TPLP **20** (4): 552–586, 2020. © Cambridge University Press 2020
doi:10.1017/S1471068420000010 First published online 24 February 2020

*Rethinking Defeasible Reasoning: A Scalable Approach**

MICHAEL J. MAHER
Reasoning Research Institute, Australia
(*e-mail:* michael.maher@reasoning.org.au)

ILIAS TACHMAZIDIS, GRIGORIS ANTONIOU and STEPHEN WADE
University of Huddersfield, UK
(*e-mails:* i.tachmazidis@hud.ac.uk, g.antoniou@hud.ac.uk, s.j.wade@hud.ac.uk)

LONG CHENG
Dublin City University, Ireland
(*e-mail:* long.cheng@dcu.ie)

submitted 19 May 2019; accepted 6 January 2019

Abstract

Recent technological advances have led to unprecedented amounts of generated data that originate from the Web, sensor networks, and social media. Analytics in terms of defeasible reasoning – for example, for decision making – could provide richer knowledge of the underlying domain. Traditionally, defeasible reasoning has focused on complex knowledge structures over small to medium amounts of data, but recent research efforts have attempted to parallelize the reasoning process over theories with large numbers of facts. Such work has shown that traditional defeasible logics come with overheads that limit scalability. In this work, we design a new logic for defeasible reasoning, thus ensuring scalability by design. We establish several properties of the logic, including its relation to existing defeasible logics. Our experimental results indicate that our approach is indeed scalable and defeasible reasoning can be applied to billions of facts.

KEYWORDS: defeasible reasoning, parallel peasoning, scalability

1 Introduction

Recent technological advances have led to unprecedented amounts of generated data that originate from the Web, sensor networks, and social media. Once these data are stored, the challenge becomes developing solutions for efficient processing of the vast amounts of data in order to extract additional value. Analytics in terms of reasoning – for example, for decision making – should be performed using rule sets that would allow the aggregation, visualization, understanding, and interpretation of given datasets and their interconnections. Specifically, one should use rules able to encode inference semantics, as well as commonsense and practical conclusions in order to infer new and useful knowledge based on the data.

* We thank the referees for their comments, which helped improve this paper.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Various monotonic logics have been implemented with this large scale of data in mind. Work includes Datalog (Leone *et al*. 2019; Condie *et al*. 2018; Martinez-Angeles *et al*. 2013), $\mathcal{EL}^+$ (Mutharaju *et al*. 2015), OWL Horst (Kim and Park 2015; Urbani *et al*. 2012), RDFS (Heino and Pan 2012; Goodman *et al*. 2011; Oren *et al*. 2009), and Fuzzy logics (Zhou *et al*. 2013; Liu *et al*. 2011), scaling reasoning up to billions of facts. For a comprehensive overview of existing approaches on large-scale reasoning, readers are referred to Antoniou *et al*. (2018).

Nevertheless, it should be pointed out that available data often come from heterogeneous sources that are not necessarily controlled by the data engineer, and therefore may contain imperfect, incomplete, or conflicting information. Other reasons for imperfect data may be faults in sensors or the communication infrastructure. It is evident that monotonic reasoning is not suited for such data processing, which subsequently led to the study of large-scale nonmonotonic reasoning. In particular, logic programs under the well-founded semantics (Gelder *et al*. 1991) have been addressed (Tachmazidis and Antoniou 2013; Tachmazidis *et al*. 2014). However, this semantics can only indirectly address conflicting information.

Defeasible reasoning provides facilities to directly address conflicting information. While computationally simple, it has found numerous applications in the modeling of legal reasoning (Prakken 1997; Governatori and Maher 2017), regulations (Antoniou *et al*. 1999; Islam and Governatori 2018), business rules (Grosof *et al*. 1999), contracts (Governatori and Pham 2009), negotiation (Skylogiannis *et al*. 2007), and business process compliance management (Governatori *et al*. 2006; Hashmi *et al*. 2018).

However, its scalability is still in question. Propositional defeasible logics can be executed in linear time Maher (2001); (Billington *et al*. 2010) but that algorithm does not easily support parallelism, nor does it easily extend to first-order defeasible logics. Implementations of defeasible logic for big data have been limited to subsets of the full logic Tachmazidis *et al*. (2012a,b). Both approaches have been applied to billions of facts, but neither approach was able to capture the general case. A fundamental problem in existing defeasible logics identified in (Tachmazidis *et al*. 2012b) is that the notion of provable-failure-to-prove, which is central to these logics, requires the generation and retention of a prohibitive amount of negative derivation conclusions. This inhibits the scalability of implementations of such logics.

In this work, we propose a novel approach for defeasible reasoning over large data by defining a scalable defeasible logic. The new inference rules of the logic avoid reliance on provable-failure-to-prove by building solely on definitely and defeasibly provable conclusions. With this approach, the new logic provides scalability by design. The result is a reasoning process that is comparable in terms of scalability with existing methods for monotonic logics. In the context of this work, a scalable method allows large-scale inference computation by utilizing parallel and distributed settings over big data. Experimental results highlight the scalability properties of the proposed logic, while showing that our approach can scale up to 1 billion facts over a real-world case study.

The paper is structured as follows. Section 2 provides a brief outline of defeasible logic. Section 3 discusses an existing implementation of defeasible logic over stratified rule sets, to demonstrate the general process of inferring defeasible conclusions in a distributed setting and the problems that arise. The new logic is introduced in Section 4, while Section 5 establishes its theoretical properties. Section 6 describes the implementation

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

and Section 7 the experimental evaluation. We conclude in Section 8. Proofs are available in the appendices.

2 Defeasible logics

A defeasible theory D is a triple $(F, R, >)$ where F is a finite set of facts (literals), R a finite set of rules, and $>$ a superiority (or priority) relation (a binary acyclic relation) on R, specifying when one rule overrides another, given that both are applicable. Rules and facts may be labeled, to enable reference to them. The set of labels is denoted by $\Lambda(D)$.

A rule r consists (a) of its antecedent (or body) $A(r)$ which is a finite set of literals, (b) an arrow, and (c) its consequent (or head) $C(r)$ which is a literal. There are three types of rules: strict rules, defeasible rules, and defeaters represented by a respective arrow $\rightarrow$, $\Rightarrow$, and $\rightsquigarrow$. Strict rules are rules in the classical sense: whenever the premises are indisputable (e.g., facts) then so is the conclusion. Defeasible rules are rules that can be defeated by contrary evidence. Defeaters are rules that cannot be used to draw any conclusions; their only use is to prevent some conclusions.

A literal is a possibly negated predicate symbol applied to a sequence of variables and constants. We will require that any variable in the head of a rule also occurs in the body, and that every fact is variable-free, a property known as *range-restricted*[1]. Given a fixed finite set of constants, any rule is equivalent to a finite set of variable-free rules, and any defeasible theory is equivalent to a variable-free defeasible theory, for the purpose of semantical analysis. We refer to variable-free defeasible theories, etc as *propositional*, since there is only a syntactic difference between such theories and true propositional defeasible theories. Consequently, we will formulate definitions and semantical analysis in propositional terms. However, for computational analyses and implementation, we will also address defeasible theories that are not propositional.

Given a set R of rules, we denote the set of all strict rules in R by R_s, and the set of strict and defeasible rules in R by R_{sd}. $R[q]$ denotes the set of rules in R with consequent q. 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). A *conclusion* takes the forms $+d\,q$ or $-d\,q$, where q is a literal and d is a tag indicating which inference rules were used. Given a defeasible theory D, $+d\,q$ expresses that q can be proved via inference rule d from D, while $-d\,q$ expresses that it can be established that q cannot be proved from D.

Example 1

To demonstrate defeasible theories, we consider the representation of the Tweety problem as a defeasible theory. The defeasible theory D consists of the rules and facts

$$\begin{array}{rrcl} r_1: & bird(X) & \Rightarrow & fly(X) \\ r_2: & penguin(X) & \Rightarrow & \neg fly(X) \\ r_3: & penguin(X) & \rightarrow & bird(X), \\ e: & bird(eddie) & & \\ f: & penguin(tweety) & & \end{array}$$

and a priority relation $r_2 > r_1$.

[1] This is not a requirement of defeasible theories, it simply eases discussion and implementation. It is a common requirement in work on deductive elements of databases and is not very restrictive in practice.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Here r_1, r_2, r_3, e, f are labels and r_3 is (a reference to) a strict rule, while r_1 and r_2 are defeasible rules, and e and f are facts. Thus, $F = \{e, f\}$, $R_s = \{r_3\}$ and $R_{sd} = R = \{r_1, r_2, r_3\}$ and $>$ consists of the single tuple (r_2, r_1). The rules express that birds usually fly (r_1), penguins usually don't fly (r_2), and that all penguins are birds (r_3). In addition, the priority of r_2 over r_1 expresses that when something is both a bird and a penguin (i.e., when both rules can fire), it usually cannot fly (i.e., only r_2 may fire, it overrules r_1). Finally, we are given the facts that *eddie* is a bird and *tweety* is a penguin.

As an example of a defeasible logic, in Antoniou *et al.* (2001), a defeasible logic now called $DL(\partial)$ is defined with the following inference rules, phrased as conditions on proofs[2]

$+\Delta$) If $P(i+1) = +\Delta q$, then either
(1) $q \in F$; or
(2) $\exists r \in R_s[q]\ \forall a \in A(r), +\Delta a \in P[1..i]$.

$-\Delta$) If $P(i+1) = -\Delta q$, then
(1) $q \notin F$, and
(2) $\forall r \in R_s[q]\ \exists a \in A(r), -\Delta a \in P[1..i]$.

These two inference rules concern reasoning about definitive information, involving only strict rules and facts. They define conventional monotonic inference. The next rules refer to defeasible reasoning.

$+\partial$) If $P(i+1) = +\partial q$, then either
(1) $+\Delta q \in P[1..i]$; or
(2) The following three conditions all hold.
(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 $t > s$.

$-\partial$) If $P(i+1) = -\partial q$, then
(1) $-\Delta q \in P[1..i]$, and
(2) either
(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 not$(t > s)$.

$+\partial q$ is a *consequence* of a defeasible theory D if there is a proof containing $+\partial q$.

In the $+\partial$ inference rule, (1) ensures that any monotonic consequence is also a defeasible consequence. (2) allows the application of a rule (2.1) with head q, provided that monotonic inference provably cannot prove ${\sim}q$ (2.2) and every competing rule either provably fails to apply (2.3.1) or is overridden by an applicable rule for q (2.3.2). The $-\partial$ inference rule is the strong negation (Antoniou *et al.* 2000) of the $+\partial$ inference rule. It establishes when a literal is provably not provable in the logic.

To demonstrate these inference rules, we apply them to the Tweety defeasible theory in the previous example.

Example 2

We infer $+\Delta penguin(tweety)$ by application of (1) of the $+\Delta$ inference rule, and then $+\Delta bird(tweety)$ by application of (2) of that rule. We also infer $+\Delta bird(eddie)$. From

[2] Here, D is a defeasible theory $(F, R, >)$, q is a variable-free literal, P denotes a proof (a sequence of conclusions constructed by the inference rules), $P[1..i]$ denotes the first i elements of P, and $P(i)$ denotes the i^{th} element of P.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

the $-\Delta$ inference rule, we infer $-\Delta penguin(eddie)$, $-\Delta fly(eddie)$, $-\Delta\neg fly(eddie)$, $-\Delta fly(tweety)$, and $-\Delta\neg fly(tweety)$, establishing that these literals cannot be definitely established.

All $+\Delta$ conclusions can also be derived defeasibly, using (1) of the $+\partial$ inference rule. Also note that $-\partial penguin(eddie)$ is derived because $-\Delta penguin(eddie)$ and there is no (instance of) a rule with head $penguin(eddie)$, so (1) and (2.1) of the $-\partial$ inference rule are satisfied. Consequently, we can infer $-\partial\neg fly(eddie)$ because (1) and (2.1) of the $-\partial$ inference rule are satisfied by $-\Delta\neg fly(eddie)$ and $-\partial penguin(eddie)$. Finally, we can now infer $+\partial fly(eddie)$ by (2) of the $+\partial$ inference rule because r_1 and $+\partial bird(eddie)$ combine to satisfy (2.1), $-\Delta\neg fly(eddie)$ satisfies (2.2), and (2.3) is satisfied because the only (instance of a) rule for $fly(eddie)$, r_2, has $penguin(eddie)$ in its body, and we derived $-\partial penguin(eddie)$.

In contrast, we infer $+\partial\neg fly(tweety)$ because r_2 and $+\partial penguin(tweety)$ satisfy (2.1), $-\Delta fly(tweety)$ satisfies (2.2), and (2.3) is satisfied because the only rule $s = r_1$ with head $\neg fly(tweety)$ is overruled by $t = r_2$ in (2.3.2) using $+\partial penguin(tweety)$. Without the priority statement, we would not infer $+\partial\neg fly(tweety)$, and instead infer $-\partial\neg fly(tweety)$ (as well as $-\partial fly(tweety)$), thus being unable to come to any positive conclusion about the ability of *tweety* to fly.

A tag/inference rule d in a logic is *consistent* if, for every defeasible theory D in the logic and every proposition q, we do not have both consequences $+dq$ and $+d\neg q$ unless we also have consequences $+\Delta q$ and $+\Delta\neg q$. This property expresses that defeasible reasoning does not cause inconsistencies: any inconsistency in consequences is caused by inconsistency in the monotonic part of the defeasible theory. We say a logic is consistent if its main inference rule is consistent. $DL(\partial)$ is consistent, as are the other logics in Billington *et al.* (2010).

3 Parallel stratified defeasible reasoning

In order to facilitate the discussion in the following sections, we first need to discuss fundamental notions of parallel stratified defeasible reasoning as presented in Tachmazidis *et al.* (2012b).

A rule set is *stratified* if all of its predicates can be assigned a rank such that: (a) no predicate depends on one of equal or greater rank, and (b) no predicate is assigned a rank not equal to its complement. Note that a predicate that is found in the head of a rule *depends* on the predicates that are found in the body of the same rule.

Consider the following stratified rule set:

$$\begin{array}{lrcl} r_1: & r(X,Z), s(Z,Y) & \Rightarrow & q(X,Y) \\ r_2: & t(X,Z), u(Z,Y) & \Rightarrow & \neg q(X,Y) \\ & r_1 > r_2 & & \end{array}$$

where predicates $r(X,Z)$, $s(Z,Y)$, $t(X,Z)$, and $u(Z,Y)$ are assigned to rank 0, while both $q(X,Y)$ and $\neg q(X,Y)$ are assigned to rank 1.

Predicates that are assigned to rank 0 do not appear in the head of any rule, and thus, only a transformation of given facts into $+\Delta$ and $+\partial$ conclusions is required. Given the facts $r(a,b)$, $s(b,b)$, $t(a,e)$, $u(e,b)$, and $u(e,g)$, this transformation will create the

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

following conclusions (assuming a *key-value* storage, where the *key* stores the conclusion itself while the *value* stores the knowledge about the conclusion):

$$< r(a,b),\ (+\Delta,+\partial) > \qquad < s(b,b),\ (+\Delta,+\partial) >$$
$$< t(a,e),\ (+\Delta,+\partial) > \qquad < u(e,b),\ (+\Delta,+\partial) >$$
$$< u(e,g),\ (+\Delta,+\partial) >$$

For rank 1, defeasible reasoning needs to be performed in order to resolve the conflict between $q(X,Y)$ and $\neg q(X,Y)$. Due to the nature of defeasible reasoning, parallel reasoning is performed in two passes. The first pass computes applicable rules for $q(X,Y)$ and $\neg q(X,Y)$. Notice that unlike monotonic reasoning, in defeasible reasoning, applicable rules might not lead to new conclusions. Hence, the second pass performs the actual defeasible reasoning and computes for each literal whether it is definitely or defeasibly provable.

The following is based on a distributed system with two nodes. However, the same process is applicable to any parallel and distributed setting. Note that in order to perform parallel and distributed defeasible reasoning, each node requires a complete knowledge of the given rule set, thus enabling both parallel rule applications (first pass) and parallel defeasible reasoning (second pass). Consider the following distribution of the aforementioned conclusions (for the sake of readability, all knowledge is assumed to be stored in memory):

node 1	node 2
$< r(a,b),\ (+\Delta,+\partial) >$	$< s(b,b),\ (+\Delta,+\partial) >$
$< t(a,e),\ (+\Delta,+\partial) >$	$< u(e,b),\ (+\Delta,+\partial) >$
	$< u(e,g),\ (+\Delta,+\partial) >$

Considering the first pass, namely computing applicable rules, joins on common arguments for rule r_1 (resp. r_2) can only be performed if literals $r(a,b)$ and $s(b,b)$ (resp. $t(a,e)$, $u(e,b)$ and $u(e,g)$) are located in the same node, performing joins on argument b (resp. argument e). Thus, the existing knowledge needs to be shuffled as follows:

node 1	node 2
$< r(a,b),\ (+\Delta,+\partial) >$	$< t(a,e),\ (+\Delta,+\partial) >$
$< s(b,b),\ (+\Delta,+\partial) >$	$< u(e,b),\ (+\Delta,+\partial) >$
	$< u(e,g),\ (+\Delta,+\partial) >$

Note that *key-value* shuffling for the first pass can be performed according to the hash value of the join argument (argument Z in rules r_1 and r_2), namely after applying a hash function, argument b is assigned to node 1 while argument e is assigned to node 2. In this way, joins can be performed locally in each node and in parallel since each node works independently. Such computation will lead to the following knowledge base:

node 1	node 2
$< r(a,b),\ (+\Delta,+\partial) >$	$< t(a,e),\ (+\Delta,+\partial) >$
$< s(b,b),\ (+\Delta,+\partial) >$	$< u(e,b),\ (+\Delta,+\partial) >$
$< q(a,b),(+\partial,r_1) >$	$< u(e,g),\ (+\Delta,+\partial) >$
	$< q(a,b),(\neg,+\partial,r_2) >$
	$< q(a,g),(\neg,+\partial,r_2) >$

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

where $< q(a,b), (+\partial, r_1) >$ means that $q(a,b)$ is supported by rule r_1, $< q(a,b), (\neg, +\partial, r_2) >$ means that $\neg q(a,b)$ is supported by rule r_2, and $< q(a,g), (\neg, +\partial, r_2) >$ means that $\neg q(a,g)$ is supported by rule r_2.

At this point, neither $q(a,b)$ nor $\neg q(a,b)$ can be concluded since the required knowledge for defeasible reasoning is scattered among different nodes. Thus, all knowledge for $q(a,b)$ and $\neg q(a,b)$ must be located in a single node. Hence, the second pass, namely defeasible reasoning, groups all relevant data for each potential conclusion in a single node, with different nodes performing reasoning (in parallel) over different conclusions. Thus, the knowledge will be shuffled as follows:

node 1	node 2
$< r(a,b),\ (+\Delta, +\partial) >$	$< t(a,e),\ (+\Delta, +\partial) >$
$< s(b,b),\ (+\Delta, +\partial) >$	$< u(e,b),\ (+\Delta, +\partial) >$
$< q(a,b), (+\partial, r_1) >$	$< u(e,g),\ (+\Delta, +\partial) >$
$< q(a,b), (\neg, +\partial, r_2) >$	$< q(a,g), (\neg, +\partial, r_2) >$

This *key-value* shuffling for the second pass can be performed according to the hash value of the *key*, namely after applying a hash function, literal $q(a,b)$ is assigned to node 1 while literal $q(a,g)$ is assigned to node 2. Note that during the second pass, only knowledge for literals $q(a,b)$, $\neg q(a,b)$, and $\neg q(a,g)$ is relevant (while literals $r(a,b)$, $s(b,b)$, $t(a,e)$, $u(e,b)$, and $u(e,g)$ are ignored). Notice that conclusions $q(a,b)$ and $\neg q(a,g)$ are computed in parallel by node 1 and node 2, respectively. Finally, after performing defeasible reasoning, the knowledge about applicable rules is replaced by the final conclusions. Thus, the final knowledge base will contain the following conclusions:

node 1	node 2
$< r(a,b),\ (+\Delta, +\partial) >$	$< t(a,e),\ (+\Delta, +\partial) >$
$< s(b,b),\ (+\Delta, +\partial) >$	$< u(e,b),\ (+\Delta, +\partial) >$
$< q(a,b), (+\partial) >$	$< u(e,g),\ (+\Delta, +\partial) >$
	$< \neg q(a,g), (+\partial) >$

For a more elaborate description of parallel stratified defeasible reasoning, readers are referred to Tachmazidis (2015).

Note that in Tachmazidis *et al.* (2012b), only positive conclusions are computed in order to ensure scalability. In theory, provable-failure-to-prove (e.g., $-\Delta$ and $-\partial$ inference rules) could be computed by first calculating applicable rules and then applying conflict resolution. However, such computation does not lead to scalable solutions. Consider the following strict rules:

$$\begin{aligned} r^*: &\quad p(X,Z),\ q(Z,Y) &\rightarrow &\quad p(X,Y) \\ r': &\quad q(X,Z), p(Z,V), t(V,Y) &\rightarrow &\quad q(X,Y) \end{aligned}$$

In order to establish that $p(a,b)$ is not definitely provable ($-\Delta$) every possible instantiated rule needs to be checked, namely for every value of Z either $p(a,Z)$ or $q(Z,b)$ should be established as $-\Delta$. For the aforementioned rule (r^*), if there are N constants in the given dataset, N instantiated rules need to be checked for each $-\Delta$ conclusion.

In general, for N constants in the given dataset and k variables in a given rule that do not appear in the head of the rule (e.g., the variable Z in the aforementioned rule r^*), every $-\Delta$ conclusion (say $-\Delta p(X,Y)$) will require N^k instantiated rules to be checked.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

By checking every possible instantiated rule, a significant overhead is introduced that can become prohibitive even for relatively small datasets (e.g., if $N = 10^5$ and $k = 3$, then each conclusion would require the computation of 10^{15} rules).

Once, all relevant rules are computed, all available information for each literal (such as $p(a, b)$) must be processed by a single node (containing all relevant information for the literal to be proved). However, this leads to memory and load balancing problems. For example, if conclusion $-\Delta p(c, b)$ depends on 10^5 instantiated rules (where $N = 10^5$ and $k = 1$, for rule r^*), while conclusion $-\Delta q(c, b)$ depends on 10^{10} instantiated rules (where $N = 10^5$ and $k = 2$, for rule r'), then there is a clear difference in the amount of information that needs to be processed by each node (a node computing a $-\Delta p(c, b)$ conclusion is expected to terminate significantly faster than a node computing a $-\Delta q(c, b)$ conclusion).

This problem motivates the definition of a new logic.

4 A scalable defeasible logic

The defeasible logic $DL(\partial_{||})$ involves three tags: Δ, which we have already seen; λ, an auxiliary tag; and $\partial_{||}$, which is the main notion of defeasible proof in this logic.

For a defeasible theory D, we define P_Δ to be the set of consequences in the largest proof satisfying the proof condition $+\Delta$, and call this the Δ *closure*. It contains all $+\Delta$ consequences of D.

Once P_Δ is computed, we can apply the $+\lambda$ inference rule. $+\lambda q$ is intended to mean that q is potentially defeasibly provable in D. The $+\lambda$ inference rule is as follows:

$+\lambda$: We may append $P(i+1) = +\lambda q$ if either
(1) $+\Delta q \in P_\Delta$ or
(2) (2.1) $\exists r \in R_{sd}[q]\ \forall \alpha \in A(r) : +\lambda\alpha \in P(1..i)$ and
(2.2) $+\Delta{\sim}q \notin P_\Delta$

Using this inference rule, and given P_Δ, we can compute the λ closure P_λ, which contains all $+\lambda$ consequences of D.

$+\partial_{||}q$ is intended to mean that q is defeasibly provable in D. Once P_Δ and P_λ are computed, we can apply the $+\partial_{||}$ inference rule.

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

The $\partial_{||}$ closure $P_{\partial_{||}}$ contains all $\partial_{||}$ consequences of D.

Notice that the structure of the inference rule for $\partial_{||}$ is the same as the structure of the inference rule for ∂. However, there are important differences to note:

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

- The inference rule for $\partial_{||}$ uses the closures P_Δ and P_λ in addition to the single proof P to which it is applied. These closures are precomputed. In contrast, in $DL(\partial)$, the proof P incorporates both Δ and ∂ conclusions.
- At (2.3.1), the $\partial_{||}$ inference rule refers to λ rather than ∂; in terms of which conclusions are drawn, this is the most significant variation from ∂.
- Furthermore, at (2.2) and (2.3.1), the $\partial_{||}$ inference rule does not use negative tags, such as $-\Delta$, which represent provable-failure-to-prove. Instead, $\partial_{||}$ uses $\notin P$, which represents failure to prove at the meta level, rather than from within the logic. This use of $\notin P$ is only possible because it refers to closures that have already been computed.

Since the proof rules of our logic do not require $-\lambda$ or $-\partial_{||}$ conclusions, we do not present the inference rules for $-\lambda$ and $-\partial_{||}$ here. They are in Appendix C.

It is straightforward to see that λ is not consistent. Nevertheless, $DL(\partial_{||})$ is consistent. The proof is in Appendix A.

Proposition 3
The inference rule $+\partial_{||}$ is consistent.

Inference rules ∂ and $\partial_{||}$ employ the notion of "team defeat", where it does not matter which rule overrides an opposing rule, as long as all opposing rules are overridden. This is expressed in (2.3.2). We can also have a version of $\partial_{||}$ with "individual defeat", where all opposing rules must be overridden by the same rule, which we denote by $\partial^*_{||}$. The inference rule for $+\partial^*_{||}$ replaces (2.3.2) in $\partial_{||}$ by $r > s$. It, too, is consistent.

To demonstrate the use of $DL(\partial_{||})$, we provide a simple example.

Example 4
Consider the following defeasible theory describing reachability in a directed graph where some edges may be broken.

$$\begin{array}{lrcl} r: & reachable(X), link(X,Y) & \rightarrow & reachable(Y) \\ s: & edge(X,Y) & \Rightarrow & link(X,Y) \\ t: & broken(X,Y) & \Rightarrow & \neg link(X,Y) \end{array}$$

$$\begin{array}{llll} reachable(a). & & & \\ edge(a,b). & edge(b,c). & edge(b,e). & edge(c,a). \\ edge(c,d). & edge(d,e). & edge(e,d). & edge(f,e). \\ broken(c,d). & broken(b,e). & & \end{array}$$

with $t > s$.

In this defeasible theory, r is a strict rule, s and t are defeasible rules with t overriding s when both are applicable, and there are facts defining the predicates *edge* and *broken*, as well as a fact for *reachable* identifying the starting point for the reachability calculation.

All the facts are known definitely, so they appear in P_Δ; there are no other facts in P_Δ because there is no definite information about *link*, and so the only strict rule cannot fire. In addition to all the facts, P_λ contains $link(X,Y)$ for each $edge(X,Y)$ fact, and $\neg link(X,Y)$ for each $broken(X,Y)$ fact. P_λ also contains $reachable(X)$ for every X that is reachable from a, ignoring the information about broken edges.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

$P_{\partial_{||}}$ contains all the facts, and $link(X, Y)$ for each unbroken edge and $\neg link(X, Y)$ for each broken edge. The superiority relation $t > s$ ensures that $+\partial_{||}\neg link(X, Y)$ appears and $+\partial_{||} link(X, Y)$ does not appear, for each broken edge. $P_{\partial_{||}}$ also contains $reachable(X)$ for every X that is reachable from a, via only unbroken edges.

If we compare $DL(\partial_{||})$ with $DL(\partial)$ on this defeasible theory, we find that they agree on the defeasible conclusions. Similarly, on the Tweety theory (Examples 1 and 2), the two logics agree on the defeasible conclusions.

The new inference rules provide a scalability advantage when compared to existing inference rules like ∂. As pointed out in Tachmazidis *et al.* (2012b), provable-failure-to-prove (e.g., $-\Delta$ and $-\partial$ inference rules) inhibits the scalability of existing defeasible logics. We saw a little of this in Example 2 where many negative conclusions were needed to derive positive conclusions, but the greater issue arises when rules have variables local to the body of the rule (as discussed in Section 3).

On the other hand, as illustrated in Section 6, the new inference rules that are proposed in this work result in a defeasible logic that is comparable in terms of scalability to existing monotonic logics, and thus able to benefit from available optimizations (readers are referred to (Antoniou *et al.* 2018) for a comprehensive overview of existing large-scale reasoning methods).

5 Properties of the logic

In this section, we address properties of $DL(\partial_{||})$: the computational complexity of the inference problem, the relative expressiveness of the logic, and the relative inference strength of the logic compared to existing defeasible logics.

5.1 Computational complexity

We first formalize the inference problem for defeasible logics.

The Inference Problem for a Defeasible Logic
Instance
A defeasible logic L, a defeasible theory T, a tag/inference rule d, and a literal q.
Question
Is $+dq$ derivable from T using the inference rules of L?

The computational complexity of inference reflects the difficulty of a scalable implementation. We show that $DL(\partial_{||})$ has linear complexity for propositional defeasible theories, but exponential for arbitrary defeasible theories. We have three inference rules to consider. As a result of the structure of the inference rules, it is straightforward to compute the consequences of Δ, and then λ, efficiently.

The inference problem for ∂ in propositional defeasible logic has linear complexity Maher (2001), and we use the same techniques to show that the inference problem for $\partial_{||}$ also has linear complexity. The proof is available in Appendix B.

Theorem 5
The set of all consequences of a propositional defeasible theory can be computed in time linear in the size of the defeasible theory. Consequently, the inference problem for propositional $DL(\partial_{||})$ can be solved in linear time.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

However, when variables are permitted in rules, the inference problem is EXPTIME-complete.

Corollary 6
The set of all consequences of a defeasible theory can be computed in time exponential in the size of the defeasible theory. Furthermore, the inference problem for defeasible theories is EXPTIME-complete.

From a scalability point of view, the potential for parallelism is important. Unfortunately, the inference problem for $DL(\partial_{||})$ is not parallelizible in a theoretical sense, even for propositional defeasible theories. Inference of $\partial_{||}$ consequences of propositional defeasible theories is P-complete, which is generally regarded as a sign that the problem is not parallelizible (i.e., not computable in poly-log time with polynomially many processors), unless all polynomial-time problems are parallelizible. Actually, inference of $+\Delta q$ is already P-complete, so all defeasible logics are not parallelizible in this sense. But the proof extends to practically every defeasible logic, even without strict rules.

Theorem 7
The inference problem for propositional defeasible logics is P-complete.

The proof is by reduction of the Horn satisfiability problem, which is P-complete (Cook and Nguyen 2010).

5.2 Relative expressiveness

Relative expressiveness of defeasible logics is defined in terms of the ability of one logic to simulate another (Maher 2012, 2013), even in the presence of some additions to a theory. The *addition* of two defeasible theories $(F_1, R_1, >_1) + (F_2, R_2, >_2)$ is $(F_1 \cup F_2, R_1 \cup R_2, >_1 \cup >_2)$. Let $\Sigma(D)$ denote the vocabulary of propositions and $\Lambda(D)$ denote the vocabulary of labels for D. Given a theory D and a possible simulating theory D', an addition A is required to be *modular*: $\Sigma(A) \cap \Sigma(D') \subseteq \Sigma(D)$, $\Lambda(D) \cap \Lambda(A) = \emptyset$, and $\Lambda(D') \cap \Lambda(A) = \emptyset$. This property ensures that the addition A cannot interfere with auxiliary propositions in D', nor can it interfere by overruling rules in D or D'.

A defeasible theory D_1 in logic L_1 is *simulated* by D_2 in L_2 with respect to a class C of additions if, for every modular addition A in C, $D_1 + A$ and $D_2 + A$ have the same consequences in $\Sigma(D_1 + A)$, modulo tags[3]. The classes C of additions considered in Maher (2013) are the empty theory, theories consisting only of facts, theories consisting only of rules, and arbitrary theories. These represent progressively stronger notions of simulation.

We say a logic L_1 can be simulated by a logic L_2 with respect to a class C, if every theory in L_1 can be simulated by some theory in L_2 with respect to additions from C. We say L_2 is *more (or equal) expressive than* L_1 w.r.t. C, if L_1 can be simulated by L_2 with respect to C. L_2 is *strictly more expressive than* L_1 w.r.t. C, if L_2 is more expressive than L_1 and L_1 is not more expressive than L_2, w.r.t. C.

To see the necessity of the restriction to modular additions, we present the following example.

[3] That is, D_1 in L_1 might produce $+d_1 q$ while D_2 in L_2 produces $+d_2 q$, due to different inference rules in the different logics, but the set of literals q that are derived is the same.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Example 8
Consider a conventional defeasible logic L_1 that is to be simulated by a similar logic L_2 that allows only two literals in the body of a rule. A theory D_1 consisting of a single rule

$$r: \quad b_1, b_2, b_3 \quad \Rightarrow \quad h$$

in L_1 might be represented as D_2:

$$\begin{array}{lccc} s: & b_1, b_2 & \Rightarrow & tmp \\ r: & tmp, b_3 & \Rightarrow & h \end{array}$$

in L_2. However, if an addition A were permitted to include the fact tmp (and b_3) then $D_1 + A$ cannot infer h, but $D_2 + A$ can infer h. Similarly, if A contains facts b_1, b_2, b_3 and

$$s': \quad \Rightarrow \quad \neg tmp$$

with $s' > s$ then $D_1 + A$ can infer h, but $D_2 + A$ cannot, because r' overrules r. In either case, L_2 does not simulate L_1, despite the close similarity of the two logics.

Thus if non-modular additions were permitted, only simulations that do not use auxiliary predicates and labels are possible, and the notion of relative expressiveness would be useless.

In this section, we investigate the relative expressiveness of $DL(\partial_{||})$. We first show that every theory in $DL(\partial)$ can be simulated in $DL(\partial_{||})$. On the other hand, $DL(\partial)$ cannot simulate $DL(\partial_{||})$. In fact, there is a single defeasible theory whose behavior in $DL(\partial_{||})$ cannot be simulated in $DL(\partial)$. Thus, $DL(\partial)$ is less expressive than $DL(\partial_{||})$.

Theorem 9
$DL(\partial)$ is strictly less expressive than $DL(\partial_{||})$ when there are no additions. More specifically,

- every defeasible theory in $DL(\partial)$ can be simulated by a defeasible theory in $DL(\partial_{||})$
- there is a defeasible theory D whose consequences in $DL(\partial_{||})$ cannot be expressed by any defeasible theory in $DL(\partial)$.

The argument for the second part is based on the following defeasible theory D:

$$\begin{array}{ccc} & \Rightarrow & p \\ \neg p & \rightarrow & \neg p \end{array}$$

with empty superiority relation.

$+\lambda p$ and $+\partial_{||} p$ are consequences of D, as is $-\Delta p$, while $-\Delta\neg p$ is not a consequence. However, there is no defeasible theory D' in which $+\partial p$ and $-\Delta p$ are consequences but $-\Delta\neg p$ is not. See the proof in Appendix C for details. The argument for the second part applies equally to logics $DL(\partial^*)$, $DL(\delta)$, $DL(\delta^*)$ (defined in Billington *et al.* (2010)) because their inference conditions all have the structure that is used in the proof. Essentially, this result arises from the fact that inference in $DL(\partial_{||})$ uses, for (2.2), the condition $+\Delta{\sim}q \notin P_\Delta$, whereas the usual defeasible logics use $-\Delta{\sim}q \in P_\Delta$. Thus, all these logics are not more expressive than $DL(\partial_{||})$, under any kind of addition.

This theorem suggests that defeasible theories in $DL(\partial)$ (and other logics) could be transformed into theories of $DL(\partial_{||})$ and then executed more scalably. However, the proof does not provide such a transformation. Furthermore, the overhead of such a transfor-

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

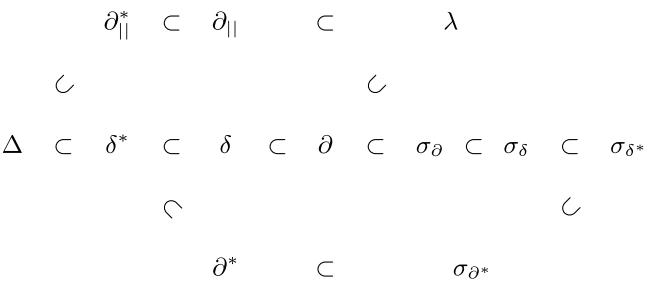


Fig. 1. Ordering of defeasible logic inference rules by relative inference strength. The only unrepresented relation is $\partial^* \subset \lambda$.

mation and the expansion in size of the theory could negate the scalability advantages. Nevertheless, this remains an avenue for future research.

Although $DL(\partial_{||})$ is able to simulate $DL(\partial)$ when there are no additions, it is unable to achieve a simulation when rules can be added.

Theorem 10
$DL(\partial_{||})$ is not more expressive than $DL(\partial)$ with respect to addition of rules.

The question of whether $DL(\partial_{||})$ can simulate $DL(\partial)$ w.r.t. addition of facts remains open.

5.3 Relative inference strength

We compare the inference strength of the new inference rules to the rules of existing defeasible logics. We write $d_1 \subseteq d_2$ if, for every defeasible theory T and literal q, if $+d_1 q$ is inferred from T then also $+d_2 q$ is inferred from T. This expresses that d_2 has greater inference strength than d_1, in the sense that any literal d_1 can infer can also be inferred by d_2. We can also view this inclusion as saying that d_1 is an under-approximation of d_2, or that d_2 is an over-approximation of d_1. We write $d_1 \subset d_2$ (i.e., the inclusion is *strict*) if $d_1 \subseteq d_2$ and there is a defeasible theory T and literal q such that $+d_2 q$ is inferred from T but $+d_1 q$ is not.

The relationship between the inference rules introduced in this paper and those of other defeasible logics is presented in Figure 1. (We follow the notation of Governatori and Maher (2017) for the inference rules σ_X.) The figure omits $\partial^* \subset \lambda$, which is difficult to include in such a two-dimensional representation. Examples show that all the containments are strict, and no containments are missing. The proof relies on results available in Appendix A.

Theorem 11
The containments illustrated in Figure 1 hold and are strict. In addition, $\partial^* \subset \lambda$ holds. There are no other missing containments in the figure.

In general, relative inference strength provides an indication of how brave/cautious a logic is in making inferences. The results show only that $DL(\partial_{||})$ is incomparable to existing logics. Nevertheless, the containment $\partial^*_{||} \subset \partial_{||}$ is noteworthy, since ∂^* and ∂ are incomparable.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

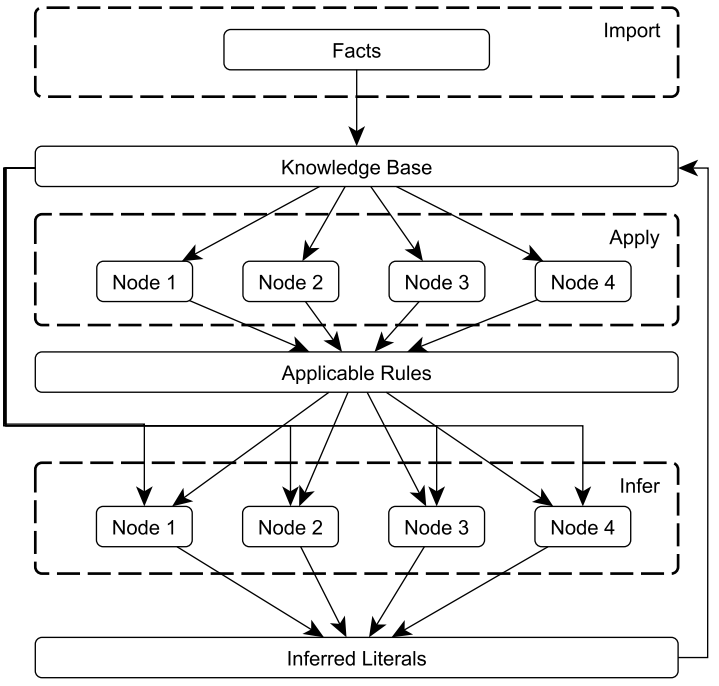


Fig. 2. Parallelizing the Import-Apply-Infer model.

6 Implementation

In this section, we present a generic approach for computing the new inference rules by building on previous work. Moreover, we outline the implementation for a real-world case study.

6.1 Import-apply-infer

An implementation of the new inference rules should first compute the Δ closure, subsequently the λ closure, and finally the $\partial_{||}$ closure. It is evident that the Δ closure computation is conventional rule application, starting from initial facts and repeatedly applying rules until no new conclusion is derived. Large-scale closure computation utilizing parallel and distributed settings over big data possesses unique challenges, with a wide range of challenges already addressed in the literature for various logics, including Datalog (Condie *et al.* 2018), $\mathcal{EL}^+$ (Mutharaju *et al.* 2015), OWL Horst (Kim and Park 2015), and RDFS (Heino and Pan 2012).

For λ and $\partial_{||}$ inference rules, we propose a three-step method called *import-apply-infer*, which can be parallelized as depicted in Figure 2. Essentially, the first step (*import*) reuses existing knowledge that could be considered as facts. Most parallel frameworks provide an efficient data transformation process, thus *import*'s scalability should be considered self-evident. The second step (*apply*) computes all currently applicable rules based on already proved literals. Following data partitioning, data are divided in chunks with each chunk assigned to a node (four nodes in Figure 2), thus finding matching literals within each node (e.g., *p(X,Z)* and *q(Z,Y)* match on argument Z for rule r^* in Section 3).

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Notice that *apply* follows the same rule application pattern as the first pass in Section 3. The third step (*infer*) resolves existing conflicts (e.g., "team defeat"), thus proving and adding new literals to the knowledge base. Notice that *infer* follows the same conflict resolution pattern as the second pass in Section 3.

Upon close inspection, λ and $\partial_{||}$ inference rules are variations (in terms of algorithmic computation) of inference rules presented in Tachmazidis *et al.* (2012b). Thus, the scalability findings of Tachmazidis *et al.* (2012b) in terms of a single computation of steps *import*, *apply*, and *infer* are applicable to this work as well. Considering closure computation, the *import* step is computed once at the beginning of the process, while steps *apply* and *infer* are computed repeatedly until no new conclusion is derived. Note that a generic implementation of a parallel reasoner is deferred to future work.

For the λ *closure*, clause (1) of the λ inference rule corresponds to the *import* step where literals in P_Δ are treated as given facts, (2.1) of the inference rule corresponds to the *apply* step as applicable rules are computed based on already proved λ predicates, and (2.2) of the inference rule corresponds to the *infer* step, where a literal q is proved only if $+\Delta{\sim}q \notin P_\Delta$ (with P_Δ already pre-computed). In terms of scalability, the *import* step requires importing existing knowledge, which is as scalable as the system's data storage, the *apply* step is as scalable as any rule application (including monotonic reasoning), and the *infer* step is basic data filtering where knowledge for each literal (both q and ${\sim}q$) is processed in parallel by different nodes in the cluster. Note that for any given rule set the knowledge for a specific literal is significantly smaller than main memory capacity, while the large number of literals ensures a high degree of parallelization and scalability.

For the $\partial_{||}$ *closure*, clause (1) of the $\partial_{||}$ inference rule corresponds to the *import* step where literals in P_Δ are treated as given facts, (2.1) of this inference rule corresponds to the *apply* step as applicable rules are computed based on already proved $\partial_{||}$ predicates, and clauses (2.2) and (2.3) correspond to the *infer* step, where a literal q is proved only if $+\Delta{\sim}q \notin P_\Delta$ (with P_Δ already pre-computed), and either $\exists\alpha \in A(s) : +\lambda\alpha \notin P_\lambda$ (where $s \in R[{\sim}q]$ and P_λ is already pre-computed) or q overrides ${\sim}q$ through "team defeat". In terms of scalability, $\partial_{||}$ *closure* follows a similar pattern as λ *closure* for all three steps. Note that although the *infer* step for $\partial_{||}$ *closure* requires more complex computations compared to the λ *closure*, the amount of processed data for each literal is still significantly smaller than main memory capacity, with the large number of literals ensuring a high degree of parallelization and scalability.

6.2 Apache spark

We have used Spark[4] in our implementations. The main reason is that the platform is very well suited to parallel data processing in distributed environments. It is elastic in terms of both storage (through the use of HDFS) and computation, which is in contrast with the conventional data systems where each node has to be carefully tuned to its specifications (Cheng *et al.* 2019). This makes Spark be able to greatly simplify the parallel programming of data applications. Namely, developers only need to focus on the design of high-level workflows and can ignore the underlying parallel executions. To handle the complex workflows in our implementation, we have applied Spark SQL (Armbrust *et al.* 2015) in our data processing. Spark SQL is a module in Apache Spark that inte-

[4] https://spark.apache.org/

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

grates relational processing with Spark's functional programming API. Here, we briefly introduce the core abstract of Spark SQL's API – the *DataFrame*.

A DataFrame in Spark SQL is a distributed collection of rows with the same schema. It can be seen as a table in a relational database, while its data are distributed over all computing nodes. A DataFrame can be manipulated and can also perform relational operations over data with existing Spark programs. Currently, DataFrames have supported all the common relational operators, such as projection, filter, join, and aggregations. Moreover, they also enable applications to run SQL queries programmatically and return the result as a DataFrame. Similar to the fundamental data structure of Spark (i.e., RDD), DataFrames are lazy. Namely, in the case that DataFrame object represents a logical plan to compute a dataset, no real parallel execution will occur until an output action such as *save* is called. This mechanism enables Spark SQL to use data structure information in order to perform rich optimization across all operations that were used to build the DataFrame (Armbrust *et al*. 2015), which is also the main reason why Spark SQL can provide a highly efficient execution solution for data applications.

6.3 FDA use case with spark

The experimental evaluation is based on a FAERS (FDA Adverse Event Reporting System – US Food and Drug Administration) case study, initially developed for RuleRS (Islam and Governatori 2018). More details on this use case are given in Section 7.1. We have implemented the logic using Spark specifically for this use case, and made our code, for the evaluated algorithms in this work, publicly available[5]. The approach reuses fundamental concepts of Tachmazidis *et al*. (2012b), but is more specific to the ruleset, implements the new logic, and uses Spark. Due to the nature of the FDA rule set, reasoning consists mainly of reporting: (a) obligation conclusions (applicable to all FDA cases) that follow from the given rule set, and (b) identified predicates for each FDA case. Note that other rule sets might require a more elaborate reasoning implementation.

The basic structure of the implementation is described in Algorithm 1. First, a set of obligation conclusions, that is a set of obligations that need to be concluded for all FDA cases (such as the obligation to report "Patient age", i.e., "obl_report_Patient_age_to_FDA"), is loaded in memory (line 1). Note that obligation conclusions are manually extracted from the FDA rule set. In order to allow each node in the cluster to perform reasoning independently by providing all required information on the given rule set, the set of obligation conclusions needs to be broadcast (line 2). Prior to applying SQL queries, facts are initially set to an empty DataFrame (line 3). Subsequently, each input file in the extracted FDA dataset (lines 4–7) is loaded into a corresponding DataFrame (line 5), and SQL queries are executed (line 6), using Spark SQL, in order to extract predicates (facts) that will be used for reasoning. Spark SQL ensures parallel evaluation of given SQL queries, while the developer needs only to define the queries using the Spark SQL API. Prior to performing reasoning, generated facts are grouped based on their *primaryid*, namely each FDA case is handled separately (line 8). Data grouping is performed in parallel by Spark. Note that lines 3–8 should be considered as the *import* step.

[5] https://github.com/longcheng11/dReasoning

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Algorithm 1 Spark implementation

```
1: FDA_obl_conclusions = Seq("Obligation conclusions")
2: broadcast(FDA_obl_conclusions)
3: factsDF = ∅
4: for each inputPath in FDA_Dataset do
5:     inputDF = load(inputPath)
6:     factsDF += inputDF.SQL_Queries_To_Facts()
7: end for
8: factsDF = factsDF.Group_By_Primary_ID()
9: conclusions = ∅
10: for each primaryID in factsDF do
11:     conclusions += reasoning(FDA_obl_conclusions, factsDF.getFacts(primaryID))
12: end for
13: conclusions.count()
```

Considering the reasoning process itself, conclusions are initially set to an empty DataFrame (line 9), while reasoning over each *primaryid* (in parallel) adds new conclusions (lines 10–12). Essentially, each *primaryid* is evaluated by a different node in the cluster, thus ensuring parallelism. Note that lines 9–12 should be considered as steps *apply* and *infer*. Finally, the number of conclusions is counted (line 13). Counting the number of final conclusions instead of materialising the output allows a better focus on the runtime performance of a given implementation. At the end of Algorithm 1, final conclusions are stored in memory and could be readily used for further processing if required.

7 Experimental results

In this section, we present the results of our experimental evaluation on a commodity cluster. We conduct a quantitative evaluation of our implementation.

7.1 Methodology

The evaluation of our approach is based on the RuleRS (Islam and Governatori 2018) FAERS (FDA Adverse Event Reporting System – US Food and Drug Administration) case study. The FAERS is a database that contains adverse event reports, medication error reports, and product quality complaints resulting in adverse events that were submitted to FDA. The database is designed to support the FDA's post-marketing safety surveillance program for drug and therapeutic biologic products[6].

Dataset. FAERS publishes quarterly data files[7], which include:

- *DEMO*: Demographic and administrative information.
- *DRUG*: Drug information from the case reports.
- *OUTC*: Patient outcome information from the reports.
- *REAC*: Reaction information from the reports.
- *RPSR*: Information on the source of the reports.

[6] https://www.fda.gov/drugs/surveillance/questions-and-answers-fdas-adverse-event-reporting-system-faers

[7] https://fis.fda.gov/extensions/FPD-QDE-FAERS/FPD-QDE-FAERS.html

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Table 1. *Input details*

Copies	Size (GB)	Distinct cases	Rows	Facts
1	3	5,285,699	43,791,158	96,925,980
3	9	15,857,097	131,373,474	290,777,940
6	18	31,714,194	262,746,948	581,555,880
12	36	63,428,388	525,493,896	1,163,111,760

Table 2. *Number of rows per file*

DEMO	DRUG	OUTC	REAC	RPSR	Whole
5,285,792	19,087,015	3,649,558	15,525,084	243,709	43,791,158

Table 1 describes the details of the used input. The original dataset consists of data published between the third quarter of 2014 and the second quarter of 2018 (a total of four calendar years), which corresponds to 3 GB of storage space, 5,285,699 distinct FDA cases (with each case indicated by a unique *primaryid*), 43,791,158 rows in the consolidated CSV files (for more details see Table 2), and 96,925,980 generated facts by the SQL queries (when all SQL queries were applied). Note that details of applied SQL queries are described below.

The initial dataset allows reasoning over 97M facts, which would not highlight the full potential of the proposed method. For scalability purposes, copies of the aforementioned dataset were generated by adjusting the *primaryid* field, where 3, 6, and 12 copies correspond to 291M, 582M, and 1.16 billion facts, respectively. Note that the *primaryid* field is adjusted by appending a counter, namely for three copies the following input:

primaryid	caseid	rpsr_cod
100208273	10020827	FGN

would be transformed into:

primaryid	caseid	rpsr_cod
1002082731	10020827	FGN
1002082732	10020827	FGN
1002082733	10020827	FGN

Note that the first step from one copy to three copies is counterintuitive in terms of scalability (not a power of two); however, it still provides interpretable results while allowing an evaluation of up to 1.16 billion facts (for 12 copies).

Rule set. The rule set consist of rules that are manually converted from *U.S. ELECTRONIC CODE OF FEDERAL REGULATIONS, Title 21: Food and Drugs, PART 310-NEW DRUGS, Subpart D-Records and Reports US Government (2014)*[8]. As discussed in Islam and Governatori (2018), the regulations: (a) specify the records and reports

[8] https://www.ecfr.gov/cgi-bin/text-idx?SID=7bf64fa0b8f5d9185244a769699c5e13&mc=true&node=se21.5.310_1305&rgn=div8

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

concerning adverse drug experiences on marketed prescription drugs for human use without approved new drug applications, and (b) include reporting requirements for Manufacturers, Packers, and Distributors (MPD) and information reported on various life-threatening serious and unexpected adverse drug experience for Individual Case Safety Report (ICSR).

Consider, for example, the provision (as part of informed on ICSRs) prescribing to report electronically to FDA as ICSRs to include "Patient age" while reporting to FDA. Its formal representation in Defeasible Deontic Logic is:

$$r_1 : \text{[OAPNP]}\ \mathit{report_on_ICSRs_to_FDA}(\mathrm{X}) \ \Rightarrow\ \text{[OAPNP]}\ \mathit{report_Patient_age_to_FDA}(\mathrm{X})$$

where [OAPNP] is a deontic operator expressing obligation, while using the defeasible logic as defined in this work we have:

$$r_1 : \mathit{obl_report_on_ICSRs_to_FDA}(\mathrm{X}) \ \Rightarrow\ \mathit{obl_report_Patient_age_to_FDA}(\mathrm{X})$$

Note that obligation conclusions such as *obl_report_Patient_age_to_FDA(X)* in rule r_1 are loaded in memory and broadcast to each node in the implementation in order to ensure parallelism (see lines 1–2 in Algorithm 1).

SQL Queries. In Islam and Governatori (2018), predicate extraction (facts generation) is performed through SQL queries over a PostgreSQL database, while in our experiments such SQL queries are part of the implementation using Spark SQL. Each query represents a single predicate, which is eventually passed as an input parameter to the reasoner. Note that the reasoner is implemented using Spark, based on Algorithm 1.

The following query illustrates sample test predicates *ICSRs_contain_Patient_age* in PostgreSQL:

```
SELECT primaryid,
CASE WHEN age IS NOT NULL THEN 'report_Patient_age_to_FDA'
     ELSE '-report_Patient_age_to_FDA'
END
FROM DEMO14Q1
```

Such query evaluation can be implemented using Spark SQL by loading in memory a DataFrame, called *demoDF*, containing records found in file *DEMO* where attribute *age* must be defined in a given row. The aforementioned SQL query can be translated in Spark SQL as follows (note that in this work only positive literals are relevant):

```
demoDF
 .where(demoDF.col("age").isNotNull && demoDF.col("age") =!= "")
 .select(demoDF.col("primaryid").as("argument_X"))
 .withColumn("predicate",lit("report_Patient_age_to_FDA"))
```

The following input (note that file *DEMO* contains more columns, which are not included below for readability purposes):

primaryid	caseid	...	age	...
100051922	10005192	...	21	...

would be transformed into:

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Table 3. *Number of SQL queries*

DEMO	DRUG	OUTC	REAC	RPSR	Whole
13	5	1	1	1	21

Table 4. *Number of conclusions (in millions)*

Copies	DEMO	DRUG	OUTC	REAC	RPSR	Whole
1	310.390	291.485	150.876	280.142	10.781	335.355
3	931.171	874.455	452.627	840.426	32.344	1,006.065
6	1,862.343	1,748.910	905.253	1,680.852	64.688	2,012.130
12	3,724.665	3,497.801	1,810.491	3,361.686	129.376	4,024.237

argument_X	predicate
100051922	report_Patient_age_to_FDA

which essentially represents the fact *report_Patient_age_to_FDA(100051922)*.

The number of executed queries for each file is included in Table 3, where it is clear that the majority of queries are executed over files *DEMO* and *DRUG*. Note that all SQL queries are included in our publicly available implementation (see Section 6.3).

All rules are transformed into the notation defined in this work with the reasoner being implemented in Spark specifically for this rule set. Both fact extraction (Spark SQL) and reasoning (Spark) are implemented within a single job as described in Algorithm 1 (see Section 6.3).

7.2 Platform

Our evaluation platform is the Kay supercomputer located at Irish Centre for High-End Computing (ICHEC). We choose up to 17 nodes from the system, and each node we have used contains a 20-core Intel Xeon Gold 6148 (Skylake) processor running at 2.4GHz with 192GB of RAM and a single 400GB SSD local disk. The operating system is Linux kernel version 3.10.0-693 and the software stack consists of Spark version 2.3.1, Hadoop version 2.7.3, Scala version 2.11.8, and Java version 1.8.0_191.

For Spark, we set the following system parameters: *spark_worker_memory* and *spark_executor_memory* are set to 160GB and *spark_worker_cores* is to 20. In all our experiments, the operations of input file reading are on the HDFS system using the SSD on each node. We measure runtime as the elapsed time from job submission to the job being reported as finished and we record the mean value based on three measurements.

7.3 Results

Table 4 provides insight in terms of reasoning. Specifically, it is evident that queries for *DEMO*, *DRUG*, and *REAC* are generating comparable numbers of conclusions, *OUTC* generates approximately half compared to the aforementioned queries, while *RPSR* generates only a fraction of conclusions. The number of conclusions for each set of queries is a function of the number of rows in the corresponding file (see Table 2) and the number of executed queries (see Table 3). However, providing the exact function that would allow

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

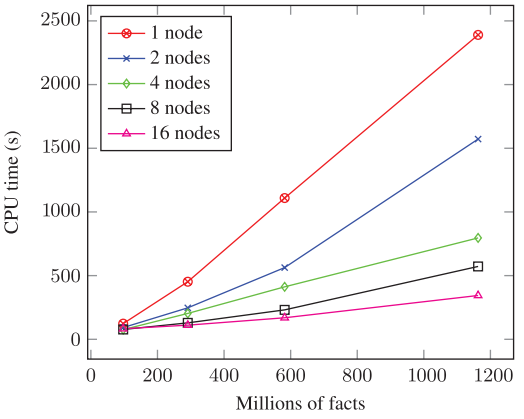


Fig. 3. Time in seconds as a function of number of facts, for various numbers of nodes.

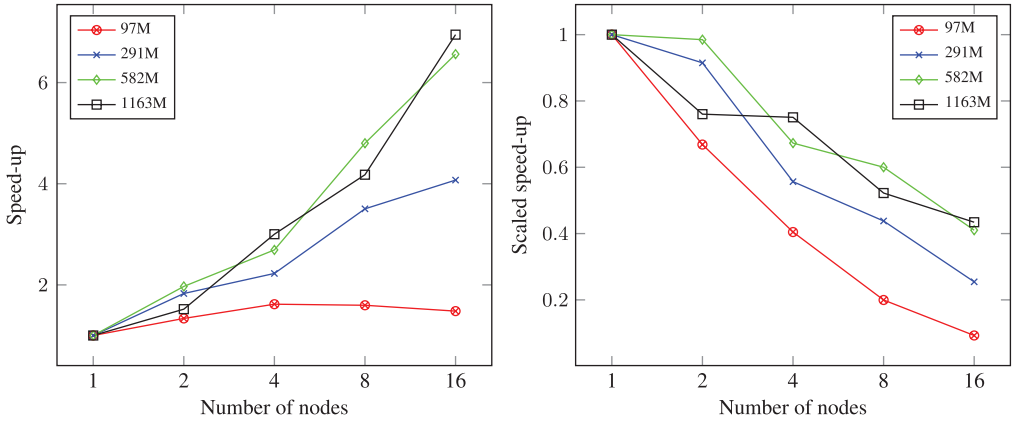


Fig. 4. Speed-up and scaled speed-up as a function of numbers of nodes, for various numbers of facts.

an accurate prediction of the number of conclusions, based on the number of rows and executed queries, is out of the scope of this work.

Figure 3 shows the scalability results of the implementation for increasing number of facts. The implementation follows a fairly linear scalability up to 1.16 billion facts when the number of nodes ranges from 1 to 16. From a practical point of view, the initial dataset (four calendar years) can be processed with 16 nodes in 83 s, while 12 copies (corresponding to almost half a century) can be processed in 5 min and 44 s. For comparison, the initial dataset would require more than 5 days with RuleRS (Islam and Governatori 2018), while auditing the 12 copies with RuleRS would require approximately 2 months. Even though the proposed approach in this work cannot be directly compared to RuleRS, given the fact that RuleRS is based on a serial implementation, our results show a significant scalability advantage of the proposed inference rules.

Figure 4 depicts speed-ups and scaled speed-ups[9] for increasing number of nodes, for various number of facts. It is evident that 97M of facts is relatively small input in order

[9] Speed-up is calculated as: $\frac{runtime_{1node}}{runtime_{Nnodes}}$, while scaled speed-up is calculated as: $\frac{runtime_{1node}}{N * runtime_{Nnodes}}$, where $runtime_{1node}$ is the required run time for one node, N is the number of nodes, and $runtime_{Nnodes}$ is the required run time for N nodes.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

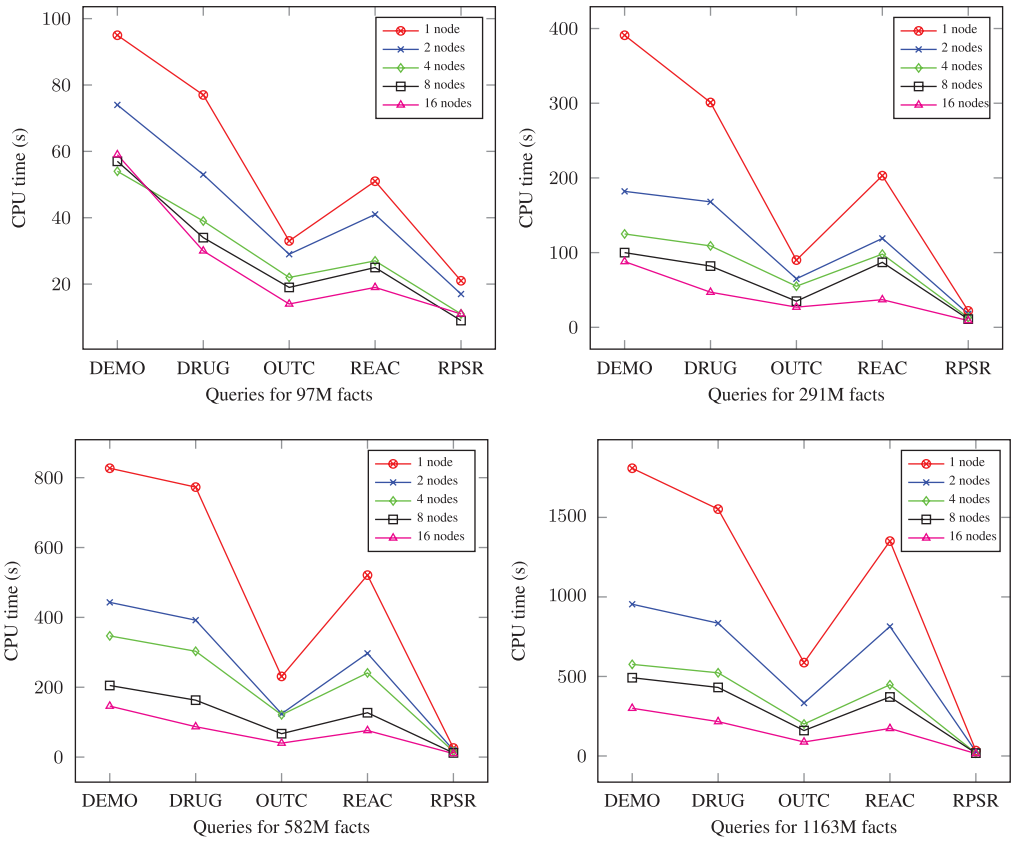


Fig. 5. Time in seconds as a function of queries, for various numbers of facts and nodes.

to show the benefits of parallelization, this is attributed to the fact that the majority of time is dedicated to reading the input. On the other hand, larger inputs highlight the advantages of the distributed implementation. However, the speed-ups are sublinear regardless of the number of facts or nodes. Nonetheless, the results are encouraging in terms of a proof of concept.

Figure 5 presents the required time in order to execute each set of queries separately (including reasoning over generated facts). The required time in declining order is as follows: *DEMO*, *DRUG*, *REAC*, *OUTC*, and *RPSR*. This is consistent with Tables 2 and 4 since larger files require more time to be read, while more conclusions mean both longer reasoning time and more generated facts from the executed queries. Finally, once the input is large enough, there is a clear trend where adding more nodes leads to faster runtimes.

8 Conclusion and future work

In this paper, we introduced a scalable defeasible logic that allows reasoning over large amounts of data. In particular, we proposed new inference rules for defeasible reasoning, discussed the theoretical properties of the new defeasible logic, and ran experiments over an FDA case study (with rules encoding FDA regulations over publicly available FDA

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

datasets). Our experimental results indicate that this method can be applied to billions of facts.

In future work, we plan to develop a generic implementation of a parallel reasoner over the logic we propose in this work. In addition, we plan to study how the proposed inference rules can be extended further in order to model more complex constructs while retaining scalability. In particular, a potential direction could be the introduction of a scalable Defeasible Deontic Logic as an alternative to the one presented in (Governatori *et al.* 2013). Another direction could be the extension of the proposed defeasible logic in this work to the BOID (Belief, Obligation, Intention, Desire) architecture Governatori and Rotolo 2008). Such approaches would facilitate reasoning in the legal context, thus providing scalable solutions for processing large amounts of legal documents.

References

Antoniou, G., Batsakis, S., Mutharaju, R., Pan, J. Z., Qi, G., Tachmazidis, I., Urbani, J. and Zhou, Z. 2018. A survey of large-scale reasoning on the web of data. *The Knowledge Engineering Review 33*, e21.

Antoniou, G., Billington, D., Governatori, G. and Maher, M. J. 1999. On the modelling and analysis of regulations. In *Proceedings of the Australasian Conference on Information Systems*, 20–29.

Antoniou, G., Billington, D., Governatori, G. and Maher, M. J. 2000. A flexible framework for defeasible logics. In *AAAI/IAAI*. AAAI Press/The MIT Press, 405–410.

Antoniou, G., Billington, D., Governatori, G. and Maher, M. J. 2001. Representation results for defeasible logic. *ACM Transactions on Computational Logic 2,* 2, 255–287.

Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A. and Zaharia, M. 2015. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD Conference*. ACM, 1383–1394.

Billington, D., Antoniou, G., Governatori, G. and Maher, M. J. 2010. An inclusion theorem for defeasible logics. *ACM Transactions on Computational Logic 12,* 1, 6.

Cheng, L., Van Dongen, B. and Van Der Aalst, W. 2019. Scalable discovery of hybrid process models in a cloud computing environment. *IEEE Trans. Services Computing*.

Condie, T., Das, A., Interlandi, M., Shkapsky, A., Yang, M. and Zaniolo, C. 2018. Scaling-up reasoning and advanced analytics on BigData. *TPLP 18,* 5–6, 806–845.

Cook, S. and Nguyen, P. 2010. *Logical Foundations of Proof Complexity*. Cambridge University Press.

Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys 33,* 3, 374–425.

Dowling, W. F. and Gallier, J. H. 1984. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming 1,* 3, 267–284.

Gelder, A. V., Ross, K. A. and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of ACM 38,* 3, 620–650.

Goodman, E. L., Jimenez, E., Mizell, D., Al-Saffar, S., Adolf, B. and Haglin, D. J. 2011. High-Performance Computing Applied to Semantic Databases. In *ESWC (2),* 31–45.

Governatori, G. and Maher, M. J. 2017. Annotated defeasible logic. *TPLP 17,* 5-6, 819–836.

Governatori, G., Milosevic, Z. and Sadiq, S. W. 2006. Compliance checking between business processes and business contracts. In *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006),* 221–232.

Governatori, G., Olivieri, F., Rotolo, A. and Scannapieco, S. 2013. Computing strong and weak permissions in defeasible logic. *Journal of Philosophical Logic 42,* 6, 799–829.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Governatori, G. and Pham, D. H. 2009. DR-CONTRACT: An architecture for e-contracts in defeasible logic. *IJBPIM 4,* 3, 187–199.

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.

Grosof, B. N., Labrou, Y. and Chan, H. Y. 1999. A declarative approach to business rules in contracts: courteous logic programs in XML. In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3–5, 1999*, 68–77.

Hashmi, M., Governatori, G., Lam, H. and Wynn, M. T. 2018. Are we done with business process compliance: state of the art and challenges ahead. *Knowledge and Information Systems 57,* 1, 79–133.

Heino, N. and Pan, J. Z. 2012. RDFS reasoning on massively parallel hardware. In *The Semantic Web - ISWC 2012 – 11th International Semantic Web Conference, Boston, MA, USA, November 11–15, 2012, Proceedings, Part I*, P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, Eds. LNCS, vol. 7649. Springer, 133–148.

Islam, M. B. and Governatori, G. 2018. RuleRS: A rule-based architecture for decision support systems. *Artificial Intelligence and Law 26,* 4, 315–344.

Kim, J. and Park, Y. 2015. Scalable owl-horst ontology reasoning using SPARK. In *2015 International Conference on Big Data and Smart Computing, BIGCOMP 2015, Jeju, South Korea, February 9–11, 2015*, 79–86.

Leone, N., Allocca, C., Alviano, M., Calimeri, F., Civili, C., Costabile, R., Fiorentino, A., Fuscà, D., Germano, S., Laboccetta, G., Cuteri, B., Manna, M., Perri, S., Reale, K., Ricca, F., Veltri, P. and Zangari, J. 2019. Enhancing DLV for large-scale reasoning. In *Logic Programming and Nonmonotonic Reasoning – 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3–7, 2019, Proceedings*, M. Balduccini, Y. Lierler, and S. Woltran, Eds. LNCS, vol. 11481. Springer, 312–325.

Liu, C., Qi, G., Wang, H. and Yu, Y. 2011. Large Scale Fuzzy pD* Reasoning Using MapReduce. In *10th International Semantic Web Conference, Bonn, Germany, October 23–27*. LNCS, vol. 7031. Springer, 405–420.

Maher, M. J. 2001. Propositional defeasible logic has linear complexity. *TPLP 1,* 6, 691–711.

Maher, M. J. 2012. Relative expressiveness of defeasible logics. *TPLP 12,* 4–5, 793–810.

Maher, M. J. 2013. Relative expressiveness of defeasible logics II. *TPLP 13,* 4–5, 579–592.

Maher, M. J., Antoniou, G. and Billington, D. 1998. A study of provability in defeasible logic. In *Proc. 11th Australian Joint Conference on Artificial Intelligence*. LNCS, vol. 1502. Springer, 215–226.

Martinez-Angeles, C. A., de Castro Dutra, I., Costa, V. S. and Buenabad-Chávez, J. 2013. A datalog engine for GPUs. In *Declarative Programming and Knowledge Management - Declarative Programming Days, KDPD 2013, Unifying INAP, WFLP, and WLP, Kiel, Germany, September 11–13, 2013, Revised Selected Papers*, M. Hanus and R. Rocha, Eds. LNCS, vol. 8439. Springer, 152–168.

Mutharaju, R., Hitzler, P., Mateti, P. and Lécué, F. 2015. Distributed and scalable OWL EL reasoning. In *The Semantic Web. Latest Advances and New Domains – 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 – June 4, 2015. Proceedings*, F. Gandon, M. Sabou, H. Sack, C. d'Amato, P. Cudré-Mauroux, and A. Zimmermann, Eds. LNCS, vol. 9088. Springer, 88–103.

Oren, E., Kotoulas, S., Anadiotis, G., Siebes, R., ten Teije, A. and van Harmelen, F. 2009. Marvin: Distributed reasoning over large-scale Semantic Web data. *J. Web Sem. 7,* 4, 305–316.

Prakken, H. 1997. *Logical Tools for Modelling Legal Argument: A Study of Defeasible Reasoning in Law*. Kluwer Academic Publishers.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Skylogiannis, T., Antoniou, G., Bassiliades, N., Governatori, G. and Bikakis, A. 2007. DR-NEGOTIATE – A system for automated agent negotiation with defeasible logic-based strategies. *Data & Knowledge Engineering 63,* 2, 362–380.

Tachmazidis, I. 2015. Large-scale reasoning with nonmonotonic and imperfect knowledge through mass parallelization. Ph.D. thesis, University of Huddersfield, UK.

Tachmazidis, I. and Antoniou, G. 2013. Computing the stratified semantics of logic programs over big data through mass parallelization. In *Theory, Practice, and Applications of Rules on the Web – 7th International Symposium, RuleML 2013, Seattle, WA, USA, July 11–13, 2013. Proceedings*, L. Morgenstern, P. S. Stefaneas, F. Lévy, A. Z. Wyner, and A. Paschke, Eds. LNCS, vol. 8035. Springer, 188–202.

Tachmazidis, I., Antoniou, G. and Faber, W. 2014. Efficient computation of the well-founded semantics over big data. *TPLP 14,* 4–5, 445–459.

Tachmazidis, I., Antoniou, G., Flouris, G. and Kotoulas, S. 2012a. Towards parallel nonmonotonic reasoning with billions of facts. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10–14, 2012*, G. Brewka, T. Eiter, and S. A. McIlraith, Eds. AAAI Press.

Tachmazidis, I., Antoniou, G., Flouris, G., Kotoulas, S. and McCluskey, L. 2012b. Large-scale parallel stratified defeasible reasoning. In *ECAI 2012 – 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27–31, 2012*, L. D. Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas, Eds. Frontiers in Artificial Intelligence and Applications, vol. 242. IOS Press, 738–743.

Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F. and Bal, H. E. 2012. Webpie: A web-scale parallel inference engine using mapreduce. *Journal of Web Semantics, 10*, 59–75.

Zhou, Z., Qi, G., Liu, C., Hitzler, P. and Mutharaju, R. 2013. Scale reasoning with fuzzy-EL+ ontologies based on MapReduce. In *Proceedings of the IJCAI-2013 Workshop on Weighted Logics for Artificial Intelligence, WL4AI-2013, Beijing, China, August 2013*, 87–93.

Appendix A Relative inference strength

Proposition 12
$\Delta \subset \partial^*_{||} \subset \partial_{||} \subset \lambda$

Proof
The first containment follows immediately from (1) of the $\partial^*_{||}$ inference rule. The only difference between $\partial^*_{||}$ and $\partial_{||}$ is in (2.3.2), and the clause for $\partial^*_{||}$ implies the clause for $\partial_{||}$. The second containment follows. The λ inference rule is essentially the $\partial_{||}$ inference rule with condition (2.3) omitted. The third containment then follows.

Strictness is shown with straightforward examples. Strictness of the first containment is shown by D consisting only of $\Rightarrow p$. Strictness of the second containment is shown by the standard example distinguishing team and individual defeat: D consists of:

$$\begin{array}{lcr} r_1: & \Rightarrow & p \\ r_2: & \Rightarrow & p \\ r_3: & \Rightarrow & \neg p \\ r_4: & \Rightarrow & \neg p \end{array}$$

with $r_1 > r_3$ and $r_2 > r_4$. Then, we can conclude $+\partial_{||}p$ but not $+\partial^*_{||}p$.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Example 13 shows the strictness of the third containment since $+\lambda q$ is proved but $+\partial_{||}q$ cannot be proved. □

It is straightforward to see that λ is not consistent.

Example 13
Consider the defeasible theory

$$\begin{array}{lcl} r: & \Rightarrow & q \\ s: & \Rightarrow & \neg q \end{array}$$

with empty superiority relation.

Then we can infer $+\lambda q$ and $+\lambda\neg q$, but cannot infer $+\Delta q$ nor $+\Delta\neg q$. Thus, λ is not consistent. Furthermore, we cannot infer $+\partial_{||}q$ nor $+\partial_{||}\neg q$.

Proposition 14
The inference rule $+\partial_{||}$ is consistent.

Proof
Suppose, for some defeasible theory D, and some proposition q, that $+\partial_{||}q$ and $+\partial_{||}\neg q$ are consequences of D.

If $+\Delta\neg q \in P_\Delta$ but $+\Delta q \notin P_\Delta$ then, when attempting to prove $+\Delta q$, neither (1) nor (2.2) of the $\partial_{||}$ inference rule hold and, thus, $+\Delta q$ cannot be proved. This contradicts our original supposition, so this case cannot occur. Similarly, the case where $+\Delta q \in P_\Delta$ but $+\Delta\neg q \notin P_\Delta$ cannot occur.

In the third case, neither $+\Delta q$ nor $+\Delta\neg q$ are consequences. Since $+\partial_{||}q$ is a consequence and (1) does not hold, (2.1) of the $\partial_{||}$ inference rule must hold for some rule r for q. Symmetrically, there is a rule s for $\neg q$ such that (2.1) holds. Consequently, by Proposition 12, for each $\alpha \in A(s)$, $+\lambda\alpha \in P_\lambda$. Hence, to infer $+\partial_{||}q$, there must be a rule t for q with $t > s$ and for each $\beta \in A(t)$, $+\partial_{||}\beta$ is provable and thus $+\lambda\beta \in P_\lambda$. But then, to infer $+\partial_{||}\neg q$, there must be a rule t' with $t' > t$ and for each $\gamma \in A(t)$, $+\partial_{||}\gamma$ is provable. And so on. This creates an infinite chain of rules, each superior to the previous rule. No rule can be repeated, since $>$ is acyclic. However, the chain cannot be infinite, since the set of rules is finite. This contradiction shows that this case cannot occur.

Thus, by exclusion, both $+\Delta q$ and $+\Delta\neg q$ are consequences, and the result is proved. □

It follows immediately from Propositions 3 and 12 that $DL(\partial^*_{||})$ also is consistent.

Corollary 15
The inference rule $\partial^*_{||}$ is consistent.

The next two examples show that ∂ and $\partial_{||}$ are incomparable in inference strength.

Example 16
Consider the defeasible theory

$$\begin{array}{lcccl} r: & & \Rightarrow & & q \\ s: & \neg q & \rightarrow & & \neg q \end{array}$$

with $r > s$.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Then $+\Delta\neg q$ cannot be inferred, and so $+\partial_{||}q$ is inferred. On the other hand, $-\Delta\neg q$ also cannot be inferred, and so $+\partial q$ cannot be inferred. Consequently, $\partial_{||} \not\subseteq \partial$.

This comes about because of the different treatments of opposing strict inferences in the two inference rules.

Example 17
Consider the defeasible theory

$$\begin{array}{lrcl} r: & & \Rightarrow & q \\ s: & & \Rightarrow & \neg q \\ t: & & \Rightarrow & p \\ u: & q & \Rightarrow & \neg p \end{array}$$

with no superiority relation.

Then we can infer $-\partial q$ and $+\lambda q$. Consequently, we can infer $+\partial p$, but not $+\partial_{||}p$. Hence, $\partial \not\subseteq \partial_{||}$.

This comes about because the inference rules for $+\partial_{||}$ and $+\partial$ differ at (2.3.1): $+\partial_{||}$ requires $+\lambda\alpha \notin P_\lambda$ while $+\partial$ requires $-\partial\alpha \in P(1..i)$.

Proposition 18
$\partial \subset \lambda$ and $\partial^* \subset \lambda$

Proof
The λ inference rule has no condition (2.3) and replaces the condition for $-\Delta{\sim}q \in P$ for ∂ with $+\Delta{\sim}q \notin P$. By the coherence of defeasible logics (Billington *et al.* 2010), $+\Delta{\sim}q \notin P$ is a weaker condition. Hence, every inference that $+\partial$ can make can be duplicated by $+\lambda$. The result then follows.

The same argument applies to show that $\partial^* \subseteq \lambda$.

Strictness in both cases is straightforward, using the same example and argument (Example 13) as in the proof of Proposition 12 for the strictness of the third containment. □

We establish the lack of any additional containments in Figure 1 using the following three examples.

Example 19
Consider the defeasible theory

$$\begin{array}{lrcl} r: & & \Rightarrow & q \\ s: & \neg q & \rightarrow & \neg q \end{array}$$

with $s > r$.

Then we can infer $+\lambda q$ and $+\partial^*_{||}q$, but not $+\sigma_{\delta^*}q$. This arises because the inference rule for $+\lambda$ and $+\partial^*_{||}$ requires only that $+\Delta{\sim}q$ is not inferred, while the inference rule for $+\sigma_{\delta^*}$ must establish $-\delta^*{\sim}q$. In this case, $-\delta^*{\sim}q$ cannot be inferred. Thus, $\partial^*_{||} \not\subseteq \sigma_{\delta^*}$.

It then follows that $\partial^*_{||} \not\subseteq X$, for any X considered in Governatori and Maher (2017) (since $X \subseteq \sigma_{\delta^*}$ Billington *et al.* (2010)), and also $\partial_{||} \not\subseteq X$ and $\lambda \not\subseteq X$ (since $\partial^*_{||} \subseteq \partial_{||}$).

We use σ_X to denote any of the support inference rules σ_{δ^*}, σ_δ, σ_{∂^*}, and σ_∂.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Example 20
Consider the defeasible theory

$$\begin{array}{lll} r: & \Rightarrow & q \\ s: & \rightarrow & \neg q \end{array}$$

with no superiority relation.

Then we can infer $+\sigma_X q$ but not $+\lambda q$. Thus $\sigma_X \not\subseteq \lambda$. This comes about because the inference rules for $+\sigma_X$ ignores the possibility of strict inference of $\sim q$, while the inference rule for $+\lambda$ does not.

Hence, $\sigma_X \not\subseteq \lambda$, for any X.

Example 21
Consider the defeasible theory

$$\begin{array}{lll} r: & \Rightarrow & p \\ s: & \Rightarrow & \neg p \\ & \Rightarrow & q \\ p & \Rightarrow & \neg q \end{array}$$

with $s > r$.

Then we can infer $+\lambda p$, and hence cannot infer $+\partial_{||} q$. On the other hand, we can infer $-\sigma_{\delta^* p}$, since $s > r$, and hence we can infer $+\delta^* q$. Thus, $\delta^* \not\subseteq \partial_{||}$.

Because $\partial^*_{||} \subseteq \partial_{||}$ and $\delta^* \subseteq X$ for every X discussed in Governatori and Maher (2017) except Δ, we can conclude that neither $\partial_{||}$ nor $\partial^*_{||}$ contains any X discussed in Governatori and Maher (2017) except Δ.

Theorem 22
The containments illustrated in Figure 1 hold and are strict. In addition, $\partial^* \subset \lambda$ holds. There are no other missing containments in the figure.

Proof
The containments on the top row of the diagram are established in Proposition 12. The containments on and between the lower two rows are established in Billington *et al.* (2010), Governatori and Maher (2017), including their strictness and the lack of any other containments among them. The containments $\partial^* \subset \lambda$ and $\partial \subset \lambda$ are established in Proposition 18.

Example 19 shows that no tag in the lower rows contains a tag in the upper row. Furthermore, Example 21 shows that $\partial^*_{||}$ and $\partial_{||}$ do not contain any tag on the lower rows, except for Δ. and Example 20 shows that λ does not contain any of the σ_X tags. Examples showing that containments are strict are straightforward and left to the reader. □

Appendix B Complexity

In this appendix, we prove results on the complexity of $DL(\partial_{||})$.

As a result of the structure of the inference rules, it is straightforward to compute the consequences of Δ and λ efficiently.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Lemma 23
The Δ and λ closures, P_Δ and P_λ, of a propositional defeasible theory can be computed in linear time.

Proof
(Sketch) The inference rule for $+\Delta$ is already treated in Maher (2001). Alternatively, this inference rule is essentially treating strict rules as definite clauses, where negative literals ($\neg p$) are considered as atoms (e.g., not_p). Such inference can be done in time linear in the size of facts and strict rules (Dowling and Gallier 1984).

Similarly, the inference rule for $+\lambda$ essentially treats strict and defeasible rules as definite clauses, with an extra condition about Δ consequences. Once the Δ consequences have been computed, it takes constant extra time for each rule to check the extra condition. Consequently, the inference of $+\lambda$ consequences takes time linear in the size of facts, strict rules, and defeasible rules.

Similarly, $-\Delta$ and $-\lambda$ consequences (see Appendix C) are also computed in linear time (although this information is not necessary for the results in this appendix). □

The inference problem for propositional $DL(\partial)$ has linear complexity Maher (2001), and we use the same techniques to show that the inference problem for propositional $DL(\partial_{||})$ also has linear complexity.

Theorem 24
The set of all consequences of a propositional defeasible theory can be computed in time linear in the size of the defeasible theory. Consequently, the inference problem for propositional $DL(\partial_{||})$ can be solved in linear time.

Proof
(Sketch) We adapt the approach of Maher (2001). This is possible largely because the structure of the inference rules for ∂ and $\partial_{||}$ are the same. First, observe that the transformations of Antoniou *et al.* (2001) for $DL(\partial)$ are also correct for $DL(\partial_{||})$. These transformation are used in Maher (2001) to reduce the input defeasible theory D to an equivalent theory in simpler form.

There are three transformations in (Antoniou *et al.* 2001) . The first, *regular*, separates strict rules from the superiority relation, and it is straightforward to see that this is valid for a wide range of defeasible logics, including $DL(\partial_{||})$. The other two, *elim_dft* and *elim_sup*, which are used to eliminate defeaters and the superiority relation, respectively, employ the same technique to achieve their respective aims: they introduce an intermediate literal in a rule that might be attacked.

For example, roughly speaking, a rule $B \Rightarrow h$ is replaced by $B \Rightarrow temp$ and $temp \Rightarrow h$, and a defeater $B' \rightsquigarrow \sim h$ is replaced by $B' \Rightarrow \sim temp$. Similarly, if we have rules $r_1 : B_1 \Rightarrow h$ and $r_2 : B_1 \Rightarrow \sim h$ with $r_1 > r_2$ then these are replaced by $B_1 \Rightarrow temp_1$, $temp_1 \Rightarrow h$, $B_2 \Rightarrow temp_2$, $temp_2 \Rightarrow \sim h$, and $temp_1 \Rightarrow \neg temp_2$, where the latter rule encodes $r_1 > r_2$. In each case, when the defeater or overriding rule is active the intermediate literal fails to be proved because it is attacked by another rule, and consequently the application of the original rule is prevented. Because the structure of the inference rules is the same for $DL(\partial_{||})$ and $DL(\partial)$, the introduction of intermediate literals and the effect of an attacking rule is the same in both logics. Thus, the technique is also correct in $DL(\partial_{||})$.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

We have already seen that P_Δ and P_λ can be computed in linear time. Now we can simplify the transformed version of D and deduce some $\partial_{||}$ consequences.

Let C be a set of consequences, initially $\emptyset$.

1. For each literal q: If $+\Delta q \in P_\Delta$, then delete all defeasible rules for $\sim q$, add $+\partial_{||} q$ to C, and delete all occurrences of q from the body of rules.
2. For each literal q: If $+\lambda q \notin P_\lambda$ and q occurs in the body of a rule, delete the rule.
3. Delete all strict rules.

Simplification 1 is justified by (2.2) and (1) of the inference rule. Simplification 2 is justified by (2.3.1) of the inference rule, and by Proposition 12 (which implies that such rules cannot be used in (2.1) for $\sim q$). Simplification 3 is justified because all definite consequences are already available in P_Δ and, as a result of the *regular* transformation, no other use is made of these rules.

The simplified theory D' incorporates all the effects of references to P_Δ and P_λ. Consequently, the transition system of Maher (2001) applies also to D' with initial consequences C, for $DL(\partial_{||})$. In fact, only the transitions numbered 2, 4, 5, and 8 are needed, since the remaining transitions involve strict rules or negative tags, though 5 is modified by dropping the reference to $-\Delta\sim q$. The simplifications above can be viewed as variants of transitions: simplification 1 corresponds to transitions 6 and 1; and simplification 2 corresponds to transition 10. Simplification 3 is essentially redundancy removal, given C. Furthermore, the data structure used in Maher (2001) to achieve linear complexity in application of the transition system is also applicable to $DL(\partial_{||})$.

Thus, all positive consequences of a propositional defeasible theory D in $DL(\partial_{||})$ can be computed in time linear in the size of D. □

Corollary 25

The set of all consequences of a defeasible theory can be computed in time exponential in the size of the defeasible theory. Furthermore, the inference problem for defeasible theories is EXPTIME-complete.

Proof

Construct a propositional defeasible theory D' from the original defeasible theory D by taking all variable-free instances of all rules using the constants that appear in D. Two instances of rules are related by the superiority relation iff the rules of which they are instances are so related. Let n be the maximum number of variables in a rule of D and c be the number of constants in D. Then there are at most c^n propositional instances of a rule of D, and at most c^{2n} derived superiority statements for each superiority statement in D. Since both n and c may be $O(|D|)$, the size of D' is $O(|D|^{2|D|})$, which is $O(2^{p(|D|)})$, for a polynomial p.

D and D' have the same consequences. By Theorem 5, the consequences of D' can by computed in linear time in the size of D', which is EXPTIME in the size of D.

The inference problem is shown EXPTIME-complete by reduction of the same problem for Datalog (see Dantsin *et al.* 2001, Theorem 4.5). Each Datalog rule is expressed as a defeasible rule. A positive literal is inferred in $DL(\partial_{||})$ iff it is inferred in Datalog. □

Theorem 26

The inference problem for propositional defeasible logics is P-complete.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

Proof

We show that the inference problem for $+\Delta$ is P-complete, by reduction of the Horn satisfiability problem, which is P-complete (Cook and Nguyen 2010). For completeness, we first specify this problem. A Horn clause is a disjunction of literals containing at most one positive literal.

The Horn Satisfiability Problem

Instance

A set H of propositional Horn clauses.

Question

Is H satisfiable, that is, is there an assignment of Boolean values to propositional variables such that each clause of H evaluates to true?

In the reduction, each of the propositional variables in the Horn satisfiability problem is represented by itself, and we add an extra propositional variable **false**. For clarity, we write the Horn clauses in the logic programming style.

For every Horn clause of the form

$$A \leftarrow B_1, \ldots, B_n$$

the defeasible theory contains the strict rule

$$B_1, \ldots, B_n \rightarrow A$$

Similarly, for every Horn clause of the form

$$\leftarrow B_1, \ldots, B_n$$

the defeasible theory contains the strict rule

$$B_1, \ldots, B_n \rightarrow \mathbf{false}$$

It is straightforward to show that $+\Delta q$ is inferred by a defeasible logic iff q is true in every model of the definite clause subset of H, and $+\Delta\mathbf{false}$ is inferred by a defeasible logic iff H is unsatisfiable.

Strict inference is a part of any defeasible logic, so the result applies to all defeasible logics. Even without a separate notion of strict inference, the proof extends easily to any inference rule that allows the chaining of defeasible or strict rules, since the superiority relation and conflicting rules do not arise in the reduction. This includes all defeasible logics we are aware of. □

Appendix C Relative expressiveness

In this appendix, we prove Theorems 9 and 10.

Relative expressiveness involves both positive and negative tags, so we first introduce the inference rules for $-\lambda$ and $-\partial_{||}$. These inference rules are a kind of negation of the corresponding positive inference, under the Principle of Strong Negation (Antoniou *et al.* 2000). However, the notion of strong negation must be extended to address expressions of the form $t\,\alpha \notin P$, which were not considered in Antoniou *et al.* (2000). In these cases, we define the strong negation of $t\,\alpha \notin P$ to be $t\,\alpha \in P$.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

The closure P_Δ must be closed under both $+\Delta$ and $-\Delta$ inference rules, that is, it must contain all $+\Delta$ and $-\Delta$ consequences.

The $-\lambda$ inference rule is as follows.

$-\lambda$: We may append $P(i+1) = -\lambda q$ if both
(1) $-\Delta q \in P_\Delta$ and
(2) (2.1) $\forall r \in R_{sd}[q]\ \exists \alpha \in A(r) : -\lambda\alpha \in P(1..i)$ or
(2.2) $+\Delta{\sim}q \in P_\Delta$

The λ closure P_λ contains all $+\lambda$ and $-\lambda$ consequences of D.

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

To prove the first part of Theorem 9, we employ an analysis introduced in Maher *et al.* (1998) and Antoniou *et al.* (2001). For each proposition p, we can identify exactly six different possible outcomes of the proof theory. With each outcome, we present a simple theory that achieves this outcome.

A: $-\Delta p \notin P_\Delta$ and $+\partial_{||}p \notin P_{\partial_{||}}$
$p \to p$
B: $+\partial_{||}p \in P_{\partial_{||}}$ and $+\partial_{||}p \notin P_\Delta$ and $-\Delta p \notin P_\Delta$
$\Rightarrow p; p \to p$
C: $+\Delta p \in P_\Delta$ (and also $+\partial_{||}p \in P_{\partial_{||}}$)
$\to p$
D: $+\partial_{||}p \in P_{\partial_{||}}$ and $-\Delta p \in P_\Delta$
$\Rightarrow p$
E: $-\Delta p \in P_\Delta$ and $+\partial_{||}p \notin P_{\partial_{||}}$ and $-\partial_{||}p \notin P_{\partial_{||}}$
$p \Rightarrow p$
F: $-\partial_{||}p \in P_{\partial_{||}}$ (and also $-\Delta p \in P_\Delta$)
$\emptyset$, the empty theory

Similarly, there are the same six possibilities for $\neg p$. We can represent the outcomes in terms of a Venn diagram in Figure C 1.

In Figure C 1, the circle on the left – containing B, C, and D – represents the literals p such that $+\partial_{||}p$ can be proved, and the ellipse inside it (i.e., C) represents the literals p such that $+\Delta p$ can be proved. The circle on the right – containing D, E, and F – represents the literals p such that $-\Delta p$ can be proved, and the ellipse inside it (i.e., F) represents the literals p such that $-\partial_{||}p$ can be proved.

Due to the relationship between p and $\neg p$, many fewer than the 36 possible combinations are possible outcomes of the proof theory.

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

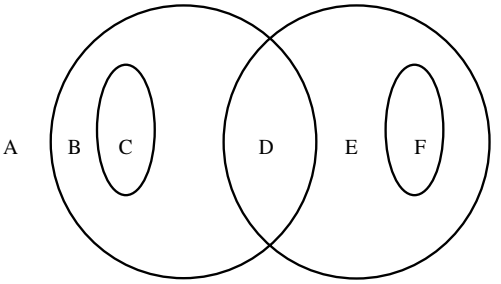


Fig. C 1. Possible outcomes for a single literal in $DL(\partial_{||})$

In what follows, p ranges over literals and $\sim p$ denotes the complement of p. We first establish some simple properties that will eliminate many combinations.

Proposition 27
Consider a defeasible theory D, with Δ closure P_Δ and $\partial_{||}$ closure $P_{\partial_{||}}$.

1. If $+\Delta p \notin P_\Delta$ and $+\Delta{\sim}p \in P_\Delta$, then $+\partial_{||}p \notin P_{\partial_{||}}$
2. If $-\Delta p \in P_\Delta$ and $+\Delta{\sim}p \in P_\Delta$, then $-\partial_{||}p \in P_{\partial_{||}}$
3. If $+\Delta p \notin P_\Delta$ and $+\Delta{\sim}p \notin P_\Delta$, then we cannot have both $+\partial_{||}p \in P_{\partial_{||}}$ and $+\partial_{||}{\sim}p \in P_{\partial_{||}}$

Proof
Statements 1 and 2 follow directly from the proof rules for $+\partial_{||}$ and $-\partial_{||}$. Statement 3 is proved as follows.

Suppose this combination is possible. Then, when applying the inference rule $\partial_{||}$ for p (and for $\sim p$), (1) does not apply and (2.2) is satisfied. We also must have $+\lambda p$ and $+\lambda{\sim}p$ because $\partial_{||} \subseteq \lambda$ and by definition of D (because we have $+\partial_{||}p$ and $+\partial_{||}{\sim}p$). There must be a rule r for p (and one for $\sim p$) such that $\forall \alpha \in A(r)\ +\partial_{||}\alpha \in P_{\partial_{||}}$. Consequently, there is a rule s for p (and one for $\sim p$) such that $\forall \alpha \in A(s)\ +\lambda\alpha \in P_\lambda$. Hence, (2.3.1) of inference rule $\partial_{||}$ does not apply, and so (2.3.2) must. That is, there is a rule r for p (and one for $\sim p$) such that $\forall \alpha \in A(t)\ +\partial_{||}\alpha \in P_{\partial_{||}}$ and $t > s$. In this way, we obtain a chain of rules: for each rule s_i for $\sim p$ there is a superior rule t_i for p, and for each rule t_i for p there is a superior rule s_i for $\sim p$. Since D is finite and $>$ is acyclic, this can never occur. □

In terms of the diagram (Figure C 1), the properties of the previous proposition have the following effects:

1. If p satisfies A, B, D, E, or F, and $\sim p$ satisfies C then p satisfies A, E, or F (Property 1). Consequently, it is not possible for p to satisfy B or D, and $\sim p$ to satisfy C.
2. If p satisfies D, E, or F, and $\sim p$ satisfies C then p satisfies F (Property 2). Consequently, it is not possible for p to satisfy D or E, and $\sim p$ to satisfy C.
3. If p satisfies B or D and $\sim p$ satisfies B or D, we have a contradiction. That is, it is not possible for p to satisfy B or D, and $\sim p$ to satisfy B or D.

These effects apply for p a positive or negative literal.

In the table in Figure C 2, we display the possible combinations of conclusions for a proposition p and its negation $\neg p$. The table is symmetric across the leading diagonal,

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

		$\neg p$					
		A	B	C	D	E	F
p	A	*Poss*	*Poss*	*Poss*	*Poss*	*Poss*	*Poss*
	B	*Poss*	*NP(3)*	*NP(1)*	*NP(3)*	*Poss*	*Poss*
	C	*Poss*	*NP(1)*	*Poss*	*NP(1)*	*NP(2)*	*Poss*
	D	*Poss*	*NP(3)*	*NP(1)*	*NP(3)*	*Poss*	*Poss*
	E	*Poss*	*Poss*	*NP(2)*	*Poss*	*Poss*	*Poss*
	F	*Poss*	*Poss*	*Poss*	*Poss*	*Poss*	*Poss*

Fig. C 2. Table of all combinations of conclusions for p and $\neg p$ in $DL(\partial_{||})$, indicating whether or not the combination is possible, and, if not, why not.

since the treatment of literals in defeasible logic is independent of the polarity of the literal. Those combinations which are possible are displayed as *Poss*. Those combinations which are not possible are displayed as *NP(i)*, where i is the property number in Proposition 27 that implies that they are impossible.

For the possible combinations, a sample theory can be exhibited by combining the sample theories for each letter (for p and $\sim p$, respectively). We leave this for the reader to verify.

It is now straightforward to compare this table, for $DL(\partial_{||})$, with the table in Maher *et al.* (1998) and Antoniou *et al.* (2001) for $DL(\partial)$. Every combination that is possible for $DL(\partial)$ is also possible for $DL(\partial_{||})$. Thus, for any defeasible theory D, and each proposition p, we can identify which combination of conclusions $DL(\partial)$ entails and simulate that behavior with the sample theory for that combination for p. This completes the proof of the first part of Theorem 9. Thus, we have

Theorem 28
$DL(\partial)$ is less expressive than $DL(\partial_{||})$ when there are no additions. More specifically,

- every defeasible theory in $DL(\partial)$ can be simulated by a defeasible theory in $DL(\partial_{||})$
- there is a defeasible theory D whose consequences in $DL(\partial_{||})$ cannot be expressed by any defeasible theory in $DL(\partial)$

Proof
The proof of the first part is established by the preceding work in this section. For the second part, let D consists of

$$\begin{array}{ccc} & \Rightarrow & p \\ \neg p & \rightarrow & \neg p \end{array}$$

with empty superiority relation.

Then, $+\lambda p$ and $+\partial_{||} p$ are consequences of D, as is $-\Delta p$, while $-\Delta\neg p$ is not a consequence.

We now show that $DL(\partial)$ cannot simulate this theory. That is, for no defeasible theory D' are both $+\partial p$ and $-\Delta p$ consequences, and $-\Delta\neg p$ not a consequence. Suppose, $+\partial p$ is a consequence of D'. Then either (1) $+\Delta p$ or (2) $-\Delta\neg p$ must be consequences of D', from the inference condition for $+\partial$. But these contradict the requirements on the Δ-consequences of D'. Thus, $DL(\partial)$ is unable to simulate the consequences of D under $DL(\partial_{||})$. □

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press

It is interesting to note that the comparison of tables identifies several different theories that might be used to establish the second part of Theorem 9. However, they all have a similar structure: a literal q is defeasibly provable, despite a loop for $\sim q$.

Theorem 29
$DL(\partial_{||})$ is not more expressive than $DL(\partial)$ with respect to addition of rules.

Proof
Let D be the empty defeasible theory $(\emptyset, \emptyset, \emptyset)$. The consequences of D in $DL(\partial)$ are $-\partial q$, $-\partial\neg q$, $-\Delta q$, and $-\Delta\neg q$ for every proposition q. Suppose, to achieve a contradiction, that D' is a simulation of D w.r.t. addition of rules in $DL(\partial_{||})$. Then, $-\Delta q$ and $-\Delta\neg q$ are consequences of D'. We will consider several additions to D.

For the addition A_1

$$r_1 : \quad \Rightarrow \quad q$$

$D + A_1$ has consequences $+\partial q$ and $-\partial\neg q$.

Then $+\partial_{||} q$ is a consequence of $D' + A_1$. It is straightforward that $-\Delta q$ and $-\Delta\neg q$ are consequences of $D' + A_1$. It follows, from the $\partial_{||}$ inference rule, that (2.3.1) or (2.3.2) holds for each rule s for $\neg q$ in $D' + A_1$.

Note that $+\lambda q \in P_\lambda$ for $D' + A_1$, by Proposition 12. Now, if $+\lambda\alpha \notin P_\lambda$ for $D' + A_1$, for some α, then also $+\lambda\alpha \notin P_\lambda$ for D', because P_λ monotonically increases with the addition of $+\lambda$ consequences. Furthermore, r_1 is not superior to any rule in D' (by definition of modular addition). Consequently, (2.3.1) or (2.3.2) holds for each rule s for $\neg q$ in D'.

For addition A_2

$$\begin{array}{lccc} r_2 : & & \Rightarrow & q \\ s_2 : & \neg q & \rightarrow & \neg q \end{array}$$

The only consequence of $D + A_2$ in $DL(\partial)$ related to q and $\neg q$ is $-\Delta q$, and hence this is the only consequence of $D' + A_2$ in $DL(\partial_{||})$ related to q and $\neg q$.

As we saw from A_1, (2.3.1) or (2.3.2) holds for each rule s for $\neg q$ in D'. (2.3.2) cannot apply to s_2, by definition of modular addition. If (2.3.1) applies to s_2 in $D' + A_2$, then $+\lambda\neg q \notin P_\lambda$ for $D' + A_2$. Because neither $+\Delta q$ nor $+\Delta\neg q$ appear in P_Δ for $D' + A_2$, this can only hold if every rule for $\neg q$ in $D' + A_2$ contains a body literal α such that $+\lambda\alpha \notin P_\lambda$ for $D' + A_2$, by the $+\lambda$ inference rule.

Now consider the application of the $+\partial_{||}$ inference rule to prove $+\partial_{||} q$ in $D' + A_2$. (2.1) is satisfied by r_2 and (2.2) is satisfied. For every rule for $\neg q$ in $D' + A_2$, (2.3.1) is satisfied, as shown in the previous paragraph. Hence, $+\partial_{||} q$ is a consequence of $D' + A_2$. However, this contradicts the supposed simulation. Thus, there is no D' that simulates D w.r.t. addition of rules in $DL(\partial_{||})$.

□

https://doi.org/10.1017/S1471068420000010 Published online by Cambridge University Press