GRIFFITH UNIVERSITY

Forgetting for defeasible logic

Author

Antoniou, Grigoris, Eiter, Thomas, Wang, Kewen

Published

2012

Conference Title

LOGIC FOR PROGRAMMING, ARTIFICIAL INTELLIGENCE, AND REASONING (LPAR-18)

DOI

10.1007/978-3-642-28717-6_9

Downloaded from

http://hdl.handle.net/10072/48474

Griffith Research Online

https://research-repository.griffith.edu.au

Forgetting for Defeasible Logic

Grigoris Antoniou[1], Thomas Eiter[2] and Kewen Wang[3]

[1] FORTH-ICS, Greece and University of Huddersfield, UK
antoniou@ics.forth.gr
[2] Institut für Informationssysteme, Technische Universität Wien, Austria
eiter@kr.tuwien.ac.at
[3] School of Information and Communication Technology, Griffith University, Australia
k.wang@griffith.edu.au

Abstract. The concept of *forgetting* has received significant interest in artificial intelligence recently. Informally, given a knowledge base, we may wish to forget about (or discard) some redundant parts (such as atoms, predicates, concepts, etc) but still preserve the consequences for certain forms of reasoning. In nonmonotonic reasoning, so far forgetting has been studied only in the context of extension based approaches, mainly answer-set programming. In this paper forgetting is studied in the context of defeasible logic, which is a simple, efficient and sceptical nonmonotonic reasoning approach.

1 Introduction

The concept of *forgetting* has received significant interest in Artificial Intelligence recently. Informally, given a knowledge base, we may wish to forget about (or discard) some redundant parts (such as atoms, predicates, concepts, etc) but still preserve the consequences for certain forms of reasoning. Forgetting has been introduced in many formalisms for knowledge representation, for instance, in propositional logic [17, 18], first-order logic [19, 31], modal logic [25, 14, 30], description logic [28, 27, 16], and logic programming [7, 26, 29]. The theory of forgetting has also been fruitfully applied in various contexts, e.g. in cognitive robotics [19], for resolving conflicts and inconsistencies [18, 7], and in particular, in ontology engineering [15].

Regarding nonmonotonic logics, so far forgetting has been studied only in the context of extension based approaches, mainly answer-set programming. On the other hand, defeasible reasoning is a nonmonotonic reasoning approach in which the gaps due to incomplete information are closed through the use of defeasible rules. Defeasible logics were introduced by Nute [23] and developed over several years [3, 10, 6]. They allow for defeasible reasoning, where a conclusion supported by a rule might be overturned by the effect of another rule; they also have a monotonic reasoning component, and a priority on rules. One feature of their design is that they are quite simple, allowing efficient reasoning; in fact, basic defeasible logics have linear time complexity [20]. Its properties have been thoroughly studied and analyzed, with strong results in terms of proof theory [22, 3] and semantics [10, 21].

Defeasible logic has recently attracted considerable interest. Its use in various application domains has been advocated, including the modeling of regulations and business

rules [2], modeling of contracts [8], legal reasoning [13], agent negotiations [9, 24], modeling of agents and agent societies [12, 11], and applications to the Semantic Web [1, 4] and ambient intelligence [5].

However, to our best knowledge, *a theory of forgetting for defeasible logic is still missing*. In this paper, we first examine a naive definition of forgetting for defeasible logic and explain why such a definition is insufficient for practical applications. Specifically, the naive definition does not preserve any syntactical information of the original theory. For this reason, we develop a new approach to forgetting for defeasible logic and establish some of its formal properties, including completeness, computational complexity, and structure preservation. Salient features of the solution provided include linear time complexity, and linear size of the output of (iterated) forgetting.

The paper is organised as follows. Section 2 presents the basics of defeasible logic. Section 3 introduces the problem of forgetting in the context of defeasible logic, and presents a first, naive solution. An analysis of the weaknesses of this approach leads to an improved approach, presented in Section 4, where its properties and computational complexity are also analyzed. We conclude the paper with plans for future work in Section 5.

2 Defeasible Logic

In this paper we restrict attention to essentially propositional defeasible logic. Rules with free variables are interpreted as rule schemas, that is, as the set of all ground instances. If q is a literal, $\sim q$ denotes the complementary literal (if q is a positive literal p then $\sim q$ is $\neg p$; and if q is $\neg p$, then $\sim q$ is p).

Rules are defined over a *language* (or *signature*) Σ, the set of propositions (atoms) and labels that may be used in the rule.

A rule $r : A(r) \hookrightarrow C(r)$ consists of its unique *label* r, its *antecedent* $A(r)$, which is a finite set of literals (possibly omitted if empty), an arrow $\hookrightarrow$ (which is a placeholder for concrete arrows to be introduced in a moment), and its *head* (or *consequent*) $C(r)$ which is a literal. In writing rules we omit set notation for antecedents, and sometimes we omit the label when it is not relevant for the context. There are three kinds of rules, each designated by a different arrow:

- $\rightarrow$ designates *strict rules* (definitional rules);
- $\Rightarrow$ designates *defeasible rules* (nonmonotonic rules that may be attacked by other rules); and
- $\rightsquigarrow$ designates *defeaters* (special rules that do not support positive conclusions but may only attack other rules).

Given a set R of rules, we denote by R_s the set of all strict rules in R, by R_{sd} the set of all strict and defeasible rules in R, and by $R[q]$ the set of rules in R with consequent q.

A *superiority relation on* R is a binary relation $>$ on R. When $r_1 > r_2$, then r_1 is called *superior* to r_2, and r_2 *inferior* to r_1. Intuitively, $r_1 > r_2$ expresses that r_1 overrules r_2, should both rules be applicable. Typically we assume $>$ to be acyclic (that is, the transitive closure of $>$ is irreflexive).

A *defeasible theory* is a triple $D = (F, R, >)$ where F is a finite set of literals (called *facts*), R a finite set of rules, and $>$ is an acyclic superiority relation on R. We call D *decisive*, if the atom dependency graph of D is acyclic.

Example 1. The following is a theory $D = (F, R, >)$ in defeasible logic, where

$F = \{\ \}$

$R = \{r_0 :\rightarrow d$
$\quad r_1 :\Rightarrow a$
$\quad r_2 :\Rightarrow \neg a$
$\quad r_3 : a \Rightarrow b$
$\quad r_4 :\Rightarrow \neg b$
$\quad r_5 : d, b \Rightarrow c$
$\quad r_6 :\Rightarrow \neg c \quad \}$,

and $r_1 > r_2 \;\; r_3 > r_4$.

A *conclusion* of a theory D in defeasible logic is a tagged literal and can have one of the following four forms:

- $+\Delta q$, which is intended to mean that q is definitely provable in D.
- $-\Delta q$, which is intended to mean that we have proved that q is not definitely provable in D.
- $+\partial q$, which is intended to mean that q is defeasibly provable in D.
- $-\partial q$, which is intended to mean that we have proved that q is not defeasibly provable in D.

If we are able to prove q definitely, then q is also defeasibly provable. This is a direct consequence of the formal definition below. It resembles the situation in, say, Reiter's default logic: a formula is sceptically provable from a default theory $T = (W, D)$ (in the sense that it is included in each extension) if it is provable from the set W of classical formulas.

Let us consider the theory in Example 1. Intuitively, d can be derived by the strict rule r_0 and thus $+\Delta d$ is provable from the theory, and so is $+\partial d$. Also, $-\Delta\neg d$ is provable, as an attempt to prove $\neg d$ fails finitely. Similarly, $+\partial a$ is provable (as $r_1 > r_2$), and so is $-\partial\neg a$ is provable (defeasible logic is sceptical, and the rule supporting a is superior to the rule supporting $\neg a$).

Provability for defeasible logic is based on the concept of a *derivation* (or *proof*) in $D = (F, R, >)$. A *derivation* is a finite sequence $P = (P(1), ..., P(n))$ of tagged literals constructed by inference rules. There are four inference rules (corresponding to the four kinds of conclusion) that specify how a derivation may be extended. ($P(1..i)$ denotes the initial part of the sequence P of length i):

$+\Delta$: We may append $P(i+1) = +\Delta q$ if either
$\quad q \in F$ or
$\quad \exists r \in R_s[q]\, \forall a \in A(r) : +\Delta a \in P(1..i)$

To prove $+\Delta q$, this means we need to establish a proof for q using facts and strict rules only. This is a deduction in the classical sense. No proofs for the negation of q need to

be considered (in contrast to defeasible provability below, where opposing chains of reasoning must be taken into account, too).

To prove $-\Delta q$, that is, that q is not definitely provable, q must not be a fact. In addition, we need to establish that every strict rule with head q is *known to be* inapplicable. Thus for every such rule r there must be at least one antecedent a for which we have established that a is not definitely provable ($-\Delta a$).

$-\Delta$: We may append $P(i{+}1) = -\Delta q$ if
$q \notin F$ and
$\forall r \in R_s[q]\, \exists a \in A(r) : -\Delta a \in P(1..i)$

It is worth noticing that this definition of nonprovability does not involve loop detection. Thus if D consists of the single rule $p \rightarrow p$, we can see that p cannot be proven, but defeasible logic is unable to prove $-\Delta p$.

$+\partial$: We may append $P(i{+}1) = +\partial q$ if either
(1) $+\Delta q \in P(1..i)$ or
(2) (2.1) $\exists r \in R_{sd}[q]\ \forall a \in A(r)$: $+\partial a \in P(1..i)$ and
(2.2) $-\Delta \sim q \in P(1..i)$ and
(2.3) $\forall s \in R[\sim q]$ either
(2.3.1) $\exists a \in A(s)$: $-\partial a \in P(1..i)$ or
(2.3.2) $\exists t \in R_{sd}[q]$ such that
$\forall a \in A(t)$: $+\partial a \in P(1..i)$ and $t > s$

Let us illustrate this definition. To show that q is defeasibly provable, we have two choices: (1) we show that q is already definitely provable; or (2) we need to argue using the defeasible part of D as well. In particular, we require that there must be a strict or defeasible rule with head q which can be applied (2.1). But now we need to consider possible attacks, that is, reasoning chains in support of $\sim q$. To be more specific: to prove q defeasibly we must show that $\sim q$ is not definitely provable (2.2). Also (2.3) we must consider the set of all rules (including defeaters) which are not known to be inapplicable and which have head $\sim q$. Essentially each such rule s attacks the conclusion q. For q to be provable, each such rule must be counterattacked by a rule t with head q with the following properties: (i) t must be applicable at this point, and (ii) t must be stronger than s. Thus each attack on the conclusion q must be counterattacked by a stronger rule.

The definition of the proof theory of defeasible logic is completed by the condition $-\partial$. It is nothing more than a strong negation of the condition $+\partial$.

$-\partial$: We may append $P(i{+}1) = -\partial q$ if
(1) $-\Delta q \in P(1..i)$ and
(2) (2.1) $\forall r \in R_{sd}[q]\ \exists a \in A(r)$: $-\partial a \in P(1..i)$ or
(2.2) $+\Delta \sim q \in P(1..i)$ or
(2.3) $\exists s \in$ R[$\sim q$] such that
(2.3.1) $\forall a \in A(s)$: $+\partial a \in P(1..i)$ and
(2.3.2) $\forall t \in R_{sd}[q]$
either $\exists a \in A(t)$: $-\partial a \in P(1..i)$ or $t \not> s$

To prove that q is not defeasibly provable, we must first establish that it is not definitely provable. Then we must establish that it cannot be proven using the defeasible part of the theory. There are three possibilities to achieve this: either we have established that none of the (strict and defeasible) rules with head q can be applied (2.1); or $\sim q$ is definitely provable (2.2); or there must be an applicable rule r with head $\sim q$ such that no possibly applicable rule s with head $\sim q$ is superior to s (2.3).

The elements of a derivation P in D are called *lines* of the derivation. We say that a tagged literal L *is provable in* $D = (F, R, >)$, denoted $D \vdash L$, if there is a derivation in D such that L is a line of P. When D is obvious from the context we write $\vdash L$.

Consider the theory in Example 1 again. Then the conclusions that can be drawn from this theory are: $-\Delta a$, $-\Delta\neg a$, $+\partial a$, $-\partial\neg a$, $-\Delta b$, $-\Delta\neg b$, $+\partial b$, $-\partial\neg b$, $-\Delta c$, $-\Delta\neg c$, $-\partial c$, $-\partial\neg c$, $+\Delta d$, $+\partial d$, $-\Delta\neg d$, $-\partial\neg d$.

Two defeasible theories D and D' are *equivalent*, denoted $D \equiv D'$, if for each tagged literal L, it holds that $D \vdash L$ iff $D' \vdash L$.

3 Forgetting in Defeasible Logic: A Naive Approach

Intuitively, given a knowledge base K and a set of atoms A, which we wish to forget from K, we are interested in obtaining a knowledge base $K' = forget(K, A)$ such that: (a) K' does not contain any occurrence of any atom in A, and (b) K' is equivalent to K for all atoms not belonging to A.

In the context of defeasible logic, this intuitive idea translates to the following task:

> Given a defeasible theory D and a set of atoms A, find a defeasible theory D' such that (a) D' does not contain any occurrences of any atom in A, and (b) $D' \vdash q \Leftrightarrow D \vdash q$ for any tagged literal q that does not contain any atom in A.

We denote such a theory D' by $forget(D, A)$; for singleton $A = \{a\}$, we also write $forget(D, a)$. Note that, by definition, any two different defeasible theories D' and D'' satisfying conditions (a) and (b) above are equivalent. Thus $forget(D, A)$ is uniquely defined, modulo equivalence of defeasible theories.

A first, naive approach to solving this problem is applying the following algorithm.

Algorithm 1

1. Compute the set $cons(D)$ of all conclusions of D.
2. Delete all conclusions from $cons(D)$ that contain an atom in A.
3. Construct a defeasible theory D' that has exactly those conclusions contained in the result of step 2. □

Before continuing, let us make an initial remark about this approach: While quite naive, it is computationally feasible, in contrast to, say, propositional logic or description logics where forgetting has already been studied; the reason is that the conclusions of a defeasible theory (step 1) can be computed in linear time, as shown in [20]. Indeed, the algorithm provided in that work constitutes the first step in our algorithm.

Thus, the only question that remains is whether step 3 can be carried out. To provide an answer to this question, we have to carefully analyze the proof theory of defeasible

Table 1. Distinct cases of provable sets of conclusions involving p and $\neg p$ (cf. [22])

Case	Set of conclusions	Rules with this outcome
1		$p \rightarrow p, \neg p \rightarrow \neg p$
2	$+\Delta\neg p, +\partial\neg p$	$p \rightarrow p, \rightarrow \neg p$
3	$-\Delta\neg p$	$p \rightarrow p, \neg p \Rightarrow \neg p$
4	$-\Delta\neg p, -\partial\neg p$	$p \rightarrow p$
5	$+\partial p, -\Delta\neg p, -\partial\neg p$	$\Rightarrow p, p \rightarrow p$
6	$+\Delta p, +\partial p, +\Delta\neg p, +\partial\neg p$	$\rightarrow p, \rightarrow \neg p$
7	$+\Delta p, +\partial p, -\Delta\neg p, -\partial\neg p$	$\rightarrow p$
8	$-\Delta p, +\partial p, -\Delta\neg p, -\partial\neg p$	$\Rightarrow p$
9	$-\Delta p, -\Delta\neg p$	$p \Rightarrow p, \neg p \Rightarrow \neg p$
10	$-\Delta p, -\Delta\neg p, -\partial\neg p$	$p \Rightarrow p$
11	$-\Delta p, -\partial p, -\Delta\neg p, -\partial\neg p$	

logic. This analysis provides a number of relationships among different conclusions; for example, if $D \vdash +\Delta p$ then also $D \vdash +\partial p$. And also $D \nvdash +\partial p$ or $D \nvdash -\partial p$ (or both). In addition, there are interrelationships between possible conclusions involving an atom p and its negation. For example, if $D \vdash +\Delta\neg p$ and $D \vdash -\Delta p$ then $D \vdash -\partial p$.

Maher et al. [22] provide a thorough analysis of the proof theory of defeasible logic, and reveal that regarding the provability of a set of conclusions involving p and its negation, there are essentially eleven distinct cases, out of $2^8 = 256$ syntactically possible sets of conclusions, which are summarized in Table 1. For each of these cases, we provide in the rightmost column a respective defeasible theory that contains only the literals p and $\neg p$. Thus we demonstrate how step 3 of Algorithm 1 can be carried out.

Let us explain a few of these cases. Case 1 is the one where no conclusion regarding p or $\neg p$ can be drawn. This outcome is achieved by strict loops for both p and $\neg p$, which are not detected by the proof theory of Section 2, therefore no conclusion can be drawn.

Case 2 represents that $D \nvdash -\Delta p$ and $D \nvdash +\partial p$ while $D \vdash +\Delta\neg p$ and $D \vdash +\partial\neg p$. The theory for it has again a loop for p, but $\neg p$ is definitely, thus also defeasibly, provable.

The theory for case 3 consists of the rules $\{p \rightarrow p, \neg p \Rightarrow \neg p\}$. Here there is a strict loop for p, but a defeasible loop for $\neg p$. The latter allows one to derive at least the definite non-provability of $\neg p$ (as there is no fact nor a strict rule with head $\neg p$).

Case 6 is the one of inconsistency, which may only occur if both p and $\neg p$ are definitely (hence also defeasibly) provable, as achieved by the theory provided. Finally, in case 10 absence of facts or strict rules achieves definite non-provability. The loop on p prevents us from deriving $-\partial p$, whereas absence of a rule with head $\neg p$ gives us $-\partial\neg p$.

Summarizing the discussion so far, all three steps of the algorithm provided in this section are feasible, and provide a solution to the problem of forgetting.

Proposition 1. *Given a defeasible theory D and a set of atoms A, Algorithm 1 computes a defeasible theory $D' = forget(D, A)$.*

Moreover, the algorithm can be evaluated efficiently: given Table 1, step 1 is the only involved step which however can be done in linear time [20].

Proposition 2. *Algorithm 1 computes $forget(D, A)$ in linear time.*

Though these theoretical results are positive, the algorithm is still naive, in the sense that it completely abandons the original structure of the knowledge base and returns an artificial theory.

Example 2. Take the theory in Example 1 and now suppose we wish to forget about b. Then Algorithm 1 returns $forget(D, A) = (F', R', >')$ where

$$R' = \{ \rightarrow d \\ \Rightarrow a \}$$

and F' and $>'$ are empty.

This approach is clearly not suitable for practical purposes. The question arises whether the theoretical properties can be achieved while maintaining the original knowledge structure to the extent possible. This question is studied in the next section.

4 Forgetting in Defeasible Logic: An Improved Approach

The idea for the improved algorithm is the following: assuming that we have the complete picture regarding the derivability of p and $\neg p$ (one of the eleven cases of the previous section), we transform the set of rules in a way that takes into account the derivability of p and $\neg p$. For example, consider the rules

$$r_1 : p, s \rightarrow \neg a$$
$$r_2 : p, q \Rightarrow a$$

and the derivability $D \vdash -\Delta p$ and $D \vdash +\partial p$. Rule r_1 cannot fire as we have established that p is not definitely derivable. And we can delete p from the body of r_2 as p is defeasibly derivable. As a result of this transformation, the resulting set of rules contains no occurrence of p, and represents the result of forgetting p. Let us now consider another interesting case. Consider the rule

$$r : p \Rightarrow s$$

and the derivability $D \vdash -\Delta p$ and $D \nvdash +\partial p$ and $D \nvdash -\partial p$. Clearly rule r cannot fire ever, as p cannot be derived. However, $-\partial p$ cannot be derived either (indicating a cyclic situation; see e.g. case 10 in Table 1). Suppose we have in our theory another rule

$$r' :\Rightarrow \neg s.$$

Rule r' does not fire to prove $\neg s$ because it is attacked by r and r cannot be discarded as we cannot prove non-derivability of its antecedent p. Simply deleting r when forgetting p would delete this attack on r', thus enabling derivability of $\neg s$ and altering the set of conclusions not involving p. The solution to this problem is to turn r into a defeater

$$r :\rightsquigarrow s.$$

This rule fails to support derivation of s but is able to attack r' and prevents it from firing.

As a final example, consider the rule

$$r : p \rightarrow q$$

and suppose $D \vdash +\partial p$ and $D \vdash -\Delta p$. In this case, the strict rule r fails to prove q strictly, but is able to prove q defeasibly. If we were to simply remove r in the process of forgetting p, we would lose this defeasible provability of q. To avoid this problem, we replace r by the defeasible rule

$$r :\Rightarrow q$$

Based on these ideas, we provide in the following the full transformation in all eleven cases of derivability of p and $\neg p$.

Case 1: $\{\,\}$

- Replace all rules $r : A \hookrightarrow q$ where p or $\neg p$ appear in A with $r : A \setminus \{p, \neg p\} \rightsquigarrow q$.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 2: $\{+\Delta\neg p, +\partial\neg p\}$

- Replace all rules $r : A \hookrightarrow q$ such that p appears in A with $r : A \setminus \{p\} \rightsquigarrow q$.
- Remove all occurrences of $\neg p$ from all rule bodies.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 3: $\{-\Delta\neg p\}$

- Replace all rules $r : A \hookrightarrow q$ such that p appears in A with $r : A \setminus \{p\} \rightsquigarrow q$.
- Replace all rules $r : A \hookrightarrow q$ such that $\neg p$ appears in A with $r : A \setminus \{\neg p\} \rightsquigarrow q$.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 4: $\{-\Delta\neg p, -\partial\neg p\}$

- Replace all rules $r : A \hookrightarrow q$ such that p appears in A with $r : A \setminus \{p\} \rightsquigarrow q$.
- Remove all rules with $\neg p$ as one of its antecedents.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 5: $\{+\partial p, -\Delta\neg p, -\partial\neg p\}$

- Replace all strict rules $r : A \rightarrow q$ with p in A by the defeasible rule $r : A \setminus \{p\} \Rightarrow q$.
- Remove all occurrences of p in the bodies of defeasible rules and defeaters.
- Remove all rules with $\neg p$ as one of its antecedents.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 6: $\{+\Delta p, +\partial p, +\Delta\neg p, +\partial\neg p\}$

- Remove all occurrences of p and $\neg p$ from all rule bodies.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 7: $\{+\Delta p, +\partial p, -\Delta\neg p, -\partial\neg p\}$

- Remove all occurrences of p from the bodies of all rules.
- Remove all rules with $\neg p$ as one of its antecedents.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 8: $\{-\Delta p, +\partial p, -\Delta\neg p, -\partial\neg p\}$

- Replace all strict rules $r : A \rightarrow q$ with p in A by the defeasible rule $r : A \setminus \{p\} \Rightarrow q$.
- Remove all occurrences of p from the bodies of all defeasible rules and defeaters.
- Remove all rules with $\neg p$ as one of its antecedents.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 9: $\{-\Delta p, -\Delta\neg p\}$

- Replace all rules $r : A \hookrightarrow q$ where p or $\neg p$ appear in A with $r : A \setminus \{p, \neg p\} \rightsquigarrow q$.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 10: $\{-\Delta p, -\Delta\neg p, -\partial\neg p\}$

- Replace all rules $r : A \hookrightarrow q$ where p appears in A with $r : A \setminus \{p\} \rightsquigarrow q$.
- Remove all rules with $\neg p$ as one of its antecedents.
- Delete all facts and rules with p or $\neg p$ in their head.

Case 11: $\{-\Delta p, -\partial p, -\Delta\neg p, -\partial\neg p\}$

- Remove all rules in which p or $\neg p$ occurs.

This leads us then to the following improved algorithm for forgetting.

Algorithm 2

1. Compute all conclusions concerning atoms in A.
2. For each atom p in A, transform the rules in D to obtain D'.
3. Delete all priority pairs $r > s$ where r or s was deleted in step 2. □

Step 1 can be carried out, in the worst case, by computing all conclusions of D. Step 2 is based on the analysis provided above.

Example 3. Reconsider the theory D in Example 1. Note that $-\Delta b$, $+\partial b$, $-\Delta\neg b$, and $-\partial\neg b\}$ are provable from D. So, by Case 8 above, application of Algorithm 2 yields the theory $D' = (\emptyset, R', >')$ as $forget(D, b)$, where

$$\begin{aligned} R' = \{\ & r_0 :\rightarrow d \\ & r_1 :\Rightarrow a \\ & r_2 :\Rightarrow \neg a \\ & r_5 : d \Rightarrow c \\ & r_6 :\Rightarrow \neg c\ \ \} \end{aligned}$$

and $r_1 >' r_2$.

Example 4. Consider the defeasible theory $D = (F, R, >)$ where

$$R = \{\ r_1 : a \Rightarrow a \\ \quad\ r_2 : a \Rightarrow b \\ \quad\ r_3 :\Rightarrow \neg b\ \ \}$$

and F and $>$ are empty. Here $\neg b$ is not defeasibly derivable because a cannot be shown to be non-provable ($-\partial a$ cannot be derived). Thus it would be a mistake to simply delete r_2 when forgetting a. Instead, Algorithm 2 correctly turns a modified rule r_2 (without a in the body) into a defeater, giving as result the defeasible theory $D' = (\emptyset, R', \emptyset)$ where R' contains the rules

$$r_2 :\rightsquigarrow b$$
$$r_3 :\Rightarrow \neg b.$$

The following proposition states that Algorithm 2 indeed provided a correct solution for forgetting.

Proposition 3. *Given a defeasible theory D and a set of atoms A, Algorithm 2 computes a defeasible theory $D' = forget(D, A)$.*

Proof. (Sketch) We want to show that for each tagged literal L whose literal is neither p nor $\neg p$, $D \vdash L$ iff $D' \vdash L$.

If $D \vdash L$, then there exists a derivation of L in D: $P = (P(1), ..., P(n))$. We use P' to denote the sequence of tagged literals obtained from P by removing every tagged literal whose literal is either p or $\neg p$. Then we can prove that P' is a derivation of L in D' by induction on the length n of P. The induction step can be done by examining the Cases 1-11 in Algorithm 2.

On the other hand, if $P' = (P'(1), ..., P'(m))$ is a derivation of L in D', we can construct a derivation P of L in D inductively as follows.

Assume that a derivation $(P(1), ..., P(u))$ of $P'(m-1)$ in D has been constructed. Again, the induction step can be done by examining the Cases 1-11 in Algorithm 2. For instance, here we consider the Case 7 in Algorithm 2: $D \vdash \{+\Delta p, +\partial p, -\Delta\neg p, -\partial\neg p\}$.

To construct a derivation of $P'(m) = L$ in D, consider four possible cases:
Case 1. $L = +\Delta q$: Then either $q \in F$ or $\exists r' \in R'_s[q]\ \forall a \in A(r') :\ +\Delta a \in P'(1..m-1)$. If $q \in F$, then it is done. If $q \notin F$, by the (last) induction assumption, $\forall a \in A(r') :\ +\Delta a \in P(1..u)$. If $r' \in R$, then $(P(1), ..., P(u), L)$ is already a derivation of L in D. So we assume that $r' \notin R$. By the Case 7 in Algorithm 2, there exists $r \in R$ such that $A(r) = A(r') \cup \{p\}$. Let $P'' = (P''(1), ..., P''(v))$ be a derivation of $+\Delta p$; then $P = (P(1), ..., P(u), P''(1), ..., P''(v), L)$ is a derivation of L in D.
Cases 2,3,4: $L = -\Delta q$, $L = +\partial q$, $L = -\partial q$: we can similarly construct a derivation of L in D. □

4.1 Semantic Properties

Algorithm 2 satisfies a number of desirable properties. One is that forgetting atoms which do not occur in the theory cause no change.

Proposition 4. *Let D be a defeasible theory and let A be a set of atoms. Then $forget(D, A) \equiv forget(D, A')$ where $A' \subseteq A$ is the set of atoms from A that occur in D; in particular $forget(D, \emptyset) = D$.*

Another such property is irrelevance of syntax with respect to equivalence.

Proposition 5. *Let D and D' be defeasible theories such that $D \equiv D'$. Then, for every set A of atoms, $forget(D, A) \equiv forget(D', A)$.*

Proof. Let D and D' be two defeasible theories that are equivalent. Then, for each tagged literal L whose literal is not in A,

$forget(D, A) \vdash L$
iff $D \vdash L$
iff $D' \vdash L$
iff $forget(D', A) \vdash L$.

Thus, $forget(D, A)$ and $forget(D', A)$ are also equivalent. □

In addition, the result is independent of whether atoms are forgotten successively (in some order), or altogether.

Proposition 6. *Let D be a defeasible theory and let $A = \{a_1, \ldots, a_n, a_{n+1}\}$ be a set of atoms. Then $forget(D, A) \equiv forget(forget(D, \{a_1, \ldots, a_n\}), a_{n+1})$.*

Proof. (Sketch) This can be shown by a simple induction on the size n of A. Let $A_n = \{a_1, \ldots, a_n\}$. We need only to observe that

$$forget(D, A) \equiv forget(D, A_n) \equiv forget(forget(D, A_n), a_{n+1}).$$ □

Note that Algorithm 2 is as little disruptive to the original theory as possible. A close inspection reveals that its only operations are to:

- remove all rules containing atoms in A in their heads;
- remove provable atoms from A from rule bodies;
- remove rules with an atom from A in their body, in cases where this atom is not provable;
- turn rules into defeaters in certain cases caused by cyclicity (see Example 2);
- remove priority pairs where one of the rules involved was deleted.

All these changes are necessary and as little obstructive as possible in the attempt to compile the derivability of literals from A into the knowledge base. The following formal property seeks to partially capture this intuitive notion of "structure preservation". It makes use of direct dependency. Formally, an atom a is *directly dependent* on an atom b iff either a and b are identical, or there is a rule r with head a or $\neg a$, such that b or $\neg b$ appears in the body of r. An atom a is *dependent* on an atom b iff there is a sequence $a_1, a_2, \ldots, a_t$ of atoms such that $t > 0$ and a_i is directly dependent on a_{i+1} for $i = 1, \ldots, t-1$.

Proposition 7. *$forget(D, A)$ differs from D only on rules whose head is directly dependent on an atom in A.*

4.2 Complexity

Like Algorithm 1, also Algorithm 2 can be run efficiently, as all steps 1-3 are feasible in linear time (for steps 2 and 3, suitable standard data structures are used).

Proposition 8. *Algorithm 2 computes $forget(D, A)$ in linear time.*

Finally, the size of the outcome remains linear, ensuring (in conjunction with Proposition 5) that the result of iterated forgetting is of linear size. The latter does not necessarily follow from the linear time complexity, as there might be an exponential increase in the number of iterations.

Corollary 1 (of Propositions 4, 6 and 8). *For every defeasible theory D and sets of atoms $A_1, \ldots, A_m$, The size of $D' = forget(forget(\cdots forget(D, A_1), \cdots), A_m)$ computed by Algorithm 2 is linear in the size of D, and D' is computed in time linear in the size of D and $A_1, \ldots, A_m$.*

Proof. (Sketch) By Proposition 4, without loss of generality the sets A_i are pairwise disjoint and nonempty. For each A_i, we can write the forgetting of $A_i = \{a_{i,1}, \ldots, a_{i,n_i}\}$ as iterated forgetting of each $a_{i,j}$, $1 \leq j \leq n_i$. Thus, we obtain that

$$D' = forget(forget(\cdots forget(D, a_{1,1}), \cdots), a_{m,n_m}).$$

By Proposition 6 again, $D' = forget(D, A)$ where $A = \bigcup_{i=1}^{m} A_i$. The result follows then from Proposition 8. □

4.3 Modularity

Let us, given any two defeasible theories $D_1 = (F_1, R_1, <_1)$ and $D_2 = (F_2, R_2, <_2)$, define their union $D_1 \cup D_2$ to be the defeasible theory $(F_1 \cup F_2, R_1 \cup R_2, <_1 \cup <_2)$.

Given a theory D and a set A of atoms, it is often the case that D is large but only a small fraction of D is relevant to A. That is, D can be split into two parts D_1 and D_2 where D_2 is irrelevant to A. In this case, to forget about A in D, one would expect to perform forgetting only on D_1. Unfortunately, this is not true in general. Consider the following theory $D = (F, R, >)$, where

$$\begin{aligned} R = \{\ & r_1 : a' \Rightarrow a \\ & r_2 : a \Rightarrow b \\ & r_3 : b \Rightarrow c \\ & r_4 : \ \rightarrow a' \ \} \end{aligned}$$

and F and $>$ are empty. Then $D = D_1 \cup D_2$ where $D_1 = (\emptyset, \{r_1, r_2, r_3\}, \emptyset)$ and $D_2 = (\emptyset, \{r_4\}, \emptyset)$. Then for $A = \{b\}$, $forget(D, A)$ is not equivalent to $forget(D_1, A) \cup D_2$: We note that $+\partial b$ is provable from D and the body of r_3 is b. So, if $A = \{b\}$ is forgotten from D, $+\partial c$ remains provable from $forget(D, A)$.

On the other hand, $+\partial b$ is not provable in D and thus when forgetting about $A = \{b\}$, the rule r_3 is deleted. As a result, even when $r_5 : a'$ is added, $+\partial c$ is still not provable from $forget(D_1, A) \cup D_2$.

We see that the solution provided by Algorithm 2 is not modular. But at least the following restricted form of splitting for forgetting holds, which can be seen from the correctness of Algorithm 2.

Proposition 9. *Let A be a set of atoms and let $D = D_1 \cup D_2$ be a defeasible theory without defeaters such that (1) no atom in D_1 depends on an atom in D_2, and (2) no atoms in A appear in D_2. Then $forget(D, A)$ is equivalent to $forget(D_1, A) \cup D_2$.*

Proof. (Sketch) Without loss of generality, we assume that $A = \{p\}$. Given a tagged literal L whose literal is q instead of p, we consider two possible cases.
Case 1. q appears only in D_1: Then

$D \vdash L$
iff $D_1 \vdash L$
iff $forget(D_1, A) \vdash L$
iff $forget(D_1, A) \cup D_2 \vdash L$.

Case 2. q appears in D_2: It can also be shown that $D \vdash L$ iff $forget(D_1, A) \cup D_2 \vdash L$. The basic idea is that, given a tagged literal L' whose literal is in D_1, we can always replace the derivation of a tagged literal L' in D_1 with a derivation of L' in $forget(D_1, A)$ or vice versa. □

As a final point, we wish to place this partial modularity result into context. There is an intuitive trade-off between dependency/derivability of literals and modularity. Intuitively, if we want to achieve unrestricted modularity, then information about the connection (derivability) between pairs of literals must be retained in the theory; but to record such information, without additional symbols, requires quadratic space. Let us consider as an example the following set of rules:

$a_i \Rightarrow b$ with $i \in \{1, \ldots, n\}$,
$b \Rightarrow c_j$ with $j \in \{1, \ldots, n\}$.

Here each c_j depends on all a_i's, and we have quadratically many "implied rules" $a_i \Rightarrow c_j$ to incorporate when forgetting about b, which cannot be done in linear time. And in lack of any further information to discriminate among different rules, it would also be unclear which of these implied rules to add, so adding none would make sense; this is exactly what Algorithm 2 does. So we would argue that Algorithm 2 is a reasonable core for linear-time forgetting algorithms in the context of defeasible logic. Adding all, or some (according to some external criteria) "implied rules" would give refinements of this basic algorithm.

5 Conclusion

In this paper we studied the problem of forgetting in the context of defeasible logic. We provided two algorithms for computing the result of forgetting a set of literals, a naive algorithm and an advanced one (Algorithm 2) whose output has several desired properties. Salient features of the solutions provided are linear time complexity, and the linear size of iterated forgetting. In addition, Algorithm 2 preserves a lot of structure of the original defeasible theory; to measure how much in formal terms (e.g., using similarly via tree edit distance) remains to be considered.

We intend to continue work on forgetting. One task is to determine a modular approach to forgetting: Algorithm 2 is not modular, as it has to recompute all conclusions related to p and $\neg p$ every time the theory is changed. An interesting question would be to define a pure transformation approach, in the spirit of step 2 of Algorithm 2, without the need for step 1. Many research questions follow on from this, including a thorough analysis of time complexity, space of the outcome, and tradeoff between modularity and dependency and derivability, continuing the discussion at the end of Section 4.

Another idea for future work is to apply forgetting to multi-context theories, and in particular to the work of [5] which is essentially a contextual defeasible logic. Finally, we might apply forgetting in the context of the Semantic Web, in particular to defeasible rule systems over RDF [1, 4], as a means of information integration and knowledge dynamics, and to deal with inconsistencies in this context.

Acknowledgments

The authors would like to thank the three anonymous referees for their helpful comments. This work was partially supported by the Australia Research Council (ARC) Discovery Projects DP1093652 and DP110101042, by an Olga Taussky Fellowship of the Wolfgang Pauli Institute (WPI) Vienna, by the Austrian Science Fund (FWF) project P20841 and by the Vienna Science and Technology Fund (WWTF) grant ICT 08-020.

References

1. G. Antoniou and A. Bikakis. DR-Prolog: A system for defeasible reasoning with rules and ontologies on the semantic web. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):233–245, 2007.
2. G. Antoniou, D. Billington, G. Governatori, and M. Maher. On the modeling and analysis of regulations. In *Proc. Australian Conference Information Systems*, pp. 20–29, 1999.
3. G. Antoniou, D. Billington, G. Governatori, and M. Maher. Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2(2):255–287, 2001.
4. N. Bassiliades, G. Antoniou, and I. Vlahavas. A defeasible logic reasoner for the semantic web. *International Journal of Semantic Web Information Systems*, 2(1):1–41, 2006.
5. A. Bikakis and G. Antoniou. Defeasible contextual reasoning with arguments in ambient intelligence. *IEEE Transactions on Knowledge and Data Engineering*, 22(11):1492–1506, 2010.
6. D. Billington, G. Antoniou, G. Governatori, and M. Maher. An inclusion theorem for defeasible logics. *ACM Transactions on Computational Logic*, 12(1):6, 2010.
7. T. Eiter and K. Wang. Semantic forgetting in answer set programming. *Artificial Intelligence*, 14:1644–1672, 2008.
8. G. Governatori. Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14(2-3):181–216, 2005.
9. G. Governatori, M. Dumas, A. ter Hofstede, and P. Oaks. A formal approach to legal negotiation. In *International Conference on Artificial Intelligence and Law*, pp. 168–177, 2001.
10. G. Governatori, M. Maher, G. Antoniou, and D. Billington. Argumentation semantics for defeasible logic. *Journal of Logic Computation*, 14(5):675–702, 2004.
11. G. Governatori, V. Padmanabhan, and A. Sattar. A defeasible logic of policy-based intention. In *Proc. 15th Australian Joint Conference on Artificial Intelligence*, page 723, 2002.

12. G. Governatori and A. Rotolo. Defeasible logic: Agency, intention and obligation. In *Proc7th International Workshop on Deontic Logic in Computer Science*, pp. 114–128, 2004.
13. G. Governatori, A. Rotolo, and G. Sartor. Temporalised normative positions in defeasible logic. In *Proc. 10th International Conference on Artificial Intelligence and Law*, pp. 25–34, 2005.
14. A. Herzig and J. Mengin. Uniform interpolation by resolution in modal logic. In *Proc. 11th European Conference on Logics in Artificial Intelligence (JELIA-08)*, pp. 219–231, 2008.
15. B. Konev, D. Walther, F. Wolter. Forgetting and uniform interpolation in large-scale description logic terminologies. In *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI-09)*, pp. 830–835, 2009.
16. R. Kontchakov, F. Wolter, and M. Zakharyaschev. Can you tell the difference between DL-Lite ontologies? In *Proc. 1th International Conference on Principles of Knowledge Representation and Reasoning (KR-08)*, pp. 285–295, 2008.
17. J. Lang, P. Liberatore, and P. Marquis. Propositional independence: Formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18:391–443, 2003.
18. J. Lang and P. Marquis. Resolving inconsistencies by variable forgetting. In *Proc. 8th International Conference on Principles of Knowledge Representation and Reasoning (KR-02)*, pp. 239–250, 2002.
19. F. Lin and R. Reiter. Forget it. In *Proc. AAAI Fall Symposium on Relevance*, pp. 154–159. New Orleans (LA), 1994.
20. M. Maher. Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming*, 1(6):691–711, 2001.
21. M. Maher. A model-theoretic semantics for defeasible logic. In *Proc. Workshop on Paraconsistent Computational Logic*, pp. 67–80, 2002.
22. M. Maher, G. Antoniou, and D. Billington. A study of provability in defeasible logic. In *Proc. 11th Australian Joint Conference on Artificial Intelligence*, pp. 215–226, 1998.
23. D. Nute. *Handbook of logic in Artificial Intelligence and Logic Programming*, volume 3, chapter Defeasible logic. Oxford University Press, 1994.
24. T. Skylogiannis, G. Antoniou, N. Bassiliades, and G. Governatori. Dr-negotiate - a system for automated agent negotiation with defeasible logic-based strategies. In *Proc. 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pp. 44–49, 2005
25. H. van Ditmarsch, A. Herzig, J. Lang, and P. Marquis. Introspective forgetting. In *Proc. 21st Australasian Joint Conference on Artificial Intelligence*, pp. 18–29, 2008.
26. K. Wang, A. Sattar, and K. Su. A theory of forgetting in logic programming. In *Proc. 20th National Conference on Artificial Intelligence (AAAI-05)*, pp. 682–687, 2005.
27. K. Wang, Z. Wang, R. Topor, J. Z. Pan, and G. Antoniou. Eliminating concepts and roles from ontologies in description logic. In *Proc. 8th International Semantic Web Conference (ISWC-09)*, pp. 666–681, 2009.
28. Z. Wang, K. Wang, R. Topor, and J. Z. Pan. Forgetting concepts in DL-Lite. In *Proc. 5th European Semantic Web Conference (ESWC-08)*, pp. 245–257, 2008.
29. Y. Zhang, N. Foo, and K. Wang. Solving logic program conflicts through strong and weak forgettings. In *Proc. 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pp. 627–632, 2005.
30. Y. Zhang and Y. Zhou. Knowledge forgetting: Properties and applications. *Artificial Intelligence*, 173(16-17):1525–1537, 2009.
31. Y. Zhou and Y. Zhang. Bounded forgetting. In *Proc. 25th AAAI Conference on Artificial Intelligence*, pp. 280–285, 2011.