On the Complexity of Temporal Defeasible Logic

Guido Governatori*
NICTA, Australia
guido.governatori@nicta.com.au

Antonino Rotolo
CIRSFID and Law Faculty,
University of Bologna
antonino.rotolo@unibo.it

Abstract

In this paper we investigate the complexity of temporal defeasible logic, and we propose an efficient algorithm to compute the extension of a temporalised defeasible theory. We motivate the logic showing how it can be used to model deadlines.

1 Introduction

Defeasible Logic (DL) (Nute 1993; Antoniou *et al.* 2001) is based on a logic programming-like language and it is a simple, efficient but flexible non-monotonic formalism capable of dealing with many different intuitions of non-monotonic reasoning. DL has a linear complexity (Maher 2001) and has also efficient implementations (Bassiliades, Antoniou, & Vlahavas 2006; Lam & Governatori 2009).

Recently, DL has been extended to capture the temporal aspects of several specific phenomena, such as legal positions (Governatori, Rotolo, & Sartor 2005) and modifications (Governatori *et al.* 2005; Governatori & Rotolo 2010), deadlines (Governatori *et al.* 2007). Although Temporal Defeasible Logic (TDL) proved to be sufficiently expressive for those purposes, and many variants of it have been proposed accordingly, no systematic investigation on the proof-theoretic and computational properties of TDL has been so far carried out. This paper is a first step in this direction. In particular, we will show that an expressive variant of TDL, able to represent, e.g., different types of deadline, is computationally feasible. We will prove that it is possible to compute the complete set of consequences of any given TDL theory in linear time, thus preserving the nice computational features of standard DL.

Section 2 describes a variant of TDL, its formal language and proof theory, and a possible application to model the concept of deadline. Section 4 investigates the complexity of this logic, and proposes an efficient algorithm to compute the extension of any temporalised defeasible theory.

2 Temporal Defeasible Logic (TDL)

The language of TDL is based on the concept of *temporalised literal*, which is an expression such as l^t (or its negation, $\neg l^t$), where l is a literal and t is an element of a discrete totally ordered set $\mathscr{T}$ of instants of time $\{t_1, t_2, \ldots\}$: l^t intuitively means that l holds at time t. Given a temporalised literal l the complement $\sim l$ is $\neg p^t$ if $l = p^t$, and p^t if $l = \neg p^t$.

A *rule* is an expression $lbl : A \hookrightarrow^x m$, where lbl is a unique label of the rule, A is a (finite, possibly empty) set of temporalised literals, $\hookrightarrow \in \{\rightarrow, \Rightarrow, \rightsquigarrow\}$, m is a temporalised literal and x is either π or τ signaling whether we have a *persistent* or *transient* rule. *Strict rules*, marked by the arrow $\rightarrow$, support indisputable conclusions whenever their antecedents, too, are indisputable. *Defeasible rules*, marked by $\Rightarrow$, can be defeated by contrary evidence. *Defeaters*, marked by $\rightsquigarrow$, cannot lead to any conclusion but are used to defeat some defeasible rules by producing evidence to the contrary. A *persistent* rule is a rule whose conclusion holds at all instants of time after the conclusion has been derived, unless interrupting events occur; *transient* rules establish the conclusion only for a specific instant of time. Thus $ex_1 : p^5 \Rightarrow^\pi q^6$ means that if p holds at 5, then q defeasibly holds at time 6 and continues to hold after 6 until some event overrides it. The rule $ex_2 : p^5 \Rightarrow^\tau q^6$ means that, if p holds at 5, then q defeasibly holds at time 6 but we do not know whether it will persist after 6.

We will use some abbreviations. Given a rule r and a set R of rules, $A(r)$ denotes the antecedent of r while $C(r)$ denotes its consequent; R^π denotes the set of persistent rules in R, and $R[\psi]$ the set of rules with consequent ψ. R_s, R_{sd} and R_{dft} are respectively the set of strict rules, the set of strict and defeasible rules, and the set of defeaters in R.

Note that we assume that defeaters are only transient: if a persistent defeasible conclusion is blocked at t by a transient defeater, such a conclusion no longer holds after t unless another applicable rule reinstates it (see below).

There are in TDL three kinds of features: facts, rules, and a superiority relation among rules. Facts are indisputable statements, represented by temporalised literals. The superiority relation ($\prec$) provides information about the relative strength of rules, i.e., about which rules can overrule which other rules. A knowledge base that consists of these items is called a TDL theory.

Definition 1. *A* TDL theory *is a structure* $(F, R, \prec)$*, where* F *is a finite set of facts,* R *is a finite set of rules and* $\prec$ *is an acyclic binary relation over* R*.*

TDL is based on a constructive inference mechanism

*NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

based on tagged conclusions. Proof tags indicate the strength and the type of conclusions. The strength depends on whether conclusions are indisputable (the tag is Δ), namely obtained by using facts and strict rules, or they are defeasible (the tag is ∂). The type depends on whether conclusions are obtained by applying a persistent or a transient rule: hence, conclusions are also tagged with π (persistent) or τ (transient).

Provability is defined below and is based on the concept of a derivation (or proof) in a TDL theory D.

Definition 2. *Given a TDL theory D, a* proof *P from D is a finite sequence of tagged temporalised literals such that:*

(1) Each tag is one of the following: $+\Delta^{\pi}$, $-\Delta^{\pi}$, $+\partial^{\pi}$, $-\partial^{\pi}$, $+\Delta^{\tau}$, $-\Delta^{\tau}$, $+\partial^{\tau}$, $-\partial^{\tau}$;

(2) The proof conditions definite provability *and* defeasible provability *given below are satisfied by the sequence P.*

Given a proof P we use $P(n)$ to denote the n-th element of the sequence, and $P[1..n]$ denotes the first n elements of P.

The meaning of the proof tags is as follows:

- $+\Delta^{\pi}p^{t_p}$ (resp. $+\Delta^{\tau}p^{t_p}$): we have a definite derivation of p holding from time t_p onwards (resp. p holds at t_p);
- $-\Delta^{\pi}p^{t_p}$ ($-\Delta^{\tau}p^{t_p}$): we can show that it is not possible to have a definite derivation of p holding from time t_p onwards (resp. p holds at t_p);
- $+\partial^{\pi}p^{t_p}$ (resp. $+\partial^{\tau}p^{t_p}$): we have a defeasible derivation of p holding from time t_p onwards (resp. p holds at t_p);
- $-\partial^{\pi}p^{t_p}$ (resp. $-\partial^{\tau}p^{t_p}$): we can show that it is not possible to have a defeasible derivation of p holding from time t_p onwards (resp. p holds at t_p).

The inference conditions for negative proof tags ($-\Delta$ and $-\partial$) are derived from the inference conditions for the corresponding positive proof tags by applying the Principle of Strong Negation introduced by (Antoniou *et al.* 2006). For space reasons, in what follows we will list only the positive version of the inference rules.

Definite Provability

If $P(n+1) = +\Delta^{x}p^{t_p}$, then
1) $p^{t_p} \in F$ if $x = \tau$; or
2) $\exists r \in R^{x}_{s}[p^{t'_p}]$ such that $\forall a^{t_a} \in A(r) : +\Delta^{y}a^{t_a} \in P[1..n]$.

where:

(a) $y \in \{\pi, \tau\}$;
(b) if $x = \pi$, then $t'_p \leq t_p$;
(c) if $x = \tau$, then $t'_p = t_p$.

If the conclusion is transient (if $x = \tau$), the above conditions are the standard ones for definite proofs in DL, which are just monotonic derivations using forward chaining. If the conclusion is persistent ($x = \pi$), p can be obtained at t_p or, by persistence, at any time t'_p before t_p. Finally, notice that facts lead to strict conclusions, but are taken not to be persistent. Consider this theory:

$$(F = \{p^{t_1}\}, R = \{p^{t_1} \rightarrow^{\pi} q^{t_1}\}, \prec = \emptyset)$$

We can derive $+\Delta^{\pi}q^{t_1}$, and $+\Delta^{\pi}q^{t}$ for $t > t_1$.

Defeasible Provability

If $P(n+1) = +\partial^{x}p^{t_p}$, then
1) $+\Delta^{x}p^{t_p} \in P[1..n]$ or
2) $-\Delta^{x}{\sim}p^{t_p} \in P[1..n]$ and
2.1) $\exists r \in R^{x}_{sd}[p^{t'_p}]$ such that
$\forall a^{t_a} \in A(r) : +\partial^{y}a^{t_a} \in P[1..n]$, and
2.2) $\forall s \in R^{y}[{\sim}p^{t_{\sim p}}]$ either
2.2.1) $\exists b^{t_b} \in A(s), -\partial^{y}b^{t_b} \in P[1..n]$ or
2.2.2) $\exists w \in R^{y}[p^{t_{\sim p}}]$ such that
$\forall c^{t_c} \in A(w) : +\partial^{y}c^{t_c} \in P[1..n]$ and $s \prec w$.

where

(i) $y \in \{\pi, \tau\}$;
(ii) if $x = \pi$, then $t'_p \leq t_{\sim p} \leq t_p$;
(iii) if $x = \tau$, then $t'_p = t_{\sim p} = t_p$.

Defeasible derivations run in three phases. In the first phase we put forward a supported reason (rule) for the conclusion we want to prove. Then in the second phase we consider all possible (actual and not) reasons against the desired conclusion. Finally in the last phase, we have to rebut all the counterarguments. This can be done in two ways: we can show that some of the premises of a counterargument do not obtain, or we can show that the argument is weaker than an argument in favour of the conclusion. If $x = \tau$, the above conditions are essentially those for defeasible derivations in DL. If $x = \pi$, a proof for p can be obtained by using a persistent rule which leads to p holding at t_p or at any time t'_p before t_p. In addition, for every instant of time between the t'_p and t_p, p should not be terminated. This requires that all possible attacks were not triggered (clause 2.2.1) or are weaker than some reasons in favour of the persistence of p (clause 2.2.2).

Consider the following theory.

$$\begin{aligned}(&F = \{a^{t_1},\ b^{t_3},\ c^{t_3},\ d^{t_4}\},\\ &R = \{r_1 :\ a^{t_1} \Rightarrow^{\pi} e^{t_1},\ r_2 :\ b^{t_3} \Rightarrow^{\pi} \neg e^{t_3},\ r_3 :\ c^{t_3} \rightsquigarrow^{\tau} e^{t_3},\\ &\qquad r_4 :\ d^{t_4} \Rightarrow^{\tau} \neg e^{t_4}\},\\ &\succ = \{r_3 \succ r_2,\ r_1 \succ r_4\})\end{aligned}$$

At time t_1, r_1 is the only applicable rule; accordingly we derive $+\partial^{\pi}e^{t_1}$. At time t_2 no rule is applicable, and the only derivation permitted is the derivation of $+\partial^{\pi}e^{t_2}$ by persistence. At time t_3 both r_2 and r_3 are applicable, but r_4 is not. If r_2 prevailed, then it would terminate e. However, it is rebutted by r_3, so we derive $+\partial^{\pi}e^{t_3}$. Finally at time t_4, rule r_4 is applicable, thus we derive $+\partial^{\tau}\neg e^{t_4}$ and $-\partial^{\pi}e^{t_4}$, which means that r_4 terminates e. Notice that, even if r_4 is weaker than r_1, the latter is not applicable at t_4, thus it does not offer any support to maintain e.

Proposition 1. *Let D be a TDL theory where the transitive closure of* $\prec$ *is acyclic. For every* $\# \in \{\Delta, \partial\}$, $x, y \in \{\pi, \tau\}$*:*

- *It is not possible that both* $D \vdash +\#^{x}p^{t}$ *and* $D \vdash -\#^{y}p^{t}$*;*
- *if* $D \vdash +\partial^{x}p^{t}$ *and* $D \vdash +\partial^{y}{\sim}p^{t}$*, then* $D \vdash +\Delta^{x}p^{t}$ *and* $D \vdash +\Delta^{y}{\sim}p^{t}$.

Proposition 1 shows the soundness of TDL: it is not possible to derive a tagged conclusion and its opposite, and that we cannot defeasibly prove both p and its complementary unless the definite part of the theory proves them; this

means that inconsistency can be derived only if the theory we started with is inconsistent, and even in this case the logic does not collapse to the trivial extensions (i.e., everything is provable).

Definition 3. *Let HB_D be the Herbrand Base for a TDL theory D. The extension of D (denoted by E^D) is the 4-tuple $(\Delta^+, \Delta^-, \partial^+, \partial^-)$, where $\#^{\pm} = \{p^t \mid p \in HB_D, D \vdash \pm\#^x p^t, t \in \mathscr{T}\}$, $\# \in \{\Delta, \partial\}$, and $x \in \{\pi, \tau\}$.*

We will refer to Δ^+ and Δ^- as the definite extension, to ∂^+ and ∂^- as the defeasible extension, to Δ^+ and ∂^+ as the positive extension, and to Δ^- and ∂^- as the negative extension.

3 Modelling Deadlines in TDL

In this section we illustrate how the logic at hand can model deadlines. An alternative analysis has been developed by (Governatori *et al.* 2007), but the logic used there was based on introducing in DL temporal intervals, thus posing complexity limitations. Here, we show that a more efficient variant of TDL can do the same job. The idea of deadline refers to the notion of obligation, which is parametrized by temporal instants. Consider the following example.

Example 1. *Customers must pay within 30 days, after receiving the invoice.*

Example 1 states an obligation for customers to pay within 30 days upon the receipt of an invoice. To keep our presentation light, we do not explicitly introduce modal operators to capture obligations (Governatori *et al.* 2007; Governatori, Rotolo, & Sartor 2005). Accordingly, an expression like OBLpay^{t_1}, meaning that it is obligatory to pay at time t_1, is logically treated here as a standard temporalised literal.

We can distinguish *achievement obligations*, like Example 1, from *maintenance obligations*, like Example 2 below. For an achievement obligation, a certain condition must occur at least once before the deadline. For maintenance obligations, a certain condition must obtain during all instants before the deadline. Consider the following example.

Example 2. *Customers must keep a positive balance, for 30 days after opening a bank account.*

In Example 2, the deadline only signals that the obligation is terminated. A violation occurs when the obliged state does not obtain at some point before the deadline.

Example 1 can be represented as follows. The deadline refers to an obligation triggered by receipt of the invoice (inv_{init}): such an obligation is persistent. After that the customer is obliged to pay. The obligation terminates only when it is complied with (inv_{term}). Note that the obligation itself may even persist after the deadline (like in Example 1). Generally, a deadline signals that a violation of the obligation has occurred (rule inv_{viol}).

inv_{init} $\quad \text{get\_invoice}^{t_1} \Rightarrow^{\pi} \text{OBLpay}^{t_1}$
inv_{term} $\quad \text{OBLpay}^{t_2}, \text{pay}^{t_2} \rightsquigarrow^{\tau} \neg\text{OBLpay}^{t_2+1}$
inv_{viol} $\quad \text{get\_invoice}^{t_1}, \text{OBLpay}^{t_1+30} \Rightarrow^{\tau} \text{viol(inv)}^{t_1+30}$

Suppose that the set of facts is $\{\text{get\_invoice}^1, \text{pay}^{20}\}$. We can derive $+\partial^{\tau}\text{get\_invoice}^1$, which makes rule inv_{init} applicable, leading to $+\partial^{\pi}\text{OBLpay}^1$, that is, an obligation to pay applies persistently. Rule inv_{term} terminates the obligation at 21. Therefore rule inv_{viol} is applicable, and we cannot derive a violation: $-\partial^{\tau}\text{viol(inv)}^{30}$.

Example 2 can be represented as follows.

pos_{init} $\quad \text{open\_account}^{t_1} \Rightarrow^{\pi} \text{OBLpositive}^{t_1}$
pos_{term} $\quad \text{open\_account}^{t_1} \rightsquigarrow^{\tau} \neg\text{OBLpositive}^{t_1+30}$
pos_{viol} $\quad \text{OBLpositive}^{t_2}, \neg\text{positive}^{t_2} \Rightarrow^{\tau} \text{viol(pos)}^{t_2}$

Notice that we may have other examples where a maintenance obligation holds indefinitely (and not only for 30 days). In those cases, the clause pos_{term} is not needed and no termination is required.

By definition, maintenance obligations do not persist after the deadline. But achievement obligations often do persist, until they are achieved. However, this is not the case for all achievement obligations.

Example 3. *A wedding cake must be delivered, before the wedding party.*

In Example 3, the obligation to deliver the cake does not persist after the deadline, since the wedding guests will have no use for it. (Of course, the couple who ordered the cake, are entitled not to pay for the cake, or even claim damages after the deadline has passed without delivery.)

wed_{init_1} $\quad \text{order}^{t_1} \Rightarrow^{\pi} \text{OBLcake}^{t_1}$
wed_{init_2} $\quad \text{wedding}^{t_2} \rightsquigarrow^{\tau} \neg\text{OBLcake}^{t_2}$
wed_{term} $\quad \text{OBLcake}^{t_3}, \text{cake}^{t_3} \rightsquigarrow^{\tau} \neg\text{OBLcake}^{t_3+1}$
wed_{viol} $\quad \text{wedding}^{t_2}, \text{OBLcake}^{t_2} \Rightarrow^{\tau} \text{viol(wed)}^{t_2}$

Note that we have here two *init* rules, which jointly state the time interval within which the cake should be delivered: wed_{init_1} produces a persistent obligation to deliver the cake, while wed_{init_2} states that after the wedding there is no longer such an obligation.

4 Computing Consequences in TDL

In this section we present an algorithm to compute the extension of a TDL theory. We show that the time complexity of the algorithm is linear to the size of the theory. Following the idea of (Maher 2001) the algorithm is based on a series of (theory) transformations that allow us (1) to assert whether a literal is provable or not (and the strength of its derivation) (2) to progressively reduce and simplify a theory.

At this point we have to make precise what we mean for two theories to be equivalent, and we formally define the notion of transformation.

Definition 4. *Two theories D and D' are* equivalent *if and only if they have the same extension, namely $D \equiv D'$ iff $E^D = E^{D'}$. Similarly $D_1 \equiv_{\Sigma} D_2$ means that D_1 and D_2 have the same consequences in the language Σ.*

Definition 5. *A* transformation *is a mapping from TDL theories to TDL theories. A transformation T is* correct *iff for all TDL theories D, $D \equiv_{\Sigma} T(D)$ (Σ is the language of D).*

The key ideas behind the approach depends on the following properties that allow us to transform theories into 'simpler' theories.

Proposition 2. *Let D be a theory in TDL. For $x, y \in \{\pi, \tau\}$: (1) If $D \vdash +\partial^x p^t$, then*

$$D \cup \{r : p_1^{t_1}, \ldots, p_n^{t_n}, p^t \Rightarrow^y q\} \equiv D \cup \{r : p_1^{t_1}, \ldots, p_n^{t_n} \Rightarrow^y q\}.$$

(2) if $D \vdash -\partial^x p^t$, *then* $D \cup \{r : p_1^{t_1}, \ldots, p_n^{t_n}, p^t \Rightarrow^y q\} \equiv D$.

The meaning of 1 in the above proposition is that once we have established that a temporalised literal is positively provable we can remove it from the body of rules without affecting the set of conclusions we can derive from the theory. Similarly 2 states that we can safely remove rules from a theory when one of the elements in the body of the rules is negatively provable.

Proposition 3. *Let D be a TDL theory. If* $r : \Rightarrow^x p^t \in R$ *and* $R[\sim p] = \emptyset$, *then* $D \vdash +\partial^x p^t$ *and* $D \vdash -\partial^x \sim p^t$.

This proposition gives us the main criterion to assess whether we can prove a literal. The result is restricted to theories where the superiority relation is empty. (Antoniou *et al.* 2001) showed that for basic defeasible logic it is always possible to transform a theory into an equivalent theory where the superiority relation is empty. In Section 4.1 we extend the result of (Antoniou *et al.* 2001) to cover TDL.

We compute the extension of a TDL theory in three phases:

1. in the first phase we remove the superiority relation by creating an equivalent theory where $\prec = \emptyset$;
2. in the second phase we use the theory obtained from the first phase to compute the definite extension;
3. in the third and final phase we use the theory from the first phase and the definite extension to generate the theory to be used to compute the defeasible extension.

4.1 Removing Superiority Relation

We are now ready to give the transformation to empty the superiority relation.

Definition 6. *Let* $D = (F, R, \prec)$ *be a TDL theory. Let* Σ *be the language of D. Define* $elimsup(D) = (F, R', \emptyset)$, *where*

$$\begin{aligned} R' = \{ & s^+_{r_1,r_2} : \neg inf^+(r_1)^0 \Rightarrow^\tau inf^+(r_2)^0, \\ & s^-_{r_1,r_2} : \neg inf^+(r_1)^0 \Rightarrow^\tau inf^-(r_2)^0 \mid r_2 \prec r_1\} \cup \\ & \{r_a : A(r) \hookrightarrow^\tau \neg inf^+(r)^0, \\ & r_c : \neg inf^+(r)^0 \hookrightarrow^x p^t \mid A(r) \hookrightarrow^x p^t \in R_{sd}[p]\} \cup \\ & \{r_a : A(r) \Rightarrow^\tau \neg inf^-(r)^0, \\ & r_c : \neg inf^-(r)^0 \rightsquigarrow^x p^t \mid A(r) \rightsquigarrow^x p^t \in R_{dft}[p]\}. \end{aligned}$$

For each r, $inf^+(r)$ *and* $inf^-(r)$ *are new atoms not in* Σ, *and so are the labels associated to the rules obtained from the transformation of the superiority relation. Furthermore all new atoms and labels are distinct.*

The transformation is essentially identical to the transformation given in (Antoniou *et al.* 2001) to remove the superiority relation. The only difference is that for r_c we use a rule of the same type as the rule r_a and r_c replace. The reason why we have both inf^+ and inf^- literals is that strict and defeasible rules can be used in all phases of a defeasible derivation, while defeaters cannot be used to support a conclusion. Thus inf^+ are to support conclusions while inf^- to prevent them.

Proposition 4. *The transformation elimsup is correct.*

Proof. The proof is essentially the same as that given in (Antoniou *et al.* 2001). □

4.2 Computing the Definite Extension

Before giving the algorithm to compute the definite extension we have to introduce some auxiliary notation to refer to the data structures needed for the algorithm.

Definition 7. *Let D be a TDL theory and* H_D *be the set of literals in D. For each* $a \in H_D$ *we have the following sets:*

- $ptimes(a) = \{t : \exists r \in R^\pi[a] \text{ and } C(r) = a^t\}$;
- $ttimes(a) = \{t : \exists r \in R^\tau[a] \text{ and } C(r) = a^t\}$;
- $times(a) = ptimes(a) \cup ttimes(a)$.

We use the same abbreviations as those of Section 2. Thus, e.g., $ptimes_s(a)$ is the set of instants associated to $R_s[a]$.

In the presentation of the algorithms we will use intervals to give a compact representation for sets of contiguous instants. We will use both proper intervals, i.e., intervals with both start and end time, and punctual intervals, i.e., intervals corresponding to singletons. We will use $[t,t']$ for a proper interval and $[t]$ for a punctual interval.

Definition 8. *Given an interval I we say that* $t^* \in I$ *iff (1) if* $I = [t,t']$, $t < t'$ *and* $t \le t^* < t'$ *or (2) if* $I = [t]$ *and* $t^* = t$.

We adopt a compact but isomorphic representation for the extensions, namely, elements of extensions are now pairs (l,I) were l is a literal and I is an interval; thus (l,I) corresponds to the set of temporalised literals l^t such that $t \in I$.

In the presentation of the algorithms, given a set S to be manipulated in a cycle of the computation, we use S' to denote the set S after the computation of the cycle.

Algorithm: *ComputeDefinite*(D)
Input: a TDL theory D
Output: a theory D' and the definite extension of D
ComputeDefinte(D):
while $H_D \neq \emptyset$ or $R_s \neq \emptyset$ or $H'_D \neq H_D$ or $(\Delta^+)' \neq \Delta^+$ or $R' \neq R$
  for each literal $a \in H_D$:
    if $R_s[a] = \emptyset$, **then** $\Delta^- = \Delta^- \cup \{(a,[0,\infty])\}$
    if $R_s[a] = \emptyset$ **and** $R_s[\sim a] = \emptyset$, **then** $H_D = H_D - \{a, \sim a\}$
  for each rule $r \in R_s$
    if $l^t \in A(r)$, $(l,I) \in \Delta^+$ **and** $t \in I$, **then** $A(r) = A(r) - \{l^t\}$
    let $S = \{s \in R_s : \exists a^{t'} \in A(s) : \ t' < \min(ptimes_s(a))$ and $t' \notin ttimes_s(a)\}$
      $R_s = R_s - S$
    if $A(r) = \emptyset$ **and** $C(r) = a^t$, **then**
      if $r \in R^\pi$, **then**
        $\Delta^+ = \Delta^+ \cup \{(a,[t,\infty])\}$
        $R_s = (R_s - \{s \in R_s : a^{t'} = C(s), t' > t\}) - \{r\}$
      if $r \in R^\tau$, **then**
        $\Delta^+ = \Delta^+ \cup \{(a,[t])\}$
        $R_s = (R_s - \{s \in R_s : a^t = C(s)\}) - \{r\}$
for each literal $a \in H_D$
  let $t_p = \min\{\min\{t : (a,[t,\infty]) \in \Delta^+\}, \infty\}$
  until $times(a) = \emptyset$ or $\min(times(a)) > t_p$
    let $t_d = \max\{\max\{t : (a,[t',t]) \in \Delta^-\}, 0\}$
      $\Delta^- = \Delta^- \cup \{(a,[t_d + 1, \min(times(a))])\}$
      $times(a) = times(a) - \{\min(times(a))\}$

At each cycle the algorithm *ComputeDefinite* scans the set of literals in search of temporalised literals for which there are no rules for them. This happens in two cases: (i) there are no rules for a temporalised literal or (ii) all the persistent rules for the literal have a greater time. For each of such temporalised literals *ComputeDefinite* adds them to the negative definite extension of the theory, and removes all rules where at least one of these literals occurs.

Then *ComputeDefinite* scans the set of rules in search of rules with an empty body. In case of a positive match the algorithm adds the conclusion of the rule to the positive definite extension (with an open ended interval for a persistent rule and with a punctual interval otherwise). Finally the algorithm removes such temporalised literals matching the newly added conclusions from the body of rules.

We repeat the cycle until (1) there are no more literals to be examined, or (2) the set of strict rules is empty, or (3) no addition to the extension happened in the cycle.

Proposition 5. *ComputeDefinite is correct.*

Proof. Propositions 2 and 3 can be proved for definite conclusions and strict rules. Then, the correctness of *ComputeDefinite* follows immediately. □

4.3 Computing the Defeasible Extension

We are now ready to give the algorithm to compute the defeasible extension of a theory. We first give some subroutines corresponding to theory transformation to be used in the main algorithm.

The first algorithm we consider is concerned with literals to be tagged with $-\partial$.

Algorithm: *discard*(l,I)
Input: a literal l and an interval I
discard(l,I):
  $\partial^- = \partial^- \cup \{(l,I)\}$
  let $S = \{s : l^t \in A(s), t \in I\}$
    $R = R - S$
  persistence(S)

The algorithm *discard* adds a literal to the negative defeasible extension and then removes rules for which we have already proved that some literal in the antecedent of the rules is not provable. The literal is parametrised by an interval. This means that the operation is performed for all instances of the literal temporalised with an instant in the interval. The transformation corresponding to it is justified by Proposition 2.2. The algorithm further calls the subroutine *persistence* that updates the state of the extension of a theory.

The next algorithm concerns defeasible provable literals.

Algorithm: *proved*(l,r,I)
Input: a literal l, a rule r, and an interval I
proved(l,r,I):
  $\partial^+ = \partial^+ \cup \{(l,I)\}$
  discard$(\sim l, I)$
  for each $s \in R$, **if** $l^t \in A(s)$ **and** $t \in I$, **then** $A(s) = A(s) - \{l^t\}$
  $R = R - \{r\}$

As a first step the algorithm inserts a provable literal in the positive defeasible extension of the theory. Then *proved* calls *discard* with the complementary literal. The next step is to remove all the instances of the literal temporalised with an instant in the interval I from the body of rules. Finally we remove the rule for the set of rules. The transformations implemented by this algorithm are correct and are justified by Propositions 2.1, and 3.

Algorithm: *persistence*(S)
Input: a set of rule S
persistence(S):
  for each $(l,[t,t']) \in \partial^+$
    if $s \in S$ **and** $C(s) = \sim l^{t'}$, **then**
      if $t^* = \min(\{k \in times(\sim l), k > t'\})$, **then**
        replace $(l,[t,t'])$ in ∂^+ with $(l,[t,t^*])$
        proved$(l,\emptyset,[t',t^*])$

The above algorithm updates the state of literals in the extension of a theory after we have removed rules we know cannot longer be fired (i.e., at least one literal in the antecedent of the rule is provable with $-\partial^x$). As we have seen in Section 2 a conclusion proved using a persistent rule persists until it is terminated by another (applicable) rule for its complementary. Thus an entry $(l,[t,t'])$ in ∂^+ means that l holds from t to $t'-1$. This means that there is a rule for $\sim l^{t'-1}$. When we insert $(l,[t,t'])$ in ∂^+ we do not know if the rule for $\sim l^{t'-1}$ is applicable or not. The set S passed as parameter to the algorithm is the set of rules we have discovered to be no longer applicable. At this point we can update the entry for l in ∂^+, and we can set it to t'', where t'' is the next instant for which we have a rule for $\sim l$. Consider, for example, a theory where the rules for p and $\neg p$ are: $r: \Rightarrow^\pi p^1$, $s: q^5 \Rightarrow^\tau \neg p^{10}$, $v: \Rightarrow^\pi \neg p^{15}$. In this theory we can prove $+\partial^\pi p^t$ for $1 \le t < 10$, no matter whether q is provable or not at 5. Suppose that we discover that $-\partial^x q^5$. Then we have to remove rule s. In the resulting theory from this transformation can prove $+\partial^\pi p^t$ for $1 \le p < 15$. Thus we can update the entry for l from $(l,[1,10])$ to $(l,[1,15])$.

Algorithm: *ComputeDefeasible*(D)
Input: a TDL theory D:
Output: a theory D' and the defeasible extension of D.
ComputeDefeasible(D):
while $H_D \neq \emptyset$ or $R \neq \emptyset$ or $H'_D \neq H_D$ or $(\partial^\pm)' \neq \partial^\pm$ or $R' \neq R$
  for each literal $a \in H_D$:
    1) **if** $R_{sd}[a] = \emptyset$, **then**
      $\partial^- = \partial^- \cup \{(a,[0,\infty])\}$
      $R = R - \{r : a^t \in A(r)\}$ for any t.
    2) **if** $R_{sd}[a] = \emptyset$ **and** $R_{sd}[\sim a] = \emptyset$, **then** $H_D = H_D - \{a, \sim a\}$
  for each rule $r \in R$
    3) **let** $S = \{s \in R : \exists a^{t'} \in A(s) : \ t' < \min(ptimes(a))$ and $t' \notin ttimes(a)\}$
      $R = R - S$
      persistence(S)
    4) **if** $A(r) = \emptyset$ **and** $C(r) = a^t$, **then**
      if $r \in R^\tau$, **then**
        if $t \notin times(\neg a)$ **and** $r \notin R_{dft}$, **then** *proved*$(a,r,[t])$
        else *discard*$(\sim a,[t])$
      if $r \in R^\pi$, **then**
        if $R[\sim a] = \emptyset$, **then** *proved*$(a,r,[t,\infty])$
        else
          if $t \in times(\sim a)$, **then** *discard*$(\sim a,[t])$
          else
            let $t^* = \min(\{k \in times(\sim a)\} \cup \{\infty\})$
              proved$(a,r,[t,t^*])$

Steps 1)–3) are essentially the same as the first three steps in *ComputeDefinite* (with the difference that when we eliminate a rule we update the state of the extension instead of waiting to the end as we did for definite extensions).

At step 4) we search for rules with empty body. Suppose we have one of such rules, let us say a rule for l^t. No matter what type of rule we have, we know that the complementary of l, i.e., $\sim l$, cannot be proved at t. So we add $(\sim l,[t])$ to ∂^-. At this stage we still have to determine whether we can insert l in ∂^+ and the instant/interval associated to it. We

have a few cases. The rule is a defeater. Defeaters cannot be used to prove conclusions, so in this case, we are done. If the rule is transient, then it can prove the conclusion only at t, and we have to see if there are transient rules for $\sim l^t$ or persistent rules for $\sim l^{t'}$ such that $t' \leq t$. If there are we have to wait to see if we can discard such rules. Otherwise, we can add $(l,[t])$ to ∂^+. Finally, in the last case the rule is persistent. What we have to do in this case is to search for the minimum time greater or equal to t in the rules for $\sim l$, and we can include $(l,[t,t'])$ in ∂^+.

Proposition 6. *Let $D = (\emptyset, R, \emptyset)$ be a TDL theory such that there is no strict rule with empty body in R; then the transformation ComputeDefeasible(D) is correct.*

Proof. Since there are no facts and no strict rules with empty body, strict rules behave exactly as defeasible rules. Then the proposition follows from Propositions 3 and 4. □

4.4 Computing the Extension

To compute the full extension of a TDL theory D we use the following series of transformation:

1. $D_\prec = elimsup(D)$;
2. Let D_s, Δ_s^+, Δ_s^- be, respectively, the theory, the positive definite extension, and the negative definite extension obtained from *ComputeDefinite*$(D_\prec)$;[1]
3. $D' = (\emptyset, R_\prec - \{r : a^t \in A(r), (a,I) \in \Delta^+, \text{ and } t \in I\}, \emptyset)$. Let $\partial^+ = \Delta_s^+$. The defeasible extension is obtained from *ComputeDefeasible*(D').

We call the transformation *ComputeExtension*.

Theorem 7. *ComputeExtension is correct.*

Proof. The correctness follows from the correctness of the three steps (Propositions 4, 5 and 6). □

Theorem 8. *Given a TDL theory D where $\prec = \emptyset$, the extension of D can be computed in linear time, i.e., $O(|R| * |H_D| * |\mathscr{T}_D|)$, where $\mathscr{T}_D$ is the set of distinct instants in D.*

Proof. For *ComputeDefinite* and *ComputeDefeasible* we have that for each literal a the set $R[a]$ can be implemented as an hash table with pointers to the rules, where each rule is implemented as an ordered list of pairs. The sets of *ptimes* and *ttimes* can be constructed as indexes. This means that the information stored in them can be accessed (in the form required by the algorithms *ComputeDefinite* and *ComputeDefeasible*) in linear time.

The algorithms *ComputeDefinite* and *ComputeDefeasible* alternate linear scans of the sets of literals and the set of rules. During the scan of the literals they identify rules that can be removed from the set of rules; during the scan of the rules they identify literals that can be removed at the next scan of the literals. This means that the number of times we have to scan the set of rules is bounded by the number of literals.

The final step of *ComputeDefinite* is to populate the negative definite extension. The number of times we have to repeat this cycle for a literal is bounded by the number of times the literal appears as head of a rule with a distinct instant, which is then bounded by the number of rules for the literal. Thus the complexity of *ComputeDefinite* is $O(|R| * |H_D|)$.

Every time we have to remove a rule in the algorithm *ComputeDefeasible*, we call *peristence* to update the extension. For each cycle the number of time we call the procedure is bounded by the number of instants such that there is a particular literal with that instant in the head of a rule.

Hence, the complexity of *ComputeExtension* is given by $O(|R| * |H_D| * |\mathscr{T}_D|)$. □

Notice that the *elimsup* produces a theory D' where $|R'| = O(|R| + |\prec|)$ and $H_{D'} = O(|H_D| + |R|)$, thus the result of Theorem 8 gives us $O((|R| + |\prec|) * (|H_D| + |R|) * |\mathscr{T}_D|)$, and a resulting polynomial (quadratic) complexity, for defeasible theories where the superiority is not empty. However, it is possible to use the argument of (Maher 2001) to show that the number of cycles we have to perform is bounded by the number of instances of literals occurring in the theory multiplied by the instant of times explicitly occurring in the theory.

5 Discussion and Implementation

TDL is an extension of basic defeasible logic (Antoniou *et al.* 2001) for which (Maher 2001) proved that the complexity is linear. The extension is twofold: on the syntactic side, literals are labelled with timestamps, on the conceptual side TDL introduces persistent and transient conclusions. The idea of persistent conclusions is that once a conclusion has been classified as persistent then it continues to hold until there are some reasons to terminate it. From a computational point of view, we can propagate persistent conclusions from one instant to the successive instant unless there are some reasons that prevent the propagation. Based on this intuition, if we restrict the language to rules with the form $a_1^{t_1}, \ldots, a_n^{t_n} \Rightarrow b^t$, such that $\max(\{t_1, \ldots, t_n\}) \leq t$, then we can devise the following procedure (Governatori & Terenziani 2007) to compute the extension of a theory.

- At time 0, consider the sub-theory restricted to the rules whose consequent is labelled by 0. Then use the algorithms given in (Maher 2001) to compute the extension of the sub-theory at time 0.
- At time $n+1$, consider the extension at time n. Then for each positive conclusion (i.e., conclusion whose proof tag is $+\partial$) p_i:
  - introduce a rule $r^n_{p_i} : \Rightarrow^\tau p_i$
  - introduce an instance of the superiority relation $r^n_{p_i} \prec s$ for each s such that $C(s) = \sim p_i^{n+1}$;
  - remove p_i^n from the body of rules where it occurs;

  For each negative conclusion q_j remove rules where q_j appears in the body. Compute the extension for the sub-theory restricted to the rules whose consequent is labelled with $n+1$.

It is immediate to see that the above theory compute the extension of a theory D in the interval $[0,t]$ in $O(|D| * t)$-time (where $|D|$ is the size of the theory, i.e., the number of instances of literals occuring in it). It is clear that the above procedure applies an incremental swap over the timeline, and the problem with it is that it cannot look backward.

[1] In this step we also remove facts, by replacing each fact a^t with the rule $\rightarrow^\tau a^t$.

Therefore the major limitation of it is that it cannot handle rules where the time of the conclusion precedes the time of some of its antecedent. Unfortunately, we believe that the restriction the above procedure relies upon is a very strong one. It excludes some important application areas, for example, among other, legal reasoning where retroactivity is not unusual and diagnosis where rules typically have the format excluded by the restriction that the conclusion should not precedes the antecedents.

An efficient Java implementation of TDL based on the algorithms presented in this paper is discussed in (Rubino & Rotolo 2009; Rubino 2009). Initial experiments with the implementation confirm the linearity (and scalability) of the approach. The implementation has been tested on both synthetic theories (designed to test particular features of TDL) and concrete theories obtained from real life scenarios.

On the synthetic side, TDL and the proposed algorithms can account for compact encodings and efficient computations. For example consider the theories:

$$T_1 : \Rightarrow^{\pi} p^0 \qquad T_2 : \Rightarrow^{\tau} p^0, p^i \Rightarrow^{\tau} p^{i+1} \text{ for } 0 \leq i \leq 99$$

The two theories are equivalent (i.e., they generate the same extension) in the time interval $[0,100]$, but, trivially, our algorithms compute the extension much more quickly when given the first theory as input than when the second theory is used as input. On the practical side, of particular interest is the formalisation in TDL of the Road Traffic Restriction Regulation of the Italian town of Piacenza (Rubino 2009; Governatori, Rotolo, & Rubino 2010).

6 Summary and Related Work

We described an expressive variant of TDL. We showed that it can be used to capture many aspects of the concept of deadline. Despite its expressiveness, this logic is computationally feasible: we proposed an algorithm to compute conclusions in TDL and proved that the time complexity of the algorithm is linear to the size of the theory.

Typically there are two mainstream approaches to reasoning with and about time. A point based approach, as in the present paper, and an interval based approach (Allen 1984). Notice that that the current approach is able to deal with constituents holding in an interval of time: an expression $\Rightarrow a^{[t_1,t_2]}$ meaning that a holds between t_1 and t_2 can just be seen as a shorthand of the pair of rules $\Rightarrow^{\pi} a^{t_1}$ and $\rightsquigarrow^{\tau} \neg a^{t_2}$.

Non-monotonicity and temporal persistence are covered by a number of different formalisms, some of which are quite popular and mostly based on variants of Event Calculus or Situation Calculus combined with non-monotonic logics (see, e.g., (Shanahan 1997; Turner 1997)). TDL has some advantages over many of them. While TDL is able to cover many different aspects, it is possible in TDL to compute the set of consequences of any given theory in linear time to the size of the theory. To the best of our knowledge, no logic with the same coverage of TDL is so efficient.

Anyway, we would like to point out that interval and duration based defeasible reasoning has been developed by (Augusto & Simari 2001; Governatori & Terenziani 2007). (Governatori & Terenziani 2007) focus on duration and periodicity and relationships with various forms of causality. (Augusto & Simari 2001) proposed a sophisticated interaction of defeasible reasoning and standard temporal reasoning (i.e., mutual relationships of intervals and constraints on the combination of intervals). In both cases no complexity results were presented, but it seems that those systems cannot enjoy the same nice computational properties of the logic studied here, since both are based on complex temporal structures.

References

Allen, J. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23:123–154.

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

Antoniou, G.; Billington, D.; Governatori, G.; and Maher, M. J. 2006. Embedding defeasible logic into logic programming. *Theory and Practice of Logic Programming* 6:703–735.

Augusto, J., and Simari, G. 2001. Temporal defeasible reasoning. *Knowledge and Information Systems* 3:287–318.

Bassiliades, N.; Antoniou, G.; and Vlahavas, I. 2006. A defeasible logic reasoner for the Semantic Web. *International Journal on Semantic Web and Information Systems* 2:1–41.

Governatori, G., and Rotolo, A. 2010. Changing legal systems: Legal abrogations and annulments in defeasible logic. *Logic Journal of IGPL* 18(1):157–194.

Governatori, G., and Terenziani, P. 2007. Temporal extensions to defeasible logic. In *Proc. Australian AI 2007*, 476–485. Springer.

Governatori, G.; Palmirani, M.; Riveret, R.; Rotolo, A.; and Sartor, G. 2005. Norm modifications in defeasible logic. In *JURIX 2005*. Amsterdam: IOS Press. 13–22.

Governatori, G.; Hulstijn, J.; Riveret, R.; and Rotolo, A. 2007. Characterising deadlines in temporal modal defeasible logic. In *Proc. Australian AI 2007*, 486–496.

Governatori, G.; Rotolo, A.; and Rubino, R. 2010. Implementing temporal defeasible logic for modeling legal reasoning. In *Proc. JSAI-isAI 2009 Workshop*, LNAI. Springer.

Governatori, G.; Rotolo, A.; and Sartor, G. 2005. Temporalised normative positions in defeasible logic. In *ICAIL'05*, 25–34. ACM Press.

Lam, H., and Governatori, G. 2009. The making of SPINdle. In *Proc. RuleML 2009*, LNCS, 315–322. Springer.

Maher, M. J. 2001. Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming* 1:691–711.

Nute, D. 1993. Defeasible logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3. Oxford University Press. 353–395.

Rubino, R., and Rotolo, A. 2009. A java implementation of temporal defeasible logic. In *Proc. RuleML 2009*, LNCS, 298–305. Springer.

Rubino, R. 2009. *Una implementazione della logica defeasible temporale per il ragionamento giuridico*. Ph.D. Dissertation, CIRSFID, University of Bologna.

Shanahan, M. 1997. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. Cambridge, MA: MIT Press.

Turner, H. 1997. Representing actions in logic programs and default theories: A situation calculus approach. *Journal of Logic Programming* 31(1-3):245–298.