Temporally Annotated Extended Logic Programs

Anastasia Analyti
Institute of Computer Science, FORTH-ICS,
Heraklion, Greece
analyti@ics.forth.gr

Ioannis Pachoulakis
Dept. of Applied Informatics & Multimedia, TEI of Crete,
Heraklion, Greece
ip@epp.teicrete.gr

***Abstract*— Extended logic programs (ELPs) are a set of logic rules with strong negation ¬ allowed in the bodies or head of the rules and weak negation ~ allowed in the bodies of the rules. ELPs enable for various forms of reasoning that cannot be achieved by definite logic programs. Answer Set Programming provides a widely acceptable semantics for ELPs. However, ELPs do not provide information regarding the temporal intervals that derived ELP literals or weakly negated ELP literals are valid. In this paper, we associate ELP rules with their validity temporal interval, resulting in a *temporally annotated logic program*. A ground temporal literal has the form *L*:*i*, where *L* is a ground ELP literal or weakly negated ELP literal and *i* is a temporal interval. We define (simple) entailment and maximal entailment of a ground temporal literal *L*:*i* from a temporally annotated logic program *C*. Both kinds of entailment are based on Answer Set Programming. Additionally, we provide an algorithm that for an ELP literal or a weakly negated ELP literal *L* returns a list with all temporal intervals *i* such that a temporally annotated logic program *C* maximally entails *L*:*i*. Based on this algorithm, the answer of various kinds of temporal queries can be provided.**

Keyword*: *Extended logic programs, validity temporal intervals, temporal inference, query answering

I. INTRODUCTION

Extended logic programs (ELPs) are a set of logic rules with strong negation ¬ allowed in the bodies or head of the rules and weak negation ~ allowed in the bodies of the rules [1]. ELP rules are assumed to be valid at the time of their evaluation but no historical information is derived as it is not known if these rules were valid in the past. However, each ELP rule is usually valid at a certain interval of time. In this paper, we associate ELP rules with their validity temporal interval. Thus, derived ELP literals and weakly negated ELP literals are associated with the temporal intervals at which they are valid. A temporal interval has the form $[t_1,t_2]$, where t_1,t_2 are time points s.t. $t_1 \leq t_2$ and it includes all time points t s.t. $t_1 \leq t \leq t_2$ In this paper, time points are years but they can also be dates, date times, etc. In general, we assume that the set of time points is a discrete linearly ordered domain that can be mapped to the set of integers through a bijective mapping.

In particular, ELP logic rules, associated with their validity temporal interval, form a *temporally annotated logic program*. A ground temporal literal has the form *L*:*i*, where *L* is a ground ELP literal or weakly negated ELP literal and *i* is a temporal interval. We define (*simple*) *entailment* of a ground temporal literal *L*:*i* from a temporally annotated logic program *C* expressing that according to *C*, *L* is true during all time points within *i*. We define *maximal entailment* of a ground temporal literal *L*:*i* from a temporally annotated logic program *C*, expressing that according to *C*, *L* is true during all time points within *i* but *L* is not true at the time point before *i* and at the time point after *i*. Both kinds of entailment are based on Answer Set Programming [1]. The complexities of simple and maximal entailment of ground temporal literals are provided showing that simple temporal entailment is not harder than entailment of a ground ELP literal from an ELP logic program, based on Answer Set Programming.

We provide an algorithm that for an ELP literal or a weakly negated ELP literal *L* returns a list with all temporal intervals *i* such that a temporally annotated logic program *C* maximally entails *L*:*i*. Based on this algorithm, the answer of various kinds of temporal queries can be provided. For example, the user may request the maximal temporal intervals that a number of ELP literals or weakly negated ELP literals hold concurrently within a temporal interval of interest. Additionally, the user may request the maximal temporal intervals that a number of ELP literals or weakly negated ELP literals hold, provided that these intervals are associated with complex relations concerning their duration, start and end points. In particular, we define four types of simple temporal queries. Additionally, we define a complex temporal query as a conjunction of simple temporal queries and a *filter* condition that provides for various kinds of checks regarding the duration, start and end points of the temporal intervals returned by the query. The filter condition can express any combination of Allen's interval algebra relations [2].

To the best of our knowledge, there is no work concerning entailment from ELP rules associated with their validity temporal intervals. The rest of the paper is organized as follows: In Section II, we define temporally annotated logic programs and entailment of temporal literals. Additionally, we provide an algorithm that for an ELP literal or weakly negated ELP literal *L* returns a list with all temporal intervals *i* such that a temporally annotated logic program *C* maximally entails

L:*i*. In Section III, we define various kinds of temporal queries and their answers. In Section IV, we present related work. Finally, Section V concludes the paper.

II. Temporally Annotated Logic Programs & Entailment

In this section, we define temporally annotated logic programs and entailment of temporal literals.

We consider a vocabulary *V*=<*Pred*,*Const*>, where *Pred* is a set of predicate symbols and *Const* is a set of constants. We consider a set of variable symbols *Var*. Variables are preceded by the question mark symbol "?". Additionally, we consider a maximal temporal interval [t_{min}, t_{max}], within which all temporal inferences are made.

Definition 1 [temporally annotated logic rule]
A *temporally annotated logic rule* over a vocabulary *V* is an ELP rule over *V* associated with a temporal interval *i*, i.e. it has the form *i*:*r* where *r* is an ELP rule and *i*=[t_1,t_2], where $t_1 \leq t_{min}$ and $t_2 \leq t_{tmax.}$

Definition 2 [temporally annotated logic program]
A *temporally annotated logic program C* over a vocabulary *V* is a set of temporally annotated logic rules over *V*.

Example 1 An example temporally annotated logic program *C* over *V*=<*Pred*, *Const*>, where *Pred* are all predicates appearing in *C* and *Const* are all the constants appearing in *C*, is the following (we consider t_{min}=1988 and t_{max}=2012):

[1990,1994]: *has_job*(*Mary*, *Hairdresser*).
[1995,2002]: *has_job*(*Mary*, *Secretary*).
[2006,2009]: *has_job*(*Mary*, *Hairdresser*).
[2001,2003]: *has_job*(*Peter*, *Garbage_collector*).
[2005,2008]: *has_job*(*Peter*, *Builder*).
[1980,1992]: *heavy_job*(*Hairdresser*).
[1980,2000]: *heavy_job*(*Garbage_collector*).
[1999,2012]: *heavy_job*(*Builder*).
[1988, 1991]: *vacation_days*(?*x*,29) ← *has_job*(?*x*,?*y*), *heavy_job*(?*y*).
[1992, 2012]: *vacation_days*(?*x*,27) ← *has_job*(?*x*,?*y*), *heavy_job*(?*y*).
[1988, 1991]: *vacation_days*(?*x*,25) ← *has_job*(?*x*,?*y*), ~ *heavy_job*(?*y*).
[1992, 2012]: *vacation_days*(?x,22) ← *has_job*(?*x*,?*y*), ~ *heavy_job*(?*y*).

Predicate *has_job*(*x*,*y*) expresses that *x* has as job *y*. Predicate *heavy_job*(*x*) expresses that *x* is a heavy and unhealthy job. Predicate *vacation_days*(*x*,*y*) expresses that *x* is entitled to *y* vacations days per year.

Convention: In the sequel, by *C*, we will denote a temporally annotated logic program and, by *V*=<*Pred*,*Const*>, we will denote the vocabulary of *C*.

Below, we define the temporal projection of *C* at a certain time point *t*.

Definition 3 [temporal projection]
The temporal projection of *C* at a time point *t* is the extended logic program $C(t)=\{r \mid i{:}r \in C \text{ and } t \in i\}$.

Example 2 Consider the temporally annotated logic program *C* of Example 1. Then, *C*(1994) is the following extended logic program:

has_job(*Mary*, *Hairdresser*).
heavy_job(*Garbage_collector*).
vacation_days(?*x*,27) ← *has_job*(?*x*,?y), *heavy_job*(?*y*).
vacation_days(?*x*,22) ← *has_job*(?*x*,?*y*), ~ *heavy_job*(?*y*).

A temporal literal over *V* has the form *L*:*i*, where *L* is an ELP literal or a weakly negated ELP literal over *V* and *i* a temporal interval or variable, called *temporal variable*.

Let *P* be an extended logic program of a vocabulary *V*=<*Pred*,*Const*> and *L* be an ELP literal or a weakly negated ELP literal over *V*, we write $P \models^{ASP} L$ if the grounded version of *P* over the constants in *Const* entails *L* under Answer Set Programming [1].

Definition 4 [temporal entailment]
We say that *C* (*simply*) *entails* a ground temporal literal *L*:*i*, denoted by $C \models L{:}i$, if for all $t \in i$, $C(t) \models^{ASP} L$. We say that *C* *maximally entails* a temporal literal *L*:[t_1,t_2], denoted by $C \models_{\max} L{:}[t_1,t_2]$, if

1. $C \models L{:}[t_1,t_2]$,
2. if $t_1 > t_{min}$ then $C(t_1\text{-}1) \not\models^{ASP} L$, and
3. if $t_2 < t_{max}$ then $C(t_2\text{+}1) \not\models^{ASP} L$.

Example 3 Consider the temporally annotated logic program *C* of Example 1. Then, $C \models$ *vacation_days*(*Mary*,22):[2006,2008] and $C \models_{\max}$ *vacation_days*(*Mary*,22):[2006, 2009]. Additionally, $C \models$ *vacation_days*(*Peter*,27):[2006, 2007] and $C \models_{\max}$ *vacation_days*(*Peter*,27): [2005,2008].

Proposition 1 Let *L*:*i* be a ground temporal literal over *V*. Deciding if $C \models L{:}i$ is:

1. co-NP-complete, in the case that *C* does not contain variables or the number of variables of each rule of *C* is less than a constant.
2. co-NEXPTIME-complete, in the general case.

Note that entailment of a ground ELP literal from an ELP program *P*, under Answer Set Programming is (i) co-NP-complete, in the case that *P* does not contain variables or the number of variables of each rule of *P* is less than a constant

and (ii) co-NEXPTIME-complete, in the general case [3]. Therefore, entailment of a ground temporal literal from a temporally annotated logic program does not increase the computational complexity over Answer Set Programming.

Now, we provide a few definitions. Let $i=[t_1, t_2]$ be a temporal interval. We define $start(i)=t_1$ and $end(i)=t_2$. Let L be an ELP literal or a weakly negated ELP literal, we denote by $pred(L)$, the predicate appearing in L. Let r be an ELP rule. We denote by $Head(r)$, the head of r.

Below, we present the algorithm *FindMaximalIntervals*(C, L) that, for an ELP literal or a weakly negated ELP literal L, returns a list with all temporal intervals i such that $C \models_{\max} L{:}i$. This algorithm calls the algorithm *GetIntervals*(C,p) which returns a list of the maximal temporal intervals i that define predicate p, i.e. for all $t \in i$, p appears in the head of a rule of $C(t)$. The list of returned temporal intervals i is sorted by $start(i)$.

Algorithm 1 *GetIntervals*(C,p), where $p \in Pred$, first gets all intervals i s.t. there exist rule $i{:}r \in C$ with $pred(Head(r)) = p$. Then, it orders these intervals i based on $start(i)$ and puts them in a list IL. Afterwards, it fetches intervals from IL in the stored order and combines the intervals that overlap or are consecutive, creating maximal intervals that puts them in a new list IL'. Finally, it returns IL'.

Algorithm 1 *GetIntervals*(C,p)
Input: a temporally annotated logic program C and a predicate $p \in Pred$
Output: a sorted list of the maximal temporal intervals i that define p

Let $S=\{i \mid i{:}r \in C$ and $p=pred(Head(r))\}$.
If $S=\{\}$ then return(*empty list*).
Order intervals $i \in S$ by $start(i)$ and put them in list IL.
Let IL' be the empty list.
Get first interval i from IL.
$current\_start=start(i)$.
$current\_end=end(i)$.
For each interval i taken in sequence from the list IL do
  If $start(i) \leq current\_end+1$ and $end(i)>current\_end$ then
    $current\_end=end(i)$.
  If $start(i) > current\_end+1$ or i is the last interval in IL then
    Add at the end of IL' the temporal interval [$current\_start$, $current\_end$].
    $current\_start=start(i)$.
    $current\_end=end(i)$.
  If $start(i) > current\_end+1$ and i is the last interval in IL then
    Add at the end of IL' the temporal interval [$start(i)$, $end(i)$].
EndFor
return(IL').

Example 4 Consider the temporally annotated logic program C of Example 1. Then, *GetIntervals*(C, *has_job*) returns the list [[1990,2003], [2005,2009]].

Algorithm 2 *FindMaximalIntervals*(C,L), where L is an ELP literal or a weakly negated ELP literal over V, first calls algorithm *GetIntervals*(C, $pred(L)$) which returns a list IL of maximal intervals i that define $pred(L)$, ordered by $start(i)$ (line 2). Assume that IL is the empty list (lines 3-7). If L is a weakly negated literal then the algorithm returns the list containing $[t_{min}, t_{max}]$. This is because in this case $C \models_{\max} L{:}[t_{min}, t_{max}]$. Otherwise, the algorithm returns the empty list. This is because in this case, there is no interval i s.t $C \models_{\max} L{:}i$. If IL is not the empty list then the algorithm fetches the first interval i of IL and if L is a weakly negated ELP literal and $start(i) \neq t_{min}$ then it puts the interval $[t_{min}, start(i)\text{-}1]$ in a new list IL' (line 11). This is because for all time points $t \in [t_{min}, start(i)\text{-}1]$, it holds that $C(t) \models^{ASP} L$. Then, it fetches each interval $[t_s, t_e]$ from IL in the stored order.

Afterwards, it examines each time point $t \in [t_s, t_e]$ and creates maximal temporal intervals i within $[t_s, t_e]$ such that for each $t' \in i$, it holds that $C(t) \models^{ASP} L$ (lines 13-21). These intervals are stored in order in IL' with the difference that if the first of these intervals is consecutive with the previous interval in IL' then it combines these. Note that if $[t_s, t_e]$ is the last temporal interval in IL, L is a weakly negated literal, and $t_e < t_{max}$ then, for each $t \in [t_e+1, t_{max}]$, it holds that $C(t) \models^{ASP} L$. Thus, if $C(t_e) \models^{ASP} L$ then it combines interval $[t_e+1, t_{max}]$ with the last interval in IL' (line 25), otherwise it adds interval $[t_e+1, t_{max}]$ to the end of IL' (line 27). If $[t_s, t_e]$ is not the last temporal interval in IL and L is a weakly negated literal then (i) if $C(t_e) \models^{ASP} L$ and $[t'_s, t_e]$ is the last interval in IL' then it replaces it by the interval $[t'_s, t''_s\text{-}1]$ (line 31) and (ii) if $C(t_e) \not\models^{ASP} L$ then it adds $[t_e+1, t''_s\text{-}1]$ to IL' (line 33), where $[t''_s, t''_e]$ is the next to $[t_s, t_e]$ interval in IL. This, is because for all $t \in [t_e+1, t''_s\text{-}1]$, it holds that $C(t) \models^{ASP} L$. Then, this process continues until all intervals $[t_s, t_e]$ from IL are fetched. In the latter case, the temporal interval list IL' is returned.

Algorithm 2 *FindMaximalIntervals*(C,L)
Input: a temporally annotated logic program C and an ELP literal or a weakly ELP literal L over V
Output: a sorted list of maximal temporal intervals i s.t. for each $t \in i$, $C(t) \models^{ASP} L$

(1) $flag$=TRUE.
(2) IL= *GetIntervals*(C, $pred(L)$).
(3) If IL is the empty list then
(4) If L is a weakly negated ELP literal then
(5) return(*a list containing* $[t_{min},t_{max}]$).
(6) else
(7) return(*empty list*).
(8) Let IL' be the empty list.
(9) Let i be the first item in IL.
(10) If L is a weakly negated ELP literal and

start(i) $\neq t_{min}$ then
(11) Add $[t_{min}$, *start*(i)-1] to the end of *IL'*.
(12) For each interval $[t_s,t_e]$ taken in sequence from the list *IL* do
(13) For each time point $t=t_s, ..., t_e$ do
(14) If $C(t) \models^{ASP} L$ then
(15) If *IL'* is not empty and *flag*=TRUE then
(16) Take the last item $[t'_s, t'_e]$ from *IL'* and replace it by $[t'_s,t]$.
(17) else
(18) Add interval $[t,t]$ to the end of *IL'*.
(19) else /* $C(t) \not\models^{ASP} L$ */
(20) *flag*=FALSE.
(21) EndFor
(22) If $[t_s,t_e]$ is the last interval in *IL* then
(23) If *L* is a weakly negated ELP literal and $t_e \neq t_{max}$ then
(24) If *IL'* is not empty and *flag*=TRUE then
(25) Take the last item $[t'_s, t_e]$ from *IL'* and replace it by $[t'_s,t_{max}]$.
(26) else
(27) Add interval $[t_e+1,t_{max}]$ to the end of *IL'*.
(28) else /* $[t_s,t_e]$ is not the last interval in *IL* */
(29) If *L* is a weakly negated ELP literal then
(30) If *IL'* is not empty and *flag*=TRUE then
(31) Take the last interval $[t'_s, t_e]$ from *IL'* and replace it by $[t'_s, t''_s\text{-}1]$, where $[t''_s,t''_e]$ is the next interval to $[t_s,t_e]$ in *IL*.
(32) else
(33) Add $[t_e+1,t''_s\text{-}1]$ to the end of *IL'* where $[t''_s,t''_e]$ is the next interval to $[t_s,t_e]$ in *IL*.
(34) *flag*=TRUE.
(35) else /* *L* is not a weakly negated ELP literal */
(36) *flag*=FALSE.
(37) EndFor
(38) return(*IL'*).

Example 5 Consider the temporally annotated logic program *C* of Example 1. Then, *FindMaximalIntervals*(*C has_job*(*Mary*, *Hairdresser*)) returns the list [[1990,1994], [2006, 2009]] and *FindMaximalIntervals*(C, ~*has_job*(*Mary*, *Hairdresser*)) returns the list [[1988, 1989], [1995,2005], [2010,2012]].

Based on Algorithm 2, the following complexity results are derived.

Proposition 2 Let *L*:*i* be a ground temporal literal over *V*. Deciding if $C \models^{max} L{:}i$ is:

1. in P^{NP}, in the case that *C* does not contain variables or the number of variables of each rule of *C* is less than a constant, and
2. in $P^{NEXPTIME}$, in the general case.

We would like to note that we have investigated the case to define, for a temporally annotated logic program *C*, interpretations *I* containing temporal literals *L*:*i* s.t. if *L*:*i*, *L*:*i'* $\in I$ then temporal intervals *i,i'* have no common time points and they are not consecutive. We can decide if *I* is an answer set of *C*, using techniques similar to Answer Set Programming, but now we work on temporal intervals and not on time points. Then, $C \models L{:}[t_1,t_2]$, if for all answer sets *M* of *C* it holds that $M \models L{:}[t_1,t_2]$. Additionally, $C \models_{max} L{:}[t_1,t_2]$, if (i) for all answer sets *M* of *C*, it holds that $M \models L{:}[t_1,t_2]$, (ii) there is an answer set *M* of *C* such that $M \not\models L{:}[t_1\text{-}1,t_1\text{-}1]$, and (iii) there is an answer set *M* of *C* such that $M \not\models L{:}[t_2+1,t_2+1]$. However, this technique has higher complexity and does not provides answer sets in the case that it exists a time point *t* such that *C*(*t*) has no (consistent) answer set.

III. Query Answering

Below, we define the queries that can be imposed to a temporally annotated logic program *C* over a vocabulary *V*=<*Pred*, *Const*> and their answers.

First, we define the intersection of a set of temporal intervals. The intersection of the temporal intervals $[t_1, t'_1],...,[t_n, t'_n]$ is the interval $[max(\{t_1,...,t_n\}), min(\{t'_1,..., t'_n\})]$, if $max(\{t_1,...,t_n\}) \leq min(\{t'_1,...,t'_n\})$. Otherwise, it is undefined.

A simple temporal query of type 1 has the form *SQ*=*L*:*i*, where *L*:*i* is a temporal literal with *i* being a temporal interval. The answers of *SQ* w.r.t. *C*, denoted by $Ans_C(SQ)$, is the set of mappings *v* from the variables of *L* to *Const* s.t. $C \models v(L{:}i)$.

A simple temporal query of type 2 has the form *SQ*=*L*:?*i*, where *L*:?*i* is a temporal literal with ?*i* being a temporal variable. The answers of *SQ* w.r.t. *C*, denoted by $Ans_C(SQ)$, is the set of mappings *v* from the variables of *L* to *Const* and from ?*i* to the set of temporal intervals s.t $C \models_{max} v(L{:}?i)$.

The answer to this type of queries can be provided using Algorithm 2.

A simple temporal query of type 3 has the form $SQ=L_1{:}?i \wedge ... \wedge L_n{:}?i$, where $L_i{:}?i$ is a temporal literal with ?*i* being a temporal variable. The answers of *SQ* w.r.t. *C*, denoted by $Ans_C(SQ)$, is the set of mappings *v* s.t. if $v_i \in Ans_C(L_i{:}?i)$, for *i*=1,...,*n*, s.t. $v_1,...,v_n$ coincide on the common variables of $L_1,...,L_n$ then *v* coincides with v_i on the variables of L_i, for *i*=1,...,*n*, and maps ?*i* to the intersection of the temporal intervals $v_1(?i),...,v_n(?i)$, if such intersection exists.

A simple temporal query of type 4 has the form $SQ=L_1{:}?i \wedge ... \wedge L_n{:}?i \wedge included(?i, [t_1,t_2])$, where $L_1{:}?i \wedge ... \wedge L_n{:}?i$ is a simple temporal query of type 3 and t_1,t_2 are time points. The answers of *SQ* w.r.t. *C*, denoted by $Ans_C(SQ)$, is the set of mappings *v* s.t. if $u \in Ans_C(L_1{:}?i \wedge ... \wedge L_n{:}?i)$ then *v* coincides with *u* on the variables of L_i, for *i*=1,...,*n*, and maps ?*i* to the intersection of the temporal intervals *u*(?*i*) and $[t_1,t_2]$, if such intersection exists.

Example 6 Consider the temporally annotated logic program *C* of Example 1. Consider the simple temporal query of type 1 *SQ*=*vacation_days*(*Mary*,?*x*):[1994,2001], requesting the vacations days that Mary is entitled to continuously during the temporal interval [1994,2001]. Then, $Ans_C(SQ)$ is the mapping *v* s.t. *v*(?*x*)=22.

Consider the simple temporal query of type 2 SQ= *vacation_days*(*Mary*,?*x*):?*i*, requesting the vacations days that Mary is entitled to and their maximal temporal intervals. Then, $Ans_C(SQ)$ is the set of mappings (i) v_1 s.t. $v_1(?x)$=29 and $v_1(?x)$=[1990,1991], (ii) v_2 s.t. $v_2(?x)$=27 and $v_2(?i)$= [1992, 1992], (iii) v_3 s.t. $v_3(?x)$=22 and $v_3(?i)$= [1993, 2002], and (iv) v_4 s.t. $v_4(?x)$=22 and $v_4(?x)$=[2006, 2009].

Consider the simple temporal query of type 3 SQ=*has_job*(*Mary*,?*x*):?i $\wedge$ *has_job*(*Peter*,?*y*):?*i*, requesting the jobs of Mary and Peter and their common validity temporal intervals. Then, $Ans_C(SQ)$ is the set of mappings (i) v_1 s.t. $v_1(?x)$=*Secretary*, $v_1(?y)$=*Garbage_collector*, and $v_1(?i)$=[2001,2002] and (ii) v_2 s.t. $v_2(?x)$=*Hairdresser*, $v_2(?y)$= *Builder*, and $v_2(?i)$=[2006,2008].

Consider the simple temporal query of type 4 SQ=*vacation_days*(*Mary*,?*x*):?*i* $\wedge$ *included*(?*i*,[1991, 2001]), requesting the vacations days that Mary is entitled to and their maximal temporal intervals, within the temporal interval of interest [1991, 2001]. Then, $Ans_C(SQ)$ is the set of mappings (i) v_1 s.t. $v_1(?x)$=29 and $v_1(?x)$=[1991,1991], (ii) v_2 s.t. $v_2(?x)$=27 and $v_2(?i)$=[1992,1992], and (iii) v_3 s.t. $v_3(?x)$=22 and $v_3(?i)$=[1993,2001].

A complex *pt*-query has the form $CQ=SQ_1 \wedge \ldots \wedge SQ_n \wedge filter$, where SQ_i are simple temporal queries, each having a different temporal variable, and *filter* is an expression of the following EBNF grammar:

term:=*duration*(?*i*) | *start*(?*i*) | *end*(?*i*) | *c*,
where ?*i* is a temporal variable and *c* is a decimal.
complex_term:= term | complex_term (+ | - | * | /) complex_term
comparison:= complex_term (< | > | = | ≤ | ≥ | ≠) complex_term
filter:= comparison | *filter* ($\wedge$|$\vee$) filter,

such that each temporal variable appearing in *filter* appears in $SQ_1 \wedge \ldots \wedge SQ_n$. The answers of CQ w.r.t. C, denoted by $Ans_P(CQ)$, is the set of mappings v s.t. (i) if $u_i \in Ans_P(SQ_i)$, for i=1,…,n, s.t. $u_1,\ldots,u_n$ coincide on the common variables of $SQ_1,\ldots, SQ_n$ then v coincides with u_i on the variables of SQ_i, for i=1,…,n, and (ii) v(*filter*) holds.

Note that all Allen's interval algebra relations and their combinations can be expressed by a *filter* expression. For example, the Allen's algebra relation *overlaps*(?*i*,?*j*) can be expressed by the filter (*start*(?*i*) < *start*(?*j*)) $\wedge$ (*start*(?*j*) < *end*(?*i*)).

Example 7 Consider the temporally annotated logic program C of Example 1. Consider the complex query CQ= *has_job*(*Mary*,?*x*):?*i* $\wedge$ *has_job*(*Peter*,?*y*):?*i'* $\wedge$ (*start*(?*i*) ≤ *end*(?*i'*)) $\wedge$ (*end*(?*i*) ≥ *start*(?*i'*)), requesting the job of Mary and its validity temporal interval and the job of Peter and its validity temporal interval, provided that these intervals have common time points. Then, $Ans_C(SQ)$ is the set of mappings (i) v_1 s.t. $v_1(?x)$=*Secretary*, $v_1(?i)$=[1995,2002], $v_1(?y)$= *Garbage_collector*, and $v_1(?i')$=[2001, 2003] and (ii) v_2 s.t. $v_2(?x)$=*Hairdresser*, $v_2(?i)$=[2006,2009], $v_2(?y)$=*Builder*, and $v_2(?i')$=[2005,2008].

Consider the complex query

$$CQ = has\_job(Mary, ?x) : ?i \wedge has\_job(Peter, ?y) : ?i' \wedge$$
$$((start(?i) > start(?i')) \wedge (end(?i) < end(?i')) \wedge (end(?i) - start(?i) > 2) \vee$$
$$(start(?i') \geq start(?i)) \wedge (end(?i') \leq end(?i)) \wedge (end(?i') - start(?i') > 2) \vee$$
$$(start(?i) \leq start(?i')) \wedge (start(?i') \leq end(?i)) \wedge (start(?i') - end(?i) > 2) \vee$$
$$(start(?i') \leq start(?i)) \wedge (start(?i) \leq end(i')) \wedge (start(?i) - end(?i') > 2))$$

requesting the job of Mary and its validity temporal interval and the job of Peter and its validity temporal interval, provided that these intervals have more than 2 common time points. Then, $Ans_C(SQ)$ is the mapping v s.t. $v(?x)$= *Hairdresser*, $v(?i)$=[2006,2009], $v(?y)$=*Builder*, and $v(?i')$= [2005,2008].

IV. Related Work

Below, we review related work.

In [4], the authors present a framework to incorporate temporal reasoning into RDFS [5][6]. The author associate RDF triples with their validity temporal interval and apply the RDFS inference rules (which are always valid). Like our work, their semantics is based on time points and not on temporal intervals. Yet, [4] does not consider strong and weak negation and validity intervals on logic rules. Additionally, it does not support simple queries of type 3 and type 4 and the filter condition is limited.

Note that our approach can also be applied to RDFS, as RDFS inference rules can be expressed through definite rules [7].

In [8], the authors present a general framework for representing, reasoning, and querying with annotated data on the Semantic Web. They show that their formalism can be instantiated on the temporal, fuzzy, and provenance domain. The authors associate RDF triples with their validity temporal intervals and apply the RDFS inference rules (which are always valid). Unlike our work, their semantics is based on temporal intervals. Yet, [8] does not consider strong and weak negation and validity intervals on logic rules. Additionally, it does not support simple queries of type 4. Moreover, our query answering is more efficient, since during query answering, we directly work on maximal temporal intervals. In [8], all temporal intervals returned by the query are considered and then the maximal ones are returned. Further, our semantics are different than [8]. For example, consider the temporal RDF triples $p(a,b)$: [1990, 2000] and $q(c,d)$: [1995, 2010]. Then, according to [8], the answer to query $p(a,b)$:?*i* $\wedge q(c,d)$:?*i'* $\wedge$ *end*(?i) < *start*(?*i'*) will provide the mapping v s.t. $v(?i)$=[1990, 1994] and $v(?i')$=[1995, 2010]. In our case, we will provide no answers, since 2000 > 1995.

In [9], the authors extend RDF graphs with temporal information, by associating RDF triples with their validity

interval. They consider any entailment regime that can be expressed through definite rules $A_0 \leftarrow A_1,..., A_n$, where A_i is an RDF triple. Each such rule is replaced by the temporal rule $A_0:[max(t_1,...,t_n),\min(t'_1,...,t'_n)] \leftarrow A_1:[t_1,t'_1]$,..., $A_n:[t_n, t'_n]$. These rules are applied recursively, until a fixpoint is reach. Then, maximal validity temporal intervals for each derived RDF triple are produced. Yet, [9] does not consider strong and weak negation and validity intervals on logic rules. Additionally, it does not support simple temporal queries of type 4 and the filter condition is left unspecified.

Work in [10] provides a framework to support spatial and temporal analysis over semantic web data. With respect to the temporal component [10] is similar to [9], as it also computes the maximal validity temporal intervals of derived RDF triples, using the RDFS entailment rules. Yet, [10] does not consider strong and weak negation and validity intervals on logic rules.

In [11], [12], the authors extend the RDFS and ter-Horst entailment rules [13] (which extend RDFS with terms from the OWL [14] vocabulary) with temporal information. In particular, they support inference rules having the general form of these, supported by [9]. However, they dot consider a query language. Additionally, they do not consider strong and weak negation and validity intervals on logic rules.

In [15], we presented a semantics for provenance and temporally annotated definite logic programs. However, [15] does not consider strong and weak negation, and reasons based on temporal intervals and not time points. The query language presented here is a restriction of the query language presented in [15] on the temporal component, with the difference that negated atoms in the queries are supported in the present work.

In [16], the authors present a temporal algebra, where the validity temporal interval of two joined relational tuples with associated temporal intervals i_1 and i_2 is the intersection of i_1 and i_2. This temporal algebra operation is also adopted by TSQL2 [17]. In general, TSQL2 is an extension of SQL that supports temporal and non-temporal tables. It also provides a temporal relational algebra that can undertake temporal selection of data and temporal joins based on temporal intersection.

V. Conclusion

In this paper, we considered extended logic programming rules, associated with their validity temporal intervals, forming a *temporally annotated logic program*.

We defined (simple) entailment and maximal entailment of a ground temporal literal $L:i$ from a temporally annotated logic program C. Both kinds of entailment are based on Answer Set Programming. The complexities of simple and maximal entailment of ground temporal literals are provided. Additionally, we provided an algorithm that for an ELP literal or a weakly negated ELP literal L returns a list with all temporal intervals i such that a temporally annotated logic program C maximally entails $L:i$. Based on this algorithm, the answer of various kinds of temporal queries can be provided.

Note that we do not support operations, such as *next, until, since, sometimes,* and *always*, supported by temporal logic (for an overview, see [18]). Additionally, we do not support inferences such that ``if something is true in one temporal interval then something else is true in another temporal interval", as supported by [19]. Yet, these works do not support the inferences made by our own model.

As future work, we plan to consider logic programs annotated over multiple domains and not just the temporal domain.

.

Appendix: Proof of Propositions

Proof of Proposition 1

Hardness: Let P be an extended logic program over a vocabulary $V=<Pred,Const>$. Consider the temporally annotated logic program C over V that is derived from P by associating all rules with the validity temporal interval $[t,t]$. Let L be an ELP literal over V. Then, $P \models^{ASP} L$ iff $C \models L:[t,t]$. In [3], it is shown that deciding if $P \models^{ASP} L$ is (i) co-NP-complete, in the case that P does not contain variables or the number of variables of each rule of P is less than a constant and (ii) co-NEXPTIME-complete, in the general case. Therefore, deciding if a temporally annotated logic program $C \models L:i$, for a temporal literal $L:i$, is (i) co-NP-hard, in the case that C does not contain variables or the number of variables of each rule of C is less than a constant and (ii) co-NEXPTIME-hard, in the general case.

Membership: Guess a time point t within the temporal interval i and an interpretation I of $C(t)$ over constants in *Const*. Deciding if I is an answer set of $C(t)$ and $I \not\models L$ is in (i) P, in the case that $C(t)$ does not contain variables or the number of variables of each rule of $C(t)$ is less than a constant and (ii) EXPTIME, in the general case [3]. Thus, deciding if $C \not\models L:i$ is in (i) NP, in the case that C does not contain variables or the number of variables of each rule of C is less than a constant and (ii) NEXPTIME, in the general case. Therefore, deciding if $C \models L:i$ is in (i) co-NP, in the case that C does not contain variables or the number of variables of each rule of C are less than a constant and (ii) co-NEXPTIME, in the general case.

Proof of Proposition 2

In [3], it is shown that entailment of an ELP literal or a weakly negated ELP literal L from an extended logic program P, under Answer Set Programming, is (i) co-NP-complete, in the case that P does not contain variables or the number of variables of each rule of P is less than a constant and (ii) co-NEXPTIME-complete, in the general case. Note that Algorithm 2 runs in polynomial time by calling oracles

deciding if $C(t) \mid=^{ASP} L$. Therefore, the complexity of Algorithm 2 is in (i) P^{NP}, in the case that C does not contain variables or the number of variables of each rule of C is less than a constant and (ii) $P^{NEXPTIME}$, in the general case.

REFERENCES

[1] M. Gelfond and V. Lifschitz, "Classical Negation in Logic programs and Disjunctive Databases", New Generation Computing, 9, 1991, pp. 365-385.

[2] J. F. Allen, "Maintaining Knowledge about Temporal Intervals", Communications of the ACM, 1983, pp. 832–843.

[3] Evgeny Dantsin, T. Eiter, G. Gottlob, and A. Voronkov: "Complexity and expressive power of logic programming". ACM Comput. Surv. 33(3),2001,pp. 374-425.

[4] C. Gutierrez, C. A. Hurtado, and A. A. Vaisman, "Introducing Time into RDF", IEEE Transactions on Knowledge and Data Engineering, 19(2), 2007, 207-218.

[5] G. Klyne and J. J. Carroll, "Resource Description Framework (RDF): Concepts and Abstract Syntax", W3C Recommendation, 10 February 2004, available at http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[6] Patrick Hayes, "RDF Semantics", W3C Recommendation, 10 February 2004, available at http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

[7] G. Ianni, A. Martello, C. Panetta, and G. Terracina, "Efficiently Querying RDF(S) Ontologies with Answer Set Programming", Journal of Logic and Computation, 19(4), 2009, pp. 671-695.

[8] A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia, "A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data", Journal of Web Semantics , 11, 2012, pp. 72–95.

[9] B. Motik, "Representing and Querying Validity Time in RDF and OWL: A Logic-Based Approach", 9th International Semantic Web Conference (ISWC-2010), 2010, pp. 550-565.

[10] M. Perry, A. P. Sheth, F. Hakimpour, and P. Jain, "Supporting Complex Thematic, Spatial and Temporal Queries over Semantic Web Data", 2nd International Conference on GeoSpatial Semantics (GeoS-2007), 2007, pp. 228-246.

[11] H.Krieger, "A Temporal Extension of the Hayes and ter Horst Entailment Rules for RDFS and OWL", AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning, 2011.

[12] H.Krieger, "A Temporal Extension of the Hayes/ter Horst Entailment Rules and a Detailed Comparison with W3C's N-ary Relations", Deutsches Forschungszentrum fur Kunstliche Intelligenz GmbH Technical Report, RR-11-02, 2011.

[13] H. J. ter Horst, "Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary, Journal of Web Semantics, 3(2-3), 2005, pp. 79-115.

[14] G. Antoniou and F. van Harmelen, A semantic web primer, MIT Press, 2004.

[15] A. Analyti and I. Pachoulakis, "Provenance and Temporally Annotated Logic Programming", accepted to the International Journal of Advanced Research in Artificial Intelligence (IJARAI), 2012.

[16] D. Dey, T. M. Barron, and V. C. Storey, "A Complete Temporal Relational Algebra", VLDB Journal, 5(3), 1996, pp. 167-180.

[17] R.T. Snodgrass , The TSQL2 Temporal Query Language, Springer, 1995.

[18] M. A. Orgun and W. Ma, "An Overview of Temporal and Modal Logic Programming", First International Conference on Temporal Logic (ICTL-1994), 1994, pp. 445-479.

[19] T.W. Fruhwirth, "Temporal Annotated Constraint Logic Programming", Journal of Symbolic Computation, 22(5/6), 1996, pp.555-583.

AUTHORS PROFILE

Anastasia Analyti earned a B.Sc. degree in Mathematics from University of Athens, Greece and a M.Sc. and Ph.D. degree in Computer Science from Michigan State University, USA. She worked as a visiting professor at the Department of Computer Science, University of Crete, and at the Department of Electronic and Computer Engineering, Technical University of Crete. Since 1995, she is a principal researcher at the Information Systems Laboratory of the Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH-ICS). Her current interests include reasoning on the Semantic Web, modular web rule bases, non-monotonic-reasoning, faceted metadata and semantics, conceptual modelling, contextual organization of information, information integration and retrieval systems for the web, interoperability of heterogeneous and distributed information bases, and biomedical information systems. She has participated in several research projects and has published over 55 papers in refereed scientific journals and conferences.

Ioannis Pachoulakis received a B.Sc. in Physics (1988) at the University of Crete, Greece, and a Ph.D. in Astrophysics (1996) and an M.Sc. in Engineering (1998), both from the University of Pennsylvania in the U.S.A. Since 2001, he serves as an Assistant Professor at the Department of Applied Informatics and Multimedia at TEI of Crete with mainstream interests in realistic multimedia applications, virtual reality and multimedia applications for science.