Extraction of Defeasible Proofs as Explanations

Luca Pasetto[1], Matteo Cristani[2], Guido Governatori[3], Francesco Olivieri[3] and Edoardo Zorzi[2]

[1]*Department of Computer Science, University of Luxembourg, Esch-sur-Alzette, Luxembourg*
[2]*Department of Computer Science, University of Verona, Verona, Italy*
[3]*Independent Researcher*

Abstract

Houdini is a Defeasible Deontic Logic reasoner that has been recently developed in Java. The algorithm employed in Houdini follows the proof conditions of the logic to conclude propositional and deontic literals, and is an efficient solution that provides the full extension of a theory. This computation is made in a forward-chaining complete way. Effectiveness is a fundamental property of the adopted approach, but we are also interested in providing an explicit reference to the reasoning that is employed to reach a conclusion. This reasoning is a proof that corresponds to an explanation for that conclusion, and such a proof is less natural to identify in a non-monotonic framework like Defeasible Logic than it would be in a classical one. Depending on the formalism and on the algorithm, the process of reconstructing a proof from a derived conclusion can be cumbersome. Intuitively, a proof consists of a support argument in favour of a literal to be concluded. However, it is necessary also to show that this argument is strong enough, either because the are no arguments against it, or because those arguments are weaker than it. In this paper, with a slight modification of the algorithm of Houdini, we show that it is possible to extract a proof for a defeasible literal in polynomial time, and that such a proof results minimal in its depth.

Keywords

Defeasible Logic, Proof extraction, Non-monotonic reasoning, Explainable AI

1. Introduction

The problem of reconstructing a proof for a specific conclusion in automated reasoning systems has been considered by scholars in the past for a large spectrum of logical frameworks, including non-monotonic ones. However, the investigation has been carried out only to a very limited extent for the case of Defeasible Logic in [1, 2], where the reconstruction process is based only on the proof system, and not on an algorithmic method.

Based on the algorithmic approach to the computation of the extension of a defeasible theory of [3], the authors of this work, along with others, have built an implementation, the *Houdini* system, that also operates on other aspects of the logical framework, including deontic operators and numeric constraints [4].

The aim of this work is to develop a method for providing an *explanation* for a defeasible literal that Houdini has computed to be in the extension by extracting a proof for it. It is thus

7th Workshop on Advances in Argumentation in Artificial Intelligence, November 09, 2023, Rome, Italy

luca.pasetto@uni.lu (L. Pasetto); matteo.cristani@univr.it (M. Cristani); guido@governatori.net (G. Governatori); francesco.olivieriphd@gmail.com (F. Olivieri); edoardo.zorzi@univr.it (E. Zorzi)

part of a stream that is carried out by some of the authors of this paper, along with others, on the topic of explainability in non-monotonic frameworks [5, 6].

The concept of explanation that we adopt here is based on and extends the classical notion of proof. We explain how a given conclusion is obtained by finding a chain from the assumptions to the conclusion, and moreover including in the proof all the information that is needed to beat possible counter-arguments. The actual proof system, and consequently the algorithm and its implementation, does not generate proof trees or other structures that can be used to explain the derivation, but rather a complex graph that is proving *all the literals* in the extension. Therefore, the proof needs to be extracted.

In order to introduce the reader to the conceptual framework shortly mentioned above, we devise an example, that is informally described here and reconsidered formally in Examples 2 and 3.

Example 1. Let us consider the following fictional situation, described in natural language, of a taxonomist seeing a platypus for the first time and debating whether it should be classified as a mammal or not based on a number of its characteristics. First of all, the taxonomist observes some of the characteristics of the platypus:

- The taxonomist notices a new animal, that we now know as a *platypus*;
- The taxonomist observes that the animal has the ability to produce milk to nourish its young;
- The taxonomist observes that the animal has its body covered in fur, providing insulation and protection from the environment;
- The taxonomist observes that the animal is warm-blooded;
- The taxonomist observes that the animal lays eggs;
- The taxonomist observes that males of the animal have a pair of venomous spurs;
- The taxonomist observes that the animal has a bill resembling that of a duck.

The taxonomist knows these rules that typically hold for classifying an animal:

- Producing milk is usually a mammalian trait;
- Having a layer of fur or hair is usually a mammalian trait;
- Being warm-blooded is usually a mammalian trait;
- Having a fur-covered body and being warm-blooded imply the ability to regulate the body temperature;
- Being able to regulate the body temperature is usually a mammalian trait;
- Laying eggs is usually a trait of birds, and not of mammals;
- Producing venom is usually a trait of reptiles, and not of mammals;
- Having a duck-like bill is usually a trait of birds, and not of mammals.

If the taxonomist considers all of the rules to have the same importance, he will not be able to take a decision in this case, as the rules that can be applied lead to decide that the platypus is both a mammal and a non-mammal. Indeed, the taxonomist then comes up with the following preferences for the rules:

- Producing milk is more important than laying eggs;
- Having a layer of fur is more important than producing venom;
- Being warm-blooded is more important than having a duck-like bill;
- Being able to regulate the body temperature is more important than producing venom;
- Being able to regulate the body temperature is more important than having a duck-like bill.

Based on these preferences, the taxonomist can then decide that the platypus can be classified as a mammal. The chain of reasoning is the following:

> *The taxonomist has observed that the animal has a number of mammalian traits, and he focuses on three: the ability to produce milk to nourish its young; its body is covered in fur; and it is warm-blooded. The taxonomist has also observed that the animal has three non-mammalian traits: it lays eggs; its males have a pair of venomous spurs; and it has a bill resembling that of a duck. Since each one of the existing non-mammalian traits is beated by one of the chosen three mammalian traits, it is possible to conclude that the animal is plausibly a mammal.*

Example 1 is a case of a derivation obtained by chaining simple steps and by comparing different arguments. In this paper, we provide a technique for the reconstruction of a proof for a literal derived in the extension of a Defeasible Logic theory by expanding the representation used during the computation of the extension and adopting a simple approach of graph exploration. We then prove that the entire approach can be conducted effectively, in polynomial time on deterministic machines.

The rest of the paper is structured as follows. Section 2 provides a brief introduction to Defeasible Logic; Section 3 describes Houdini and the algorithm used in the system; Section 4 discusses the problem of interest, the algorithm and its computational properties; Section 5 provides reference to relevant literature; and finally Section 6 draws conclusions and identifies possible future developments.

2. Defeasible Logic

Defeasible Logic (DL) [7, 8] is a simple, flexible, and efficient rule-based non-monotonic formalism. Its strength lies in its constructive proof theory, as DL allows to draw meaningful conclusions from a (potentially) conflicting and incomplete knowledge base. In non-monotonic systems, more accurate conclusions can be obtained when more pieces of information become available. Throughout the years, many variants of DL have been proposed for the modelling of different application areas: agents [9, 10], legal reasoning [11, 12], and business processes [13] in the scope of regulatory compliance.

We start with the language. Let PROP be a set of propositional atoms, and Lab be a set of arbitrary labels (the names of the rules). The set of *literals* is $\mathrm{Lit} = \mathrm{PROP} \cup \{\neg p \mid p \in \mathrm{PROP}\}$. We shall use lower-case Roman letters to denote literals, and lower-case Greek letters to denote rules.

The *complement* (or *opposite*) of a literal l is denoted by $\sim l$. If l is a positive literal p then $\sim l$ is $\neg p$, and if l is a negative literal $\neg p$ then $\sim l$ is p. The definition of a defeasible theory is the standard one in DL ([8]).

Definition 1 (Defeasible theory). A *defeasible theory* D is a tuple $(F, R, >)$, being $F \subseteq \mathrm{Lit}$ the set of facts, R the set of rules, and $>$ the superiority relation.

The set of facts F denotes simple pieces of information that are considered to be always true, like "Sylvester is a cat", formally $cat(Sylvester)$. The set of rules R contains three types of rules: *strict rules*, *defeasible rules*, and *defeaters*. The *superiority relation*[1] $>\subseteq R \times R$ is an irreflexive relation to solve conflicts in case of potentially contradicting information. The superiority relation is not an order, as it is not transitive.

A defeasible theory is *finite* if F and R are finite. A *rule* $r \in R$ is an expression $r\colon A(r) \hookrightarrow C(r)$ such that:

1. $r \in \mathrm{Lab}$ is the *unique name* of the rule;
2. $A(r) \subseteq \mathrm{Lit}$ is the (possibly empty) set of the *antecedents*[2];
3. $\hookrightarrow \in \{\rightarrow, \Rightarrow, \leadsto\}$ denotes, resp., strict rules, defeasible rules, and defeaters;
4. $C(r) \in \mathrm{Lit}$ is the *consequent*, a single literal.

A strict rule is a rule in the classical sense: whenever the premises are indisputable, so is the conclusion. The statement "All computer scientists are humans" is formulated through the strict rule

$$r\colon \mathit{CScientist}(X) \rightarrow \mathit{Human}(X),$$

since there is no exception to it. As in [8], we consider only a propositional version of this logic, and we do not take function symbols into account. Every expression with variables represents the finite set of its variable-free instances.

Defeasible rules represent statements that can be defeated by contrary evidence: the statement "Computer scientists travel to the city of the conference that accepted their paper" is thus represented with the defeasible rule

$$s\colon \mathit{CScientist}, \mathit{PaperAccepted} \Rightarrow \mathit{TravelConference}$$

Defeaters are special rules whose only purpose is to prevent the derivation of the opposite conclusion; the statement "During pandemic travels might be prohibited" is hence represented by the defeater

$$t\colon \mathit{Pandemic} \leadsto \neg \mathit{TravelConference}$$

In this case, using a defeater is preferable to using a defeasible rule, as we simply want to "block" the derivation of travelling.

We use some conventional abbreviations. The set of strict rules is R_{s}, the set of strict and defeasible rules is R_{sd}. $R[l]$ is the set of rules whose consequent is literal l. We say that a literal l *appears in* D, if there exists a rule such that l is either one of its antecedents, or the consequent.

A *conclusion* of a defeasible theory D is a *tagged formula*, an expression of the form $\pm\#l$, $l \in \mathrm{Lit}$ and $\# \in \{\Delta, \partial\}$, with the following meanings:

[1] Also referred to as *priority* or *preference relation*.

[2] We shall interchangeably use *antecedent* to refer to $A(r)$.

- $+\Delta l$ means that l is *strictly proved*, i.e., there is a strict proof of l in D;
- $-\Delta l$ means that l is *strictly refuted*, i.e., a strict proof of l does not exist in D;
- $+\partial l$ means that l is *defeasibly proved*, i.e., there is a defeasible proof of l in D;
- $-\partial l$ means that l is *defeasibly refuted*, i.e., a defeasible proof of l does not exist in D.

We refer to $\pm\Delta$ and $\pm\partial$ as *proof tags*. The definition of Δ describes forward chaining of strict rules, while ∂ represents the non-monotonic part of the logic.

Given a defeasible theory D, a *proof* P *of length* n *in* D is a finite sequence $P(1), P(2), \ldots, P(n)$ of tagged formulas of the type $+\Delta l$, $-\Delta l$, $+\partial l$, $-\partial l$, where the proof conditions defined hereafter hold; $P(1..n)$ denotes the first n steps of P. Proofs are based on the notions of a rule being *applicable* or *discarded*. A rule is *applicable* when all the elements in its antecedent have been proved; a rule is *discarded* if at least one of such elements has been refuted.

Definition 2 (Applicable & Discarded). Given a defeasible theory $D = (F, R, >)$, a rule $r \in R$ is

- $\#$*-applicable*, $\# \in \{\Delta, \partial\}$ at $P(n+1)$ iff $\forall a_i \in A(r).\ +\#a_i \in P(1..n)$;
- $\#$*-discarded*, $\# \in \{\Delta, \partial\}$ at $P(n+1)$ iff $\exists a_i \in A(r).\ -\#a_i \in P(1..n)$.

When it is not needed or it is clear from the context, we will omit Δ*-applicable*, Δ*-discarded*, ∂*-applicable*, ∂*-discarded*, and just say that a rule is applicable or discarded. The proof conditions hereafter are also standard in DL.

$+\Delta l$: If $P(n+1) = +\Delta l$ then
    (1) $l \in F$, or
    (2) $\exists r \in R_s[l]$ s.t. r is Δ-applicable.

A literal is *strictly proved* if (1) it is a fact, or (2) there exists a Δ-applicable strict rule for it.

$+\partial l$: If $P(n+1) = +\partial l$ then
    (1) $+\Delta l \in P(1..n)$, or
    (2.1) $-\Delta{\sim}l \in P(1..n)$, and
    $\exists r \in R_{\mathrm{sd}}[l]$ s.t.
        (2.2) r is ∂-applicable, and
        $\forall s \in R[{\sim}l]$ then either
            (2.3.1) s is ∂-discarded, or
            (2.3.2) $\exists t \in R[l]$ s.t.
                (2.3.2.1) t is ∂-applicable and
                (2.3.2.2) $t > s$.

A literal is *defeasibly proved* if (1) it has been strictly proved, or (2.1) the opposite has not been strictly proved and (2.2) there exists a ∂-applicable rule (r for l) such that every attack (s for the opposite ${\sim}l$) is either (2.3.1) ∂-discarded or (2.3.2) defeated by a ∂-applicable rule (ζ for l).

Negative proof tags $-\Delta$ and $-\partial$ are omitted as they are obtained by applying the *strong negation principle* to their positive counter-parts [14], which applies the function that simplifies a formula by moving all negations to an innermost position in the resulting formula, replacing the positive tags with the respective negative tags, and vice versa. The last notions of this section are those of extension of a theory, and of equivalence of theories. Informally, an *extension* is everything that is derived and refuted, and two theories are *equivalent* when they have the same extension.

Definition 3 (Extension). Consider two defeasible theories D and D'. The set of positive and negative conclusions of D is the *extension* $E(D) = (+\varDelta, -\varDelta, +\partial, -\partial)$, where $\pm\# = \{l | \, l$ appears in D and $D \vdash \pm\# l\}$, $\# \in \{\varDelta, \partial\}$ and we use symbol $\vdash$ to indicate that there is a derivation of l from D.

We call $+\varDelta$, $-\varDelta$, $+\partial$, $-\partial$ the *extension sets* of a theory.

Definition 4 (Theory equivalence). Given two defeasible theories D and D', they are *equivalent* iff $E(D) \equiv E(D')$.

3. Houdini

In this section, we briefly describe the DL reasoner Houdini. While this paper focuses just on Defeasible Logic, Houdini also works with its deontic version, Defeasible Deontic Logic. At the current state, Houdini is a web-app based on Java with a single endpoint and a responsive and user-friendly UI. The choice of programming language fell on Java due to its high maintainability and portability, as well as the existence of mature web frameworks that are suitable to develop straightforward applications, that is, easy to understand for nonprofessional people such as students and researchers with limited knowledge of programming. In particular, Houdini is open source, available on request, and developed by using Java 11 and Spring Boot 2.7.1.

Front-end checks with colour-coded error messages guide users and let them correct ill-formed inputs, forbidden characters, and plain wrong syntax. These controls are responsive and can fire an error message at the top of the screen with each single inserted character and mouse click. Input fields are generated dynamically and can be eliminated with the press of a button so as to have a clean interface devoid of too many unused spaces. At the start, users are presented with three different kinds of empty fields wherein they can insert the elements of the theory one-by-one, or an upload button which can be used to load a JSON file with the whole theory specified. The syntax required for this file is easy to understand and mimics the structure of a hand-inserted theory.

As an example, the following rule is syntactically correct and is allowed by the system: “a, [O]b, x > 3 =>[P] ∼c". The body is “a, [O]b, x > 3" and so it is made up of three elements: the literal “a", a proposition marked with the deontic tag O with variable name “b" and parameter “x" and a logical constraint “x > 3".

Superiority relations are simply two rule names separated by a “>” character: the left-side indicates the superior rule, whereas the right-side the inferior one. A rule name is simply the character “r" plus a number. For example, “r5 > r1" is a valid superiority relation. Once the user inserts the data Houdini parses and validates it, then it computes the extension sets $\pm\#_m$, $\# \in \{\varDelta, \partial\}$, $m \in \{\mathsf{C}, \mathsf{O}, \mathsf{P}\}$, and finally it displays these sets in an appropriate result page.

Reasoning about and computing the extension sets of a Defeasible Deontic Logic theory requires, first, a clear and precise representation of it. Hence, Houdini expects users to provide data which complies with a precise syntax that cannot lead to any ambiguity. In particular, a theory must be provided by following a context-free grammar specification which is then used by Houdini's custom parser (generated using ANTLR 4 parse tree listeners) to validate it, server-side. This is the first operation done on the inputted data.

If the input is ill-formed, an error is returned. Client-side checks are also performed to avoid unnecessary parsing, and to deliver quicker feedback. Houdini's parser not only validates the user data but also builds, incrementally, as it traverses the tree, the necessary internal data structures that are later used by the reasoner to compute the extension sets. Some optimisations are also employed, both on the grammar side, such as grouping and prioritising the most common expressions (chosen empirically), and on the parser side.

Moreover, mathematical expressions (which can appear inside a rule head) and logical expressions (which can appear inside a rule body logical constraint) are transformed in their respective Reverse Polish Notation (RPN) versions to speed-up things later when such expressions must be evaluated multiple times with actual arguments, as RPN expressions are easily evaluated and with minimal memory requirements.

The reasoner module, which is performed after the parsing phase, implements the core functionality of Houdini: computing the extension of the given defeasible deontic theory. It is composed by two sub-modules: *Strict Reasoner* and *Defeasible Reasoner*, whose behaviour follows the proof conditions provided above.

The main algorithm, the reasoner module, acts on Literal and Rule instances which have been initialised by the parser and which contain, together with a general Theory container instance, all the necessary information to work with. For example, a Literal instance contains or contains references to: all the rules wherein it appears as head or in the body, its deontic modality, its opposite literal, whether at the current moment it has been assigned by the reasoner to an extension set ($\pm\Delta$ and $\pm\partial$) or it is still undecided, and other information. On the other hand, a Rule instance contains or contains references to: all the literals appearing in its body and head, extra information about its head literal (deontic mode, parameters, logical expressions, …), and other information.

Also, all these classes implement methods called by the reasoner to compute the extension sets. The decision of having a strict pairing between data and methods that can act inside objects is due to the need to have to use similar but not identical functions depending on the type of the literal, its mode, the presence of mathematical expressions in the head of the rule, etc.

4. Proof Reconstruction in Defeasible Logic

When the algorithm halts because it has computed the extension of the given defeasible theory D, for a literal p such that $D \vdash +\Delta p$ or $D \vdash +\partial p$, in order to explain why p has been derived (either definitely or defeasibly), it is necessary to reconstruct its proof.

The *strict proof tree* of literal p such that $D \vdash +\Delta p$ is defined inductively as follows.

Definition 5 (Strict proof tree). Given a defeasible theory $D = (F, R, >)$ and a literal p such that $D \vdash +\Delta p$, the *strict proof tree* $\Pi(+\Delta p)$ of p is defined as follows:

- if $D \vdash +\Delta p$ because $p \in F$, the *strict proof tree* $\Pi(+\Delta p)$ consists of a node labeled by p;
- if $D \vdash +\Delta p$ because there is a rule $r \in R_s[p]$ s.t. $\forall a \in A(r) : D \vdash +\Delta a$, the *strict proof tree* $\Pi(+\Delta p)$ consists of a node labeled by $+\Delta p$ with a subtree $\Pi(+\Delta a)$ for each $a \in A(r)$.

After this definition, in the rest of the paper we focus on defeasible proofs. For a defeasible proof of $+\partial p$, we make the choice of showing in the proof only the chains that end up with $+\partial p$, and not the chains that attack these, but we make sure to include in the proof the information on how the attacks are beaten. The *defeasible proof tree* of literal p such that $D \vdash +\partial p$ is defined inductively as follows.

Definition 6 (Defeasible proof tree). Given a defeasible theory $D = (F, R, >)$ and a literal p such that $D \vdash +\partial p$, the *defeasible proof tree* $\Pi(+\partial p)$ of p is defined as follows:

- if $D \vdash +\partial p$ because $D \vdash +\Delta p$, the *defeasible proof tree* $\Pi(+\partial p)$ consists of a node labeled by $+\partial p$ with a subtree $\Pi(+\Delta p)$;
- if $D \vdash +\partial p$ because there is a rule $r \in R_{sd}[p]$ s.t. $\forall a \in A(r)\colon D \vdash +\partial a$ and for every $s \in R[\sim p]$ that is activated, there exists a rule $t \in R_{sd}[p]$ s.t. $\forall b \in A(t)\colon D \vdash +\partial b$ and $t > s$, the *defeasible proof tree* $\Pi(+\partial p)$ consists of a node labeled by $+\partial p$ with a subtree $\Pi(+\partial a)$ for each $a \in A(r)$, and for every $s \in R[\sim p]$ that is activated and defeated by its corresponding rule $t \in R[p]$, a subtree $\Pi(+\partial a)$ for each $a \in A(t)$.

In case that a rule is a hypothesis, that is, it is a defeasible rule without premises, we show it in the proof by connecting the concluded literal to a special node ε that we call the *dummy fact*. Since we are working in a non-monotonic context, the algorithm can activate more than one rule concluding for the same literal. Indeed, the algorithm is a forward chaining one, and if a rule can be activated, then it will be. For this reason, when we do proof reconstruction on a derived literal, we might end up with a directed acyclic graph (DAG) instead of a tree.

The *defeasible proof DAG* of literal p such that $D \vdash +\partial p$ is defined inductively as follows.

Definition 7 (Defeasible proof DAG). Given a defeasible theory $D = (F, R, >)$ and a literal p such that $D \vdash +\partial p$, the *defeasible proof DAG* $\Gamma(+\partial p)$ of p is defined as follows:

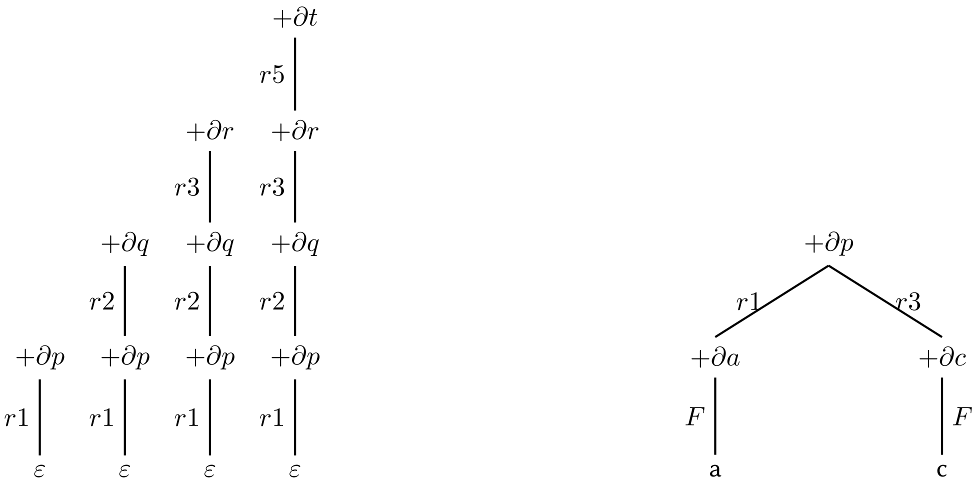
- if $D \vdash +\partial p$ because $D \vdash +\Delta p$, the *defeasible proof DAG* $\Gamma(+\partial p)$ consists of a node labeled by $+\partial p$ with an edge from $\Gamma(+\Delta p)$;
- if $D \vdash +\partial p$ because there is a rule $r \in R_{sd}[p]$ s.t. $\forall a \in A(r)\colon D \vdash +\partial a$ and for every $s \in R[\sim p]$ that is activated, there exists a rule $t \in R_{sd}[p]$ s.t. $\forall b \in A(t)\colon D \vdash +\partial b$ and $t > s$, the *defeasible proof DAG* $\Gamma(+\partial p)$ consists of a node labeled by $+\partial p$ with an edge from $\Gamma(+\partial a)$ for each $a \in A(r)$, and for every $s \in R[\sim p]$ that is activated and defeated by its corresponding rule $t \in R[p]$, an edge from $\Gamma(+\partial a)$ for each $a \in A(t)$.

Also here, in the case of a hypothesis rule, we show it in the proof by connecting the concluded literal to a special node ε that we call the *dummy fact*.

In general, the reconstruction process described in the definitions above is not trivial to perform after having computed the extension. Indeed, if a proof has been built by applying only one rule at each step, as in a direct chain from facts towards a literal that is concluded directly without attacks against it, then the tree can be easily reconstructed in polynomial time. However, it is not immediate for more complicated cases.

Let us exhibit some examples of reconstructed proofs, that will enlighten the approach we can adopt to solve the issue mentioned above.

Example 2. Consider the following theory $D = (\emptyset, R, >)$, where $R = \{r1\colon\ \Rightarrow p,\ r2\colon\ p \Rightarrow q,\ r3\colon\ q \Rightarrow r,\ r4\colon\ s \Rightarrow \neg r,\ r5\colon\ r \Rightarrow t,\ r6\colon\ \neg p \Rightarrow \neg t\}$, and $>$ is empty. The extension of D is such that $+\partial = \{p,\ q,\ r,\ t\}$. Figure 1a shows the corresponding proof trees.



(a) The proof trees reconstructed from the extension of Example 2.

(b) The proof tree reconstructed for $+\partial p$ in Example 3.

Example 3. Consider the following theory $D = (F, R, >)$, where $F = \{a,\ b,\ c\}$, $R = \{r1\colon\ a \Rightarrow p,\ r2\colon\ b \rightsquigarrow \neg p,\ r3\colon\ c \Rightarrow p,\ r4\colon\ c \Rightarrow \neg p,\ \}$, with $r1 > r4$ and $r3 > r2$. The extension of D is such that $+\partial = \{a,\ b,\ c,\ p\}$. Figure 1b shows the proof tree for $+\partial p$.

In order to compute the proof DAG, we need some further information. In particular, for each derived literal that we want to extract the proof of, we need to know *when* the literal has been derived *for the first time* in that reasoning chain. This information can be managed to identify when a rule has actually been used to derive a literal, and build the above mentioned DAG.

Before going into the details of the algorithm that deals with the annotation needed to compute the reconstruction, we need to discuss an issue that may appear complicated: *cycles*. In a defeasible theory, we may have dependency circularity, namely there might be a derivation that starts from facts and goes towards the same literal conclusion twice (or even more times). This, in principle, could cause computational unfeasibility: we may generate a number of derivation chains that is potentially infinite, and since the cycles that are generated can be combined in sequences, this can give rise to combinatorial effects, transforming the reconstruction of proofs into a search problem in the graph of derivation chains.

This can be remedied by a single technique. For a literal that has been concluded once, we can ignore it if it is derived again afterwards, as this is not relevant for the existence of a derivation chain. In fact, we derive the following lemma, whose proof is a straightforward consequence of the above reasoning and is therefore omitted.

Lemma 1. *Any derivation chain obtained in a forward-chaining computation of the propositional extension of a defeasible theory contains a cycle-free derivation chain.*

As a consequence of Lemma 1, we can conceive a solution to the marking problem for these chains, that in turn leads us to the algorithm for computing the proof DAG proposed below. The idea is to compute the extension as usually, and to also add to each literal in the extension sets a number representing the *first appearance* of that literal during the computation. We call this number the *derivation depth* of the considered literal.

By means of this annotation, a simple and straightforward modification of the original algorithm implemented in Houdini (and more generally, in any forward-chaining method based on [3]), we can manage the computation of the proof DAG of Definition 7 in the following manner. First of all, we compute the extension with the described annotation, yielding an *annotated dependency graph* (ADG). This graph is a slight modification of known dependency graphs [15]. Given a defeasible theory $D = (F, R, >)$, the ADG for D is built as follows:

- For every $f \in F$, build a node and annotate it with derivation depth $d = 0$;
- For every literal p that is derived by rule $r \in R$ from literals whose nodes are already in the ADG, create an edge e that connects each $a \in A(r)$ to a new node annotated with derivation depth $d = d' + 1$, where $d' = max(depth(A(r)))$;
- The edge e of the previous step is marked with the label of the applied rule r.

The graph obtained via the above method is labelled by a label function L that associates a set of rules to an edge. Therefore, given a defeasible theory $\mathcal{D} = (F, R, >)$ we have that such a graph is defined by $G = \langle V, E, L \rangle$, where L is a function that goes from E to 2^R. However, this labelling does not generate an exponential growth of the computational cost, as discussed below. The process above, that starts from facts and computes the extension with annotations for the derivation depth, has the exact same computational cost of the process without the annotation.

We are now able to sketch the algorithm for proof reconstruction. The basic concept is to start with a concluded literal p for which we aim at reconstructing the proof, and provide subsequent expansions within the ADG while following the links in a way that does not incur in the issue of circular dependencies. In particular, we expand a literal λ towards another literal λ' that is connected to λ *if, and only if*, the literal depth of λ' is less than the depth of λ. In this way, after a finite number of steps, each of which can compute, at maximum, a subset of the literals (and therefore with a maximum number of steps that is $O(n)$, being n the number of literals), we reach a set of literals with depth 0. These literals are therefore facts, and they trigger the theory in a way that derives the concluded literal p, and they also trigger the arguments against p, that are however proven to be weaker.

The description sketched above is defined formally in Algorithm 1. We denote by $\delta(\lambda)$ the depth of the literal λ in a defeasible theory. We can keep this value undefined for the literals that are not proven in the theory.

Based on the reasoning we reported above, we can prove the following theorem.

Algorithm 1 Algorithm that computes the proof DAG of a literal.

```
Require: $G = \langle V, E, L\rangle$ ADG of $\mathcal{D}$, and a literal $\lambda$ proven by $\mathcal{D}$
Ensure: $S = \langle V_S, E_S, L_S\rangle$, subgraph of $G$ s.t. $S = \Gamma(\lambda)$
 1: $V_S = \{\lambda\}$
 2: $E_S,\ L_S = \emptyset$
 3: $B = \{\alpha \in V\ s.t.\ (\alpha, \lambda) \in E\}$
 4: while $(B \neq \emptyset)$ do
 5:     $B' = \emptyset$
 6:     for $(y \in B)$ do
 7:         for $(x \in V\ s.t.\ (x, y) \in E)$ do
 8:             $e = (x, y)$
 9:             if $(\delta(x) < \delta(y))$ then
10:                 $E_S = E_S \cup \{e\}$
11:                 for $(r \in L(e))$ do
12:                     $B' = B' \cup A(r)$
13:                     $L_S = L_S \cup \{(e, r)\}$
14:                 end for
15:             end if
16:         end for
17:     end for
18:     $B = B'$
19:     $V_S = V_S \cup B$
20: end while
```

Theorem 1. *Algorithm 1 correctly computes a proof DAG for a literal λ in a defeasible theory $\mathcal{D}$ in $O(n^2 \cdot m)$, with n the number of literals and m the number of rules in D.*

Proof. Consider a graph G that is the ADG of a defeasible theory D, and literal λ, proven in $\mathcal{D}$.

First of all, we can easily show that the graph S outputted by Algorithm 1 is a DAG. By construction, there is only one vertex in S that has no in-edge, and there exist at least one vertex that has no out-edge. Let us assume, by contradiction, that we can build a cycle that starts and ends in one vertex different from the root and the leaves found by the above reasoning. This means that on the cycle in Lines 6-16 we generate one vertex $e = (x, y)$ that is entered twice in one sub-cycle 10-13 in B'. This is impossible, for it would have been the case that simultaneously $\delta(x) < \delta(y)$ and $\delta(y) < \delta(x)$. This proves the first claim.

The fact that the obtained DAG S actually is a proof DAG can be easily proven by observing that the set of facts in G, excluding the dummy fact ε (that manages the hypotheses), trigger the rules in the theory. By construction of the chains in backward search provided by cycle 4-19, when no hypothetical rules exist in the theory, this leads to a chain from facts to λ, that, being λ proven in $\mathcal{D}$, by construction of the graph G in input to Algorithm 1, is unbeaten.

When the theory contains hypotheses, we need to observe that the dummy fact ε triggers *only* the rules that are taken from G in S, not all the triggered rules, and therefore we cannot ensure automatically the above reasoning to be applicable. However, since the triggered hypotheses are a *proper subset* of the hypotheses, and the literal λ is proven by $\mathcal{D}$ we can conclude the same.

The complexity of the algorithm is determined by the execution of the cycles in Algorithm 1. Although the labels are subsets of rules, there is no combinatorial explosion, as the rules are used in the labelling only once, with the first derivation of the conclusion of the rule. If a label "consumes" some of the available rules, no further usage of those rules can occur in subsequent labelling. Therefore, the process ends in polynomial time. □

We consider a proof to be *minimal* when it is minimal in its depth. By applying the reconstruction process by Algorithm 1 and with the proof of Theorem 1, we can ensure that the extracted proof is minimal. The proof of this is omitted for the sake of space.

Corollary 1. Algorithm 1 computes a minimal proof.

We conclude this section by providing a formalisation of the theory of Example 1 from Section 1, and in Figure 2 we then show the DAG and the proof tree for the conclusion $+\partial m$.

Example 4. Consider the theory of Example 1, $D = (F,\ R,\ >)$, with $F = \{p\}$ and $R = \{r1\colon\ p \Rightarrow k,\ r2\colon\ p \Rightarrow f,\ r3\colon\ p \Rightarrow w,\ r4\colon\ p \Rightarrow e,\ r5\colon\ p \Rightarrow v,\ r6\colon\ p \Rightarrow b,\ r7\colon\ k \Rightarrow m,\ r8\colon\ f \Rightarrow m,\ r9\colon\ w \Rightarrow m,\ r10\colon\ f, w \Rightarrow t,\ r11\colon\ t \Rightarrow m,\ r12\colon\ e \Rightarrow \neg m,\ r13\colon\ v \Rightarrow \neg m,\ r14\colon\ b \Rightarrow \neg m\ \}$, and superiority relations: $r7 > r12$, $r8 > r13$, $r9 > r14$, $r11 > r13$, and $r11 > r14$. We can claim that $+\partial m$ holds. Figure 2 shows the proof tree for $+\partial m$.

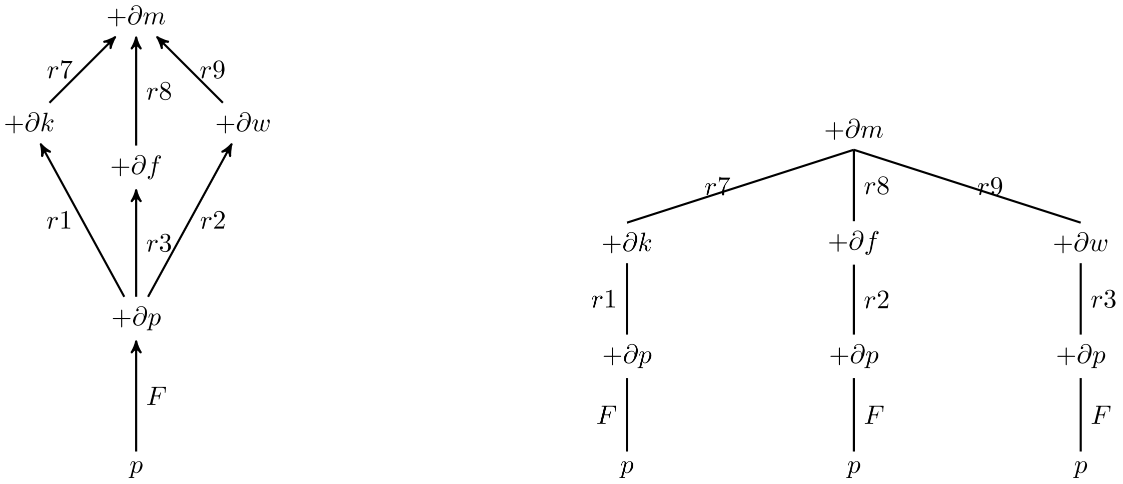


Figure 2: The DAG and proof tree reconstructed for $+\partial m$ in Example 4.

5. Related work

Proof reconstruction is a well-studied topic that has received much attention in the area of automated reasoning. It has been treated in SMT solvers such as Z3 [16], first-order theorem provers such as E [17], SPASS [18] and Vampire [19], higher-order theorem provers such as Leo-III [20], interactive theorem provers such as Isabelle [21] and HOL Light [22], and in combinations of interactive theorem provers with other systems [23, 24, 25]. Also paramodulation methods employed for higher-order derivations are central [26]. A problem for proof reconstruction in automated reasoning is when data, which can be either clauses in classical logic or rules in our case, gets eliminated during the reasoning process because it is redundant. This is not the case with Houdini, that does not perform such deletion operations.

Closer to our field of interest, non-monotonic reasoning, are the investigations on the difference between proving and exhibiting a proof, as in [27] and [28]. The specificity of non-monotonic systems lies on the need for a proof that takes into consideration not only a positive argument in favour of a given thesis, but also the arguments against that thesis, that should be shown to be beaten by the positive argument.

In parallel to the research on proof reconstruction, a discussion has started on the topic of *argument reconstruction*, where proof and argument are dual concepts, in the sense that the skeptical framework consider a proof *only* when there are arguments for it. Walton's influential research on the nature of argumentation [29] has considered since the very beginning argument reconstruction as one of the most interesting research questions in argumentation theory. Some recent technical advancements have been brought by Aakhus *et al.* in [30, 31] while designing strategies for large scale discourses.

The need for argument reconstruction and for technical solutions to it has been an inspiration for this work, as it has been for other studies, such as the significant research by Wieland [32]. In his study, the author addresses the problem of how to choose an argument among many when they all appear to be plausible in deriving a conclusion. His approach looks in a critical way at the foundational studies made by Ryle, and is based on the notion of *regress*, which shares commonalities with our technique.

6. Conclusions and further work

In this paper, we dealt with the problem of computationally reconstructing proofs in a reasoner for propositional Defeasible Logic (DL). We have shown that the problem shares issues with proof reconstruction in automated reasoning for classical logic and argument reconstruction in argumentation theory. The solution that we propose is rooted in the principles of skeptical non-monotonic reasoning and in the definition of proof in Defeasible Logic. Moreover, the given algorithm is computationally feasible and outputs a proof that is minimal in its depth.

There are two directions that are still to explore. First of all, we are interested in exploring proof reconstruction for Defeasible Deontic Logic. Deontic reasoning requires more subtleties in the reconstruction process, but then the exhibition of such proofs could be useful in a variety of legal contexts where a reasoning system can be used. The second direction lies on the possibility of introducing *information asymmetry* as in game theory. The fact that we can provide a proof that is accepted in a dialogue entirely lies on the agreement about the underlying background. If the background is the law, this is typically managed with an epistemic pure operator, that asserts the knowledge of facts, rules and superiority relation by each of the discussants. In other cases, it may be possible to disagree among discussants about the background, that generates a quite different dialogue convergence protocol. A proposition in this context can be justified by proposing an argument, but also by rejecting counterarguments within a common base or by negotiation on this base. In case the base is formalised in Defeasible Logic, all of these elements could be computed by a proof reconstruction algorithm like the one showed in this paper.

References

[1] N. Bassiliades, G. Antoniou, G. Governatori, Proof explanation in the dr-device system, LNCS-LNAI 4524 LNCS (2007) 249–258. doi:10.1007/978-3-540-72982-2_19.

[2] G. Antoniou, A. Bikakis, N. Dimaresis, M. Genetzakis, G. Georgalis, G. Governatori, E. Karouzaki, N. Kazepis, D. Kosmadakis, M. Kritsotakis, G. Lilis, A. Papadogiannakis, P. Pediaditis, C. Terzakis, R. Theodosaki, D. Zeginis, Proof explanation for the semantic web using defeasible logic, LNCS-LNAI 4798 LNAI (2007) 186–197. doi:10.1007/978-3-540-76719-0_21.

[3] G. Antoniou, D. Billington, G. Governatori, M. Maher, Representation results for defeasible logic, ACM Transactions on Computational Logic 2 (2001) 255–287. doi:10.1145/371316.371517.

[4] M. Cristani, G. Governatori, F. Olivieri, L. Pasetto, F. Tubini, C. Veronese, A. Villa, E. Zorzi, Houdini (unchained): an effective reasoner for defeasible logic, in: Proceedings of the 6th Workshop on Advances in Argumentation in Artificial Intelligence, volume 3354 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022.

[5] G. Governatori, F. Olivieri, A. Rotolo, M. Cristani, Stable normative explanations, Frontiers in Artificial Intelligence and Applications 362 (2022) 43–52. doi:10.3233/FAIA220447.

[6] G. Governatori, F. Olivieri, A. Rotolo, M. Cristani, Inference to the stable explanations, LNCS-LNAI 13416 LNAI (2022) 245–258. doi:10.1007/978-3-031-15707-3_19.

[7] D. Nute, Defeasible logic, in: Handbook of Logic in Artificial Intelligence and Logic Programming, volume 3, Oxford University Press, 1987.

[8] G. Antoniou, D. Billington, G. Governatori, M. Maher, Representation results for defeasible logic, ACM Trans. Comput. Log. 2 (2001) 255–287. doi:10.1145/371316.371517.

[9] G. Governatori, F. Olivieri, S. Scannapieco, A. Rotolo, M. Cristani, The rationale behind the concept of goal, Theory Pract. Log. Program. 16 (2016) 296–324. doi:10.1017/S1471068416000053.

[10] M. Cristani, F. Olivieri, L. Pasetto, Revising ethical principles and norms in hybrid societies: Basic principles and issues, in: G. Jezic, J. Chen-Burger, M. Kusek, R. Sperka, R. J. Howlett, L. C. Jain (Eds.), Agents and Multi-Agent Systems: Technologies and Applications 2021, Springer Singapore, Singapore, 2021, pp. 103–113.

[11] G. Governatori, F. Olivieri, A. Rotolo, S. Scannapieco, Computing strong and weak permissions in defeasible logic, J. Philos. Log. 42 (2013) 799–829. doi:10.1007/s10992-013-9295-1.

[12] M. Cristani, C. Tomazzoli, F. Olivieri, L. Pasetto, An ontology of changes in normative systems from an agentive viewpoint, in: F. De La Prieta, P. Mathieu, J. A. Rincón Arango, A. El Bolock, E. Del Val, J. Jordán Prunera, J. Carneiro, R. Fuentes, F. Lopes, V. Julian (Eds.), Highlights in Practical Applications of Agents, Multi-Agent Systems, and Trust-worthiness. The PAAMS Collection, Springer International Publishing, Cham, 2020, pp. 131–142.

[13] F. Olivieri, M. Cristani, G. Governatori, Compliant business processes with exclusive choices from agent specification, in: PRIMA, volume 9387 of *LNCS*, Springer, 2015, pp. 603–612. doi:https://10.1007/978-3-319-25524-8_43.

[14] G. Governatori, V. Padmanabhan, A. Rotolo, A. Sattar, A defeasible logic for modelling policy-based intentions and motivational attitudes, Log. J. IGPL 17 (2009) 227–265. doi:10.

1093/jigpal/jzp006.

[15] L. Aceto, W. Fokkink, C. Verhoef, Structural operational semantics, in: J. Bergstra, A. Ponse, S. Smolka (Eds.), Handbook of Process Algebra, Elsevier Science, Amsterdam, 2001, pp. 197–292. doi:https://doi.org/10.1016/B978-044482830-9/50021-7.

[16] L. M. de Moura, N. S. Bjørner, Proofs and refutations, and z3, in: LPAR Workshops, 2008.

[17] S. Schulz, S. Cruanes, P. Vukmirović, Faster, higher, stronger: E 2.3, in: P. Fontaine (Ed.), Proc. of the 27th CADE, Natal, Brasil, number 11716 in LNAI, Springer, 2019, pp. 495–507.

[18] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, P. Wischnewski, Spass version 3.5, in: R. A. Schmidt (Ed.), Automated Deduction – CADE-22, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 140–145.

[19] L. Kovács, A. Voronkov, First-order theorem proving and vampire, in: N. Sharygina, H. Veith (Eds.), Computer Aided Verification, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 1–35.

[20] A. Steen, C. Benzmüller, Extensional higher-order paramodulation in leo-iii, Journal of Automated Reasoning 65 (2021) 775 – 807. doi:10.1007/s10817-021-09588-x.

[21] A. Krauss, C. Sternagel, R. Thiemann, C. Fuhs, J. Giesl, Termination of isabelle functions via termination of rewriting, LNCS-LNAI 6898 LNCS (2011) 152 – 167. doi:10.1007/978-3-642-22863-6_13.

[22] C. Kaliszyk, J. Urban, Proch: Proof reconstruction for hol light, LNCS-LNAI 7898 LNAI (2013) 267 – 274. doi:10.1007/978-3-642-38574-2_18.

[23] S. Böhme, Proof reconstruction for Z3 in Isabelle/HOL, in: 7th International Workshop on Satisfiability Modulo Theories (SMT '09), 2009.

[24] S. Böhme, A. C. J. Fox, T. Sewell, T. Weber, Reconstruction of z3's bit-vector proofs in hol4 and isabelle/hol, volume 7086 LNCS, 2011, p. 183 – 198. doi:10.1007/978-3-642-25379-9_15.

[25] N. Sultana, C. Benzmüller, L. C. Paulson, Proofs and reconstructions, in: C. Lutz, S. Ranise (Eds.), Frontiers of Combining Systems, Springer International Publishing, Cham, 2015, pp. 256–271.

[26] A. Asperti, E. Tassi, Higher order proof reconstruction from paramodulation-based refutations: The unit equality case, LNCS-LNAI 4573 LNAI (2007) 146 – 160. doi:10.1007/978-3-540-73086-6_14.

[27] R. Blanco, Z. Chihani, D. Miller, Translating between implicit and explicit versions of proof, LNCS-LNAI 10395 LNAI (2017) 255 – 273. doi:10.1007/978-3-319-63046-5_16.

[28] H. Barbosa, J. C. Blanchette, M. Fleury, P. Fontaine, Scalable fine-grained proofs for formula processing, Journal of Automated Reasoning 64 (2020) 485 – 510. doi:10.1007/s10817-018-09502-y.

[29] D. N. Walton, Dialogue theory for critical thinking, Argumentation 3 (1989) 169 – 184. doi:10.1007/BF00128147.

[30] M. Aakhus, M. G. Benovitz, Argument reconstruction and socio-technical facilitation of large scale argumentation, volume 363, 2008, p. 77 – 81. doi:10.1145/1479190.1479201.

[31] M. Aakhus, M. Lewiriski, Argument analysis in large-scale deliberation, 2011. doi:10.1075/z.163.12aak.

[32] J. W. Wieland, Regress argument reconstruction, Argumentation 26 (2012) 489 – 503. doi:10.1007/s10503-012-9264-9.