = \pi_i|_{fr(r_i)}(Head(r_i))$ **then**
7 /* *This rule application is redundant* */
8 **if** $\mathbb{s} = (\pi_i(Body(r_i)) \Rrightarrow \pi_j(Head(r_i))) \notin \mathcal{V}$ **then**
9 /* $\Rrightarrow$ *stands for the type of implication in* r_i */
10 Add $\mathbb{s}$ to $\mathcal{V}$;
11 **else**
12 **if** $\mathbb{s} = (\pi_i(Body(r_i)) \Rrightarrow \pi_i(Head(r_i))) \notin \mathcal{V}$ **then**
13 Add $\mathbb{s}$ to $\mathcal{V}$;
14 **if** *No new facts are generated for the last breadth-first step* **then**
15 break;
16 /* *Adding Edges* */
17 **foreach** $\mathbb{s} \in \mathcal{V}$ **do**
18 **if** $Rule(\mathbb{s}) \in \mathcal{R}_{\rightsquigarrow}$ **then**
19 Create support edges from $\mathbb{s}$ to all $\mathbb{s}'$ s.t. $Conc(\mathbb{s}) \in Premise(\mathbb{s}')$;
20 Create attack edges from $\mathbb{s}$ to all $\mathbb{s}'$ s.t. $Conc(\mathbb{s})$ and a premise $f \in Premise(\mathbb{s}')$ are in conflict;
21 **else**
22 Create attack edges from $\mathbb{s}$ to all $\mathbb{s}'$ s.t. $Conc(\mathbb{s})$ and $Conc(\mathbb{s}')$ are in conflict;
23 **return** $\mathbb{SG}_{\mathcal{KB}}$;

This section extended the Statement Graph reasoning reasoning mechanism to existential rules. In the next section we present the first defeasible reasoning tool for existential rules that can handle various variants of defeasible reasoning and we evaluate its performance compared to the existing tools.

4.3.2 ELDR Tool and Evaluation

In order to expand the usability and appeal of defeasible reasoning, we propose an implementation of Statement Graph called **ELDR** (Existential Language for Defeasible Reasoning) that provides defeasible reasoning with first order existential rules, ambiguity blocking or propagating, with or without team defeat, and without consistent answers (the support for consistent answers will be addressed in the next chapter along with Repair Semantics).

ELDR Implementation. In order to express the different types of implication (strict, defeasible, and defeater) we extended the DLGP format [Baget et al., 2015] to allow for strict "<-", defeasible "<=", and defeater "<~" implications. ELDR[1] relies on the Datalog$^{\pm}$ dedicated inference engine called GRAAL [Baget et al., 2015] to run a frontier chase on the knowledge base and build the statement graph (as described in the previous Algorithm 4.1). The following Table 4.1 indicates what variants of defeasible reasoning ELDR tool can handle compared to the exiting tools.

Feature		*ASPIC**	*DeLP**	SPINdle	Flora-2	DEFT	**ELDR**
Ambiguity	Prop.	✓	✓	✓	-	✓	✓
	Block.	-	-	✓	✓	-	✓
Team Defeat	TD	-	✓	✓	✓	✓	✓
	noTD	✓	-	-	-	-	✓
Floating Conclusions	FC	-	-	-	-	-	-
	noFC	✓	✓	✓	✓	✓	✓
Consistent Derivation	Directly	-	✓	-	-	✓	-
	Indirectly	✓	-	✓	✓	-	✓
Existential Rules	S-FES	-	-	-	-	✓	✓
	FUS	-	-	-	-	-	-
	GBTS	-	-	-	-	-	-
Cycles	Support	-	✓	✓	✓	✓	✓
	Attack	✓	✓	✓	✓	✓	✓
Rule Block		✓	-	✓	✓	-	✓
Preference	$\succ$	-	✓	✓	✓	✓	✓
	$\mathbb{R}$	✓	-	-	-	-	-
Non-gound Queries		-	-	-	✓	-	-

Table 4.1: Classification results with ELDR (✓indicates the tool supports the feature).

[1] open source implementation available at `https://github.com/hamhec/eldr`

Theory		*ASPIC**	*DeLP**	SPINdle	Flora-2	DEFT	ELDR
ambiguity(n)	n = 50	0.44 (false)	148.17 (false)	0.11 (*false*) 0.09 (*true*)	1.06 (*true*)	0.11 (*false*)	**0.09 (false)** **0.09 (true)**
	n = 2000	∞	*T.O.*	∞	**18.942 (true)**	∞	∞
team(n)	n = 4	**0.22 (false)**	301.19 (true)	0.28 (true)	4.35 (true)	**0.28 (true)**	0.31 (false) 0.29 (true)
	n = 7	*T.O.*	*T.O.*	**109.46 (true)**	*T.O.*	201.917 (true)	**155.36 (false)** 155.04 (true)
cyclicSupp(n)	n = 1000	∞	291.37 (true)	**0.35 (true)**	5.712 (true)	0.44 (true)	0.38 (true)
	n = 10000	∞	*T.O.*	**26.61 (true)**	51.72 (true)	288.71 (true)	241.61 (true)
cyclicConf(n)	n = 5	0.62 (true)	55.89 (true)	0.90 (true)	0.92 (true)	**0.10 (true)**	0.83 (true)
	n = 1000	*T.O.*	*T.O.*	*T.O.*	*T.O.*	**79.52 (true)**	257.94 (true)
chain(n)	n = 600	108.05 (true)	99.46 (true)	**0.24 (true)**	2.35 (true)	0.33 (true)	0.41 (true)
	n = 10000	∞	*T.O.*	**16.04 (true)**	45.44 (true)	288.71 (true)	215.81 (true)
tree(n, 5)	n = 2	0.04 (true)	193.64 (true)	**0.03 (true)**	0.83 (true)	0.022 (true)	0.07 (true)
	n = 7	∞	*T.O.*	∞	211.94 (true)	182.83 (*true*)	**167.19 (true)**
dag(n, 10)	n = 1	∞	239.75 (true)	**7.51 (true)**	18.41 (true)	19.53 (true)	11.84 (true)
	n = 20	∞	*T.O.*	∞	*T.O.*	*T.O.*	**186.96 (true)**

Table 4.2: Execution time in seconds (selected results). 'true' and 'false' indicate query entailment and are used to check support of the feature (the best time is shown in bold)

Tool Evaluation. In order to assess the performance of Statement Graphs and their labeling functions we use the benchmark defined in Chapter 3.3. Handling more variants of defeasible reasoning is a desirable feature as long as performance does not significantly suffer. As shown in Table 4.2, ELDR is the only tool that allows for ambiguity blocking without team defeat for any of the considered languages. It can handle support and attack cycles and has better performance than DEFT and almost as good as SPINdle as shown in Figures 4.13, 4.14, 4.15, 4.16, 4.17, and 4.18.

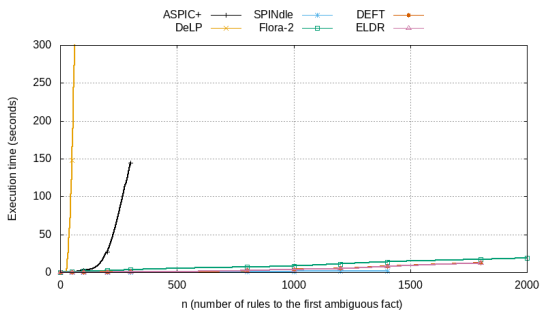

Figure 4.13: Response time for *ambiguous*(n)

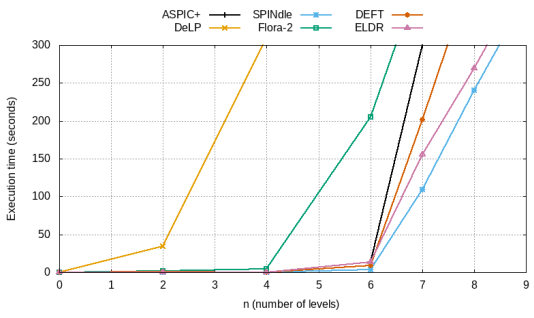

Figure 4.14: Response time for *team*(n)

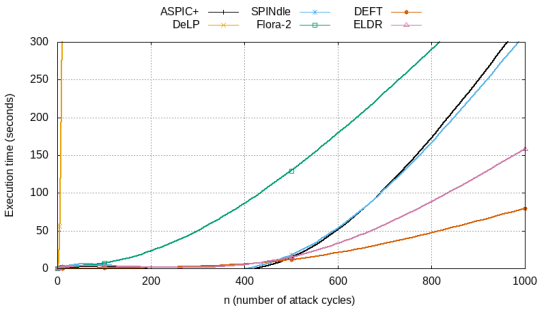

Figure 4.15: Response time for $cyclicConf(n)$

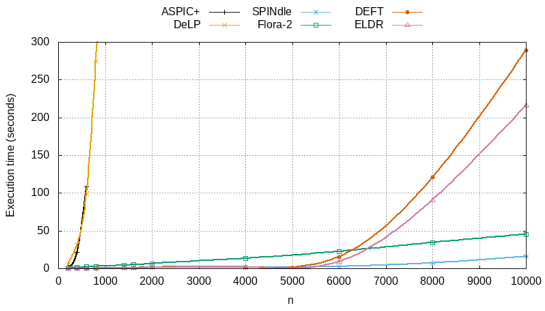

Figure 4.16: Response time for $chain(n)$

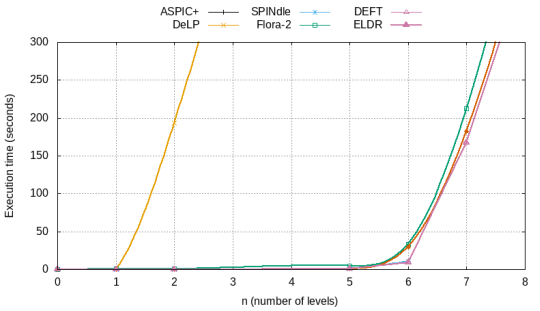

Figure 4.17: Response time for $tree(n, 5)$

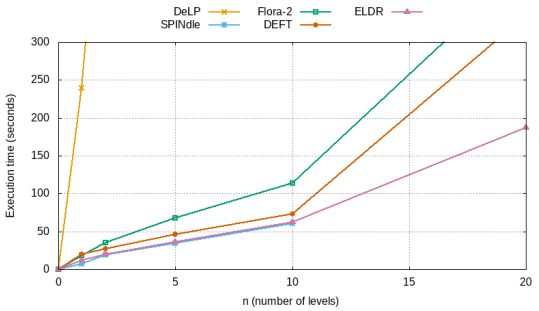

Figure 4.18: Response time for $dag(n, 10)$

4.4 Summary

In this chapter we presented a new formalism called Statement Graph that represents rule applications as "statements" with attack and support relations between them. By applying a flexible labeling function on edges, different variants of defeasible reasoning can be obtained (ambiguity propagating or blocking, with or without team defeat, and potentially with support and attack cycles). We first defined Statement Graph for the propositional language $\mathcal{L}_p$ and the first order language without existential quantifier $\mathcal{L}_\forall$ to obtain equivalent entailment results as defeasible logics and argumentation grounded semantics. We then defined Statement Graph for first order language with existential rules $\mathcal{L}_{\forall\exists}$ and presented its construction algorithm that prevents the derivation loss problem.

In order to fill the gaps of defeasible reasoning implementations we presented a tool for existential rules called *ELDR* based on Statement Graphs and showed that it is the first tool to provide defeasible reasoning with ambiguity blocking or propagating, with or without team defeat, and with failure-by-looping. We ran the benchmark defined in Chapter 3 and showed

that ELDR has similar and sometimes better performance results compared to existing tools.

In order to introduce the contribution of the next chapter we make, at this point, the observation that defeasible reasoning is not the only way to handle conflicts in knowledge bases, another well known technique is "Repair Semantics" [Lembo et al., 2010, Baget et al., 2016]. Having a tool that allows reasoning with both methods would be, thus, of great practical value. In the next chapter, we show the versatility of Statement Graph by presenting labellings for repair semantics, we compare them with defeasible reasoning variants and combine both techniques to produce new semantics that we evaluate with regards to human reasoning.

Chapter 4 in a Nutshell

- *Statement Graphs are a formalism able to represent Defeasible Logics via flexible labeling functions (ambiguity blocking or propagation, with or without team defeat, and with failure-by-looping).*
- *Statement Graphs can be constructed with different logical languages $\mathcal{L}_p$, $\mathcal{L}_\forall$, and $\mathcal{L}_{\forall\exists}$. For existential rules they are constructed using a frontier chase and account for derivation loss to represent most of the discussed features for defeasible reasoning.*
- *ELDR is an implementation of Statement Graphs and is the first tool that allows for ambiguity blocking or propagation, with or without team defeat for the languages $\mathcal{L}_p$, $\mathcal{L}_\forall$, and $\mathcal{L}_{\forall\exists}$.*

5

Statement Graph and Repair Semantics

Conflicts in knowledge representation cause severe problems, notably due the principle of explosion (from falsehood anything follows). These conflicts arise from two possible sources, either the facts are incorrect (known as *inconsistence*), or the rules themselves are contradictory (known as *incoherence*). In order to preserve the ability to reason in presence of conflicts, several approaches can be used, in particular *Defeasible Reasoning* (that we discussed in previous chapters) and *Repair Semantics*. These two approaches stem from different needs and address conflicts in different ways. However, as we will show in this chapter, they can be *compared* and *combined*. We start by defining under which conditions Repair Semantics and Defeasible Reasoning can be compared, then we show that Statement Graphs not only can represent certain Repair Semantics but also combine them with Defeasible Reasoning intuitions to obtain new semantics. We run an experimental evaluation to see how these new semantics coincide with human reasoning, and discuss the ability of Statement Graph to represent certain human reasoning tasks such as the suppression task.

Research Questions in this Chapter

- *Under which conditions can Defeasible Reasoning and Repair Semantics be compared? Do their semantics coincide?*
- *Can the different intuitions of Defeasible Reasoning be combined with those of Repair Semantics to obtain new semantics? If so, how do these new semantics compare to human reasoning? Can Statement Graphs represent them?*
- *Can Statement Graphs represent other forms of human reasoning that Defeasible Reasoning and Repair Semantics cannot represent (such as the suppression task)?*

5.1 Repair Semantics

Defeasible reasoning originates from the need to reason with incomplete knowledge by "filling the gaps in the available information by making some kind of plausible (or desirable) assumptions" [Billington et al., 2010]. Repair semantics [Lembo et al., 2010] on the other hand originate from the need to handle inconsistency that arises due to merging (among others) of different data sources. It has also been applied to "Ontology-Based Data Access" [Poggi et al., 2008] where an ontology is used to access a set of data sources.

Defeasible Reasoning and Repair Semantics are generally seen as two inherently distinct approaches that answer different problems, are studied by distant communities and, to the best of our knowledge, have never been explicitly put together. A key difference between Defeasible Reasoning and Repair Semantics is that the former was designed for incoherence while the latter was designed for inconsistence. However, since inconsistence is a special type of incoherence [Flouris et al., 2006], Defeasible Reasoning can be applied to inconsistent but coherent knowledge, and thus it can be compared to the Repair Semantics as we will see in Section 5.2.3.

Contrary to Defeasible Reasoning that considers different types of implications (strict and defeasible), Repair Semantics only consider definite strict implications, any conflict that arises in a knowledge base is due to "corrupt", erroneous data. The initial set of facts can be seen as defeasible facts. In order to handle conflicts, Repair Semantics construct all possible ways of "repairing" the set of facts while preserving as much information as possible (in the sense of set inclusion).

Definition 5.1 (Repair). *Let $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ be a knowledge base that contains only strict rules, a repair is an inclusion-maximal subset of facts $\mathcal{D} \subseteq \mathcal{F}$ that is consistent with the rules and negative constraints (i.e. $\mathit{models}(\mathcal{D}, \mathcal{R} \cup$*

$\mathcal{N}) \neq \emptyset$*). We denote the set of repairs of a knowledge base by* $\mathit{repairs}(\mathcal{KB})$.

Example 5.1. *Consider the following* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *that describes a simplified legal situation: If there is a scientific evidence incriminating a defendant then he is responsible for the crime, if there is a scientific evidence absolving a defendant then he is not responsible for the crime. A defendant is guilty if responsibility is proven. If a defendant is guilty then he will be given a sentence. If a defendant has an alibi then he is innocent. There is a scientific evidence "e1" incriminating a female defendant "alice", while another scientific evidence "e2" is absolving her of the crime. She also has an alibi. Is Alice innocent (i.e.* $Q_1() = \mathit{innocent}(\mathit{alice})$*)? Is she guilty (i.e.* $Q_2() = \mathit{guilty}(\mathit{alice})$*)?*

- $\mathcal{F} = \{\mathit{incrim}(e1, \mathit{alice}),\ \mathit{absolv}(e2, \mathit{alice}),\ \mathit{alibi}(\mathit{alice}),\ \mathit{female}(\mathit{alice})\}$
- $\mathcal{R} = \{r_1 : \forall X, Y\ \mathit{incrim}(X, Y) \rightarrow \mathit{resp}(Y),$
  $r_2 : \forall X, Y\ \mathit{absolv}(X, Y) \rightarrow \mathit{notResp}(X),$
  $r_3 : \forall X\ \mathit{resp}(X) \rightarrow \mathit{guilty}(X),$
  $r_4 : \forall X\ \mathit{alibi}(X) \rightarrow \mathit{innocent}(X),$
  $r_5 : \forall X\ \mathit{guilty}(X) \rightarrow \exists Y\ \mathit{sentence}(X, Y)\}$
- $\mathcal{N} = \{\forall X\ \mathit{resp}(X) \wedge \mathit{notResp}(X) \rightarrow \bot,\ \forall X\ \mathit{guilty}(X) \wedge \mathit{innocent}(X) \rightarrow \bot\}$

The Statement Graph representation of this example is shown in Figure 5.1.

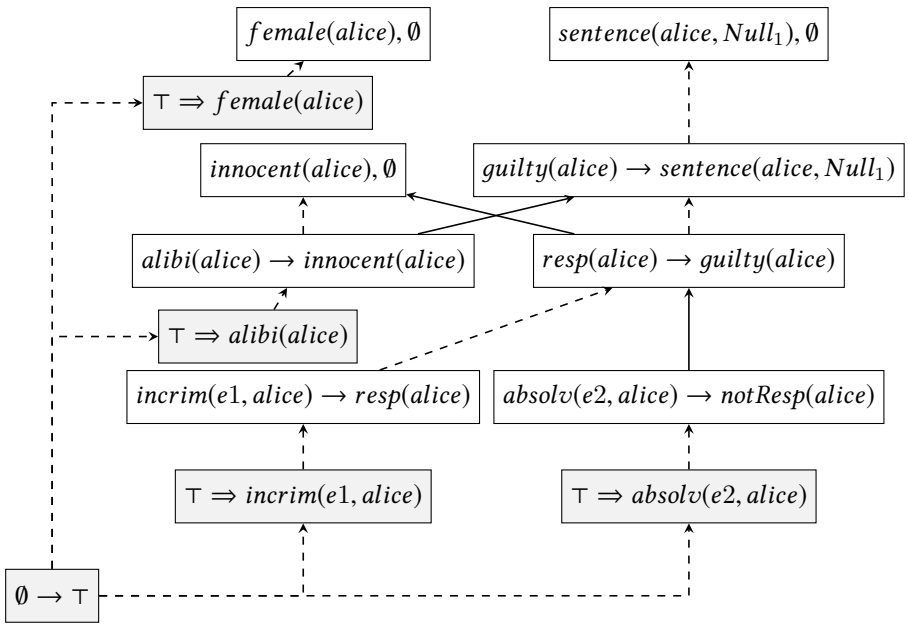

Figure 5.1: Example 5.1's Statement Graph.

The saturated set of facts resulting from a frontier chase is:

- $\mathcal{F}^* = \{incrim(e1, alice), absolv(e2, alice), alibi(alice), female(alice), resp(alice), notResp(alice), guilty(alice), innocent(alice), sentence(alice, Null_1)\}$.

This knowledge base is ***inconsistent****, because a negative constraint is applicable, (thus* $models(\mathcal{F}, \mathcal{R} \cup \mathcal{N}) = \emptyset$*). This inconsistency is due to an erroneous set of facts (either one of the evidences, the alibi, or all of them are not valid). The repairs constructed from the starting set of facts are:*

- $\mathcal{D}_1 = \{absolv(e2, alice), alibi(alice), female(alice)\}$

- $\mathcal{D}_2 = \{incrim(e1, alice), female(alice)\}$

5.1.1 AR and IAR Repair Semantics

A well-known semantics for inconsistent databases is *consistent answers* [Arenas et al., 1999] (also known as All Repairs (AR) semantics [Lembo et al., 2010]) which considers "entailed" atoms that can be derived from all possible repairs built from the initial set of facts.

Definition 5.2 (AR Semantics [Lembo et al., 2010]). *Given a knowledge base* $\mathcal{KB}$*, a query* Q *is AR entailed (denoted* $\mathcal{KB} \models_{AR} Q$*) if and only if it is entailed from every repair i.e.* $\forall \mathcal{D} \in repairs(\mathcal{KB}) : \mathcal{D} \cup \mathcal{R} \models Q$.

Example 5.2 (Cont'd Example 5.1). *The fact* $female(alice)$ *can be entailed from both repairs, therefore* $\mathcal{KB} \models_{AR} female(alice)$.

AR semantics is the most accepted Repair Semantics [Lembo and Ruzzi, 2007], however it comes with a high computational cost. Different approximations of AR semantics with a lower complexity have been defined, most notably, the IAR semantics (Intersection of All Repairs).

The Intersection of All Repairs semantics [Lembo et al., 2010] is the most skeptical of the Repair Semantics, it follows the principle of "*when in doubt throw it out*". In this semantics a fact is entailed if it is entailed by the intersection of all repairs constructed from the starting set of facts. This condition is equivalent to stating that this fact is not involved in any conflict [Lembo et al., 2010], meaning that it does not "lead"[1] to any conflict and is not generated from facts that are in conflict with other facts (i.e. ambiguous ones).

Definition 5.3 (IAR Semantics [Lembo et al., 2010]). *Given a knowledge base* $\mathcal{KB}$*, a query* Q *is IAR entailed (denoted* $\mathcal{KB} \models_{IAR} Q$*) if and only if it is entailed from the intersection of all repairs i.e.* $\bigcap repairs(\mathcal{KB}) \cup \mathcal{R} \models Q$ *where* $\bigcap repairs(\mathcal{KB})$ *is the intersection of the repairs of* $\mathcal{KB}$.

Example 5.3 (Cont'd Example 5.1). *The intersection of all repairs is* $\bigcap repairs(\mathcal{KB}) = \{female(alice)\}$*, therefore* $\mathcal{KB} \models_{IAR} female(alice)$.

[1] A fact f is said to *lead* to a conflict if there is a derivation relying on (i.e. containing) f that generates conflicting atoms.

Any fact that is IAR entailed is also AR entailed.

Proposition 5.1 (IAR and AR [Lembo et al., 2010]). *Given a knowledge base $\mathcal{KB}$ and a query Q, if $\mathcal{KB} \models_{IAR} Q$ then $\mathcal{KB} \models_{AR} Q$.*

However, the inverse is not necessarily true as shown in the following example.

Example 5.4 (AR vs. IAR). *Consider the following knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ stating that Bob is cat and a dog. Dogs and cats are animals. Is Bob an animal $Q = animal(bob)$?*

- $\mathcal{F} = \{dog(bob),\ cat(bob)\}$
- $\mathcal{R} = \{r_1 : \forall X\ dog(X) \rightarrow animal(X),\ r_2 : \forall X\ cat(X) \rightarrow animal(X)\}$
- $\mathcal{N} = \{\forall X\ dog(X) \wedge cat(X) \rightarrow \bot\}$

The repairs constructed from the initial set of facts are: $\mathcal{D}_1 = \{dog(bob)\}$ and $\mathcal{D}_2 = \{cat(bob)\}$. The query $Q = animal(bob)$ is AR entailed ($\mathcal{KB} \models_{AR} Q$) because $animal(bob)$ can be derived from all repairs (i.e. $\mathcal{D}_1 \cup \mathcal{R} \models animal(bob)$ and $\mathcal{D}_2 \cup \mathcal{R} \models animal(bob)$). However, the intersection of the repairs is empty (i.e. $\bigcap repairs(\mathcal{KB}) = \emptyset$), therefore Q is not IAR entailed ($\mathcal{KB} \not\models_{IAR} Q$).

5.1.2 CAR and ICAR Repair Semantics

AR and IAR semantics are based on the set of repairs constructed from the initial set of facts. Another way of handling conflicts is to consider the repairs constructed after saturating the initial set of facts using all possible rule applications. Applying the same intuition behind AR semantics to these repairs gives the Closed ABox Repair Semantics (CAR) that considers a query entailed if it can be derived from all the repairs constructed after saturating the initial set of facts.

Definition 5.4 (CAR Semantics [Lembo et al., 2010]). *Given a knowledge base $\mathcal{KB}$, a query Q is CAR entailed (denoted $\mathcal{KB} \models_{CAR} Q$) if and only if it is entailed from every repair constructed from the saturated set of facts i.e. $\forall \mathcal{D} \in repairs^*(\mathcal{KB}) : \mathcal{D} \cup \mathcal{R} \models Q$ where $repairs^*(\mathcal{KB})$ denotes the set of repairs constructed from $\mathcal{F}^*$.*

Example 5.5 (Cont'd Example 5.1). *The repairs constructed from the saturated set of facts are:*

- $\mathcal{D}'_1 = \{absolv(e2, alice),\ alibi(alice),\ female(alice),\ notResp(alice),\ sentence(alice, Null_1)\}$
- $\mathcal{D}'_2 = \{incrim(e1, alice),\ female(alice),\ resp(alice),\ guilty(alice),\ sentence(alice, Null_1)\}$

The fact $\mathit{sentence}(\mathit{alice}, \mathit{Null}_1)$ *is not AR entailed, however it is CAR entailed since* $\mathcal{D}'_1$ *and* $\mathcal{D}'_2$ *entails it (i.e.* $\mathcal{KB} \models_{CAR} \mathit{sentence}(\mathit{alice}, \mathit{Null}_1)$*).*

It might seem at first that CAR semantics yields unintuitive results such as in Example 5.1 where it is CAR entailed that Alice is sentenced while it is not CAR entailed that she is found guilty. However, the CAR semantics can be seen as an approximation of the AR semantics since it has a lower computational cost and any fact that is AR entailed is also CAR entailed (CAR semantics can be seen as an upper bound to AR semantics) [Lembo et al., 2010].

Another semantics is the Intersection of Closed ABox Repairs (ICAR) semantics which is an approximation of the CAR semantics with better complexity. A fact is ICAR entailed if it can be derived from the intersection of the repairs constructed from the saturated set of facts.

Definition 5.5 (ICAR Semantics [Lembo et al., 2010]). *Given a knowledge base* $\mathcal{KB}$*, a query* Q *is ICAR entailed (denoted* $\mathcal{KB} \models_{ICAR} Q$*) if and only if it is entailed from the intersection of all repairs constructed from the saturated set of facts i.e.* $\bigcap \mathit{repairs}^*(\mathcal{KB}) \cup \mathcal{R} \models Q$ *where* $\bigcap \mathit{repairs}^*(\mathcal{KB})$ *is the intersection of the repairs constructed from* $\mathcal{F}^*$ *of* $\mathcal{KB}$*.*

Example 5.6 (Cont'd Example 5.5). *The intersection of all repairs constructed from* $\mathcal{F}^*$ *is* $\bigcap \mathit{repairs}^*(\mathcal{KB}) = \{\mathit{female}(\mathit{alice}), \mathit{sentence}(\mathit{alice}, \mathit{Null}_1)\}$*, therefore* $\mathcal{KB} \models_{ICAR} \mathit{female}(\mathit{alice})$ *and* $\mathcal{KB} \models_{ICAR} \mathit{sentence}(\mathit{alice}, \mathit{Null}_1)$*.*

Same as for IAR and AR semantics, every atom that is ICAR entailed is also CAR entailed while the inverse is not necessarily true. CAR can be seen as an upper bound approximation of AR, and the same applies for ICAR and IAR as described by the following Proposition 5.2.

Proposition 5.2 (CAR, AR, ICAR, and IAR [Lembo et al., 2010]). *Given a knowledge base* $\mathcal{KB}$ *that only contains defeasible facts, strict rules and no preferences, and a query* Q*:*

- *if* $\mathcal{KB} \models_{AR} Q$ *then* $\mathcal{KB} \models_{CAR} Q$
- *if* $\mathcal{KB} \models_{ICAR} Q$ *then* $\mathcal{KB} \models_{CAR} Q$
- *if* $\mathcal{KB} \models_{IAR} Q$ *then* $\mathcal{KB} \models_{ICAR} Q$

Semantics can be compared based on the notion of *productivity* [Baget et al., 2016]. We say that a semantics $\models_1$ is less productive than $\models_2$ (represented as $\models_1 \rightarrow \models_2$) if it results in fewer conclusions being drawn (i.e. if $\models_1 f$ then $\models_2 f$). Figure 5.2 displays the productivity relation between the discussed Repair Semantics.

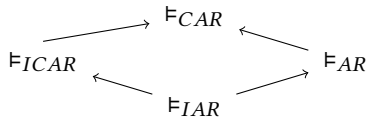

Figure 5.2: Productivity of discussed Repair Semantics [Baget et al., 2016].

The complexity of Repair Semantics depends on the cost of checking consistency [Lembo et al., 2015].

Proposition 5.3 (AR, CAR, IAR, and ICAR complexity for the Frontier Chase). *Given a knowledge base expressed in $\mathcal{L}_{\forall\exists}$ with only defeasible facts, strict rules, and no preferences, ground query answering for AR and ICAR has* coNP-complete *data complexity and* 2ExpTime-complete *combined complexity using a Frontier chase. For IAR and ICAR it is* PTime-complete *data complexity and* 2ExpTime-complete *combined complexity using a Frontier chase.*

Proof. We prove AR and CAR complexity using the same proof of [Lembo et al., 2015] for backward chaining. Showing that a query Q is not entailed under AR semantics by a KB can be done by guessing a repair $\mathcal{D} \subseteq \mathcal{F}$ of $\mathcal{KB}$ such that $\mathcal{D} \cup \mathcal{R} \nvdash Q$. Checking that $\mathcal{D}$ is consistent, that for every fact $f \in \mathcal{F} \backslash \mathcal{D}$: $\mathcal{D} \cup \{f\}$ is inconsistent, and that $\mathcal{D} \cup \mathcal{R} \nvdash Q$ has PTime-complete data complexity for the Frontier chase. Thus, query answering is coNP. For the lower bound, a proof for hardness by reduction from SAT is presented in [Lembo et al., 2015]. The same proof is applied for CAR by guessing a repair $\mathcal{D} \subseteq \mathcal{F}^*$ from the saturated set of facts.

IAR can be computed by finding all derivations for the atom $\bot$ then removing any initial fact in those derivations. This has PTime-complete data complexity and 2ExpTime-complete combined complexity for the Frontier chase as shown in Proposition **??** on page ??. ICAR can be computed by first saturating the initial set of facts then applying same principle as IAR on the saturated set of facts. □

The complexity of the considered Repair Semantics for the skolem-FES fragment of existential rules is summarized in Table 5.1.

Semantics	Combined Complexity	Data Complexity
AR	2ExpTime-complete	coNP-complete
IAR	2ExpTime-complete	PTime-complete
CAR	2ExpTime-complete	coNP-complete
ICAR	2ExpTime-complete	PTime-complete

Table 5.1: Complexity of query answering of the considered Repair Semantics for the Frontier chase

The discussed semantics are not the only Repair Semantics, different ones can be defined based on how repairs are constructed and handled. For more information, the interested reader is referred to the detailed analysis and comparison in [Baget et al., 2016].

5.1.3 Defeasible Reasoning and Repair Semantics

Repair semantics make the assumption that the set of rules is coherent because an incoherent set of rules might lead to an empty set of repairs [Deagustini et al., 2015] as shown in the following Example 5.7.

Example 5.7 (Incoherence). *Consider the following* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$*:*

- $\mathcal{F} = \{penguin(kowalski)\}$
- $\mathcal{R} = \{ r_1 : \forall X\, penguin(X) \rightarrow bird(X),\ r_2 : \forall X\, bird(X) \rightarrow fly(X),$ $r_3 : \forall X\, penguin(X) \rightarrow notFly(X)\}$
- $\mathcal{N} = \{\forall X\, fly(X) \wedge notFly(X) \rightarrow \bot\}$

The set of rules is incoherent because no set of facts (even outside $\mathcal{F}$*) that makes all rules in* $\mathcal{R}$ *applicable prevents the application of the negative constraint. No repair from the initial set of facts can be constructed since* $models(\{penguin(kowalski)\}, \mathcal{R} \cup \mathcal{N}) = \emptyset$*.*

This assumption of coherence is a key difference between Repair Semantics and Defeasible Reasoning. However, these two approaches are similar in the sense that they can both be applied to potentially inconsistent knowledge bases with a coherent set of rules, i.e. knowledge bases with *defeasible facts and only strict rules* (here we do not consider a preference relation between facts since preferences are not considered in most Repair Semantics).

Furthermore each method is based on inherently different intuitions, for example, Defeasible Logics consider 'entailed' any information derived before a contradiction happens, while Repair Semantics consider 'not entailed' any information leading to a contradiction as shown in the following example.

Example 5.8 (Cont'd Example 5.1).

- *Defeasible Logics:* $\mathbb{SG}^{BDL}_{\mathcal{KB}}$ *and* $\mathbb{SG}^{PDL}_{\mathcal{KB}}$ *are shown in Figures 5.3 and 5.4 respectively (top statement has been omitted for clarity). There are no direct attacks within the initial set of facts, therefore they are all defeasibly entailed in Defeasible Logics. In ambiguity blocking* $innocent(alice)$ *is entailed because* $guilty(alice)$ *is ambiguous while in ambiguity propagation it is not entailed. The remaining facts rely on ambiguous ones, therefore they are not entailed.*
- *Dialectical Trees: The only fact that is entailed is* $female(alice)$ *because its argument* $\langle\{\top \Rightarrow female(alice)\}, female(alice)\rangle$ *is not attacked.*

- *Grounded Semantics:* *The only entailed facts are the initial set of facts.*

Table 5.2 summarizes the entailed facts for each approach.

Semantics	Entailed Facts
$\models_{\text{BDL}}$	$female(alice), incrim(e1, alice), absolv(e2, alice), innocent(alice)$
$\models_{\text{PDL}}$	$female(alice), incrim(e1, alice), absolv(e2, alice)$
$\models_{DT}$	$female(alice)$
$\models_{GS}$	$female(alice), incrim(e1, alice), absolv(e2, alice)$
$\models_{AR}$	$female(alice)$
$\models_{IAR}$	$female(alice)$
$\models_{CAR}$	$female(alice), sentence(alice, Null_1)$
$\models_{ICAR}$	$female(alice), sentence(alice, Null_1)$

Table 5.2: Entailed facts of Example 5.1

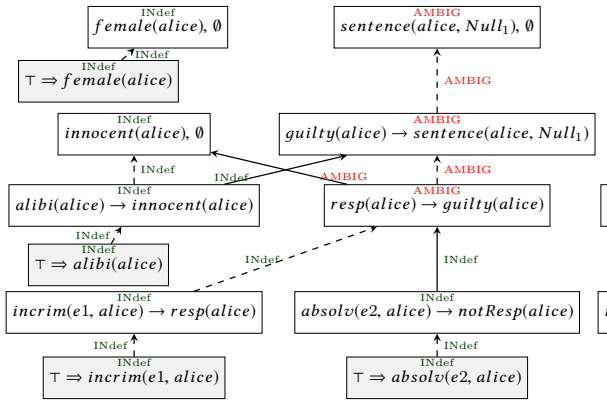

Figure 5.3: BDL applied to Example 5.1's Statement Graph.

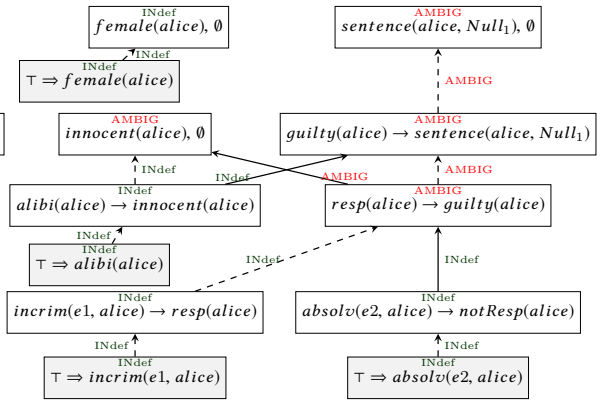

Figure 5.4: PDL applied to Example 5.1's Statement Graph.

While Defeasible Logics do not yield the same results as Repair Semantics, a productivity relation can be established between Defeasible Logics and IAR semantics: any fact that is IAR entailed is also entailed in Defeasible Logics (with both ambiguity propagation and blocking).

Proposition 5.4 (PDL, BDL, and IAR). *Given a knowledge base $\mathcal{KB}$ that only contains defeasible facts, strict rules and no preferences, and a ground query Q:*

1. *if $\mathcal{KB} \models_{IAR} Q$ then $\mathcal{KB} \models_{PDL} Q$*
2. *if $\mathcal{KB} \models_{IAR} Q$ then $\mathcal{KB} \models_{BDL} Q$*

3. if $\mathcal{KB} \models_{PDL} Q$ *then* $\mathcal{KB} \models_{BDL} Q$

Sketch. IAR semantics considers entailed any fact that does not lead to a conflict and is not generated from one. In Defeasible Logics this means that the fact is supported and is not ambiguous. The productivity relation between $\models_{\text{PDL}}$ and $\models_{\text{BDL}}$ can be extracted from the inclusion theorem in [Billington et al., 2010] (cf. detailed proof 5.3 in Section 7.2.3 on page xv). □

Since both Dialectical Trees and IAR semantics, in presence of strict rules discard facts leading to a conflict, one might be tempted to assume that Dialectical Trees give equivalent results to IAR semantics under the restriction of defeasible facts, strict rules and no preferences. However, consider the following example (similar to Example 3.7).

Example 5.9 (Dialectical Trees vs. IAR). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ *expressed in* $\mathcal{L}_p$ *for simplicity.*

- $\mathcal{F} = \{\top \Rightarrow a, \top \Rightarrow b, \top \Rightarrow d\}$
- $\mathcal{R} = \{a \wedge b \rightarrow \neg c, d \rightarrow c\}$

The repairs that can be constructed from the initial set of facts are $\mathcal{D}_1 = \{a, b\}$ *and* $\mathcal{D}_2 = \{d\}$*, their intersection is empty, therefore no fact is entailed in IAR semantics. However, the argument* $\langle \{\top \Rightarrow a\}, a \rangle$ *has no defeater (since there is no fact* f *such that* $\{f, a\} \cup \mathcal{F}_{\rightarrow} \cup \mathcal{R}_{\rightarrow}$ *is inconsistent) and is therefore warranted in Dialectical Trees, thus* $\mathcal{KB} \models_{DT} a$*.*

Nevertheless, we can show that under the restrictions of defeasible facts, strict rules, and no preferences, entailment using Dialectical Trees is strictly more productive than IAR semantics. More precisely, In the following Proposition 5.5 we show that any fact that is IAR entailed is also entailed using Dialectical Trees. Furthermore, under these restrictions, entailment using Defeasible Logics is strictly more productive than entailment using Dialectical Trees.

Proposition 5.5 (Dialectical Trees, PDL and IAR). *Given a knowledge base* $\mathcal{KB}$ *that only contains defeasible facts, strict rules and no preferences, and a ground query* Q*:*

1. if $\mathcal{KB} \models_{IAR} Q$ *then* $\mathcal{KB} \models_{DT} Q$

2. if $\mathcal{KB} \models_{DT} Q$ *then* $\mathcal{KB} \models_{PDL} Q$

Proof. cf. Proof 5.5 in Section 7.2.3 on page xv. □

Figure 5.5 represents the productivity relations between the discussed Defeasible Reasoning and Repair Semantics approaches along with their data complexity.

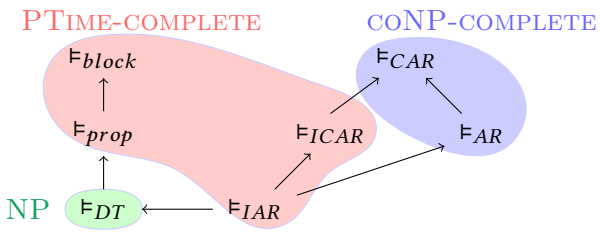

Figure 5.5: Productivity and data complexity of the discussed Defeasible Reasoning and Repair Semantics approaches (only defeasible facts, strict rules, and no preferences).

5.2 Statement Graph Labellings for Repair Semantics

Statement Graphs can be used to represent IAR and ICAR Repair Semantics using labeling functions with IN, OUT, and AMIBG labels. The idea here is to first detect conflicts, then apply a second pass from the query statements to the initial facts in order to discard any source of ambiguity.

5.2.1 Labeling for IAR

The intuition behind IAR is to reject any fact that would lead to a conflict (by applying rules) or that is generated from a conflict [Lembo et al., 2010]. From an SG point of view, any statement that is attacked, or that supports statements that lead to an attack, or that is only supported by an ambiguous statement, is discarded. This can be obtained by first detecting all conflicts, then discarding any statement that either leads to a conflict or is generated from conflicting atoms.

In order to detect conflicts using Statement Graphs, we need to ensure that all conflicts are represented. Algorithm 4.1 for Statement Graph construction ensures that no rule application is lost. Given that statements attack each other on the premise, the upper most statements with no outgoing edges might generate conflicting atoms, that is why any statement with no outgoing edges must be linked to a query statement as shown in the following example.

Example 5.10 (Representing all conflicts). *Consider the knowledge base $\mathcal{KB}$ of Example 5.1 and its $\mathbb{SG}_{\mathcal{KB}}$ in Figure 5.1. If the query statement $(innocent(alice) \rightarrow \emptyset)$ is not added, then the conflict between $(resp(alice) \rightarrow guilty(alice))$ and $(alibi(alice) \rightarrow innocent(alice))$ would not be detected. That is why we add to any rule application statement $\pi(Body(r)) \Rrightarrow \pi(Head(r))$ that has no outgoing edges a query statement of the form $(\pi(Head(r)) \rightarrow \emptyset)$.*

Now that all conflicts are accounted for, we first apply PDL to detect ambiguous statements, then backwardly broadcast this ambiguity to any statement that is linked (by a support or attack edge) to an ambiguous

statement (cf. Figure 5.6). Labellings for Defeasible Logics start from fact statements and propagate upward towards query statements, however, for Repair Semantics, the labellings have to conduct a second pass from query statements and propagate downward towards fact statements. We use the labeling function '**IAR**' to obtain entailment results equivalent to IAR [Lembo et al., 2010]. IAR is defined as follows: edges have the same label as their source statements (i.e. given an edge e, IAR(e) = IAR($Source(e)$). Given a statement $\mathbb{s}$:

(a) ***IAR***($\mathbb{s}$) = ***IN*** *iff* *IAR*($\mathbb{s}$) $\neq$ *AMBIG and* *PDL*($\mathbb{s}$) = *INdef* [2].

(b) ***IAR***($\mathbb{s}$) = ***AMBIG*** *iff either* *PDL*($\mathbb{s}$) = *AMBIG or* $\exists e \in \mathcal{E}_S^+(\mathbb{s}) \cup \mathcal{E}_A^+(\mathbb{s})$ *such that* *IAR*($Target(e)$) = *AMBIG.*

(c) ***IAR***($\mathbb{s}$) = ***OUT*** *iff* *PDL*($\mathbb{s}$) = *UNSUP.*

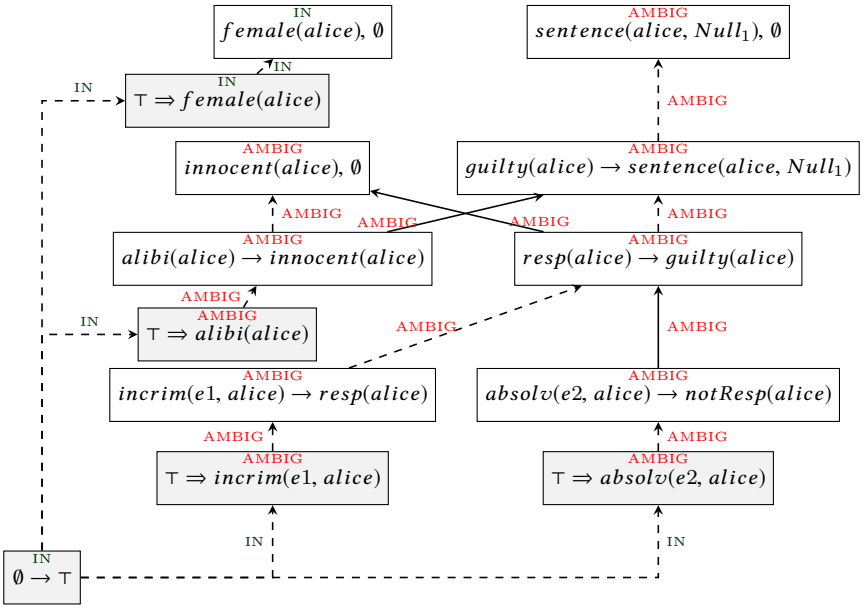

Figure 5.6: IAR applied to Example 5.1's Statement Graph.

A statement is labeled AMBIG if it was labeled ambiguous by PDL or if it leads to an ambiguous statement. Otherwise, it is IN if it has an IN complete support and is not attacked (i.e. PDL labels it INdef). The equivalence between IAR labeling function and IAR semantics is shown in the following proposition.

Proposition 5.6 (Statement Graphs and IAR). *Let $\mathcal{KB}$ be a knowledge base that contains only defeasible facts, strict rules and no preferences. Given a ground query Q:*

[2]Since all starting facts are considered defeasible, no statement can be labeled INstr.

1. $\mathcal{KB} \models_{IAR} Q$ *iff* $\mathbb{SG}^{IAR}_{\mathcal{KB}}\langle(Q \rightarrow \emptyset)\rangle = IN$.

2. $\mathcal{KB} \not\models_{IAR} Q$ *iff* $\mathbb{SG}^{IAR}_{\mathcal{KB}}\langle(Q \rightarrow \emptyset)\rangle \in \{AMBIG, OUT\}$.

Proof. cf. Proof 5.6 in Section 7.2.3 on page xv. □

5.2.2 Labeling for ICAR

The intuition behind ICAR is to reject any fact that would lead to a conflict while accepting those that would not lead to a conflict even if they were generated from ambiguous facts [Lembo et al., 2010]. From an SG point of view, any statement that is attacked or that supports statements that lead (by a support or attack edge) to attacked ones is considered "ambiguous". This is done by first applying PDL to detect ambiguous and accepted statements then the ICAR labeling starts from query statements and progresses downward towards fact statements. We use the labeling function '**ICAR**' to obtain entailment results equivalent to ICAR [Lembo et al., 2010]. ICAR is defined as follows: given an edge e, ICAR(e) = ICAR($Source(e)$). Given a statement $\mathbb{s}$:

(a) $\boldsymbol{ICAR}(\mathbb{s}) = \boldsymbol{IN}$ *iff* $ICAR(\mathbb{s}) \neq AMBIG$ *and* $PDL(\mathbb{s}) \in \{INdef, AMBIG\}$.

(b) $\boldsymbol{ICAR}(\mathbb{s}) = \boldsymbol{AMBIG}$ *iff*

1. *either* $PDL(\mathbb{s}) = AMBIG$ *and* $\exists e \in \mathcal{E}^{-}_{A}(\mathbb{s})$ *s.t.* $PDL(e) \in \{IN, AMBIG\}$,
2. *or* $\exists e \in \mathcal{E}^{+}_{S}(\mathbb{s}) \cup \mathcal{E}^{+}_{A}(\mathbb{s})$ *such that* $ICAR(Target(e)) = AMBIG$.

(c) $\boldsymbol{ICAR}(\mathbb{s}) = \boldsymbol{OUT}$ *iff* $PDL(\mathbb{s}) = UNSUP$.

A statement is labeled AMBIG if it was labeled ambiguous by PDL and it is attacked, or if it leads to an ambiguous statement. It is labeled IN if it was labeled INdef or AMBIG by PDL and is not attacked and does not lead to an ambiguous statement. Figure 5.7 shows ICAR labeling applied to the Statement Graph of Example 5.1.

Proposition 5.7 (Statement Graphs and ICAR). *Let $\mathcal{KB}$ be a knowledge base that contains only defeasible facts, strict rules and no preferences. Given a ground query Q:*

1. $\mathcal{KB} \models_{ICAR} Q$ *iff* $\mathbb{SG}^{ICAR}_{\mathcal{KB}}\langle(Q \rightarrow \emptyset)\rangle = IN$.

2. $\mathcal{KB} \not\models_{ICAR} f$ *iff* $\mathbb{SG}^{ICAR}_{\mathcal{KB}}\langle(Q \rightarrow \emptyset)\rangle \in \{AMBIG, OUT\}$.

Proof. cf. Proof 5.7 in Section 7.2.3 on page xvi. □

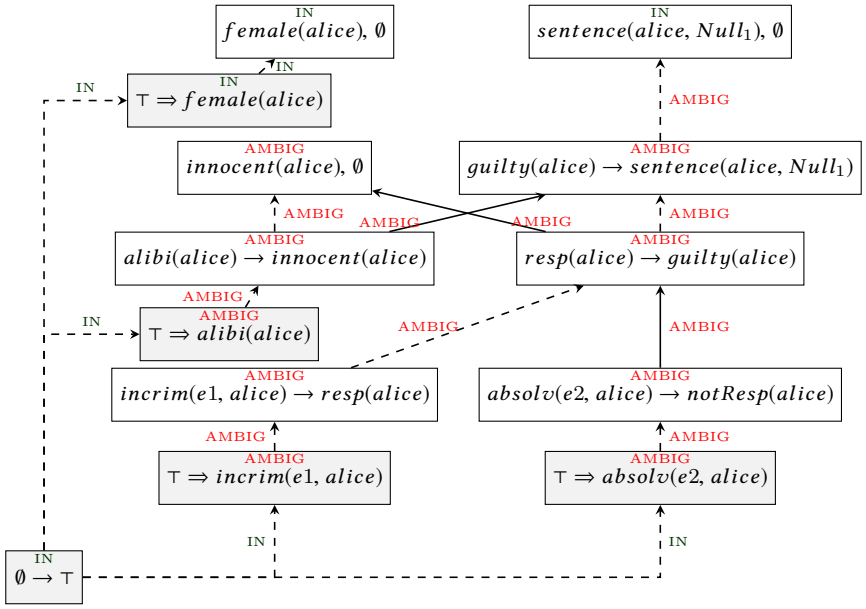

Figure 5.7: ICAR applied to Example 5.1's Statement Graph.

5.2.3 Combining Defeasible Reasoning and Repair Semantics

Defeasible Logics and Repair Semantics stem from two inherently different domains, and as such, to the best of our knowledge, they were never studied together. In this section, we first show by an experiment that there is practical value in considering intuitions from both domains. Then we show that SGs are adequately equipped to combine such notions.

5.2.3.1 Experiment

In order to get an idea of what intuitions humans follow in an abstract context, we ran an experiment with 41 participants in which they were told to place themselves in the shoes of an engineer trying to analyze a situation based on a set of sensors. These sensors (with unknown reliability) give information about the properties of an object called "o", e.g. "Object 'o' has the property P" (which could be interpreted for example as 'o' is red). Also, as an engineer, they have a knowledge that is always true about the relations between these properties, e.g. "All objects that have the property P, also have the property Q". Some of the properties cannot be true at the same time on the same object, e.g. "An object cannot have the properties P and T at the same time". Using abstract situations allowed us to avoid unwanted effects of *a priori* knowledge while at the same time representing formal concepts (facts, rules and negative constraints) in a textual simplified manner. A transcript of the original text that the experiment participants have received is shown in the following Example 5.11.

Example 5.11 (Situation 1). *Textual representation: Three sensors are respectively indicating that "o" has the properties S, Q, and T. We know that any object that has the property S also has the property V. Moreover, an object cannot have the properties S and Q at the same time, nor the properties V and T at the same time.* ***Question:*** *Can we say that the object "o" has the property T?*

Let us also provide here the logical representation of the above text. Please note that the participants have not also received the logical transcript.

- $\mathcal{F} = \{s(o), q(o), t(o)\}$
- $\mathcal{N} = \{\forall X\, s(X) \wedge q(X) \rightarrow \bot, \forall X\, v(X) \wedge t(X) \rightarrow \bot\}$
- $\mathcal{R} = \{\forall X\, s(X) \rightarrow v(X)\}$
- *Query* $Q() = t(o)$

Participants were shown in a random order 5 situations containing inconsistencies[3]. For each situation, the participants were presented with a textual description of an inconsistent knowledge base and a query (without the logical representation). Possible answers for a query were "Yes" (entailed) or "No" (not entailed). The 41 participants were second year university students in computer science with no formal background in logic, 12 females and 29 males aged between 17 and 46 years old.

Table 5.3 presents the situations and the semantics under which their queries are entailed (✓) or not entailed (−). The "% of Yes" column indicates the percentage of participants that answered "Yes". The aim of each situation is to identify if a set of semantics coincide with the majority. For example the query in Situation 1 (Example 5.11) is only entailed under $\models_{block}$. Not all cases can be represented, for example $\models_{IAR} f$ and $\not\models_{prop} f$, due to *productivity* (c.f. Section 5.1.3).

Situations	$\models_{block}$	$\models_{prop}$	$\models_{DT}$	$\models_{IAR}$	$\models_{ICAR}$	% of "Yes"	$\models_{IAR}^{block}$
#1	✓	-	-	-	-	73.17%	✓
#2	✓	✓	-	-	-	21.95%	-
#3	✓	✓	-	-	✓	21.95%	-
#4	-	-	-	-	✓	4.87%	-
#5	✓	✓	✓	✓	✓	78.04%	✓

Table 5.3: Situations Entailment and Results.

From the results in Table 5.3, we observe that blocking and IAR are the most intuitive (Situations 1 and 5), however blocking alone is not sufficient as shown by Situations 2 and 3, and IAR alone is not sufficient either (Situation 1). One possible explanation is that participants are using a semantics that is a mix of IAR and ambiguity blocking ($\models_{IAR}^{block}$). Such a semantics is absent from the literature as it is interestingly in between Repair Semantics

[3]Situations and detailed results are available in Section 7.1 on page i.

and Defeasible Reasoning. Please note that this experiment serves as an illustration of the possible applications of the new semantics, it does not claim to be a throughout validation of the link between IAR with ambiguity blocking and human reasoning. In fact, defining what is "more intuitive" is almost a lost cause (even between logicians) given the complexity of human reasoning [Horty et al., 1987]. That is why relying on abstract situations rather than real world examples might possibly isolate some of the unknown variables, remove background knowledge, and allow a (rather limited) view on what human considers a valid conclusion.

5.2.3.2 Combining Ambiguity Blocking with IAR/ICAR

Combining ambiguity blocking and IAR or ICAR can be done by applying the same intuitions as IAR or ICAR on an SG which has been labeled using BDL rather than PDL. This would amount to replacing PDL by BDL in the definitions of IAR and ICAR labellings. The resulting labeling functions are denoted IAR_{block} and ICAR_{block} respectively and are defined as follows.

- IAR:
  (a) $\boldsymbol{IAR}_{block}(\mathbb{s}) = \boldsymbol{IN}$ *iff* $IAR_{block}(\mathbb{s}) \neq AMBIG$ *and* $BDL(\mathbb{s}) = INdef.$
  (b) $\boldsymbol{IAR}_{block}(\mathbb{s}) = \boldsymbol{AMBIG}$ *iff either* $BDL(\mathbb{s}) = AMBIG$ *or* $\exists e \in \mathcal{E}_S^+(\mathbb{s}) \cup \mathcal{E}_A^+(\mathbb{s})$ *such that* $IAR_{block}(Target(e)) = AMBIG.$
  (c) $\boldsymbol{IAR}_{block}(\mathbb{s}) = \boldsymbol{OUT}$ *iff* $BDL(\mathbb{s}) = UNSUP.$
- ICAR:
  (a) $\boldsymbol{ICAR}_{block}(\mathbb{s}) = \boldsymbol{IN}$ *iff* $ICAR_{block}(\mathbb{s}) \neq AMBIG$ *and* $BDL(\mathbb{s}) \in \{INdef, AMBIG\}.$
  (b) $\boldsymbol{ICAR}_{block}(\mathbb{s}) = \boldsymbol{AMBIG}$ *iff*
    1. *either* $BDL(\mathbb{s}) = AMBIG$ *and* $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $BDL(\boldsymbol{e}) \in \{IN, AMBIG\}$,
    2. *or* $\exists e \in \mathcal{E}_S^+(\mathbb{s}) \cup \mathcal{E}_A^+(\mathbb{s})$ *such that* $ICAR_{block}(Target(e)) = AMBIG.$
  (c) $\boldsymbol{ICAR}_{block}(\mathbb{s}) = \boldsymbol{OUT}$ *iff* $BDL(\mathbb{s}) = UNSUP.$

As it turns out, IAR_{block} fully coincides with the answers given by the majority of the participants in our experiment.

Notation 5.1 (Ambiguity Blocking IAR and ICAR). *Let f be a fact in a $\mathcal{KB}$ that contains only defeasible facts, strict rules and no preferences:*

- $\mathcal{KB} \vdash_{IAR}^{block} f$ *iff* $\mathbb{SG}_{\mathcal{KB}}^{IAR_{block}}\langle(\{f\}, \emptyset)\rangle = IN.$
- $\mathcal{KB} \nvdash_{IAR}^{block} f$ *iff* $\mathbb{SG}_{\mathcal{KB}}^{IAR_{block}}\langle(\{f\}, \emptyset)\rangle \in \{AMBIG, OUT\}.$
- $\mathcal{KB} \vdash_{ICAR}^{block} f$ *iff* $\mathbb{SG}_{\mathcal{KB}}^{ICAR_{block}}\langle(\{f\}, \emptyset)\rangle = IN.$
- $\mathcal{KB} \nvdash_{ICAR}^{block} f$ *iff* $\mathbb{SG}_{\mathcal{KB}}^{ICAR_{block}}\langle(\{f\}, \emptyset)\rangle \in \{AMBIG, OUT\}.$

To illustrate this semantics, consider Example 5.12.

Example 5.12. *Applying* IAR_{block} *on Example 5.1's SG (cf. Figure 5.8) gives* $\mathcal{KB} \models^{block}_{IAR} female(alice) \wedge innocent(alice)$. *Note that the difference with BDL is that* $\mathcal{KB} \not\models^{block}_{IAR} incrim(e1, alice)$ *and* $\mathcal{KB} \not\models^{block}_{IAR} absolv(e2, alice)$. *The difference with IAR is that* $\mathcal{KB} \not\models_{IAR} innocent(alice)$.

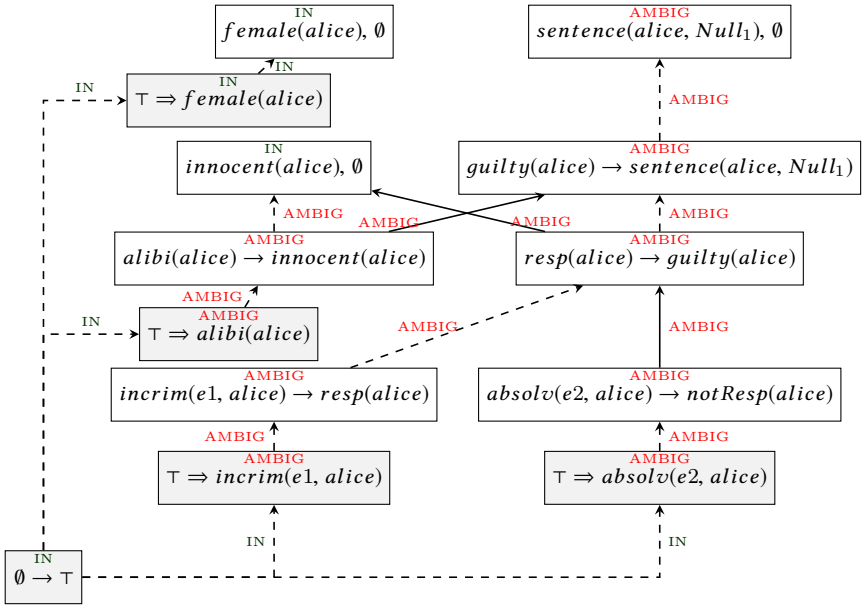

Figure 5.8: IAR applied to Example 5.1's Statement Graph.

Facts entailed by IAR are also entailed by IAR with ambiguity blocking and the same applies for ICAR as shown in the following proposition.

Proposition 5.8. *Let* $\mathcal{KB}$ *be a knowledge base that contains only defeasible facts, strict rules and no preferences. Given a ground query* Q*:*

1. *if* $\mathcal{KB} \models_{IAR} Q$ *then* $\mathcal{KB} \models^{block}_{IAR} Q$
2. *if* $\mathcal{KB} \models^{block}_{IAR} Q$ *then* $\mathcal{KB} \models^{block}_{ICAR} Q$
3. *if* $\mathcal{KB} \models_{ICAR} Q$ *then* $\mathcal{KB} \models^{block}_{ICAR} Q$

Proof. cf. Proof 5.8 in Section 7.2.3 on page xvii. □

Computing the labellings for IAR, ICAR, IAR_{block}, and $ICAR_{block}$ using SGs amounts to two breadth first graph traversals which can be done in polynomial time [Ausiello et al., 2001], combined with the SG construction, it has *PTime – complete* data complexity and 2ExpTime-complete combined complexity as shown in Proposition 4.8. Productivity comparison of Defeasible Reasoning and Repair Semantics is shown in Figure 5.9.

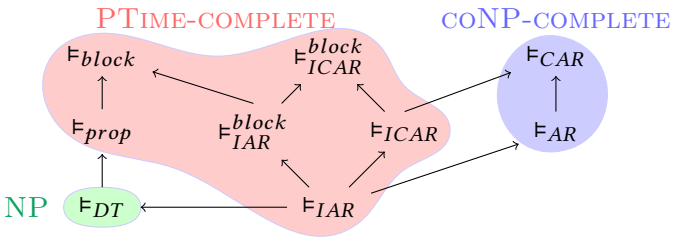

Figure 5.9: Productivity and data complexity of different semantics under FES fragment of existential rules with only defeasible facts, strict rules, and no preferences.

5.3 Human Reasoning

We conclude this chapter by a discussion on how Statement Graphs can be used to capture other forms of human reasoning, namely, the *suppression task*. A psychological study conducted in [Byrne, 1989] shows that people tend to change (suppress) previously drawn conclusions when additional information becomes available even if from a logical point of view, the new information should not affect reasoning. This suppression effect has longly been studied in cognitive computer science and can be presented in different forms. We are interested in the modus-ponens suppression task explained in Example 5.13.

Example 5.13 ([Byrne, 1989]). *Consider the following Situation 1 :*

1. *"If Lisa has an essay to write, she will study late in the library".*
2. *"Lisa has an essay to write".*

- *Will Lisa study late in the library?*

Most subjects (96%) conclude that she will study late in the library. However, if subjects receive an additional information (Situation 2):

3. *"If the library stays open, she will study late in the library".*

Only a minority (38%) concludes she will study late in the library. This is called the modus-ponens suppression effect because the rule (1) is no longer applied even though it logically should be.

This study shows that, much like non-monotonic reasoning, conclusions can be suppressed in human reasoning in presence of additional information. Since humans tend to rely on background knowledge, the first step of human reasoning is reasoning towards an appropriate logical representation of the

situation [Stenning and Lambalgen, 2005]. This is an especially important step since the suppression effect can occur either as a consequence of a suitable reasoning mechanism, or due to specific logical representation of the situation [Ragni et al., 2017]. Extracting what background knowledge has been used by humans to reason on a situation is hard and subjective, therefore, one is only left with intuition to justify a certain representation. That is why we use the "plain" direct representation [Ragni et al., 2017] where each sentence is a rule and the background knowledge rule *"if the library does not stay open then Lisa will not study late in the library"* (the contrapositive of rule (3.)) is added as shown in Example 5.14.

Example 5.14. *Consider a representation of Example 5.13 such that:*

- *"essay" denotes "She has an essay to write".*
- *"library" denotes "She will study late in the library".*
- *"open"denotes "Library stays open".*

Let $\mathcal{KB}_1 = (\mathcal{F}, \mathcal{R}_1, \emptyset, \emptyset)$ *be the representation of Situation 1:*

- $\mathcal{F} = \{\top \rightarrow essay\}$
- $\mathcal{R}_1 = \{r_1 : essay \Rightarrow library\}$

Let $\mathcal{KB}_2 = (\mathcal{F}, \mathcal{R}_2, \mathcal{N}, \succ)$ *be the representation of Situation 2:*

- $\mathcal{R}_2 = \mathcal{R}_1 \cup \{r_2 : open \Rightarrow library,\ r_3 : \neg open \Rightarrow \neg library\}$.
- $r_3 \succ r_1$.

Figures 5.10 and 5.11 display the Statement Graphs of the logical representations of Situations 1 and 2.

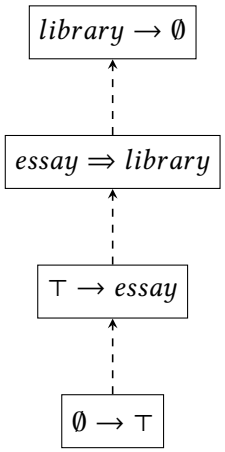

Figure 5.10: $\mathbb{SG}_{\mathcal{KB}_1}$ of Example 5.14.

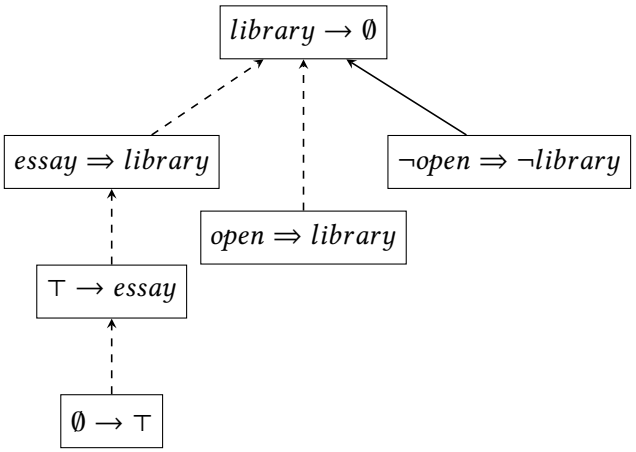

Figure 5.11: $\mathbb{SG}_{\mathcal{KB}_2}$ of Example 5.14.

Defeasible reasoning (in all its variants) **cannot represent the suppression task** as "*library*" is derivable and not defeasibly contested (i.e $\mathcal{KB}_1 \vdash \mathit{library}$ and $\mathcal{KB}_2 \vdash \mathit{library}$). Most logicians would agree that *library* should still be logically derivable in Situations 1 and 2 [Dietz et al., 2014, Stenning and Van Lambalgen, 2012, Ragni et al., 2017].

A possible explanation for the modus-ponens suppression effect is that humans consider unsupported counter-arguments as valid attacks: they think that the library might be closed and therefore cannot conclude that Lisa will study in the library. Let us show in the remainder of the section how such reasoning behavior could be captured by a labeling function of the SGs. More precisely, this can be represented by a labeling function (denoted SUP) that takes into account UNSUP attack edges if they are superior to the support edges, which makes the attacked statement AMBIG. For simplicity, we define SUP using the same definitions of BDL for the labels INstr, OUTstr, OUTdef, AMBIG and UNSUP. Keep in mind however that a labeling function based on the same intuition as SUP but using ambiguity propagating (with or without team defeat) would also effectively model the modus-ponens suppression effect. Given an edge e, SUP(e) = SUP($\mathit{Source}(e)$). Given a statement $\mathbb{s}$:

(a) $\boldsymbol{SUP}(\mathbb{s}) = \boldsymbol{INstr}$ *if* $\mathbb{s}$ *has an INstr complete support, has a strict rule* $\mathit{Rule}(\mathbb{s})$*, and* $\nexists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $\mathit{SUP}(\boldsymbol{e}) = \mathit{INstr}$.

(b) $\boldsymbol{SUP}(\mathbb{s}) = \boldsymbol{OUTstr}$ *iff* $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $\mathit{SUP}(\boldsymbol{e}) = \mathit{INstr}$.

(d) $\boldsymbol{SUP}(\mathbb{s}) = \boldsymbol{OUTdef}$ *iff* $\mathit{BDL}(\mathbb{s}) \neq \mathit{OUTstr}$ *and* $\mathbb{s}$ *has an INstr or INdef complete support and*

1. *either* $\exists f \in \mathit{Premise}(\mathbb{s})$ *where* $\nexists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $\mathit{SUP}(\boldsymbol{e_s}) = \mathit{INstr}$ *and* $\forall \boldsymbol{e'_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $\mathit{SUP}(\boldsymbol{e'_s}) \in \{\mathit{INdef}, \mathit{AMBIG}\}$, $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *attacking* $\mathbb{s}$ *on* f *s.t.* $\mathit{SUP}(\boldsymbol{e}) = \mathit{INdef}$ *and* $\boldsymbol{e}$ *is superior to* $\boldsymbol{e'_s}$.
2. *or* $\exists \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $\mathit{SUP}(\boldsymbol{e}) = \mathit{INdef}$ *attacking the rule application of* $\mathbb{s}$ *and* $\mathit{Rule}(\mathbb{s})$ *is not a strict rule and* $\mathit{Rule}(\mathit{Source}(\boldsymbol{e})) \succ \mathit{Rule}(\mathbb{s})$.

(e) $\boldsymbol{SUP}(\mathbb{s}) = \boldsymbol{AMBIG}$ *iff* $\mathit{SUP}(\mathbb{s}) \notin \{\mathit{INstr}, \mathit{OUTstr}, \mathit{INdef}, \mathit{OUTdef}, \mathit{UNSUP}\}$ *and* $\mathbb{s}$ *is not part of a support cycle and is part of an attack cycle.*

(f) $\boldsymbol{SUP}(\mathbb{s}) = \boldsymbol{UNSUP}$ *iff* $\mathit{SUP}(\mathbb{s}) \notin \{\mathit{INstr}, \mathit{OUTstr}, \mathit{INdef}, \mathit{OUTdef}, \mathit{AMBIG}\}$ *and* $\mathbb{s}$ *is part of a support cycle.*

The key difference for representing the modus-ponens suppression effect is to change the INdef labeling to account for unsupported attack edges, that is why we add the conditions (c.3) and (c.4) stating that for any attack edge **e** s.t. SUP(**e**) = UNSUP, if **e** attacks a premise then there must exist a support edge for that premise that is not inferior to **e**, and if **e** attacks the rule, then the rule must not be inferior to **e**.

(c) ***BDL***$(\mathbb{s})$ = ***INdef*** *iff* $BDL(\mathbb{s}) \notin \{INstr, OUTstr\}$ *and* $\mathbb{s}$ *has a INstr or INdef complete support and*

1. $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *that undercut* $\mathbb{s}$ *on a premise* $f \in \textit{Premise}(\mathbb{s})$ *s.t.* $BDL(\boldsymbol{e}) = INdef$, $\exists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $BDL(\boldsymbol{e_s}) = INstr$ *or* $BDL(\boldsymbol{e_s}) = INdef$ *and* $\boldsymbol{e_s}$ *is superior to* $\boldsymbol{e}$.
2. *and* $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $BDL(\boldsymbol{e}) = INdef$ *and* $\boldsymbol{e}$ *attacks the rule application of* $\mathbb{s}$, $\textit{Rule}(\mathbb{s})$ *is either a strict rule or* $\textit{Rule}(\mathbb{s}) \succ \textit{Rule}(\textit{Source}(\boldsymbol{e}))$.
3. *and* $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t* $SUP(\boldsymbol{e}) = UNSUP$ *and* $\boldsymbol{e}$ *undercuts* $\mathbb{s}$ *on* f, $\exists \boldsymbol{e_s} \in \mathcal{E}_S^-(\mathbb{s})$ *for* f *s.t.* $\boldsymbol{e_s}$ *is not inferior to* $\boldsymbol{e}$.
4. *and* $\forall \boldsymbol{e} \in \mathcal{E}_A^-(\mathbb{s})$ *s.t.* $SUP(\boldsymbol{e}) = UNSUP$ *and* $\boldsymbol{e}$ *attacks the rule application of* $\mathbb{s}$, $\textit{Rule}(\mathbb{s})$ *is either a strict rule or* $\textit{Rule}(\textit{Source}(\boldsymbol{e})) \not\succ \textit{Rule}(\mathbb{s})$.

Applying the SUP labeling function yields $\mathbb{SG}^{\text{SUP}}_{\mathcal{KB}_1}\langle(\textit{library} \rightarrow \emptyset)\rangle$ = INdef and $\mathbb{SG}^{\text{SUP}}_{\mathcal{KB}_2}\langle(\textit{library} \rightarrow \emptyset)\rangle$ = AMBIG (as shown in Figures 5.12 and 5.13) which correctly models the modus ponens suppression effect.

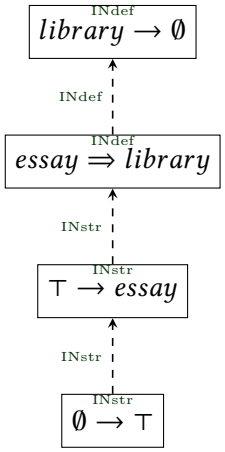

Figure 5.12: $\mathbb{SG}^{\text{SUP}}_{\mathcal{KB}_1}$ of Example 5.14.

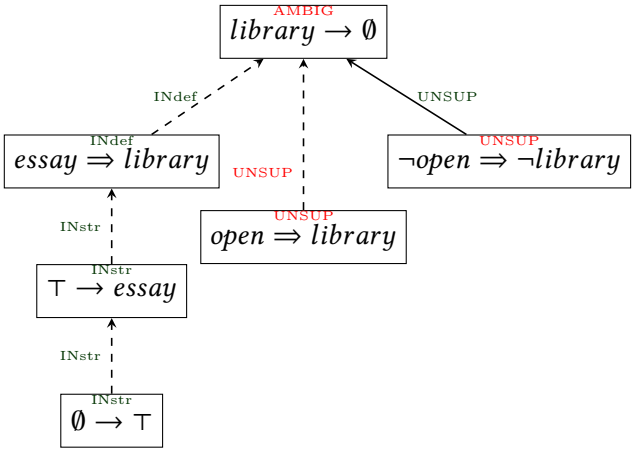

Figure 5.13: $\mathbb{SG}^{\text{SUP}}_{\mathcal{KB}_2}$ of Example 5.14.

Please note that we defined SUP using BDL but there is no proof that human use ambiguity blocking with team defeat. However we make this assumption for simplicity. A labeling function based on the same intuition as SUP but using ambiguity propagating (with or without team defeat) would also effectively model the modus-ponens suppression effect. Empirical data and more testing are needed to justify one or the other.

Finally, let us note again that while the suppression effect can occur either as a consequence of a suitable reasoning mechanism or due to specific logical representation of the situation [Ragni et al., 2017], we made sure to use the "plain" representation where only the background knowledge is added [Ragni et al., 2017]. Other representations can be used such as the

"necessary condition" by using the rule "$essay \wedge open \Rightarrow library$" or the weak completion semantics and adding an abnormality predicate [Dietz et al., 2014].

5.4 Summary

Defeasible logics and Repair Semantics stem form different needs and are studied in distinct domains. A first attempt to apply Defeasible Reasoning to the Database domain is made in [Deagustini et al., 2015] where it is shown that Repair Semantics produce no useful answers under incoherence. However, the link between defeasible logics and Repair Semantics when both can be applied is not studied. In this chapter we investigated the link between Defeasible Reasoning and Repair Semantics, we showed that they are not so different and can both be applied to inconsistent but coherent knowledge bases under the restriction of defeasible facts, strict rules, and no preferences. We started by showing the productivity link between theses techniques (summarized in Figure 5.5), namely:

- Proposition 5.4: everything that is IAR entailed is also entailed by Defeasible Reasoning with ambiguity propagation and by grounded semantics; the same cannot be said for ICAR semantics.
- Proposition 5.5: everything that is IAR entailed is also entailed by Dialectical Trees, and everything that is entailed by Dialectical Trees is also entailed by Defeasible Reasoning with ambiguity propagation and by grounded semantics (cf. Figure 5.5).

We then showed how Statement Graphs can represent IAR and ICAR semantics by first applying the labeling function for Defeasible Reasoning with ambiguity propagation then discarding statements that lead to conflicts. Afterward, we ran an experiment with 41 participants in order to understand what intuitions humans follow in abstract inconsistent situations. The results seem to indicate that ambiguity blocking and IAR are the most intuitive (cf. Table 5.3), however blocking alone or IAR alone are not sufficient to account for all entailment results. A possible explanation is that participants are using a semantics that is a mix of IAR and ambiguity blocking. Such a semantics is absent from the literature as it is interestingly "in-between" Repair Semantics and Defeasible Reasoning. This experiment serves as an illustration to the possible applications of combining both approaches, but it does not claim to be a thorough validation of human reasoning since defining what is "more intuitive" is particularly difficult.

Motivated by the experiment results, we defined labellings that combines IAR or ICAR with ambiguity blocking, we studied their complexity and productivity links with regards to Defeasible Reasoning and Repair Semantics techniques (summarized in Figure 5.9). These new semantics display the

flexibility of Statement Graphs and the next step was to check if SGs can represent human reasoning that neither Defeasible Reasoning nor Repair Semantics can represent, namely, the suppression task.

The suppression task is a psychological study conducted in [Byrne, 1989] that shows that people tend to change (suppress) previously drawn conclusions when additional information becomes available, even if from a logical point of view, this new information should not affect reasoning. A possible explanation for this effect is that human tend to consider exception even if they are not supported. From an SG point of view, this means that attacks from unsupported statements should be considered valid attacks if they are superior to the support edges. We defined a labeling function called SUP that is able to represent the suppression effect under the "plain" logical representation of the situation presented to the participants.

The results in this chapter allowed us to make the link between Defeasible Reasoning, Repair Semantics, and some forms of human reasoning. The labellings for IAR and ICAR are included in the ELDR tool giving more options for conflict-tolerant reasoning with existential rules.

Chapter 5 in a Nutshell

- *Defeasible Reasoning and Repair Semantics can be compared under the restrictions of strict rules, defeasible facts and no preferences (cf. Propositions 5.4, 5.5 and Figure 5.9 that display the productivity links between the different techniques).*
- *Statement Graphs can represent the IAR and ICAR Repair Semantics by first applying the labeling function for Defeasible Reasoning with ambiguity propagation then removing statements that lead to conflicts.*
- *Statement Graphs provide a unifying representation to obtain equivalent entailment as some Defeasible Reasoning and Repair Semantics techniques, which allows us to combine the intuitions of both approaches. Specifically, IAR and ICAR Repair Semantics can be combined with the ambiguity blocking intuition of Defeasible Reasoning. The resulting semantics seems to coincide with human reasoning under abstract situations as supported by the empirical results of the experiment in Section 5.2.3.*
- *Statement Graphs can be applied to other forms of human reasoning that neither Defeasible Reasoning nor Repair Semantics can represent, namely, the suppression task where valid logical conclusions are suppressed. We proposed to represent this effect by considering unsupported attacks as valid if they are more preferred.*

6 Conclusion

In this thesis we set out to answer a seemingly simple research problem stemming from the need to *preserve the ability to answer queries in presence of conflicts in a knowledge base expressed using existential rules*. Conflicts can arise at two different levels: *inconsistence* when the factual knowledge is incorrect, and *incoherence* when the rules themselves are contradictory. The problem of inconsistence has been addressed for existential rules using the various Repair Semantics. The problem of incoherence has not been addressed in a throughout manner for existential rules, since repair semantics fail to provide answers when the set of rules is incoherent.

Research Question

How can we preserve the ability to reason in an inconsistent or incoherent knowledge base expressed using existential rules?

Our research hypothesis is that Defeasible Reasoning provides a way to retain the ability to reason in presence of inconsistence and incoherence. However, there is no universal way of reasoning in presence of conflict, that is why we have to account for the different intuitions that can be adopted. Our research question can be refined as follows:

1. How can Defeasible Reasoning techniques be applied to Existential Rules?
2. Can we provide formalisms and tools that allow for defeasible reasoning with different intuitions under existential rules?

6.1 Scope

Since our research problem vast, its scope had to be restricted in order to address it in a reasonable manner:

- *Existential Rules* are a first order logical language that emerged from the intersection of Knowledge Representation, Database Systems, and Semantic Web. It has the ability to express knowledge about "unknown" individuals. This level of expressiveness comes at the high cost of undecidability (the reasoning mechanism can be infinite), that is why different decidable fragments of existential rules have been defined: Finite Expansion Set (FES) guaranteeing a finite forward chaining mechanism (chase), Finite Unification Set (FUS) guaranteeing a finite backward chaining mechanism, and Greedy Bounded Treewidth Set (GBTS) guaranteeing a finite forward-like inference mechanism. The first choice that was made is to *focus on the forward chaining mechanism* which is arguably the most intuitive one given its ability to handle transitive rules [Rocher, 2016]. The second choice is *what type of chase do we want to use?* There are four kinds of chases: Oblivious, Frontier/Skolem, Restricted, and Core chases. While we addressed most of them for the derivation loss problem, we focused in the remainder of the thesis on the Frontier/Skolem chase which is the most used chase given its relatively low cost and its ability to stay decidable for all known concrete classes of the FES fragment [Baget et al., 2011a].

- *Defeasible Reasoning* is an efficient form of non-monotonic reasoning. There are various techniques to achieve this kind of reasoning; we focused on *Defeasible Logics* since they can represent most of the intuitions of defeasible reasoning, *Dialectical Trees* since they are the only attempt to apply defeasible reasoning to existential rules, and argumentation's *Grounded Semantics* under the instantiation of [Governatori et al., 2004] since, in particular cases, they coincide with Defeasible Logics. Of course these are not the only techniques for defeasible reasoning, argumentation can be instantiated and defined in different ways, but we focused on the previous techniques because they provide an implemented reasoning tool that we can compare to.

- *Repair Semantics* are the first inconsistency-tolerant techniques applied to existential rules. AR semantics is the most intuitive and used one [Lembo et al., 2010]. We focused on IAR and ICAR as they are a less costly approximation of AR.

6.2 Summary and Contributions

To achieve our aim we provided three main contributions:

1. We demonstrated why one cannot simply directly apply defeasible reasoning techniques to existential rules (derivation loss problem in Chapter 3).

2. We provided a unifying formalism that takes into account the specificities of existential rules and can represent most defeasible reasoning intuitions (Statement Graphs). We also defined the first benchmark for first order defeasible reasoning tools that allows the analysis and classification of the implementations. Moreover, provides a clear overview on the gaps not covered by these tools (Chapter 4).

3. We showed that Repair Semantics and Defeasible Reasoning can be seen as two particular cases of a more general mechanism of conflict-tolerant reasoning. This mechanism can be represented using Statement Graphs which can be used to combine both approaches in order to produce new semantics (Chapter 5).

In Chapter 2 we reviewed various Defeasible Reasoning techniques and provided links and connections between them such as their complexity, the intuitions they follow, and under which restrictions they might coincide.

In Chapter 3, we showed the Derivation Loss problem (where the derivation reducer of the chase might remove some rule applications and subsequently lose derivations) which prevents the direct application of defeasible reasoning techniques to existential rules. To solve this problem, we defined the notion of Graph of Atom Dependency (GAD) for the different kinds of chases, and described how it can be used to extract all derivations. Using GAD we presented the first defeasible reasoning tool for existential rules based on Dialectical Trees (called DEFT) and we compared its features and performance to other defeasible reasoning tools using a dedicated benchmark. This benchmark provides a clear view of what the tools allow for (e.g. ambiguity handling, team defeat, preferences, cycles, etc.), and thus can be used by a data engineer to choose the best tool to use depending on the data and requirements at hand; for instance, DEFT is recommended for ambiguity propagation with team defeat due to its satisfactory performance in this context. Moreover the benchmark provides insights about the current gaps that are not covered yet by any tool.

Based on the gaps we uncovered previously, and given the fact that it would be tedious to extend every defeasible reasoning technique to existential rules, we introduced in Chapter 4 a unifying formalism that is able to represent ambiguity blocking or propagation, with or without team defeat,

and that can handle cycles: Statement Graph. It also takes into account the specificities of existential rules and avoids the problem of derivation loss. Implementing Statement Graphs in a tool (called ELDR) answered most of the gaps identified in the previous chapter. This tool has been shown to have satisfactory performance given the results of the benchmark.

Following this idea of unifying these formalisms, we investigated the link between Defeasible Reasoning and Repair Semantics and showed in Chapter 5 that these approaches can be compared under the restrictions of strict rules, defeasible facts and no preferences. We studied the productivity links between Defeasible Reasoning techniques and Repair Semantics and used Statement Graphs to represent the entailment of IAR and ICAR repair semantics. Once represented in the unifying formalism, it was a natural consequence to combine them with the intuitions of defeasible reasoning. The resulting hybrid semantics seem to coincide with human reasoning as supported by the empirical experiment. Moreover we showed that Statement Graphs can represent other forms of human reasoning that neither Defeasible Reasoning nor Repair Semantics are able to represent, namely, the suppression task where valid logical conclusions are suppressed depending on the context.

6.3 Perspectives

While we successfully answered the significant research question we set out to address, our contributions open interesting avenues for future work.

1. **Existential Rules.** Forward chaining where rules are applied over a starting set of facts to generate new knowledge is not the only way to reason with existential rules. Backward chaining can also be used. In this context, the query is rewritten using the rules until it is transformed into a conjunction of starting facts. The questions that can be asked here are: is backward chaining prone to derivation loss? If so, how can this be prevented? The first question can be easily answered: yes. Indeed, while a derivation reducer removes rule applications, backward chaining removes rule rewritings which might lead to derivation loss as shown by this simple example:

   Example 6.1 (Derivation Loss in Backward Chaining). *Consider the following knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \emptyset, \emptyset)$ *stating that animals that lay eggs are generally birds, and animals that lay eggs and have feathers are definitely birds. Suppose we have an animal named Tweety that lays eggs and has feathers. Is Tweety a bird* ($Q = bird(tweety)$)*?*

   - $\mathcal{F} = \{layEggs(tweety), hasFeathers(tweety)\}$

- $\mathcal{R} = \{r_1 : \forall X\, layEggs(X) \Rightarrow bird(X),$
  $r_2 : \forall X\, layEggs(X) \wedge hasFeathers(X) \rightarrow bird(X)\}$

In backward chaining, a query is rewritten using a rule if the query can be mapped to the head of rule. If the resulting atoms are more specific than some previously generated ones then they are removed in order to ensure termination of the algorithm [Thomazo, 2013]. The sequence of rewriting for the query Q can be represented in a graph (Figure 6.1) where nodes represent rewritten queries and edges represent the rules and homomorphisms used to rewrite. The rule r_1 can be used to rewrite Q into $Q_1 = layEggs(tweety)$, and the rule r_2 can be used to rewrite Q into $Q_2 = layEggs(tweety) \wedge hasFeathers(tweety)$. The rewritten query Q_2 is more specific to Q_1 since Q_1 can be mapped to Q_2, therefore the rewriting using r_2 is removed. If we want to extract the derivations for $bird(tweety)$ we find only one derivation (the one using r_1). Thus backward chaining is prone to derivation loss.

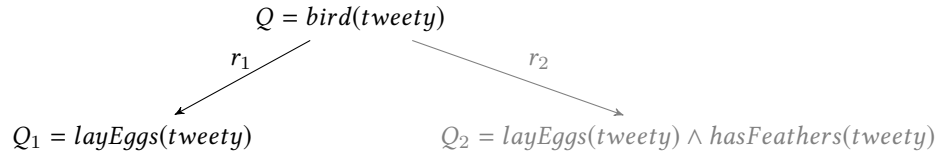

Figure 6.1: Backward chaining graph of Example 6.1 (the rewriting using r_2 is removed and displayed in gray for clarity)

Another interesting thing to consider is the fact that derivation loss is not only problematic for defeasible reasoning, it is also a problem for any situation where more than simple entailment is needed, in particular when the path of reasoning for this entailment is needed as well. For instance, we previously worked on query explanation [Hecham et al., 2017a] and in this context, it is not just entailment that is important but also the reasoning behind it. Derivation loss in this case may gravely impede the purpose of this enterprise.

Moreover, since repairs can be expressed using consistent derivations [Croitoru and Vesic, 2013], it seems that the Graph of Atom Dependency can be used to compute minimal conflicting sets by considering the query $Q = \bot$. These minimal conflicting sets can then be used to compute repairs in a manner that is more efficient that considering all possible subsets of starting facts.

2. **Defeasible Reasoning.** The discussed techniques for defeasible reasoning are not the only existing ones. An interesting research topic would be considering those that follow rationality postulates [Caminada and Amgoud, 2007]. This requires defining arguments for a

conjunction of atoms with possibly material implication to avoid disjunction. Another topic worth studying is to consider other semantics of argumentation, such as preferred semantics which seem to coincide with AR repair semantics, or other instantiations of argumentation such as Assumption Based Argumentation (ABA) [Toni, 2014] and its different ways of handling preferences.

3. **Repair Semantics.** In order to continue our work on unifying Defeasible Reasoning with Repair Semantics, it is necessary to check whether it is possible to extend Statement Graphs labeling functions to Represent AR and CAR semantics. This would allow the creation of new semantics by combining defeasible reasoning intuitions with AR and CAR (and possibly more). Furthermore, in recent work, repair semantics that include preferences over facts have been defined [Bourgaux, 2016], it would be interesting to consider how these semantics could coincide with Defeasible Reasoning. Moreover, this would allow us to combine the different intuitions on preferences (e.g. team defeat) with these semantics through the prism of Statement Graphs.

4. **Human Reasoning.** We studied in this thesis one particular form of the suppression task, namely the modus-ponens suppression task. Continuing this effort would include extending Statement Graphs to other forms of suppression task (modus-tollens, denial of antecedent, and affirmation of consequent) and potentially the selection task [Ragni et al., 2017]. Finally, we worked on defining a formal model for human reasoning [Bisquert et al., 2017, Bisquert et al., 2016, Hecham et al., 2016] based on the psychological model of Dual Process Theory [Evans, 2003, Kahneman, 2011]. This model includes a set of strict rules describing System 2 (reasoning part that is slow but precise) and defeasible rules describing System 1 (reasoning part that is fast but logically sloppy). A possible continuation of this work is to include Statement Graphs and their ability to represent different intuitions into the reasoning process of this model.

7 Appendix

This chapter contains details about the human reasoning experiment of Chapter 5 and the proofs for the propositions in different chapters.

7.1 Experiment of Chapter 5

We ran an experiment with 41 participants in which they were told to place themselves in the shoes of an engineer trying to analyze a situation based on a set of sensors, these sensors (which we ignore the reliability) give information about the properties of an object called "o", e.g. "Object 'o' has the property P" (which could be for example, 'o' is red). Also, as an engineer, they have a knowledge that is always true about the relations between these properties, e.g. "All objects that have the property P, also have the property Q". Some of the properties cannot be true at the same time on the same object, e.g. "An object cannot have the properties P and T at the same time". Using abstract situations allows us to avoid unwanted effects of a priori knowledge.

Participants were shown in a random order 5 situations containing inconsistencies. For each situation, the participant is presented with a textual description of an inconsistent knowledge base and a query. Possible answers for a query is "Yes" (entailed) or "No" (not entailed). The 41 participants are second year university students in computer science, 12 female and 29 male aged between 17 and 46 years old.

Table 5.3 represents the situations and the semantics under which their queries are entailed (✓) or not entailed (−). The "% of Yes" column indicates the percentage of participants that answered "Yes". The aim of each

situation is to identify if a set of semantics coincides with the majority, for example, the query in Situation 1 is only entailed under $\models_{block}$. Not all cases can be represented, for example $\models_{IAR} f$ and $\not\models_{prop} f$, due to *productivity*.

Situation 1. Textual representation:

- $Sensor_1$: "o" has the property S.
- $Sensor_2$: "o" has the property Q.
- $Sensor_3$: "o" has the property T.
- Any object that has the property S also has the property V.
- An object cannot have the property S and Q at the same time.
- An object cannot have the property V and T at the same time.
- **Question:** Can we say that the object "o" has the property T?

Logical representation:

- $\mathcal{F} = \{s(o), q(o), t(o)\}$
- $\mathcal{R} = \{\forall X\, s(X) \rightarrow v(X)\}$
- $\mathcal{N} = \{\forall X\, s(X) \wedge q(X) \rightarrow \bot, \forall X\, v(X) \wedge t(X) \rightarrow \bot\}$
- Query $Q = t(o)$

Situation 2. Textual representation:

- $Sensor_1$: "o" has the property W.
- $Sensor_2$: "o" has the property X.
- Any object that has the property W also has the property Y.
- Any object that has the property X also has the property Z.
- An object cannot have the property Y and Z at the same time.
- **Question:** Can we say that the object "o" has the property W?

Logical representation:

- $\mathcal{F} = \{w(o), x(o)\}$
- $\mathcal{R} = \{\forall X\, w(X) \rightarrow y(X), \forall X\, x(X) \rightarrow z(X)\}$
- $\mathcal{N} = \{\forall X\, y(X) \wedge z(X) \rightarrow \bot\}$
- Query $Q = w(o)$

Situation 3. Textual representation:

- $Sensor_1$: "o" has the property M.
- $Sensor_2$: "o" has the property J.
- $Sensor_3$: "o" has the property T.
- Any object that has the property M also has the property S.
- Any object that has the property J also has the property D.
- Any object that has the property J also has the property L.
- An object cannot have the property S and D at the same time.
- **Question:** Can we say that the object "o" has the property L?

Logical representation:

- $\mathcal{F} = \{m(o), j(o), t(o)\}$
- $\mathcal{R} = \{\forall X\, m(X) \rightarrow s(X),\ \forall X\, j(X) \rightarrow d(X),$
  $\forall X\, j(X) \rightarrow l(X)\}$
- $\mathcal{N} = \{\forall X\, s(X) \wedge d(X) \rightarrow \bot\}$
- Query $Q = l(o)$

Situation 4. Textual representation:

- $Sensor_1$: "o" has the property E.
- $Sensor_2$: "o" has the property T.
- Any object that has the property T also has the property C.
- An object cannot have the property E and T at the same time.
- **Question:** Can we say that the object "o" has the property C?

Logical representation:

- $\mathcal{F} = \{e(o), t(o)\}$
- $\mathcal{R} = \{\forall X\, e(X) \rightarrow c(X)\}$
- $\mathcal{N} = \{\forall X\, e(X) \wedge t(X) \rightarrow \bot\}$
- Query $Q = c(o)$

Situation 5. Textual representation:

- $Sensor_1$: "o" has the property F.
- $Sensor_2$: "o" has the property G.
- $Sensor_3$: "o" has the property H.
- Any object that has the property H also has the property Z.
- An object cannot have the property Z and G at the same time.
- **Question:** Can we say that the object "o" has the property F?

Logical representation:

- $\mathcal{F} = \{f(o), g(o), h(o)\}$
- $\mathcal{R} = \{\forall X\, h(X) \rightarrow z(X)\}$
- $\mathcal{N} = \{\forall X\, z(X) \wedge g(X) \rightarrow \bot\}$
- Query $Q = f(o)$

Situations	Nbr of "Yes"	Nbr of "No"	% of "Yes"
#1	30	11	73.17%
#2	9	32	21.95%
#3	9	32	21.95%
#4	2	39	4.87%
#5	32	9	78.04%

Table 7.1: Experiment Results.

7.2 Proofs

7.2.1 Chapter 2

Proposition 2.4 (Dialectical Trees and Defeasible Logic) Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R}, \mathcal{N}, \emptyset)$ expressed in $\mathcal{L}_p$ or $\mathcal{L}_\forall$ without defeater rules, strict rules or a preference relation:

- $\mathcal{KB} \vdash +\delta_{prop} f$ iff $\mathcal{KB} \models_{DT} f$.
- $\mathcal{KB} \vdash -\delta_{prop} f$ iff $\mathcal{KB} \not\models_{DT} f$.

Proof 2.4. We split the proof of (1.) in two, first we prove by induction that if $\mathcal{KB} \vdash +\delta_{prop} f$ then $\mathcal{KB} \models_{DT} f$:

- Inductive base: $\mathcal{KB} \vdash +\delta_{prop} f$ means that $Proof(1) = +\delta_{prop} f$. Given the fact that there is only defeasible rules with no preferences, this implies that there a defeasible rule of the form $\top \Rightarrow f$ (fact rule) and no fact rule for a literal in conflict with f (otherwise f would be ambiguous), which means that there is an argument $\langle \{\top \Rightarrow f\}, f \rangle$ for f that has no defeater, therefore there is a warranted argument for f, thus $\mathcal{KB} \models_{DT} f$.
- Inductive step: Let us suppose that the proposition holds for $Proof(1..i)$ and $Proof(i+1) = +\delta f$, therefore

1. there is a defeasible rule $r \in \mathcal{R}$ s.t. $Head(r) = f$ and $\forall \phi \in \mathcal{B}(r)$: $+\delta_{prop}\phi$. By the inductive hypotheses we have that $\forall \phi \in \mathcal{B}(r)$, $\mathcal{KB} \vDash_{DT} \phi$. This means that there is an argument for f such that none of its sub-arguments (except the argument itself) has a defeater, and
2. $\forall r' \in \mathcal{R}$ s.t. $Head(r')$ is in conflict with f: $\exists \phi' \in Body(r')$ s.t. there is no rule applications for ϕ', which means that r' is not applicable, implying that there is no argument attacking the argument for f, therefore it is undefeated.

From 1. and 2. there is an argument for f that is warranted, thus $\mathcal{KB} \vDash_{DT} f$.

Now we prove that if $\mathcal{KB} \vDash_{DT} f$ then $\mathcal{KB} \vdash +\delta_{prop} f$: $\mathcal{KB} \vDash_{DT} f$ means that there is an argument for f that is warranted (undefeated). Given the fact that there is only defeasible rules with no preferences, all defeats are blocking defeats, meaning that a literal can only have a warranted argument if it has no defeater since blocking defeats cannot be prevented using blocking defeaters. This implies that there is an argument for f that has no defeater i.e. there is a derivation for f and no derivation for a literal in conflict with any literal in the derivation for f, therefore $\mathcal{KB} \vdash +\delta_{prop} f$.

From (1.) the proposition (2.) directly holds (contra-positive) since $\mathcal{KB} \nvdash +\delta_{prop} f$ means $\mathcal{KB} \vdash -\delta_{prop} f$ [Billington, 1993]. □

7.2.2 Chapter 4

Proposition 4.1 Let f be a literal in a defeasible $\mathcal{KB}$ expressed in $\mathcal{L}_p$ that contains no attack or support cycles:

1. $\mathcal{KB} \vdash +\Delta f$ iff $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$
2. $\mathcal{KB} \vdash -\Delta f$ iff $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq \text{INstr}$
3. $\mathcal{KB} \vdash +\delta^{TD}_{block} f$ iff $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$
4. $\mathcal{KB} \vdash -\delta^{TD}_{block} f$ iff $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{OUTstr}, \text{OUTdef}, \text{AMBIG}, \text{UNSUP}\}$

Proof 4.1. We split the proof of (1.) in two, first we prove by induction that if $\mathcal{KB} \vdash +\Delta f$ then $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f, \emptyset)\rangle = \text{INstr}$:

- Inductive base: $Proof(1) = +\Delta f$. This implies that there is a strict rule of the form $\top \rightarrow f$, which means that $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}$ contains the statement $(\top \rightarrow f)$ that is labeled INstr since it has a support edge labeled INstr coming from the top statement, therefore the statement $(f \rightarrow \emptyset)$ has a complete strict support, thus $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$.

- Inductive step: Let us suppose that the proposition holds for $Proof(1..i)$ and $Proof(i+1) = +\Delta f$, therefore there is a strict rule $r \in \mathcal{R}_{\rightarrow}$ s.t. $Head(r) = f$ and $\forall \phi \in Body(r) : +\Delta\phi$. By the inductive hypotheses we have that $\forall \phi \in Body(r)$, $\exists \mathbb{s}' = (Body(r') \rightarrow Head(r'))$ s.t. $Head(r') = \phi$ and $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle \mathbb{s}' \rangle$ = INstr. This means that the statement $\mathbb{s} = Body(r) \rightarrow Head(r)$ has a INstr complete support i.e. $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle () \rangle$ = INstr, therefore $(f \rightarrow \emptyset)$ has a INstr complete support, thus $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle (f, \emptyset) \rangle$ = INstr.

Now we prove by induction that if $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle (f, \emptyset) \rangle$ = INstr then $\mathcal{KB} \vdash +\Delta f$:

- Inductive base: $(f \rightarrow \emptyset)$ has a INstr complete support from the top statement, this means that there is a strict rule of the form $\top \rightarrow f$ therefore $\mathcal{KB} \vdash +\Delta f$.

- Inductive step: the statement $(f \rightarrow \emptyset)$ is supported by a statement $\mathbb{s} = (r)$ s.t. $Head(r) = f$ and $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle \mathbb{s} \rangle$ = INstr. This means that $\mathbb{s}$ has a complete INstr support and is not attacked by INstr attack edge, which implies that $\forall \phi \in Body(r)$ there is a INstr statement with a strict rule for ϕ. By the inductive hypotheses we have that $\forall \phi \in Body(r) : +\Delta\phi$, therefore $\mathcal{KB} \vdash +\Delta f$.

From (1.) the proposition (2.) directly holds by contra-positive since $\mathcal{KB} \nvdash +\Delta f$ means $\mathcal{KB} \vdash -\Delta f$ [Billington, 1993].
We split the proof of (3.) in two, first we prove by induction that if $\mathcal{KB} \vdash +\delta^{TD}_{block} f$ then $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle (f \rightarrow \emptyset) \rangle \in \{\text{INstr}, \text{INdef}\}$:

- Inductive base: $Proof(2) = +\delta^{TD}_{block} f$. This implies that either $+\Delta f \in Proof(1)$ which means $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle (f \rightarrow \emptyset) \rangle$ = INstr (as shown in (1.)) or:

  1. there exist a strict or defeasible rule r s.t. $Head(r) = f$ and $\forall \phi \in Body(r)$ there is a strict or defeasible fact rule for ϕ and no rule for $\overline{\phi}$ (which means that there is a statement for ϕ that is either labeled INstr or INdef, therefore the statement (r) has INstr or INdef complete), and
  2. $\mathcal{KB} \vdash -\Delta\overline{f}$ (which means that the claim statement for $\overline{f}$ is not labeled INstr (as show in (1.)), i.e. there is no INstr attack edge on $(f \rightarrow \emptyset)$ which means it is not labeled OUTstr) and
  3. for all rules r' for $\overline{f}$ either:
     (a) $\exists \phi' \in Body(r') : -\delta^{TD}_{block}\phi' \in Proof(1)$ i.e. there is no strict or defeasible fact rule for ϕ' (which means that all statement for $\overline{f}$ do not have a INstr or INdef complete support, they are therefore not labeled INdef or INstr).
     (b) or $\exists r'' \in \mathcal{R}$ s.t. $Head(r'') = f$ and $\forall \phi \in Body(r'') : +\delta^{TD}_{block}\phi \in Proof(1)$ and $r'' \succ r'$, which means the statements (r') and

(r'') have a INstr or INdef complete support (coming from the top statement since r' and r'' are fact rules), meaning that for all the INdef attack edge on ($f \rightarrow \emptyset$) there is a support edge that is superior to it ($r'' \succ r'$).

Form 1. 2. and 3. we can see that the query statement for f has a INstr or INdef complete support and (no INstr or INdef attack edge) or (all INdef attack edges are inferior to a support edge), therefore $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$.

- Inductive step: Let us suppose that the proposition holds for $Proof(1..i)$ and $Proof(i+1) = +\delta^{TD}_{block} f$. This implies that either $+\Delta f \in Proof(i)$ (which means that $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$ as shown in 1.), or:

  1. there exits a strict or defeasible rule r s.t. $Head(r) = f$ and $\forall \phi \in Body(r) : +\delta^{TD}_{block}\phi \in Proof(1..i)$, by the inductive hypotheses we have that there is a INstr or INdef statement for ϕ, meaning that the statement (r) has a INstr or INdef complete support.
  2. $\mathcal{KB} \vdash -\Delta\overline{f}$ (which means that the claim statement for $\overline{f}$ is not labeled INstr (as show in (1.)), i.e. there is no INstr attack edge on the query statement for f).
  3. for all rules r' for $\overline{f}$ either:
     (a) $\exists \phi' \in Body(r') : -\delta^{TD}_{block}\phi' \in Proof(1..i)$, given results in [Billington, 1993] where it is shown that $\mathcal{KB} \vdash -\delta^{TD}_{block}\phi'$ means $\mathcal{KB} \nvdash +\delta^{TD}_{block}\phi'$, and by the contrapositive of the inductive hypothesis we get that $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(\phi' \rightarrow \emptyset)\rangle \notin \{\text{INstr}, \text{INdef}\}$ which means that there is no INstr or INdef complete support for (r') which implies that there is no INstr or INdef attack edge on the query statement for f.
     (b) or $\exists r'' \in \mathcal{R}$ s.t. $Head(r'') = f$ and $\forall \phi \in Body(r'') : +\delta^{TD}_{block}\phi \in Proof(1..i)$ and $r'' \succ r'$, which means the statements (r') and (r'') have a INstr or INdef complete support, meaning that for all the attack edge on the query statement for f there is a support edge that is superior to it ($r'' \succ r'$).

  Form 1. 2. and 3. we can see that the query statement for f has a INstr or INdef complete support and (no INstr or INdef attack edge) or (all INdef attack edges are inferior to a INdef support edge), therefore $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$.

Now we prove by induction that if $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$ then $\mathcal{KB} \vdash +\delta^{TD}_{block} f$.

- Inductive base: $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$ implies that there is a strict fact rule for f (which means $\mathcal{KB} \vdash +\Delta f$ given (1.)). $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow$

$\emptyset)\rangle$ = INdef means there is a defeasible fact rule for f and no strict fact rule for $\overline{f}$ and for all INdef attack edge there is a support edge that is superior to it, which implies that for every rule r' for $\overline{f}$ there is a rule r'' for f s.t. $r'' \succ r'$, therefore $\mathcal{KB} \vdash +\delta^{TD}_{block} f$.

- Inductive step: Let us suppose that the proposition holds for all premises of a rule r for f and $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$. Either $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset) = \text{INstr}$ which means that r is strict rule and its statement has INstr complete support, meaning that (from (1.)) $\forall \phi \in \mathit{Body}(r) : +\Delta\phi$ therefore $\mathcal{KB} \vdash +\delta^{TD}_{block} f$. Or $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset) = \text{INdef}$ which means that $\mathbb{s} = (f \rightarrow \emptyset)$ has:

  - either a INstr complete support and r is a defeasible rule and is superior to the rules of all edges attacking it (which means that for all INdef statements with a defeater rule r' for $\overline{f}$, $r \succ r'$). By the inductive hypothesis, $\forall \phi' \in \mathit{Body}(r') : +\delta^{TD}_{block}\phi'$ and $r \succ r'$, therefore $\mathcal{KB} \vdash +\delta^{TD}_{block} f$).
  - or a INdef complete support and
    1. for all INdef attack edge there is a support edge superior to it, which means for all INdef statements with a strict or defeasible rule r' for $\overline{f}$ there is a rule r'' for f such that $r'' \succ r'$, which implies by the inductive hypothesis that for every literal ϕ'' in the body of r'' $\mathcal{KB} \vdash +\delta^{TD}_{block}\phi''$ and $r'' \succ r'$.
    2. and for all INdef attack edges on the rule application, r is either a strict rule or r is a defeasible rule and is superior to the defeater rule attacking it, which means that for all INdef statement with a defeater rule r' for $\overline{f}$ $r \succ r'$. By the inductive hypothesis, $\forall \phi' \in \mathit{Body}(r') : +\delta^{TD}_{block}\phi'$ and $r \succ r'$.

    from 1. and 2. for all r' for $\overline{f}$ there is a rule r'' for f s.t. $r'' \succ r'$, therefore $\mathcal{KB} \vdash +\delta^{TD}_{block} f$.

  Thus if $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$ then $\mathcal{KB} \vdash +\delta^{TD}_{block} f$.

From (3.) the proposition (4.) directly holds by contrapositive since $\mathcal{KB} \nvdash +\delta^{TD}_{block} f$ means $\mathcal{KB} \vdash -\delta^{TD}_{block} f$ [Billington, 1993] and $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \notin \{\text{INstr}, \text{INdef}\}$ implies $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{OUTstr, OUTdef, AMBIG, UNSUP}\}$ given Lemma 4.1 stating that BDL is a function). □

Proposition 4.2 Let f be a literal in a defeasible $\mathcal{KB}$ expressed in $\mathcal{L}_p$ that contains no attack or support cycles:

1. $\mathcal{KB} \vdash +\Delta f$ iff $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$
2. $\mathcal{KB} \vdash -\Delta f$ iff $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq \text{INstr}$

3. $\mathcal{KB} \vdash +\delta^{TD}_{prop} f$ iff $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$

4. $\mathcal{KB} \vdash -\delta^{TD}_{prop} f$ iff $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{OUTstr}, \text{OUTdef}, \text{AMBIG}, \text{UNSUP}\}$

Proof 4.2. The definition for the INstr label of PDL is the same as BDL, therefore, given Proposition 4.1, (1.) and (2.) directly hold. Given that the only difference between PDL and BDL is the INdef label, we only need to prove that $\mathcal{KB} \vdash +\delta^{TD}_{prop} f$ iff $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$.

We split the proof in two, first we prove by induction that if $\mathcal{KB} \vdash +\delta^{TD}_{prop} f$ then $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$.

- Inductive base: $Proof(2) = +\delta^{TD}_{prop} f$. This implies that either $+\Delta f \in Proof(1)$ which means $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$ (as shown in (1.)) or:

  1. there exist a strict or defeasible rule r s.t. $Head(r) = f$ and $\forall \phi \in Body(r)$ there is a strict or defeasible fact rule for ϕ and no rule for $\overline{\phi}$ (which means that there is a statement for ϕ that is either labeled INstr or INdef, therefore the statement (r) has INstr or INdef complete), and
  2. $\mathcal{KB} \vdash -\Delta\overline{f}$ (which means that the claim statement for $\overline{f}$ is not labeled INstr (as show in (1.)), i.e. there is no INstr attack edge on $(f \rightarrow \emptyset)$ which means it is not labeled OUTstr) and
  3. for all rules r' for $\overline{f}$ either:
     (a) $\exists \phi' \in Body(r') : -\sum^{TD} \phi' \in Proof(1)$ i.e. there is no strict or defeasible fact rule for ϕ' (which means that all statement for $\overline{f}$ do not have a INstr or INdef complete support, they are therefore not labeled INdef or INstr).
     (b) or $\exists r'' \in \mathcal{R}$ s.t. $Head(r'') = f$ and $\forall \phi \in Body(r'') : +\delta^{TD}_{block} \phi \in Proof(1)$ and $r'' \succ r'$, which means the statements (r') and (r'') have a INstr or INdef complete support (coming from the top statement since r' and r'' are fact rules), meaning that for all the INdef attack edge on $(f \rightarrow \emptyset)$ there is a support edge that is superior to it $(r'' \succ r')$.

  Form 1. 2. and 3. we can see that the query statement for f has a INstr or INdef complete support and (no INstr or INdef attack edge) or (all INdef attack edges are inferior to a support edge), therefore $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$.

- Inductive step: Let us suppose that the proposition holds for $Proof(1..i)$ and $Proof(i+1) = +\delta^{TD}_{prop} f$. This implies that either $+\Delta f \in Proof(i)$ (which means that $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$ as shown in 1.), or:

  1. there exits a strict or defeasible rule r s.t. $Head(r) = f$ and $\forall \phi \in Body(r) : +\delta^{TD}_{prop} \phi \in Proof(1..i)$, by the inductive hypotheses

we have that there is a INstr or INdef statement for ϕ, meaning that the statement (r) has a INstr or INdef complete support.

2. $\mathcal{KB} \vdash -\Delta\overline{f}$ (which means that the claim statement for $\overline{f}$ is not labeled INstr (as show in (1.)), i.e. there is no INstr attack edge on the query statement for f).
3. for all rules r' for $\overline{f}$ either:
   (a) $\exists\phi' \in Body(r') : -\sum^{TD}\phi' \in Proof(1..i)$ which means that r' cannot be applied, therefore there is no INstr, INdef or AMBIG complete support for (r') which implies that there is no INstr, INdef, or AMBIG attack edge on the query statement for f.
   (b) or $\exists r'' \in \mathcal{R}$ s.t. $Head(r'') = f$ and $\forall\phi \in Body(r'') : +\delta^{TD}_{prop}\phi \in Proof(1..i)$ and $r'' \succ r'$, which means the statements (r') and (r'') have a INstr or INdef complete support, meaning that for all the INdef or AMBIG attack edge on the query statement for f there is a support edge that is superior to it $(r'' \succ r')$.

Form 1. 2. and 3. we can see that the query statement for f has a INstr or INdef complete support and (no INstr, INdef, or AMBIG attack edge) or (all INdef attack edges are inferior to a INdef support edge), therefore $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$.

Now we prove by induction that if $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$ then $\mathcal{KB} \vdash +\delta^{TD}_{prop}f$.

- Inductive base: $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$ implies that there is a strict fact rule for f (which means $\mathcal{KB} \vdash +\Delta f$ given (1.)). $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INdef}$ means there is a defeasible fact rule for f and no strict fact rule for $\overline{f}$ and for all INdef or AMBIG attack edge there is a support edge that is superior to it, which implies that for every rule r' for $\overline{f}$ where its statement has a complete INdef or AMBIG support there is a rule r'' for f s.t. $r'' \succ r'$, therefore $\mathcal{KB} \vdash +\delta^{TD}_{prop}f$.

- Inductive step: Let us suppose that the proposition holds for all premises of a rule r for f and $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$. Either $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset) = \text{INstr}$ which means that r is strict rule and its statement has INstr complete support, meaning that (from (1.)) $\forall\phi \in Body(r) : +\Delta\phi$ therefore $\mathcal{KB} \vdash +\delta^{TD}_{prop}f$. Or $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset) = \text{INdef}$ which means that $\mathbb{s} = (f \rightarrow \emptyset)$ has:

  – either a INstr complete support and r is a defeasible rule and is superior to the rules of all edges attacking it (which means that for all INdef or AMBIG statements with a defeater rule r' for $\overline{f}$, $r \succ r'$). By the inductive hypothesis, $\forall\phi' \in Body(r') : +\delta^{TD}_{prop}\phi'$ and $r \succ r'$, therefore $\mathcal{KB} \vdash +\delta^{TD}_{prop}f$).

– or a INdef complete support and

1. for all INdef or AMBIG attack edge there is a support edge superior to it, which means for all INdef or AMBIG statements with a strict or defeasible rule r' for $\overline{f}$ there is a rule r'' for f such that $r'' \succ r'$, which implies by the inductive hypothesis that for every literal ϕ'' in the body of r'' $\mathcal{KB} \vdash +\delta^{TD}_{prop}\phi''$ and $r'' \succ r'$.
2. and for all INdef or AMBIG attack edges on the rule application, r is either a strict rule or r is a defeasible rule and is superior to the defeater rule attacking it, which means that for all INdef or AMBIG statement with a defeater rule r' for $\overline{f}$ $r \succ r'$. By the inductive hypothesis, $\forall \phi' \in Body(r') : +\sum^{TD} \phi'$ and $r \succ r'$.

from 1. and 2. for all r' for $\overline{f}$ there is a rule r'' for f s.t. $r'' \succ r'$, therefore $\mathcal{KB} \vdash +\delta^{TD}_{prop}f$.

Thus if $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$ then $\mathcal{KB} \vdash +\delta^{TD}_{prop}f$.

From (3.) the proposition (4.) directly holds by contrapositive since $\mathcal{KB} \nvdash +\delta^{TD}_{prop}f$ means $\mathcal{KB} \vdash -\delta^{TD}_{prop}f$ [Billington, 1993] and $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \notin \{\text{INstr}, \text{INdef}\}$ implies $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{$OUTstr, OUTdef, AMBIG, UNSUP$\}$ given that PDL is a function).

□

Proposition 4.3 Let f be a literal in a defeasible $\mathcal{KB}$ that contains no attack or support cycles:

1. $\mathcal{KB} \vdash +\Delta f$ iff $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$
2. $\mathcal{KB} \vdash -\Delta f$ iff $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq \text{INstr}$
3. $\mathcal{KB} \vdash +\delta^{noTD}_{block}f$ iff $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$
4. $\mathcal{KB} \vdash -\delta^{noTD}_{block}f$ iff $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{OUTstr}, \text{OUTdef}, \text{AMBIG}, \text{UNSUP}\}$

Proof 4.3. (1.) and (2.) directly hold given Proposition 4.1 since they are not affected by preferences. The only difference between BDL and BDL_{noTD} is the handling of preferences, therefore we will only prove this part (the rest is already proven in Proposition 4.1).
We split the proof of (3.) in two, first we prove by induction that if $\mathcal{KB} \vdash +\delta^{noTD}_{block}f$ then $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$:

- Inductive base: $Proof(2) = +\delta^{noTD}_{block}f$. This implies that either $+\Delta f \in Proof(1)$ which means $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$ (as shown in (1.)) or:

1. there exist a strict or defeasible rule r s.t. $Head(r) = f$ and $\forall \phi \in Body(r)$ there is a strict or defeasible fact rule for ϕ and no rule for $\overline{\phi}$ (which means that there is a statement for ϕ that is either labeled INstr or INdef, therefore the statement (r) has INstr or INdef complete), and
2. $\mathcal{KB} \vdash -\Delta\overline{f}$ (which means that the claim statement for $\overline{f}$ is not labeled INstr (as show in (1.)), i.e. there is no INstr attack edge on $(f \rightarrow \emptyset)$ which means it is not labeled OUTstr) and
3. for all rules r' for $\overline{f}$ either:
   (a) $\exists \phi' \in Body(r') : -\delta^{TD}_{block}\phi' \in Proof(1)$ i.e. there is no strict or defeasible fact rule for ϕ' (which means that all statement for $\overline{f}$ do not have a INstr or INdef complete support, they are therefore not labeled INdef or INstr).
   (b) or $r \succ r'$, which means the support edge coming from (r) is superior to all the INdef attack edge on $(f \rightarrow \emptyset)$.

   Form 1. 2. and 3. we can see that the query statement for f has a INstr or INdef complete support and (no INstr or INdef attack edge) or (all INdef attack edges are inferior to the support edge coming from r), therefore $\mathbb{SG}^{\mathrm{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\mathrm{INstr}, \mathrm{INdef}\}$.

- Inductive step: Let us suppose that the proposition holds for $Proof(1..i)$ and $Proof(i+1) = +\delta^{noTD}_{block} f$. This implies that either $+\Delta f \in Proof(i)$ (which means that $\mathbb{SG}^{\mathrm{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \mathrm{INstr}$ as shown in 1.), or:

  1. there exits a strict or defeasible rule r s.t. $Head(r) = f$ and $\forall \phi \in Body(r) : +\delta^{noTD}_{block}\phi \in Proof(1..i)$, by the inductive hypotheses we have that there is a INstr or INdef statement for ϕ, meaning that the statement (r) has a INstr or INdef complete support.
  2. $\mathcal{KB} \vdash -\Delta\overline{f}$ (which means that the claim statement for $\overline{f}$ is not labeled INstr (as show in (1.)), i.e. there is no INstr attack edge on the query statement for f).
  3. for all rules r' for $\overline{f}$ either:
     (a) $\exists \phi' \in Body(r') : -\delta^{noTD}_{block}\phi' \in Proof(1..i)$, given results in [Billington, 1993] where it is shown that $\mathcal{KB} \vdash -\delta^{noTD}_{block}\phi'$ means $\mathcal{KB} \nvdash +\delta^{noTD}_{block}\phi'$, and by the contrapositive of the inductive hypothesis we get that $\mathbb{SG}^{\mathrm{BDL}}_{\mathcal{KB}}\langle(\phi' \rightarrow \emptyset)\rangle \notin \{\mathrm{INstr}, \mathrm{INdef}\}$ which means that there is no INstr or INdef complete support for (r') which implies that there is no INstr or INdef attack edge on the query statement for f.
     (b) or $r \succ r'$, which means the statements for all the attack edge on the query statement for f the support edge coming from r is superior.

Form 1. 2. and 3. we can see that the query statement for f has a INstr or INdef complete support and (no INstr or INdef attack edge) or (all INdef attack edges are inferior to the INdef support edge from (r)), therefore $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$.

Now we prove by induction that if $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$ then $\mathcal{KB} \vdash +\delta^{noTD}_{block} f$.

- Inductive base: $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INstr}$ implies that there is a strict fact rule for f (which means $\mathcal{KB} \vdash +\Delta f$ given (1.)). $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{INdef}$ means there is a defeasible fact rule for f and no strict fact rule for $\overline{f}$ and for all INdef attack edge the support edge of r is superior to it, which implies that for every rule r' for $\overline{f}$ $r \succ r'$, therefore $\mathcal{KB} \vdash +\delta^{noTD}_{block} f$.

- Inductive step: Let us suppose that the proposition holds for all premises of a rule r for f and $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$. Either $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset) = \text{INstr}$ which means that r is strict rule and its statement has INstr complete support, meaning that (from (1.)) $\forall \phi \in \mathit{Body}(r) : +\Delta\phi$ therefore $\mathcal{KB} \vdash +\delta^{noTD}_{block} f$. Or $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset) = \text{INdef}$ which means that $\mathbb{s} = (f \rightarrow \emptyset)$ has:

  - either a INstr complete support and r is a defeasible rule and is superior to the rules of all edges attacking it (which means that for all INdef statements with a defeater rule r' for $\overline{f}$, $r \succ r'$). By the inductive hypothesis, $\forall \phi' \in \mathit{Body}(r') : +\delta^{noTD}_{block}\phi'$ and $r \succ r'$, therefore $\mathcal{KB} \vdash +\delta^{noTD}_{block} f$).
  - or a INdef complete support and
    1. for all INdef attack edge the support edge coming from r superior to it, which means for all INdef statements with a strict or defeasible rule r' for $\overline{f}$, $r \succ r'$.
    2. and for all INdef attack edges on the rule application, r is either a strict rule or r is a defeasible rule and is superior to the defeater rule attacking it, which means that for all INdef statement with a defeater rule r' for $\overline{f}$ $r \succ r'$..

    from 1. and 2. for all r' for $\overline{f}$, $r \succ r'$, therefore $\mathcal{KB} \vdash +\delta^{noTD}_{block} f$.

  Thus if $\mathbb{SG}^{\text{BDL}_{noTD}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{INstr}, \text{INdef}\}$ then $\mathcal{KB} \vdash +\delta^{noTD}_{block} f$.

From (3.) the proposition (4.) directly holds by contrapositive. □

Proposition 4.5 Propositions 4.1, 4.3, 4.2, and 4.4 still hold for failure-by-looping in presence of attack and support cycles.

Proof 4.5. We prove this by construction, we start by support cycles. Suppose that evaluating a rule r for f produces a support cycle, then the premises of this rule cannot be derived i.e. $\exists \phi \in Body(r)$ such that $-\sum \phi$, if there is no other rule for f then $-\sum f$ [Billington, 2004, Lam, 2012]. This means that the statement (r) is part of a support cycle, therefore it is labeled UNSUP, meaning that $(f \rightarrow \emptyset)$ does not have a complete INstr, INdef, or AMBIG support, thus it is labeled UNSUP.

- In Defeasible Logic with ambiguity blocking (with or without team defeat) f is labeled $-\delta^{TD}_{block}$ which means that $(f \rightarrow \emptyset)$ must be labeled by BDL in $\{\text{OUTstr}, \text{OUTdef}, \text{AMBIG}, \text{UNSUP}\}$ which is the case given that $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle$ = UNSUP.
- In Defeasible Logic with ambiguity propagating (with or without team defeat) f is labeled $-\sum$, thus $-\delta_{prop}$ [Antoniou et al., 2000a] which means that $(f \rightarrow \emptyset)$ must be labeled by PDL in $\{\text{OUTstr}, \text{OUTdef}, \text{AMBIG}, \text{UNSUP}\}$ which is the case given that $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle$ = UNSUP.

Suppose that evaluating a rule r for f produces an attack cycle that cannot be avoided by checking other rules, this means that the statement (r) is part of an attack cycle and is labeled AMBIG.

- In Defeasible Logic with ambiguity blocking (with or without team defeat) f is labeled $-\delta_{block}$ which means that $(f \rightarrow \emptyset)$ must be labeled by BDL in $\{\text{OUTstr}, \text{OUTdef}, \text{AMBIG}, \text{UNSUP}\}$ which is the case given that $\mathbb{SG}^{\text{BDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle$ = AMBIG since it only has an AMBIG complete support coming from (r).
- In Defeasible Logic with ambiguity propagating (with or without team defeat) f is labeled $-\delta_{prop}$ and $+\sum$ since it is not part of a support cycle (otherwise the attack cycle can be avoided), which means that $(f \rightarrow \emptyset)$ must be labeled by PDL in $\{\text{OUTstr}, \text{OUTdef}, \text{AMBIG}, \text{UNSUP}\}$ which is the case given that $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle$ = UNSUP, and it must also be derivable to be able to attack other statement, which is the case since PDL considers AMBIG attacks.

□

7.2.3 Chapter 5

Proposition 5.4 Given a knowledge base $\mathcal{KB}$ that only contains defeasible facts, strict rules and no preferences, and a ground query Q:

1. if $\mathcal{KB} \vdash_{IAR} Q$ then $\mathcal{KB} \vdash_{\text{PDL}} Q$
2. if $\mathcal{KB} \vdash_{IAR} Q$ then $\mathcal{KB} \vdash_{\text{BDL}} Q$

3. if $\mathcal{KB} \models_{\text{PDL}} Q$ then $\mathcal{KB} \models_{\text{BDL}} Q$

Proof 5.4. We prove (1.) by contradiction. Suppose there is a fact f such that $\mathcal{KB} \models_{IAR} f$ and $\mathcal{KB} \not\models_{\text{PDL}} f$. $\mathcal{KB} \models_{IAR} f$ means that there is a derivation for f from an initial set of facts $T \subseteq \mathcal{F}$ and there is no minimal consistent set of initial facts $S \subseteq \mathcal{F}$ such that $S \cup T$ is inconsistent (i.e $models(S, \mathcal{R} \cup \mathcal{N}) \neq \emptyset$ and $models(S \cup T, \mathcal{R} \cup \mathcal{N}) = \emptyset$) [Lembo et al., 2010]. This means that f is derivable and does not rely conflicting facts, therefore the statement $(f \rightarrow \emptyset)$ has a complete INdef support and no INdef or AMBIG attack edges, i.e. $\mathbb{SG}^{\text{PDL}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle$ = INdef, thus $\mathcal{KB} \models_{\text{PDL}} f$ which is a contradiction. (3.) directly follows from the inclusion theorem in [Billington et al., 2010], it states that any literal that is $+\delta_{prop}$ is also $+\delta_{block}$. From (1.) and (3.), (2.) directly follows. □

Proposition 5.5 Given a knowledge base $\mathcal{KB}$ that only contains defeasible facts, strict rules and no preferences, and a ground query Q:

1. if $\mathcal{KB} \models_{IAR} Q$ then $\mathcal{KB} \models_{DT} Q$

2. if $\mathcal{KB} \models_{DT} Q$ then $\mathcal{KB} \models_{\text{PDL}} Q$

Proof 5.5. We prove (1.) by contradiction. Suppose there is a fact f such that $\mathcal{KB} \models_{IAR} f$ and $\mathcal{KB} \not\models_{DT} f$. $\mathcal{KB} \models_{IAR} f$ means that there is a derivation for f and there is no minimal set of initial facts S such that there is a derivation from S to atoms that make a negative constraint applicable (i.e $models(S, \mathcal{R} \cup \mathcal{N}) = \emptyset$) [Lembo et al., 2010]. This means that there is an argument for f that has no defeater (since no derivation containing f is inconsistent), thus $\mathcal{KB} \models_{DT} f$ which is a contradiction.

We prove (2.) by contradiction, Suppose there is a fact f such that $\mathcal{KB} \models_{DT} f$ and $\mathcal{KB} \not\models_{\text{PDL}} f$. $\mathcal{KB} \models_{DT} f$ means that there is an argument for f with no defeaters (since any defeater is a blocking defeater in the absence of preferences), this means that there is a derivation for f and no atom in that derivation is attacked, thus $\mathcal{KB} \not\models_{\text{PDL}} f$ which is a contradiction. □

Proposition 5.6 Let f be a fact in a $\mathcal{KB}$ that contains only defeasible facts, strict rules and no preferences:

1. $\mathcal{KB} \models_{IAR} f$ iff $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle$ = IN.

2. $\mathcal{KB} \not\models_{IAR} f$ iff $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in$ {AMBIG, OUT}.

Proof 5.6. We split the proof of (1.) in two, first we prove by contradiction that if $\mathcal{KB} \models_{IAR} f$ then $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle$ = IN: Suppose we have a fact f such that $\mathcal{KB} \models_{IAR} f$ iff $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq$ IN:

1. $\mathcal{KB} \vDash_{IAR} f$ means that there is a derivation for f from an initial set of facts $T \subseteq \mathcal{F}$ and there is no minimal consistent set of initial facts $S \subseteq \mathcal{F}$ such that $S \cup T$ is inconsistent (i.e $models(S, \mathcal{R} \cup \mathcal{N}) \neq \emptyset$ and $models(S \cup T, \mathcal{R} \cup \mathcal{N}) = \emptyset$), which means that f is not generated by conflicting atoms and is not used to generate conflicting atoms i.e. PDL($(f \rightarrow \emptyset)$) = INdef (Proposition 5.4).

2. $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq$ IN means that either:

   (a) $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle =$ OUT which is impossible given 1. (i.e. PDL$(f \rightarrow \emptyset)$ = INdef)

   (b) or $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle =$ AMBIG which means either:

      i. PDL$(f \rightarrow \emptyset)$ = AMBIG (impossible given 1.),

      ii. or $\exists e \in \mathcal{E}^{+}_{S}(\mathbb{s}) \cup \mathcal{E}^{+}_{A}(\mathbb{s})$ such that IAR($Target(e)$) = AMBIG which means that f is used to generate conflicting atoms (impossible given 1.).

Now we prove by contradiction that if $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle =$ IN then $\mathcal{KB} \vDash_{IAR} f$: Suppose we have a fact f such that $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle =$ IN and $\mathcal{KB} \nvDash_{IAR} f$:

1. $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle =$ IN means that IAR$(f \rightarrow \emptyset) \neq$ AMBIG and PDL$(f \rightarrow \emptyset)$ = IN, which means that $(f \rightarrow \emptyset)$ is not attacked (i.e. there is no chain of rule applications for an atom conflicting with f) and is not used to generate conflicting atoms (no outgoing edge leads to an AMBIG statement).

2. $\mathcal{KB} \nvDash_{IAR} f$ means that either f is generated by conflicting atoms (impossible given 1.) or is used to generate conflicting atoms (impossible given 1.).

From (1.) the proposition (2.) directly holds ($\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq$ IN means $\mathbb{SG}^{\text{IAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in$ {AMBIG, OUT} given that IAR is a function). □

Proposition 5.7 Let f be a fact in a $\mathcal{KB}$ that contains only defeasible facts, strict rules and no preferences:

1. $\mathcal{KB} \vDash_{ICAR} f$ iff $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle =$ IN.

2. $\mathcal{KB} \nvDash_{ICAR} f$ iff $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in$ {AMBIG, OUT}.

Proof 5.7. We split the proof of (1.) in two, first we prove by contradiction that if $\mathcal{KB} \vDash_{ICAR} f$ then $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle =$ IN. Suppose we have a fact f such that $\mathcal{KB} \vDash_{ICAR} f$ and $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq$ IN:

1. $\mathcal{KB} \vDash_{ICAR} f$ means that there is a derivation for f and there is no minimal consistent set of facts $S \subseteq \mathcal{F}^*$ such that $S \cup \{f\}$ is inconsistent ($models(S, \mathcal{R} \cup \mathcal{N}) \neq \emptyset$ and $models(S \cup \{f\}, \mathcal{R} \cup \mathcal{N}) = \emptyset$) i.e f is not used to generate conflicting atoms.

2. $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq \text{IN}$ means that either:

   (a) $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{OUT}$ which is impossible given 1. (there is a chain of rule applications for f i.e. $\text{PDL}(f \rightarrow \emptyset) \in \{\text{INdef}, \text{AMBIG}\}$).

   (b) or $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{AMBIG}$ which means either:

      i. $\text{PDL}(f \rightarrow \emptyset) = \text{AMBIG}$ and there is an edge attacking it (impossible given 1. i.e. there is no derivable conflicting atom with f).

      ii. or $\exists e \in \mathcal{E}^+_S(\mathbb{s}) \cup \mathcal{E}^+_A(\mathbb{s})$ such that $\text{ICAR}(Target(e)) = \text{AMBIG}$ which means that f is used to generate conflicting atoms (impossible given 1.).

Now we prove by contradiction that if $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{IN}$ then $\mathcal{KB} \vDash_{ICAR} f$: Suppose we have a fact f such that $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{IN}$ and $\mathcal{KB} \nvDash_{ICAR} f$:

1. $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle = \text{IN}$ means that $\text{ICAR}(f \rightarrow \emptyset) \neq \text{AMBIG}$ and $\text{PDL}(f \rightarrow \emptyset)) \in \{\text{IN}, \text{AMBIG}\}$, which means that $(f \rightarrow \emptyset)$ is not attacked (i.e. there is no chain of rule applications for an atom conflicting with f) and it is used to generate conflicting atoms (no outgoing edge leads to an AMBIG statement).

2. $\mathcal{KB} \nvDash_{ICAR} f$ means that either there is a chain of rule applications for an atom conflicting with f or f is used to generate conflicting atoms (impossible given 1.).

From (1.) the proposition (2.) directly holds ($\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \neq \text{IN}$ means $\mathbb{SG}^{\text{ICAR}}_{\mathcal{KB}}\langle(f \rightarrow \emptyset)\rangle \in \{\text{AMBIG, OUT}\}$ given that ICAR is a function). □

Proposition 5.8 Given a knowledge base $\mathcal{KB}$ that only contains defeasible facts, strict rules and no preferences, and a ground query Q:

1. if $\mathcal{KB} \vDash_{IAR} Q$ then $\mathcal{KB} \vDash^{block}_{IAR} Q$

2. if $\mathcal{KB} \vDash^{block}_{IAR} Q$ then $\mathcal{KB} \vDash^{block}_{ICAR} Q$

3. if $\mathcal{KB} \vDash_{ICAR} Q$ then $\mathcal{KB} \vDash^{block}_{ICAR} Q$

Proof 5.8. We prove (1.) by contradiction, suppose we have $\mathcal{KB} \vDash_{IAR} f$ and $\mathcal{KB} \nvDash^{block}_{IAR} f$:

1. $\mathcal{KB} \models_{IAR} f$ means that there is a derivation for f from an initial set of facts $T \subseteq \mathcal{F}$ and there is no minimal consistent set of initial facts $S \subseteq \mathcal{F}$ such that $S \cup T$ is inconsistent (i.e $models(S, \mathcal{R} \cup \mathcal{N}) \neq \emptyset$ and $models(S \cup T, \mathcal{R} \cup \mathcal{N}) = \emptyset$), which means that f is not generated by conflicting atoms and is not used to generate conflicting atoms i.e. PDL($(f \rightarrow \emptyset)$) = INdef (Proposition 5.4) which implies that BDL($(f \rightarrow \emptyset)$) = INdef (given Proposition 5.4).

2. $\mathcal{KB} \not\models^{block}_{IAR} f$ means that either BDL($(f \rightarrow \emptyset)$) $\neq$ INdef (impossible given 1.) or f is used to generate conflicting atoms (impossible given 1.)

We prove (2.) by contradiction, suppose we have $\mathcal{KB} \models^{block}_{IAR} f$ and $\mathcal{KB} \not\models^{block}_{ICAR} f$:

1. $\mathcal{KB} \models^{block}_{IAR} f$ means that BDL($(f \rightarrow \emptyset)$) = INdef and f is not used to generate conflicting atoms.

2. $\mathcal{KB} \not\models^{block}_{ICAR} f$ means that either BDL($f \rightarrow \emptyset$) $\neq$ INdef (impossible given 1.) or there is a derivable atom conflicting with f or f is used to generate conflicting atoms (impossible given 1.)

We prove (3.) by contradiction, suppose we have $\mathcal{KB} \models_{ICAR} f$ and $\mathcal{KB} \not\models^{block}_{ICAR} f$:

1. $\mathcal{KB} \models_{ICAR} f$ means that PDL($(f \rightarrow \emptyset)$) $\in$ {INdef, AMBIG}, there is no derivable fact conflicting with f, and f is not used to derive conflicting atoms.

2. $\mathcal{KB} \not\models^{block}_{ICAR} f$ means that BDL($(f \rightarrow \emptyset)$) $\neq$ INdef and BDL($(f \rightarrow \emptyset)$) = AMBIG and either there is a derivable fact that is conflicting with f (impossible given 1.) or f is used to generate conflicting atoms (impossible given 1.).

□

Bibliography

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc.

[Amgoud et al., 2004] Amgoud, L., Caminada, M., Cayrol, C., Lagasquie, M.-C., and Prakken, H. (2004). Towards a consensual formal model: inference part. *Deliverable D2*, 2.

[Amgoud and Vesic, 2011] Amgoud, L. and Vesic, S. (2011). A new approach for preference-based argumentation frameworks. *Annals of Mathematics and Artificial Intelligence*, 63(2):149–183.

[Antoniou, 2006] Antoniou, G. (2006). Defeasible reasoning: A discussion of some intuitions. *International Journal of Intelligent Systems*, 21(6):545–558.

[Antoniou et al., 2001] Antoniou, G., Billington, D., Governatori, G., and Maher, M. J. (2001). Representation results for defeasible logic. *ACM Transactions on Computational Logic (TOCL)*, 2(2):255–287.

[Antoniou et al., 2000a] Antoniou, G., Billington, D., Governatori, G., Maher, M. J., and Rock, A. (2000a). A Family of Defeasible Reasoning Logics and its Implementation. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 459–463.

[Antoniou et al., 1999] Antoniou, G., Billington, D., and Maher, M. J. (1999). On the analysis of regulations using defeasible rules. In *Systems Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*, pages 7–23. IEEE.

[Antoniou et al., 2000b] Antoniou, G., Maher, M. J., and Billington, D. (2000b). Defeasible logic versus logic programming without negation as failure. *The Journal of Logic Programming*, 42(1):47–57.

[Apt and Van Emden, 1982] Apt, K. R. and Van Emden, M. H. (1982). Contributions to the theory of logic programming. *Journal of the ACM (JACM)*, 29(3):841–862.

[Arenas et al., 1999] Arenas, M., Bertossi, L., and Chomicki, J. (1999). Consistent query answers in inconsistent databases. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 68–79. ACM.

[Arora et al., 1993] Arora, T., Ramakrishnan, R., Roth, W. G., Seshadri, P., and Srivastava, D. (1993). Explaining program execution in deductive

systems. In *Deductive and Object-Oriented Databases*, pages 101–119. Springer.

[Ausiello et al., 2001] Ausiello, G., Franciosa, P. G., and Frigioni, D. (2001). Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. In *Italian conference on theoretical computer science*, pages 312–328. Springer.

[Baader et al., 2005] Baader, F., Horrocks, I., and Sattler, U. (2005). Description logics as ontology languages for the semantic web. In *Mechanizing Mathematical Reasoning*, pages 228–248. Springer.

[Baget et al., 2016] Baget, J. F., Benferhat, S., Bouraoui, Z., Croitoru, M., Mugnier, M.-L., Papini, O., Rocher, S., and Tabia, K. (2016). Inconsistency-Tolerant Query Answering: Rationality Properties and Computational Complexity Analysis. In *European Conference on Logics in Artificial Intelligence*, pages 64–80. Springer.

[Baget et al., 2014a] Baget, J.-F., Garreau, F., Mugnier, M.-L., and Rocher, S. (2014a). Extending Acyclicity Notions for Existential Rules. In *ECAI*, pages 39–44.

[Baget et al., 2014b] Baget, J.-F., Garreau, F., Mugnier, M.-L., and Rocher, S. (2014b). Revisiting chase termination for existential rules and their extension to nonmonotonic negation. *arXiv preprint arXiv:1405.1071*.

[Baget et al., 2015] Baget, J.-F., Leclère, M., Mugnier, M.-L., Rocher, S., and Sipieter, C. (2015). Graal: A toolkit for query answering with existential rules. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, pages 328–344. Springer.

[Baget et al., 2011a] Baget, J.-F., Leclère, M., Mugnier, M.-L., and Salvat, E. (2011a). On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9-10):1620–1654.

[Baget et al., 2011b] Baget, J.-F., Mugnier, M.-L., Rudolph, S., and Thomazo, M. (2011b). Walking the complexity lines for generalized guarded existential rules. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 712.

[Baroni and Giacomin, 2009] Baroni, P. and Giacomin, M. (2009). Semantics of abstract argument systems. In *Argumentation in artificial intelligence*, pages 25–44. Springer.

[Bassiliades et al., 2006] Bassiliades, N., Antoniou, G., and Vlahavas, I. (2006). A defeasible logic reasoner for the semantic web. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2(1):1–41.

[Beeri and Vardi, 1981] Beeri, C. and Vardi, M. Y. (1981). The implication problem for data dependencies. In *International Colloquium on Automata, Languages, and Programming*, pages 73–85. Springer.

[Besnard and Hunter, 2001] Besnard, P. and Hunter, A. (2001). A logic-based theory of deductive arguments. *Artificial Intelligence*, 128(1-2):203–235.

[Bikakis and Antoniou, 2005] Bikakis, A. and Antoniou, G. (2005). DR-Prolog: a system for reasoning with rules and ontologies on the semantic web. In *AAAI*, volume 5, pages 1594–1595.

[Billington, 1993] Billington, D. (1993). Defeasible Logic is Stable. *Journal of logic and computation*, 3(4):379–400.

[Billington, 2004] Billington, D. (2004). A plausible logic which detects loops. In *NMR*, pages 65–71.

[Billington, 2008] Billington, D. (2008). Propositional clausal defeasible logic. *Logics in Artificial Intelligence*, pages 34–47.

[Billington et al., 2010] Billington, D., Antoniou, G., Governatori, G., and Maher, M. (2010). An inclusion theorem for defeasible logics. *ACM Transactions on Computational Logic (TOCL)*, 12(1):6.

[Bisquert et al., 2016] Bisquert, P., Croitoru, M., Dupin de Saint-Cyr, F., and Hecham, A. (2016). Substantive irrationality in cognitive systems. In *ECAI: European Conference on Artificial Intelligence*, pages 1642–1643.

[Bisquert et al., 2017] Bisquert, P., Croitoru, M., Dupin de Saint-Cyr, F., and Hecham, A. (2017). Formalizing Cognitive Acceptance of Arguments: Durum Wheat Selection Interdisciplinary Study. *Minds and Machines*, 27(1):233–252.

[Bondarenko et al., 1993] Bondarenko, A., Toni, F., and Kowalski, R. A. (1993). An assumption-based framework for non-monotonic reasoning. *Logic programming and non-monotonic reasoning (Lisbon, 1993)*, pages 171–189.

[Bourgaux, 2016] Bourgaux, C. (2016). *Inconsistency handling in ontology-mediated query answering.* PhD thesis, Université Paris-Saclay.

[Brewka, 2001] Brewka, G. (2001). On the relationship between defeasible logic and well-founded semantics. *Logic Programming and Nonmotonic Reasoning*, pages 121–132.

[Brewka and Woltran, 2010] Brewka, G. and Woltran, S. (2010). Abstract dialectical frameworks. In *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning.*

[Bryant and Krause, 2008] Bryant, D. and Krause, P. (2008). A review of current defeasible reasoning implementations. *The Knowledge Engineering Review*, 23(03):227–260.

[Byrne, 1989] Byrne, R. M. J. (1989). Suppressing valid inferences with conditionals. *Cognition*, 31(1):61–83.

[Caballero et al., 2008] Caballero, R., García-Ruiz, Y., and Sáenz-Pérez, F. (2008). A theoretical framework for the declarative debugging of datalog programs. In *Semantics in Data and Knowledge Bases*, pages 143–159. Springer.

[Cali et al., 2013] Cali, A., Gottlob, G., and Kifer, M. (2013). Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res*, 48:115–174.

[Calì et al., 2012] Calì, A., Gottlob, G., and Lukasiewicz, T. (2012). A general datalog-based framework for tractable query answering over ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:57–83.

[Cali et al., 2010a] Cali, A., Gottlob, G., Lukasiewicz, T., Marnette, B., and Pieris, A. (2010a). Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on*, pages 228–242. IEEE.

[Cali et al., 2010b] Cali, A., Gottlob, G., and Pieris, A. (2010b). Advanced processing for ontological queries. *Proceedings of the VLDB Endowment*, 3(1-2):554–565.

[Caminada, 2006] Caminada, M. (2006). On the issue of reinstatement in argumentation. In *European Workshop on Logics in Artificial Intelligence*, pages 111–123. Springer.

[Caminada and Amgoud, 2007] Caminada, M. and Amgoud, L. (2007). On the evaluation of argumentation formalisms. *Artificial Intelligence*, 171(5-6):286–310.

[Cayrol and Lagasquie-Schiex, 2005] Cayrol, C. and Lagasquie-Schiex, M.-C. (2005). On the acceptability of arguments in bipolar argumentation frameworks. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 378–389. Springer.

[Cecchi et al., 2006] Cecchi, L., Fillottrani, P., and Simari, G. R. (2006). On the complexity of DeLP through game semantics. In *11th. Intl. Workshop on Nonmonotonic Reasoning*, pages 386–394. Citeseer.

[Ceri et al., 1989] Ceri, S., Gottlob, G., and Tanca, L. (1989). What you always wanted to know about Datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166.

[Chandra et al., 1981] Chandra, A. K., Lewis, H. R., and Makowsky, J. A. (1981). Embedded implicational dependencies and their inference problem. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 342–354. ACM.

[Chesñevar et al., 2000] Chesñevar, C. I., Maguitman, A. G., and Loui, R. P. (2000). Logical models of argument. *ACM Computing Surveys (CSUR)*, 32(4):337–383.

[Croitoru and Vesic, 2013] Croitoru, M. and Vesic, S. (2013). What can argumentation do for inconsistent ontology query answering? In *International Conference on Scalable Uncertainty Management*, pages 15–29. Springer.

[Cuenca Grau et al., 2013] Cuenca Grau, B., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., and Wang, Z. (2013). Acyclicity notions for existential rules and their application to query answering in ontologies. *Journal of Artificial Intelligence Research*, 47:741–808.

[Dantsin et al., 2001] Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425.

[Dastani et al., 2005] Dastani, M., Governatori, G., Rotolo, A., and Van Der Torre, L. (2005). Preferences of agents in defeasible logic. In *Australasian Joint Conference on Artificial Intelligence*, pages 695–704. Springer.

[Deagustini et al., 2015] Deagustini, C. A., Martinez, M. V., Falappa, M. A., and Simari, G. R. (2015). On the Influence of Incoherence in Inconsistency-tolerant Semantics for Datalog+-. In *JOWO@ IJCAI*.

[Deutsch et al., 2008] Deutsch, A., Nash, A., and Remmel, J. (2008). The chase revisited. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 149–158. ACM.

[Dietz et al., 2014] Dietz, E.-A., Hölldobler, S., and Wernhard, C. (2014). Modeling the suppression task under weak completion and well-founded semantics. *Journal of Applied Non-Classical Logics*, 24(1-2):61–85.

[Dung, 1995] Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357.

[Dvovrák and Woltran, 2011] Dvovrák, W. and Woltran, S. (2011). On the intertranslatability of argumentation semantics. *Journal of Artificial Intelligence Research*, 41:445–475.

[Evans, 2003] Evans, J. S. B. T. (2003). In two minds: dual-process accounts of reasoning. *Trends in cognitive sciences*, 7(10):454–459.

[Fagin et al., 2005] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. (2005). Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124.

[Flouris et al., 2006] Flouris, G., Huang, Z., Pan, J. Z., Plexousakis, D., and Wache, H. (2006). Inconsistencies, negations and changes in ontologies. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1295. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

[Frawley et al., 1992] Frawley, W. J., Piatetsky-Shapiro, G., and Matheus, C. J. (1992). Knowledge discovery in databases: An overview. *AI magazine*, 13(3):57.

[Gallo et al., 1993] Gallo, G., Longo, G., Pallottino, S., and Nguyen, S. (1993). Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2):177–201.

[García and Simari, 2004] García, A. J. and Simari, G. R. (2004). Defeasible logic programming: An argumentative approach. *Theory and practice of logic programming*, 4(1+ 2):95–138.

[Governatori, 2005] Governatori, G. (2005). Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14(02n03):181–216.

[Governatori et al., 2004] Governatori, G., Maher, M. J., Antoniou, G., and Billington, D. (2004). Argumentation Semantics for Defeasible Logic. *Journal of Logic and Computation*, 14(5):675–702.

[Governatori and Rotolo, 2004] Governatori, G. and Rotolo, A. (2004). Defeasible logic: Agency, intention and obligation. In *International Workshop on Deontic Logic in Computer Science*, pages 114–128. Springer.

[Governatori and Rotolo, 2008] Governatori, G. and Rotolo, A. (2008). BIO logical agents: Norms, beliefs, intentions in defeasible logic. *Autonomous Agents and Multi-Agent Systems*, 17(1):36–69.

[Governatori and Rotolo, 2010] Governatori, G. and Rotolo, A. (2010). Changing legal systems: Legal abrogations and annulments in defeasible logic. *Logic Journal of the IGPL*, 18(1):157–194.

[Hecham et al., 2017a] Hecham, A., Arioua, A., Stapleton, G., and Croitoru, M. (2017a). An empirical evaluation of argumentation in explaining inconsistency tolerant query answering. *Proceedings of the 30th International Workshop on Description Logics*.

[Hecham et al., 2017b] Hecham, A., Bisquert, P., and Croitoru, M. (2017b). On the chase for all provenance paths with existential rules. In *International Joint Conference on Rules and Reasoning (RuleML+ RR 2017)*, pages 135—-150.

[Hecham et al., 2018] Hecham, A., Bisquert, P., and Croitoru, M. (2018). On a Flexible Representation of Defeasible Reasoning Variants. In *Proceedings of the 17th Conference on Autonomous Agents and MultiAgent Systems*.

[Hecham et al., 2016] Hecham, A., Croitoru, M., Bisquert, P., and Buche, P. (2016). Extending GWAPs for building profile aware associative networks. In *International Conference on Conceptual Structures*, pages 43—-58. Springer.

[Horty, 2002] Horty, J. F. (2002). Skepticism and floating conclusions. *Artificial Intelligence*, 135(1-2):55–72.

[Horty et al., 1990] Horty, J. F., Thomason, R. H., and Touretzky, D. S. (1990). A skeptical theory of inheritance in nonmonotonic semantic networks. *Artificial intelligence*, 42(2-3):311–348.

[Horty et al., 1987] Horty, J. F., Touretzky, D. S., and Thomason, R. H. (1987). A clash of intuitions: the current state of nonmonotonic multiple inheritance systems. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 476–482.

[Kahneman, 2011] Kahneman, D. (2011). *Thinking, fast and slow*. Macmillan.

[Kakas et al., 1998] Kakas, A. C., Kowalski, R. A., and Toni, F. (1998). The role of abduction in logic programming. *Handbook of logic in artificial intelligence and logic programming*, 5:235–324.

[Kravari et al., 2010] Kravari, K., Kastori, G.-E., Bassiliades, N., and Governatori, G. (2010). A contract agreement policy-based workflow methodology for agents interacting in the semantic web. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 225–239. Springer.

[Krötzsch and Rudolph, 2011] Krötzsch, M. and Rudolph, S. (2011). Extending decidable existential rules by joining acyclicity and guardedness.

In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 963. Citeseer.

[Lam, 2012] Lam, H. P. (2012). On the Derivability of Defeasible Logic.

[Lam et al., 2016] Lam, H.-P., Governatori, G., and Riveret, R. (2016). On ASPIC+ and Defeasible Logic. In *COMMA*, pages 359–370.

[Lembo et al., 2010] Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2010). Inconsistency-tolerant semantics for description logics. In *International Conference on Web Reasoning and Rule Systems*, pages 103–117. Springer.

[Lembo et al., 2015] Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2015). Inconsistency-tolerant query answering in ontology-based data access. *Web Semantics: Science, Services and Agents on the World Wide Web*, 33:3–29.

[Lembo and Ruzzi, 2007] Lembo, D. and Ruzzi, M. (2007). Consistent query answering over description logic ontologies. *Web Reasoning and Rule Systems*, pages 194–208.

[Lenzerini, 2002] Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM.

[Levesque and Brachman, 1987] Levesque, H. J. and Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational intelligence*, 3(1):78–93.

[Maher, 2001] Maher, M. J. (2001). Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming*, 1(6):691–711.

[Maher et al., 2001] Maher, M. J., Rock, A., Antoniou, G., Billington, D., and Miller, T. (2001). Efficient defeasible reasoning systems. *International Journal on Artificial Intelligence Tools*, 10(04):483–501.

[Maier and Nute, 2006] Maier, F. and Nute, D. (2006). Ambiguity propagating defeasible logic and the well-founded semantics. In *European Workshop on Logics in Artificial Intelligence*, pages 306–318. Springer.

[Maier and Nute, 2010a] Maier, F. and Nute, D. (2010a). Well-founded semantics for defeasible logic. *Synthese*, 176(2):243–274.

[Maier and Nute, 2010b] Maier, F. and Nute, D. (2010b). Well-founded semantics for defeasible logic. *Synthese*, 176(2):243–274.

[Makinson and Schlechta, 1991] Makinson, D. and Schlechta, K. (1991). Floating conclusions and zombie paths: two deep difficulties in the "directly skeptical" approach to defeasible inheritance nets. *Artificial intelligence*, 48(2):199–209.

[Marnette, 2009] Marnette, B. (2009). Generalized schema-mappings: from termination to tractability. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 13–22. ACM.

[Martinez et al., 2014] Martinez, M. V., Deagustini, A. C. D., Falappa, M. A., and Simari, G. R. (2014). Inconsistency-Tolerant Reasoning in Datalog +- Ontologies via an Argumentative Semantics. In *IBERAMIA*, volume 14, pages 15–27.

[Modgil and Prakken, 2014] Modgil, S. and Prakken, H. (2014). The ASPIC+ framework for structured argumentation: a tutorial. *Argument & Computation*, 5(1):31–62.

[Nguyen and Pallottino, 1989] Nguyen, S. and Pallottino, S. (1989). Hyperpaths and shortest hyperpaths. In *Combinatorial Optimization*, pages 258–271. Springer.

[Nute, 1988] Nute, D. (1988). *Defeasible reasoning: a philosophical analysis in prolog*. Springer.

[Onet, 2013] Onet, A. (2013). The chase procedure and its applications in data exchange. In *Dagstuhl Follow-Ups*, volume 5. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[Pandy and Chawla, 2003] Pandy, G. and Chawla, S. (2003). *A Simple and Efficient Algorithm for Hypercycle Detection*. School of Information Technologies, University of Sydney.

[Papadimitriou, 2003] Papadimitriou, C. H. (2003). *Computational complexity*. John Wiley and Sons Ltd.

[Poggi et al., 2008] Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., and Rosati, R. (2008). Linking data to ontologies. In *Journal on data semantics X*, pages 133–173. Springer.

[Pollock, 1987] Pollock, J. L. (1987). Defeasible reasoning. *Cognitive science*, 11(4):481–518.

[Prakken, 2002] Prakken, H. (2002). Intuitions and the modelling of defeasible reasoning: some case studies. In *Ninth Int Workshop on Nonmonotonic Reasoning*, pages 91–99, Toulouse.

[Prakken, 2010] Prakken, H. (2010). An abstract framework for argumentation with structured arguments. *Argument and Computation*, 1(2):93–124.

[Ragni et al., 2017] Ragni, M., Eichhorn, C., Bock, T., Kern-Isberner, G., and Tse, A. P. P. (2017). Formal Nonmonotonic Theories and Properties of Human Defeasible Reasoning. *Minds and Machines*, 27(1):79–117.

[Rahwan and Simari, 2009] Rahwan, I. and Simari, G. R. (2009). *Argumentation in artificial intelligence*, volume 47. Springer.

[Ramakrishnan and Ullman, 1995] Ramakrishnan, R. and Ullman, J. D. (1995). A survey of deductive database systems. *The journal of logic programming*, 23(2):125–149.

[Rocher, 2016] Rocher, S. (2016). *Querying Existential Rule Knowledge Bases: Decidability and Complexity*. PhD thesis, Université de Montpellier.

[Salvat and Mugnier, 1996] Salvat, E. and Mugnier, M.-L. (1996). Sound and complete forward and backward chainings of graph rules. In *International Conference on Conceptual Structures*, pages 248–262. Springer.

[Stein, 1992] Stein, L. A. (1992). Resolving ambiguity in nonmonotonic inheritance hierarchies. *Artificial Intelligence*, 55(2-3):259–310.

[Stenning and Lambalgen, 2005] Stenning, K. and Lambalgen, M. (2005). Semantic interpretation as computation in nonmonotonic logic: The real meaning of the suppression task. *Cognitive Science*, 29(6):919–960.

[Stenning and Van Lambalgen, 2012] Stenning, K. and Van Lambalgen, M. (2012). *Human reasoning and cognitive science*. MIT Press.

[Stolzenburg et al., 2003] Stolzenburg, F., García, A. J., Chesnevar, C. I., and Simari, G. R. (2003). Computing generalized specificity. *Journal of Applied Non-Classical Logics*, 13(1):87–113.

[Strass, 2013] Strass, H. (2013). Instantiating knowledge bases in abstract dialectical frameworks. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 86–101. Springer.

[Tarjan, 1972] Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160.

[Thimm, 2014] Thimm, M. (2014). Tweety - A Comprehensive Collection of Java Libraries for Logical Aspects of Artificial Intelligence and Knowledge Representation. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR'14)*.

[Thomazo, 2013] Thomazo, M. (2013). *Conjunctive Query Answering Under Existential Rules-Decidability, Complexity, and Algorithms*. PhD thesis, Université Montpellier II-Sciences et Techniques du Languedoc.

[Toni, 2014] Toni, F. (2014). A tutorial on assumption-based argumentation. *Argument & Computation*, 5(1):89–117.

[Vreeswijk, 1995] Vreeswijk, G. A. W. (1995). Interpolation of benchmark problems in defeasible reasoning. *Proceedings of 2nd World Conference on the Fundamentals in AI (WOCFAI'95)*, pages 453–468.

[Wan et al., 2015] Wan, H., Kifer, M., and Grosof, B. N. (2015). Defeasibility in answer set programs with defaults and argumentation rules. *Semantic Web*, 6(1):81–98.

[Wyner et al., 2013] Wyner, A., Bench-Capon, T., and Dunne, P. (2013). On the instantiation of knowledge bases in abstract argumentation frameworks. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 34–50. Springer.

[Yang et al., 2003] Yang, G., Kifer, M., and Zhao, C. (2003). Flora-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web. *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 671–688.

[Zhang et al., 2015] Zhang, H., Zhang, Y., and You, J.-H. (2015). Existential Rule Languages with Finite Chase: Complexity and Expressiveness. In *AAAI*, pages 1678–1685.