Chapter 1

SEMANTICS FOR TEMPORAL ANNOTATED CONSTRAINT LOGIC PROGRAMMING

Alessandra Raffaetà
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, I-56125 Pisa, Italy
raffaeta@di.unipi.it

Thom Frühwirth
Institut für Informatik, Ludwig-Maximilians-Universität (LMU)
Oettingenstrasse 67, D-80538 Munich, Germany
fruehwir@informatik.uni-muenchen.de

Abstract We investigate semantics of a considerable subset of Temporal Annotated Constraint Logic Programming (TACLP), a class of languages that allows us to reason about qualitative and quantitative, definite and indefinite temporal information using time points and time periods as labels for atoms.

After illustrating the power of TACLP with some non-trivial examples, TACLP is given two different kinds of semantics, an operational one based on meta-logic (top-down semantics) and, for the first time, a fixpoint one based on an immediate consequence operator (bottom-up semantics). We prove the top-down semantics to be sound and complete with respect to the bottom-up semantics.

Keywords: Temporal reasoning, Constraint Logic Programming, Annotated Logics, Semantics.

1. INTRODUCTION

Temporal reasoning is at the heart of human activity and not surprisingly it has raised a lot of interest in computer science, be it in the form of temporal logics [39, 21, 17], temporal programming languages [32, 34, 18, 1, 8, 11, 15] or temporal databases [36, 38, 4, 6, 9, 20, 33]. No matter if one programs with

temporal information or stores data with temporal information, in most cases the formal underpinnings will be logic, and often be variants or extensions of first order logic.

In a logical formulation and formalization of temporal information and reasoning it is quite natural to think of formulae that are labelled with temporal information and about proof procedures that take into account these labels. In other words, we separate the temporal from the non-temporal aspects. What D. Gabbay enthusiastically says in the introduction of his book on labelled deductive systems [19], that allow to label a general class of logics, "This sounds very simple but it ... makes a serious difference, like the difference between using one hand only or allowing for the coordinated use of two hands", can apply to the special case of temporal labels as well.

In our case, the logic and the labels are familiar structures: first-order logic (FOL) and lattices. The labels are called annotations, and the overall class of logics is called annotated logics [26]. Based on this framework and on constraint logic programming concepts [24, 25, 29, 30, 16], the family of temporal annotated constraint logic programming (TACLP) languages has been developed in [14, 13, 15, 35, 28].

The pieces of temporal information are given by *temporal annotations* which say at what time(s) the formula to which they are applied is valid. The annotations of TACLP make time explicit but avoid the proliferation of temporal variables and quantifiers of the first order approach. In this way, TACLP supports qualitative and quantitative (metric) temporal reasoning involving both time points and time periods (time intervals) and their duration. Moreover, it allows us to represent definite, indefinite and periodic temporal information. In Figure 1.1, an example from [14] illustrates the expressiveness and conceptual simplicity of TACLP. As shown in [14] by asking the query $murder(x, y)$ we obtain two solutions (i.e., two suspects) of the workshop murder mystery: $x = Lepov, y = Lepov$ and $x = Maringer, y = Lepov$. The first one means that Prof. Lepov could have committed suicide. This unexpected solution is found because Prof. Lepov does not have an alibi for the time of his death. The second answer means that Dr. Maringer could be the murderer, because his alibi does not hold: The copying would have taken 30 minutes, so it cannot have happened during a talk of 25 minutes.

In [15] TACLP is presented as an instance of annotated constraint logic (ACL) for reasoning about time. ACL is a generalization of generalized annotated programs [26, 27], and extends first-order languages with a distinguished class of predicates, called *constraints*, and a distinguished class of terms, called *annotations*, used to label formulae. Moreover ACL provides inference rules for annotated formulae and a constraint theory for handling annotations. One advantage of a language in the ACL framework is that its clausal fragment can be efficiently implemented: Given a logic in this framework, there is a

There is a workshop at the Plaza hotel.
(1) In the afternoon session, after the coffee break (3:00 - 3:25pm), there were four more talks, 25 minutes each - *time periods*.
Dr. Maringer gave the 3rd talk. The last talk was to be given by Prof. Lepov.
$coffeebreak$ `th` $[3{:}00, 3{:}25]$.
$talk(1, Hunon, AlgebraicSemantics...)$ `th` $[3{:}25, 3{:}50]$.
$talk(2, ..., ...)$ `th` $[3{:}50, 4{:}15]$.
$talk(3, Maringer, ...)$ `th` $[4{:}15, 4{:}40]$.
$talk(4, Lepov, P = NP)$ `th` $[4{:}40, 5{:}05]$.

(2) Prof. Lepov was found dead in his hotel room at 5:35pm - *time point*.
$founddead(Lepov)$ `at` $5{:}35$.

(3) The doctor said he was dead for one to one and a half hours - *duration and indefinite information.*
$murdered(p)$ `in` $[t_1 - 1{:}30, t_2 - 1{:}00] \leftarrow founddead(p)$ `in` $[t_1, t_2]$

There are two suspects, Dr. Kosta and Dr. Maringer. They have alibis.
(4) Dr. Kosta took the last shuttle to the airport possible to reach the 5:10pm plane - *time point.*
$boardplane(Kosta)$ `at` $5{:}10$.

(5) The shuttle from the hotel leaves every half hour between noon and 11pm - *recurrent (periodic) data.*
$shuttle$ `at` $0{:}00$.
$shuttle$ `at` $t + 30 \leftarrow 0{:}00 \leq t, t \leq 11{:}00, shuttle$ `at` t

(6) It takes at least 50 minutes to get to the airport - *duration and indefinite information.*
$onshuttle(p)$ `th` $[t_1, t_2] \leftarrow t_2 = t_1 + 50,\ shuttle$ `at` $t_1,$
$boardplane(p)$ `in` $[t_2, t_2 + 50]$

(7) During the 2nd talk Dr. Maringer realized that he had forgotten to copy his 30 slides - *relates time periods.*
So he picked up the slides from his hotel room and copied them. It takes 5 minutes to get to the room, another 5 minutes to get to the copy room from there, and 5 more minutes to get back to the lecture hall - *durations.*
A copy takes half a minute - *repeated durations.*
$copying(Maringer)$ `th` $[t_1, t_2] \leftarrow t_2 = t_1 + 5 + 5 + 30 * 0.5 + 5,$
$talk(2, p, w)$ `th` $[t_1, t_2]$

(8) *Who murdered Prof. Lepov?* The murderer is a person who is involved in the case and does not have an alibi during the time of murder.
$murder(p_1, p_2) \leftarrow murdered(p_2)$ `in` $[t_1, t_2], involved(p_1), \neg(alibi(p_1)$ `th` $[t_1, t_2])$
$involved(Kosta).$ $involved(Lepov).$ $involved(Maringer).$
$alibi(p)$ `th` $[t_1, t_2] \leftarrow (onshuttle(p) \vee copying(p) \vee talk(n, p, w))$ `th` $[t_1, t_2]$

Figure 1.1 The Workshop Murder Mystery

systematic way to make a clausal fragment executable as a constraint logic program. Both an interpreter and a compiler can be generated and implemented in standard constraint logic programming languages.

Constraint logic programming (CLP) [24, 25, 29, 30, 16] is an extension of logic programming, where in addition to predicates, which are defined by clauses and reasoned about by resolution (a form of Modus Ponens), there is a distinguished class of predicates called *constraints*. Their meaning is defined by a *constraint theory* whose reasoning capability is implemented by some efficient algorithm in the so-called *constraint solver*. In this way, efficient special-purpose algorithms can be integrated in a sound way into logic programming.

Overview of the paper. In this paper, the TACLP language is given two different kinds of semantics, an operational one based on meta-logic (top-down semantics) using a meta-interpreter and a fixpoint one obtained by extending the definition of the immediate consequence operator of CLP to deal with annotated atoms (bottom-up semantics). The meta-interpreter is proved to be sound and complete with respect to the bottom-up semantics. While top-down semantics is known from previous work on TACLP [14, 13, 15], this paper presents for the first time a bottom-up semantics and, consequently, for the first time soundness and completeness results for TACLP.

The paper is organized as follows. Section 2. introduces the TACLP framework. Section 3. defines the two semantics for TACLP and proves soundness and completeness. Section 4. shortly presents related work and Section 5. concludes the paper.

2. TEMPORAL ANNOTATED CONSTRAINT LOGIC PROGRAMMING

This subsection briefly reviews TACLP. In this paper, we consider the subset of TACLP, where time points are totally ordered, sets of time points are convex and non-empty, and only atomic formulae can be annotated. Moreover clauses are free of negation. These restrictions will become clear during this section. For a more detailed treatment of TACLP and for the general theory of ACL we refer the reader to [15].

Time can be discrete or dense. Time points are totally ordered by the relation $\leq$. We call the set of time points D and we suppose that a set of operations (such as the binary operations $+, -$) to manage such points are associated with it. We assume that the time-line is left-bounded by the number 0 and open to the future, with the symbol ∞ used to denote a time point that is later than any other. A *time period* is an interval $[r, s]$ with $0 \leq r \leq s \leq \infty, r \in D,\ s \in D$

that represents the convex, non-empty set of time points $\{t \mid r \leq t \leq s\}$[1]. Thus the interval $[0, \infty]$ denotes the whole time line.

An *annotated formula* is of the form $A\,\alpha$ where A is an atomic formula and α an annotation. In TACLP, there are three kinds of annotations based on (sets of) time points. Let t be a time point and let I be a time period.

(`at`) The annotated formula $A\,\mathtt{at}\,t$ means that A holds at time point t.

(`th`) The annotated formula $A\,\mathtt{th}\,I$ means that A holds *throughout*, i.e., at *every* time point in the time period I. The definition of a `th`-annotated formula in terms of `at` is:

$$A\,\mathtt{th}\,I \;\Leftrightarrow\; \forall t\,(t \in I \rightarrow A\,\mathtt{at}\,t).$$

(`in`) The annotated formula $A\,\mathtt{in}\,I$ means that A holds at *some* time point(s) - but we do not know exactly when - in the time period I. The definition of an `in`-annotated formula in terms of `at` is:

$$A\,\mathtt{in}\,I \;\Leftrightarrow\; \exists t\,(t \in I \wedge A\,\mathtt{at}\,t).$$

The `in` temporal annotation accounts for indefinite temporal information.

The set of annotations is endowed with a partial order relation $\sqsubseteq$ which turns it into a lattice. Given two annotations α and β, the intuition is that $\alpha \sqsubseteq \beta$ if α is "less informative" than β in the sense that for all formulae A, $A\,\beta \Rightarrow A\,\alpha$. More precisely, being an instance of ACL, in addition to Modus Ponens, TACLP has two further inference rules: the rule ($\sqsubseteq$) and the rule ($\sqcup$).

$$\frac{A\,\alpha \qquad \gamma \sqsubseteq \alpha}{A\,\gamma}\quad rule\ (\sqsubseteq) \qquad\qquad \frac{A\,\alpha \qquad A\,\beta \qquad \gamma = \alpha \sqcup \beta}{A\,\gamma}\quad rule\ (\sqcup)$$

The rule ($\sqsubseteq$) states that if a formula holds with some annotation, then it also holds with all annotations that are smaller according to the lattice ordering. The rule ($\sqcup$) says that if a formula holds with some annotation and the same formula holds with another annotation then it holds with the least upper bound of the annotations.

The lattice operation $\sqsubseteq$ is axiomatized by the following constraint theory.

(`at th`)	$\mathtt{at}\,t = \mathtt{th}\,[t,t]$		
(`at in`)	$\mathtt{at}\,t = \mathtt{in}\,[t,t]$		
(`th` $\sqsubseteq$)	$\mathtt{th}\,[s_1,s_2] \sqsubseteq \mathtt{th}\,[r_1,r_2]$	$\Leftrightarrow$	$r_1 \leq s_1,\ s_1 \leq s_2,\ s_2 \leq r_2$
(`in` $\sqsubseteq$)	$\mathtt{in}\,[r_1,r_2] \sqsubseteq \mathtt{in}\,[s_1,s_2]$	$\Leftrightarrow$	$r_1 \leq s_1,\ s_1 \leq s_2,\ s_2 \leq r_2$

[1] The results in this paper naturally extend to time lines that are bounded or unbounded in other ways and to time periods that are open on one or both sides.

The first two axioms state that $\texttt{th}\, I$ and $\texttt{in}\, I$ are equivalent to $\texttt{at}\, t$ when the time period I consists of a single time point t.[2] Next, if a formula holds at every element of a time period, then it holds at every element in all sub-periods of that period ($(\texttt{th}\ \sqsubseteq)$ axiom). On the other hand, if a formula holds at some points of a time period then it holds at some points in all periods that include this period ($(\texttt{in}\ \sqsubseteq)$ axiom).

To summarize the partial order relation on annotations, the axioms can be arranged in the following chain, assuming $r_1 \leq s_1,\ s_1 \leq s_2,\ s_2 \leq r_2$:

$$\texttt{in}\,[r_1, r_2] \sqsubseteq \texttt{in}\,[s_1, s_2] \sqsubseteq \texttt{in}\,[s_1, s_1] = \texttt{at}\, s_1 = \texttt{th}\,[s_1, s_1] \sqsubseteq \texttt{th}\,[s_1, s_2] \sqsubseteq \texttt{th}\,[r_1, r_2]$$

Now we axiomatize the least upper bound $\sqcup$ of temporal annotations over time points and time periods. As explained in [15], the least upper bound exists but sometimes may be "too large". For instance, according to the lattice, $\texttt{th}\,[1,2] \sqcup \texttt{th}\,[4,5] = \texttt{th}\,[1,5]$, but according to the definition of th-annotated formulae in terms of at , $A\,\texttt{th}\,[1,2] \wedge A\,\texttt{th}\,[4,5]$ does not imply $A\,\texttt{th}\,[1,5]$, since it does not express the validity of $A\,\texttt{at}\,3$. Also, $\texttt{in}\,[1,2] \sqcup \texttt{in}\,[2,3] = \texttt{in}\,[2,2]$, but $A\,\texttt{in}\,[1,2] \wedge A\,\texttt{in}\,[2,3] \Rightarrow A\,\texttt{at}\,2$ is not in general correct, since e.g. $A\,\texttt{at}\,1$ and $A\,\texttt{at}\,3$ may hold. From a theoretical point of view, this problem can be overcome by enriching the lattice of annotations with expressions with $\sqcup$. In practice, it suffices to consider the least upper bound for time periods that produce a different meaningful time period. Concretely, we can restrict ourselves to th annotations with overlapping time periods that do not include one another:

$$(\texttt{th}\sqcup) \qquad \texttt{th}\,[s_1, s_2] \sqcup \texttt{th}\,[r_1, r_2] = \texttt{th}\,[s_1, r_2] \ \Leftrightarrow\ s_1 < r_1, r_1 \leq s_2, s_2 < r_2.$$

Finally, the constraint theory also contains an axiomatization of the total order relation $\leq$ on D (!!!!!!! We should explain how these constraints are handled by the constraint solver).

We can now define the clausal fragment of TACLP that can be used as an efficient temporal programming language. A *TACLP program* is a finite set of TACLP clauses. A *TACLP clause* is a TACLP formula of the form:

$$A\,\alpha \leftarrow C_1, \ldots, C_n, B_1\,\alpha_1, \ldots, B_m\,\alpha_m \ \ (n, m \geq 0)$$

where A is an atom (not a constraint), α and α_i are (optional) temporal annotations, the C_j's are the constraints and the B_i's are atomic formulae. Constraints C_j cannot be annotated. As in logic programming syntax, commas "," denote conjunctions. The conclusion of the implication is called the *head* of the clause

[2] Especially in dense time, one may disallow singleton periods and drop the two axioms. This restriction has no effects on the results of the paper.

and the premise the *body* of the clause. Variables in a clause are implicitly assumed to be universally quantified at the outermost scope.

We conclude the introduction of TACLP with some examples. In the following programs, we adopt the convention of denoting variables with identifiers starting with a lower-case letter and constant symbols by identifiers starting with an upper-case letter.

Example 1 *In a company, there are managers and a secretary who has to manage their meetings. A manager is busy if he is in a meeting or if he is out.*

$$busy(p)\ \mathtt{th}\,[t_1,t_2] \leftarrow \textit{in-meeting}(p)\ \mathtt{th}\,[t_1,t_2]$$
$$busy(p)\ \mathtt{th}\,[t_1,t_2] \leftarrow \textit{out-of-office}(p)\ \mathtt{th}\,[t_1,t_2]$$

Suppose the schedule for today to be the following: Smith and Jones have a meeting at 9am and at 9:30am respectively, each lasting one hour. In the afternoon Smith goes out for lunch at 2pm and comes back at 3pm:

$$\textit{in-meeting}(Smith)\ \mathtt{th}\,[9am, 10am]. \qquad \textit{out-of-office}(Smith)\ \mathtt{th}\,[2pm, 3pm].$$
$$\textit{in-meeting}(Jones)\ \mathtt{th}\,[9{:}30am, 10{:}30am].$$

If the secretary wants to know whether Smith is busy between 9:30am and 10:30am she can ask for $busy(Smith)\ \mathtt{in}\,[9{:}30am, 10{:}30am]$. *Since Smith is in a meeting from 9am till 10am, one can indeed derive that Smith is busy. This query exploits indefinite information, since if Smith is busy in one instant of the period* $[9{:}30am, 10{:}30am]$*, then the secretary cannot schedule an appointment for him for that period.*

On the other hand, $busy(Smith)\ \mathtt{th}\,[9{:}30am, 10{:}30am]$ *does not hold, because Smith is not busy between 10am and 10:30am. Also* $busy(Smith)\ \mathtt{in}\,[10{:}30am, 1{:}30pm]$ *does not hold, because Smith is not busy in that time period at all.*

The query $(busy(Smith)\ \mathtt{th}\,[t_1,t_2],\ busy(Jones)\ \mathtt{th}\,[t_1,t_2])$ *reveals that both managers are busy throughout the time period* $[9{:}30am, 10am]$*, because this is the largest interval that is included in the time periods where both managers are busy.*

Now assume that we define

$$busy\ \mathtt{th}\,[t_1,t_2] \leftarrow busy(p)\ \mathtt{th}\,[t_1,t_2]$$

Then $busy$ *holds when either manager is busy, namely for the intervals* $[9am, 10{:}30am]$ *(which is the least upper bound of the time periods for the two overlapping meetings of Smith and Jones) and* $[2pm, 3pm]$.

In [35] TACLP is successfully applied to a system for calculating the liquid flow in a network of water tanks from some events specifying when the taps were switched on and off. The following example involving continuous change is also presented.

Example 2 *We model information about the growth of trees.*

1. *Tree 1 sprouts at time* 3.5 *(the middle of year 3).*

$sprouts(Tree1)$ `at` 3.5.

2. *Tree 1 is an oak tree.*

$tree\_type(Tree1, Oak)$.

3. *The growth rate of oak trees is 3 meters per year.*

$growth\_rate(Oak, 3)$.

4. *If a tree is of a type that has a given growth rate r, and the tree sprouts at time s then at time t it has a height, where* $h = (t - s) \times r$.

$$height(tree, h)\ \texttt{at}\ t \leftarrow \\ h = (t - s) \times r, \\ tree\_type(tree, type), growth\_rate(type, r), \\ sprouts(tree)\ \texttt{at}\ s$$

5. *If a tree has height* h *m at time* t*, where* $h \geq 6.75$*, then it is mature.*

$$mature(tree)\ \texttt{th}\,[t, \infty] \leftarrow h \geq 6.75, height(tree, h)\ \texttt{at}\ t$$

In the last clause, the maturity of the tree at an instant is implied by a constraint on the height of the tree at that instant. Height is the continuously changing quantity. The query

$mature(Tree1)\ \texttt{th}\,[6, 7]$

can be proven. This means that Tree1 is mature throughout the time period which begins at year 6 *and ends at year* 7.

The query

$mature(Tree1)\ \texttt{th}\,[t_1, t_2]$

yields $t_1 \geq 5.75, t_2 = \infty$.

3. SEMANTICS OF TACLP

In this section we define the operational (top-down) semantics of the language TACLP by presenting a meta-interpreter for it. Then we provide TACLP with a fixpoint (bottom-up) semantics, based on the definition of an immediate consequence operator, and we prove that the meta-interpreter is sound and complete with respect to the bottom-up semantics.

In the definition of the semantics, without loss of generality, we assume all atoms to be annotated with `th` or `in` labels. `at` t annotations can be replaced with `th` $[t, t]$ by exploiting the (`at th`) axiom. Each atom which is not annotated in the object level program is intended to be true throughout the whole temporal domain, and thus can be labelled with `th` $[0, \infty]$. Constraints stay unchanged.

3.1 OPERATIONAL SEMANTICS VIA META-INTERPRETER

The *vanilla* meta-interpreter [37] is the simplest application of meta-programming in logic. A general formulation of the vanilla meta-interpreter can be given by means of the $demo$ predicate used to represent provability. $demo(g)$ means that the formula g is provable in the object program.

$$\begin{aligned} & demo(Empty). \\ demo((b_1, b_2)) \quad & \leftarrow \quad demo(b_1),\ demo(b_2) \\ demo(a) \quad & \leftarrow \quad clause(a, b),\ demo(b) \end{aligned}$$

The unit clause states that the empty goal, represented by the constant symbol $Empty$, is always solved. The second clause deals with conjunctive goals. It states that a conjunction (B_1, B_2) is solved if B_1 is solved and B_2 is solved. Finally, the third clause deals with the case of atomic goal reduction. To solve an atomic goal A, a clause from the program is chosen whose head unifies with A and the body of the clause is recursively solved. An object level program P is represented at the meta-level by a set of axioms of the kind $clause(A, B)$, one for each object level clause $A \leftarrow B$ in P.

The extended meta-interpreter for our subset of TACLP is defined by the following clauses:

$$demo(Empty). \tag{1.1}$$

$$demo((b_1, b_2)) \leftarrow demo(b_1), demo(b_2) \tag{1.2}$$

$$\begin{aligned} demo(a\ \texttt{th}\ [t_1, t_2]) \leftarrow\ & s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2, \\ & clause(a\ \texttt{th}\ [s_1, s_2], b), demo(b) \end{aligned} \tag{1.3}$$

$$\begin{aligned} demo(a\ \texttt{th}\ [t_1, t_2]) \leftarrow\ & s_1 \leq t_1, t_1 < s_2, s_2 < t_2, \\ & clause(a\ \texttt{th}\ [s_1, s_2], b), demo(b), demo(a\ \texttt{th}\ [s_2, t_2]) \end{aligned} \tag{1.4}$$

$$\begin{aligned} demo(a\ \texttt{in}\ [t_1, t_2]) \leftarrow\ & t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2, \\ & clause(a\ \texttt{th}\ [s_1, s_2], b), demo(b) \end{aligned} \tag{1.5}$$

$$\begin{aligned} demo(a\ \texttt{in}\ [t_1, t_2]) \leftarrow\ & t_1 \leq s_1, s_2 \leq t_2, \\ & clause(a\ \texttt{in}\ [s_1, s_2], b), demo(b) \end{aligned} \tag{1.6}$$

$$demo(c) \leftarrow constraint(c), c \tag{1.7}$$

A clause $A\,\alpha \leftarrow B$ of a TACLP program P is represented at the meta-level by

$$clause(A\,\alpha, B) \leftarrow t_1 \leq t_2. \tag{1.8}$$

where $\alpha = \mathtt{th}\,[t_1, t_2]$ or $\alpha = \mathtt{in}\,[t_1, t_2]$.

This meta-interpreter can be written in any CLP language that provides a suitable constraint solver for temporal annotations (see Section 2. for the constraint theory). Hence the first difference with the vanilla meta-interpreter is that our meta-interpreter handles constraints which can either occur explicitly in its clauses, e.g. $s_1 \leq t_1, t_1 \leq t_2, t_2 \leq s_2$ in clause (1.3), or can come from the resolution steps. The latter kind of constraints is managed by clause (1.7) which passes each constraint C to be solved directly to the constraint solver.

The second difference is that our meta-interpreter implements not only Modus Ponens but also the rule $(\sqsubseteq)$ and the rule $(\sqcup)$. This is the reason why the third $demo$ clause of the vanilla meta-interpreter is now split into four clauses. Clauses (1.3), (1.5) and (1.6) implement the inference rule $(\sqsubseteq)$: the atomic goal to be solved is required to be labelled with an annotation which is smaller than the one labelling the head of the clause used in the resolution step. For instance, clause (1.3) states that given a clause $A\,\mathtt{th}\,[s_1, s_2] \leftarrow B$ whose body B is solvable, we can derive the atom A annotated with any $\mathtt{th}\,[t_1, t_2]$ such that $\mathtt{th}\,[t_1, t_2] \sqsubseteq \mathtt{th}\,[s_1, s_2]$, i.e., according to axiom $(\mathtt{th}\ \sqsubseteq)$, $[t_1, t_2] \subseteq [s_1, s_2]$, as expressed by the constraint $s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2$. Clauses (1.5) and (1.6) are built in an analogous way by exploiting axioms $(\mathtt{in\,th}\ \sqsubseteq)$ and $(\mathtt{in}\ \sqsubseteq)$, respectively.

Rule $(\sqcup)$ is implemented by clause (1.4). According to the discussion in Section 2., it is applicable only to `th` annotations with overlapping time periods which do not include one another. More precisely, clause (1.4) states that if we can find a clause $A\,\mathtt{th}\,[s_1, s_2] \leftarrow B$ such that the body B is solvable, and if moreover the atom A can be proved *throughout* the time period $[s_2, t_2]$ (i.e., $demo(A\,\mathtt{th}\,[s_2, t_2])$ is solvable) then we can derive the atom A labelled with any annotation $\mathtt{th}\,[t_1, t_2] \sqsubseteq \mathtt{th}\,[s_1, s_2]$. The constraints on temporal variables ensure that the time period $[t_1, t_2]$ is a *new* time period different from $[s_1, s_2]$ and $[s_2, t_2]$ and their subintervals.

Finally, in the meta-level representation of object clauses, clause (1.8), we have to add the constraint $t_1 \leq t_2$ to ensure that the head of the object clause has a well-formed, namely non-empty, annotation.

Example 3 *Consider a library database containing information about loans. Mary first borrowed the book Hamlet from May 12, 1995 to June 12, 1995 and then on June 12, 1995 she extended her loan:*

$$borrow(Mary, Hamlet)\,\mathtt{th}\,[May\ 12\ 1995, Jun\ 12\ 1995].$$
$$borrow(Mary, Hamlet)\,\mathtt{th}\,[Jun\ 12\ 1995, Aug\ 1\ 1995].$$

The period of time in which Mary borrowed Hamlet can be obtained by the query

$demo(borrow(Mary, Hamlet)\ \mathtt{th}\ [t_1, t_2])$.

By using clause (1.4), it is possible to derive the interval $[May\ 12\ 1995, Aug\ 1\ 1995]$ *(more precisely, the constraints* $May\ 12\ 1995 \leq t_1$, $t_1 < Jun\ 12\ 1995$, $Jun\ 12\ 1995 < t_2$, $t_2 \leq Aug\ 1\ 1995$ *are derived) that otherwise would be never generated. In fact, by applying clause (1.3) alone, it is possible to prove only that Mary borrowed Hamlet in the intervals* $[May\ 12\ 1995, Jun\ 12\ 1995]$ *and* $[Jun\ 12\ 1995, Aug\ 1\ 1995]$ *separately.*

3.2 FIXPOINT SEMANTICS

There are several ways of defining a bottom-up semantics of TACLP, related to the different possible choices of the semantic domain where the immediate consequence operator is defined. The simpler solution consists in using the powerset $\wp(\mathcal{A}\text{-}base^3 \times Ann)$ with set-theoretic inclusion, disregarding the partial order structure of the set of annotations Ann. Alternative solutions (as for generalized annotated programs in [26]) may consider a more abstract domain, which is obtained by endowing $\mathcal{A}$-*base*$\times Ann$ with the product order (induced by the discrete order on $\mathcal{A}$-*base* and the order on Ann) and then by taking as elements of power domain only those subsets of annotated atoms which satisfy some closure properties with respect to such an order. For instance, one can require "downward-closedness", which amounts to including subsumption in the $\mathcal{T}_P$ operator. Another possible property is "limit-closedness", namely the presence of the least upper bound of all directed sets which, from a computational point of view, amounts to consider computations which possibly require more than ω steps. For space limitations, we treat here the first, simpler solution.

The intended interpretation of constraints is defined by fixing a structure $\mathcal{A}$. In our case $\mathcal{A}$ surely contains a structure $\mathcal{D}$ (with domain D) in which we interpret the temporal constants and functions. However, TACLP programs can have constraints not only on temporal data, hence in general the structure $\mathcal{A}$ will be multi-sorted.

Let $Dom_{\mathcal{A}}$ the domain of the structure $\mathcal{A}$. An $\mathcal{A}$-*valuation* is a (multi-sorted) mapping from variables to $Dom_{\mathcal{A}}$, and its natural extension maps terms to $Dom_{\mathcal{A}}$ and formulae to formulae whose predicates have arguments ranging over $Dom_{\mathcal{A}}$. An $\mathcal{A}$-*ground instance* A' of an atom A (resp. of a constraint or of a clause) is obtained by applying an $\mathcal{A}$-valuation to the atom (resp. to the constraint or to the clause), thus producing a construct of the form $p(a_1, \ldots, a_n)$ with $a_1, \ldots, a_n$ elements from $Dom_{\mathcal{A}}$. We denote by $ground_{\mathcal{A}}(P)$ the set of $\mathcal{A}$-ground instances of clauses from a program P.

[3]The formal definition of $\mathcal{A}$-*base* is given later. Briefly, it is the natural generalization of the notion of Herbrand Base in constraint logic programming.

We first define the standard fixpoint operator of constraint logic programming and then extend it to deal with TACLP. An *$\mathcal{A}$-interpretation* for a $CLP(\mathcal{A})$ program P is a subset of the *$\mathcal{A}$-base* of P, written *$\mathcal{A}$-base*$_P$, which is the set

$$\left\{ p(a_1, \ldots, a_n) \mid \begin{array}{l} p \text{ is a } n\text{-ary user-defined predicate in } P \\ \text{and each } a_i \text{ is an element of } Dom_{\mathcal{A}} \end{array} \right\}$$

Then the standard immediate consequence operator [29] for a $CLP(\mathcal{A})$ program P is a function $T_P^{\mathcal{A}} : \wp(\textit{$\mathcal{A}$-base}_P) \rightarrow \wp(\textit{$\mathcal{A}$-base}_P)$ defined as follows:

$$T_P^{\mathcal{A}}(I) \;=\; \left\{ A \mid \begin{array}{l} A \leftarrow C_1, \ldots, C_k, B_1, \ldots, B_n, \in ground_{\mathcal{A}}(P), \\ \{B_1, \ldots, B_n\} \subseteq I,\ \mathcal{A} \models C_1, \ldots, C_k \end{array} \right\}$$

The operator $T_P^{\mathcal{A}}$ is continuous [29], and therefore it has least fixpoint which can be computed as the least upper bound of the chain $\{(T_P^{\mathcal{A}})^i\}_{i \geq 0}$ of the iterated applications of $T_P^{\mathcal{A}}$ starting from the empty set.[4] The fixpoint is denoted by $(T_P^{\mathcal{A}})^{\omega}$.

To generalize the above operator to deal with temporal annotations we consider a kind of extended interpretations, basically consisting of sets of annotated elements of $\mathcal{A}$-$base$. Formally we define the set of (semantical) annotations

$$Ann = \{\mathtt{th}\,[t_1, t_2], \mathtt{in}\,[t_1, t_2] \mid t_1 \in D, t_2 \in D, \mathcal{D} \models t_1 \leq t_2\}$$

Then given a TACLP program P, the lattice of interpretations is defined as $(\wp(\textit{$\mathcal{A}$-base}_P \times Ann), \subseteq)$ where $\wp$ is the powerset operator and $\subseteq$ is the usual relation of set-theoretic inclusion.

Definition 4 *Let P be a TACLP program, the function $\mathcal{T}_P^{\mathcal{A}} : \wp(\mathcal{A}\text{-}base_P \times Ann) \rightarrow \wp(\mathcal{A}\text{-}base_P \times Ann)$ is defined as follows.*

$$\begin{array}{l} \mathcal{T}_P^{\mathcal{A}}(I) = \\ \left\{ (A, \alpha) \mid \begin{array}{l} (\alpha = \mathtt{th}\,[s_1, s_2] \ \vee\ \alpha = \mathtt{in}\,[s_1, s_2]) \\ A\ \alpha \leftarrow C_1, \ldots, C_k, B_1\alpha_1, \ldots, B_n\alpha_n \in ground_{\mathcal{A}}(P), \\ \{(B_1, \beta_1), \ldots, (B_n, \beta_n)\} \subseteq I, \\ \mathcal{A} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, s_1 \leq s_2 \end{array} \right\} \\ \cup \\ \left\{ (A, \mathtt{th}\,[s_1, r_2]) \mid \begin{array}{l} A\ \mathtt{th}\,[s_1, s_2] \leftarrow C_1, \ldots, C_k, B_1\alpha_1, \ldots, B_n\alpha_n \in ground_{\mathcal{A}}(P), \\ \{(B_1, \beta_1), \ldots, (B_n, \beta_n)\} \subseteq I,\ (A, \mathtt{th}\,[r_1, r_2]) \in I, \\ \mathcal{A} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, s_1 < r_1, r_1 \leq s_2, s_2 < r_2 \end{array} \right\} \\ \cup \\ \left\{ (A, \mathtt{in}\,[t_1, t_2]) \mid \begin{array}{l} A\ \mathtt{th}\,[s_1, s_2] \leftarrow C_1, \ldots, C_k, B_1\alpha_1, \ldots, B_n\alpha_n \in ground_{\mathcal{A}}(P), \\ \{(B_1, \beta_1), \ldots, (B_n, \beta_n)\} \subseteq I, \\ \mathcal{A} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad s_1 \leq s_2 \end{array} \right\} \end{array}$$

[4] Formally, for a function $T : \wp(S) \rightarrow \wp(S)$ we define $T^0 = \emptyset$ and $T^{i+1} = T(T^i)$.

This definition properly extends the standard definition of the immediate consequence operator. In fact, in a sense, it captures not only the Modus Ponens rule, as the standard operator does, but also rule $(\sqcup)$ (second set in the above definition). In addition, rule $(\sqsubseteq)$ is used to prove that an annotated atom holds in an interpretation: To derive the head $A\,\alpha$ of a clause it is not necessary to find in the interpretation exactly the atoms $B_1\,\alpha_1, \ldots, B_n\,\alpha_n$ occurring in the body of the clause, but it suffices to find atoms $B_i\,\beta_i$ which implies $B_i\,\alpha_i$, i.e., such that each β_i is an annotation stronger than α_i ($\mathcal{A} \models \alpha_i \sqsubseteq \beta_i$). Finally, notice that $\mathcal{T}_P^{\mathcal{A}}(I)$ is not downward closed, namely, it is not true that if $(A, \alpha) \in \mathcal{T}_P^{\mathcal{A}}(I)$ then for all (A, γ) such that $\gamma \sqsubseteq \alpha$, we have $(A, \gamma) \in \mathcal{T}_P^{\mathcal{A}}(I)$. However such a closure is done at the end of the computation of the fixpoint of $\mathcal{T}_P^{\mathcal{A}}$. In this way the rule $(\sqsubseteq)$ is completely captured.

An important property of the $\mathcal{T}_P^{\mathcal{A}}$ operator, which is at the core of the definition of the fixpoint semantics, is continuity over the lattice of interpretations.

Theorem 5 (Continuity) *Let P be a TACLP program. The function $\mathcal{T}_P^{\mathcal{A}}$ is continuous (on $(\wp(\mathcal{A}\text{-}base \times Ann), \subseteq)$).*

Proof. The proof is a direct consequence of the definition of $\mathcal{T}_P^{\mathcal{A}}$ and of the partial order $\subseteq$ on the interpretations. For more details see the Appendix. □

The bottom-up semantics for a program P is defined as the downward closure of the least fixpoint of $\mathcal{T}_P^{\mathcal{A}}$ which by theorem 5 is the least upper bound of the chain $\{(\mathcal{T}_P^{\mathcal{A}})^i\}_{i\geq 0}$.

Definition 6 *Let P be a TACLP program. Then the fixpoint semantics of P is defined as*

$$\mathcal{F}^{\mathcal{A}}(P) = \{(A, \alpha) \mid (A, \beta) \in (\mathcal{T}_P^{\mathcal{A}})^{\omega},\ \mathcal{A} \models \alpha \sqsubseteq \beta\}$$

where $(\mathcal{T}_P^{\mathcal{A}})^{\omega} = \bigcup_{i\geq 0}(\mathcal{T}_P^{\mathcal{A}})^i$.

3.3 SOUNDNESS AND COMPLETENESS

The semantics of meta-logic is a quite debated issue (see e.g [23, 31, 3]). In the spirit of [7, 31], we define the semantics of the extended vanilla meta-interpreter by relating the semantics of an object program to the semantics of the corresponding vanilla meta-program (i.e., including the meta-level representation of the object program). When stating the correspondence between the object program and the meta-program we consider only formulae of interest, i.e., elements of $\mathcal{A}\text{-}base$ annotated with labels from Ann. We show that given a TACLP program P (object program) for any $A \in \mathcal{A}\text{-}base_P$ and any $\alpha \in Ann$, $demo(A\,\alpha)$ is provable at the meta-level if and only if (A, α) is provable in the object program. Formally, we are going to prove that

$$demo(A\,\alpha) \in (T_{\mathcal{V}}^{\mathcal{A}_m})^{\omega} \quad \Longleftrightarrow \quad (A, \alpha) \in \mathcal{F}^{\mathcal{A}}(P) \tag{1.9}$$

where $\mathcal{V}$ is the meta-program containing the meta-level representation of the object program P according to (1.8) and the clauses (1.1)-(1.7), and $T_{\mathcal{V}}^{\mathcal{A}_m}$ is the standard immediate consequence operator of CLP. It is worth noting that $\mathcal{V}$ is a $CLP(\mathcal{A}_m)$ program where the structure $\mathcal{A}_m$ in which we interpret the constraints is composed by two structures: the Herbrand structure associated with $\mathcal{V}$ and the structure $\mathcal{A}$ where the constraints of P are interpreted. Therefore it is obvious that if C is an $\mathcal{A}$-ground instance of a constraint then $\mathcal{A}_m \models C \Leftrightarrow \mathcal{A} \models C$.

Since the given meta-logical definition axiomatizes a top-down operational semantics for TACLP programs, the proof of the statement (1.9) corresponds to showing the equivalence of computing top-down and bottom-up.

In the following for simplicity we drop the reference to $\mathcal{A}$ and $\mathcal{A}_m$ in the name of the immediate consequence operators. Furthermore P denotes a TACLP program, $\mathcal{A}$ the structure where the constraints of P are interpreted, A, B elements of $\mathcal{A}$-$base_P$, α, β, γ elements of Ann and C an $\mathcal{A}$-ground instance of a constraint. All symbols may have subscripts. The proof of the next two subsections are just sketched in the main text. The details can be found in the Appendix.

Soundness In order to show the soundness of the meta-interpreter (restricted to the atoms of interest), we first prove the following lemma stating that a conjunctive goal is provable at the meta-level if its atomic conjuncts are provable at the meta-level.

Lemma 7 *Let P be a program and let $\mathcal{V}$ be the corresponding meta-interpreter. For any $B_1\,\alpha_1, \ldots, B_n\,\alpha_n$ with $B_i \in \mathcal{A}$-$base_P$ and $\alpha_i \in Ann$ and for any $C_1, \ldots, C_k$, with C_i an $\mathcal{A}$-ground instance of a constraint, the following statement holds:*

$$\textit{for all } h \quad demo((C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^h \implies \{demo(B_1\,\alpha_1), \ldots, demo(B_n\,\alpha_n)\} \subseteq T_{\mathcal{V}}^h \ \wedge\ \mathcal{A} \models C_1, \ldots, C_k$$

Proof. The proof easily follows from the definition of $T_{\mathcal{V}}$ and clauses (1.2) and (1.7) of the meta-interpreter. □

The soundness of the meta-interpreter is an easy corollary of the following theorem stating that if $demo(A\,\alpha)$ is provable at the meta-level then (A, γ) is a consequence of the program P, and $A\,\gamma \Rightarrow A\,\alpha$, i.e., the annotation α is less or equal to γ.

Theorem 8 *Let P be a program and let $\mathcal{V}$ be the corresponding meta-program. For any $A \in \mathcal{A}$-$base_P$ and $\alpha \in Ann$, the following statement holds:*

$$demo(A\,\alpha) \in T_{\mathcal{V}}^{\omega} \quad \Longrightarrow \quad \exists \gamma \in Ann : (A, \gamma) \in \mathcal{T}_P^{\omega} \ \wedge\ \mathcal{A} \models \alpha \sqsubseteq \gamma.$$

Proof. We first show that for all h

$$demo(A\,\alpha) \in T_{\mathcal{V}}^{h} \quad \Longrightarrow \quad \exists \gamma \in Ann:\ (A,\gamma) \in \mathcal{T}_P^{\omega}\ \wedge\ \mathcal{A} \models \alpha \sqsubseteq \gamma \tag{1.10}$$

The proof is by induction on h.
(Base case). Trivial since $T_{\mathcal{V}}^{0} = \emptyset$.

(Inductive case). Assume that

$$demo(A\,\alpha) \in T_{\mathcal{V}}^{h} \quad \Longrightarrow \quad \exists \gamma \in Ann:\ (A,\gamma) \in \mathcal{T}_P^{\omega}\ \wedge\ \mathcal{A} \models \alpha \sqsubseteq \gamma.$$

Then:

$$\begin{array}{ll} & demo(A\,\alpha) \in T_{\mathcal{V}}^{h+1} \\ \Longleftrightarrow & \{\text{definition of } T_{\mathcal{V}}^{\delta}\} \\ & demo(A\,\alpha) \in T_{\mathcal{V}}(T_{\mathcal{V}}^{h}) \end{array}$$

We have four cases corresponding to clauses (1.3), (1.4), (1.5) and (1.6) of the meta-interpreter. We only show the first two cases since the others are proved like for clause (1.3).

(clause (1.3)) $\{\alpha = \mathtt{th}\,[t_1,t_2]$, definition of $T_{\mathcal{V}}$ and clause (1.3)$\}$

$$\begin{array}{ll} & \{clause(A\,\mathtt{th}\,[s_1,s_2],G), demo(G)\} \subseteq T_{\mathcal{V}}^{h}\ \wedge\ \mathcal{A} \models s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2 \\ \Longrightarrow & \{\text{meta-level representation of clauses of } P\text{, according to clause (1.8)}\} \\ & \exists (B_1\,\alpha_1),\ldots,(B_n\,\alpha_n), C_1,\ldots,C_k: \\ & A\,\mathtt{th}\,[s_1,s_2] \leftarrow C_1,\ldots,C_k, B_1\,\alpha_1,\ldots,B_n\,\alpha_n \in ground_{\mathcal{A}}(P)\ \wedge \\ & demo((C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{h}\ \wedge\ \mathcal{A} \models s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2 \\ \Longrightarrow & \{\text{Lemma 7}\} \\ & A\,\mathtt{th}\,[s_1,s_2] \leftarrow C_1,\ldots,C_k, B_1\,\alpha_1,\ldots,B_n\,\alpha_n \in ground_{\mathcal{A}}(P)\ \wedge \\ & \{demo(B_1\,\alpha_1),\ldots,demo(B_n\,\alpha_n)\} \subseteq T_{\mathcal{V}}^{h}\ \wedge \\ & \mathcal{A} \models C_1,\ldots,C_k\ \wedge\ \mathcal{A} \models s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2 \\ \Longrightarrow & \{\text{inductive hypothesis}\} \\ & \exists \beta_1,\ldots,\beta_n:\ A\,\mathtt{th}\,[s_1,s_2] \leftarrow C_1,\ldots,C_k, B_1\,\alpha_1,\ldots,B_n\,\alpha_n \in ground_{\mathcal{A}}(P)\ \wedge \\ & \{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq \mathcal{T}_P^{\omega}\ \wedge\ \mathcal{A} \models \alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n\ \wedge \\ & \mathcal{A} \models C_1,\ldots,C_k\ \wedge\ \mathcal{A} \models s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2 \\ \Longrightarrow & \{\text{definition of } \mathcal{T}_P \text{ since } \mathcal{A} \models s_1 \leq s_2\} \\ & (A, \mathtt{th}\,[s_1,s_2]) \in \mathcal{T}_P(\mathcal{T}_P^{\omega})\ \wedge\ \mathcal{A} \models s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2 \\ \Longrightarrow & \{\mathcal{T}_P^{\omega} \text{ is a fixpoint of } \mathcal{T}_P \text{ and } \mathcal{A} \models s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2\} \\ & (A, \mathtt{th}\,[s_1,s_2]) \in \mathcal{T}_P^{\omega}\ \wedge\ \mathcal{A} \models \mathtt{th}\,[t_1,t_2] \sqsubseteq \mathtt{th}\,[s_1,s_2] \end{array}$$

(clause (1.4)) $\{\alpha = \mathtt{th}\,[t_1,t_2]$, definition of $T_{\mathcal{V}}$ and clause (1.4)$\}$

$$\begin{array}{ll} & \{clause(A\,\mathtt{th}\,[s_1,s_2],G), demo(G), demo(A\,\mathtt{th}\,[s_2,t_2])\} \subseteq T_{\mathcal{V}}^{h}\ \wedge \\ & \mathcal{A} \models s_1 \leq t_1, t_1 < s_2, s_2 < t_2 \\ \Longrightarrow & \{\text{meta-level representation of clauses of } P\text{, according to clause (1.8)}\} \\ & \exists (B_1\,\alpha_1),\ldots,(B_n\,\alpha_n), C_1,\ldots,C_k: \end{array}$$

$A\,\mathtt{th}\,[s_1, s_2] \leftarrow C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n \in \mathit{ground}_{\mathcal{A}}(P) \wedge$
$\mathit{demo}((C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^h \;\wedge\; \mathit{demo}(A\,\mathtt{th}\,[s_2, t_2]) \in T_{\mathcal{V}}^h \wedge$
$\mathcal{A} \models s_1 \leq t_1, t_1 < s_2, s_2 < t_2$
$\Longrightarrow$ {Lemma 7}
$A\,\mathtt{th}\,[s_1, s_2] \leftarrow C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n \in \mathit{ground}_{\mathcal{A}}(P) \wedge$
$\{\mathit{demo}(B_1\,\alpha_1), \ldots, \mathit{demo}(B_n\,\alpha_n)\} \subseteq T_{\mathcal{V}}^h \;\wedge\; \mathcal{A} \models C_1, \ldots, C_k \wedge$
$\mathit{demo}(A\,\mathtt{th}\,[s_2, t_2]) \in T_{\mathcal{V}}^h \;\wedge\; \mathcal{A} \models s_1 \leq t_1, t_1 < s_2, s_2 < t_2$
$\Longrightarrow$ {inductive hypothesis}
$\exists \beta, \beta_1, \ldots, \beta_n : \; A\,\mathtt{th}\,[s_1, s_2] \leftarrow C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n \in \mathit{ground}_{\mathcal{A}}(P) \wedge$
$\{(B_1, \beta_1), \ldots, (B_n, \beta_n), (A, \beta)\} \subseteq \mathcal{T}_P^\omega \wedge$
$\mathcal{A} \models \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, \mathtt{th}\,[s_2, t_2] \sqsubseteq \beta \;\wedge\; \mathcal{A} \models C_1, \ldots, C_k \wedge$
$\mathcal{A} \models s_1 \leq t_1, t_1 < s_2, s_2 < t_2.$
Since $\mathcal{A} \models \mathtt{th}\,[s_2, t_2] \sqsubseteq \beta$ then $\beta = \mathtt{th}\,[w_1, w_2]$ with $\mathcal{A} \models w_1 \leq s_2, t_2 \leq w_2$.

According to the relation between w_1 and s_1 we can distinguish

(a) $\mathcal{A} \models w_1 \leq s_1$. Then $\mathcal{A} \models w_1 \leq s_1, s_1 \leq t_1, t_1 < s_2, s_2 < t_2, t_2 \leq w_2$ which allows us to conclude that
$(A, \mathtt{th}\,[w_1, w_2]) \in \mathcal{T}_P^\omega \;\wedge\; \mathcal{A} \models \mathtt{th}\,[t_1, t_2] \sqsubseteq \mathtt{th}\,[w_1, w_2]$

(b) $\mathcal{A} \models s_1 < w_1$. Then $\mathcal{A} \models s_1 < w_1, w_1 \leq s_2, s_2 < t_2, t_2 \leq w_2$ then by definition of $\mathcal{T}_P$
$(A, \mathtt{th}\,[s_1, w_2]) \in \mathcal{T}_P(\mathcal{T}_P^\omega) \;\wedge\; \mathcal{A} \models s_1 \leq t_1, t_1 < s_2, s_2 < t_2, t_2 \leq w_2$
$\Longrightarrow$ $\{\mathcal{T}_P^\omega$ is a fixpoint of $\mathcal{T}_P$ and $\mathcal{A} \models s_1 \leq t_1, t_1 < t_2, t_2 \leq w_2\}$
$(A, \mathtt{th}\,[s_1, w_2]) \in \mathcal{T}_P^\omega \;\wedge\; \mathcal{A} \models \mathtt{th}\,[t_1, t_2] \sqsubseteq \mathtt{th}\,[s_1, w_2]$

We are now able to conclude the proof.

$\mathit{demo}(A\,\alpha) \in T_{\mathcal{V}}^\omega$
$\Longrightarrow \{T_{\mathcal{V}}^\omega = \bigcup_{i \geq 0} T_{\mathcal{V}}^i\}$
$\exists h : \; \mathit{demo}(A\,\alpha) \in T_{\mathcal{V}}^h$
$\Longrightarrow$ {Statement (1.10)}
$\exists \beta : (A, \beta) \in \mathcal{T}_P^\omega \;\wedge\; \mathcal{A} \models \alpha \sqsubseteq \beta$

□

Theorem 9 (Soundness) *Let P be a TACLP program and let $\mathcal{V}$ be the corresponding meta-program. For any $A \in \mathcal{A}\text{-}base_P$ and $\alpha \in Ann$, the following statement holds:*

$$\mathit{demo}(A\,\alpha) \in T_{\mathcal{V}}^\omega \qquad \Longrightarrow \qquad (A, \alpha) \in \mathcal{F}^{\mathcal{A}}(P)$$

Proof. By Theorem 8 there exists $\gamma \in Ann$ such that $(A, \gamma) \in \mathcal{T}_P^\omega$ and $\mathcal{A} \models \alpha \sqsubseteq \gamma$. Hence by definition $(A, \alpha) \in \mathcal{F}^{\mathcal{A}}(P)$. □

Completeness In order to show the completeness of the meta-interpreter we first prove two lemmata. Lemma 10 states that if an annotated atom $A\,\alpha$ is

provable at the meta-level then all A labelled with a weaker annotation ($\gamma \sqsubseteq \alpha$) are provable at the meta-level, too. Lemma 11 is a very technical lemma regarding the $\mathcal{T}_P$ operator, which is used in the proof of the main result.

Lemma 10 *Let P be a program and let $\mathcal{V}$ be the corresponding meta-program. For any $A \in \mathcal{A}\text{-}base_P$ and $\alpha \in Ann$, the following statement holds:*

$$demo(A\,\alpha) \in T_{\mathcal{V}}^{\omega} \qquad \Longrightarrow \qquad \forall \gamma \sqsubseteq \alpha.\ demo(A\,\gamma) \in T_{\mathcal{V}}^{\omega}$$

Proof. The proof is carried out by induction on the number of iteration of $T_{\mathcal{V}}$ and by considering the different cases coming from clauses (1.3), (1.4), (1.5) and (1.6) of the meta-interpreter. □

Lemma 11 *Let P be a program and let $\mathcal{V}$ be the corresponding meta-program. For any $A \in \mathcal{A}\text{-}base_P$ and $\alpha \in Ann$ and any interpretation $I \subseteq \mathcal{A}\text{-}base_P \times Ann$, the following statement holds:*

$$\begin{array}{l}
(A,\alpha) \in \mathcal{T}_P(I) \\
\quad \Longrightarrow\ (\exists (B_1,\beta_1),\ldots,(B_n,\beta_n), C_1,\ldots,C_k,\alpha_1,\ldots,\alpha_n : \\
\quad\quad clause(A\,\alpha,(C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
\quad\quad \{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq I \wedge \\
\quad\quad \mathcal{A} \models C_1,\ldots,C_k,\alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n) \vee \\
\quad\quad (\exists (B_1,\beta_1),\ldots,(B_n,\beta_n), C_1,\ldots,C_k,\alpha_1,\ldots,\alpha_n, \mathtt{th}\,[s_1,r_2], s_2, r_1 : \\
\quad\quad \alpha = \mathtt{th}\,[s_1,r_2] \wedge \\
\quad\quad clause(A\,\mathtt{th}\,[s_1,s_2],(C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
\quad\quad \{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq I \ \wedge\ (A,\mathtt{th}\,[r_1,r_2]) \in I \wedge \\
\quad\quad \mathcal{A} \models C_1,\ldots,C_k,\alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, s_1 < r_1, r_1 \leq s_2, s_2 < r_2) \\
\quad\quad (\exists (B_1,\beta_1),\ldots,(B_n,\beta_n), C_1,\ldots,C_k,\alpha_1,\ldots,\alpha_n, \mathtt{in}\,[t_1,t_2], \mathtt{th}\,[s_1,s_2] : \\
\quad\quad \alpha = \mathtt{in}\,[t_1,t_2] \wedge \\
\quad\quad clause(A\,\mathtt{th}\,[s_1,s_2],(C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
\quad\quad \{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq I \wedge \\
\quad\quad \mathcal{A} \models C_1,\ldots,C_k,\alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2)
\end{array}$$

Now we can prove the completeness of the meta-interpreter with respect to the bottom-up semantics.

Theorem 12 (Completeness) *Let P be a program and let $\mathcal{V}$ be the corresponding meta-program. For any $A \in \mathcal{A}\text{-}base_P$ and $\alpha \in Ann$, the following statement holds:*

$$(A,\alpha) \in \mathcal{F}^{\mathcal{A}}(P) \qquad \Longrightarrow \qquad demo(A\,\alpha) \in T_{\mathcal{V}}^{\omega}.$$

Proof. We first show that for all h

$$(A,\alpha) \in \mathcal{T}_P^{h} \qquad \Longrightarrow \qquad demo(A\,\alpha) \in T_{\mathcal{V}}^{\omega}. \tag{1.11}$$

The proof is by induction on h.
(Base case). Trivial since $\mathcal{T}_P^0 = \emptyset$.

(Inductive case). Assume that

$$(A, \alpha) \in \mathcal{T}_P^h \quad \Longrightarrow \quad demo(A\,\alpha) \in T_{\mathcal{V}}^{\omega}$$

Then:

$$
\begin{array}{ll}
 & (A, \alpha) \in \mathcal{T}_P^{h+1} \\
\Longleftrightarrow & \{\text{definition of } \mathcal{T}_P^{\delta}\} \\
 & (A, \alpha) \in \mathcal{T}_P(\mathcal{T}_P^h) \\
\Longrightarrow & \{\text{Lemma 11}\} \\
 & (\exists (B_1, \beta_1), \ldots, (B_n, \beta_n), C_1, \ldots, C_k, \alpha_1, \ldots, \alpha_n : \\
 & clause(A\,\alpha, (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \{(B_1, \beta_1) \ldots, (B_n, \beta_n)\} \subseteq \mathcal{T}_P^h \wedge \mathcal{A} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n) \vee \\
 & (\exists (B_1, \beta_1), \ldots, (B_n, \beta_n), C_1, \ldots, C_k, \alpha_1, \ldots, \alpha_n, \mathtt{th}\,[s_1, r_2], s_2, r_1 : \\
 & \alpha = \mathtt{th}\,[s_1, r_2] \wedge clause(A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \{(B_1, \beta_1), \ldots, (B_n, \beta_n)\} \subseteq \mathcal{T}_P^h \wedge (A, \mathtt{th}\,[r_1, r_2]) \in \mathcal{T}_P^h \wedge \\
 & \mathcal{A} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, s_1 < r_1, r_1 \leq s_2, s_2 < r_2) \vee \\
 & (\exists (B_1, \beta_1), \ldots, (B_n, \beta_n), C_1, \ldots, C_k, \alpha_1, \ldots, \alpha_n, \mathtt{in}\,[t_1, t_2], \mathtt{th}\,[s_1, s_2] : \\
 & \alpha = \mathtt{in}\,[t_1, t_2] \wedge clause(A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \{(B_1, \beta_1), \ldots, (B_n, \beta_n)\} \subseteq \mathcal{T}_P^h \wedge \\
 & \mathcal{A} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2) \\
\Longrightarrow & \{\text{inductive hypothesis}\} \\
 & (clause(A\,\alpha, (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \{demo(B_1\,\beta_1), \ldots, demo(B_n\,\beta_n)\} \subseteq T_{\mathcal{V}}^{\omega} \wedge \\
 & \mathcal{A} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n) \vee \\
 & (\alpha = \mathtt{th}\,[s_1, r_2] \wedge clause(A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \{demo(B_1\,\beta_1), \ldots, demo(B_n\,\beta_n)\} \subseteq T_{\mathcal{V}}^{\omega} \wedge demo(A\,\mathtt{th}\,[r_1, r_2]) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \mathcal{A} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, s_1 < r_1, r_1 \leq s_2, s_2 < r_2) \vee \\
 & (\alpha = \mathtt{in}\,[t_1, t_2] \wedge clause(A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \{demo(B_1\,\beta_1), \ldots, demo(B_n\,\beta_n)\} \subseteq T_{\mathcal{V}}^{\omega} \wedge \\
 & \mathcal{A} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2) \\
\Longrightarrow & \{\text{Lemma 10 and } \mathcal{A} \models \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n\} \\
 & (clause(A\,\alpha, (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \{demo(B_1\,\alpha_1), \ldots, demo(B_n\,\alpha_n)\} \subseteq T_{\mathcal{V}}^{\omega} \wedge \mathcal{A} \models C_1, \ldots, C_k) \vee \\
 & (\alpha = \mathtt{th}\,[s_1, r_2] \wedge clause(A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \{demo(B_1\,\alpha_1), \ldots, demo(B_n\,\alpha_n)\} \subseteq T_{\mathcal{V}}^{\omega} \wedge demo(A\,\mathtt{th}\,[r_1, r_2]) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \mathcal{A} \models C_1, \ldots, C_k, s_1 < r_1, r_1 \leq s_2, s_2 < r_2) \vee \\
 & (\alpha = \mathtt{in}\,[t_1, t_2] \wedge clause(A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \\
 & \{demo(B_1\,\alpha_1), \ldots, demo(B_n\,\alpha_n)\} \subseteq T_{\mathcal{V}}^{\omega} \wedge \\
 & \mathcal{A} \models C_1, \ldots, C_k, t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2)
\end{array}
$$

$\Longrightarrow$ {definition of $T_{\mathcal{V}}$ and clause (1.7) used k times and $T_{\mathcal{V}}^{\omega}$ is a fixpoint of $T_{\mathcal{V}}$}
$(clause(A\,\alpha,(C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge$
$\{demo(B_1\,\alpha_1),\ldots,demo(B_n\,\alpha_n),demo(C_1),\ldots,demo(C_k)\} \subseteq T_{\mathcal{V}}^{\omega}) \vee$
$(\alpha = \mathtt{th}\,[s_1,r_2] \wedge clause(A\,\mathtt{th}\,[s_1,s_2],(C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge$
$\{demo(B_1\,\alpha_1),\ldots,demo(B_n\,\alpha_n),demo(C_1),\ldots,demo(C_k)\} \subseteq T_{\mathcal{V}}^{\omega} \wedge$
$demo(A\,\mathtt{th}\,[r_1,r_2]) \in T_{\mathcal{V}}^{\omega} \wedge \mathcal{A} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2) \vee$
$(\alpha = \mathtt{in}\,[t_1,t_2] \wedge clause(A\,\mathtt{th}\,[s_1,s_2],(C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge$
$\{demo(B_1\,\alpha_1),\ldots,demo(B_n\,\alpha_n),demo(C_1),\ldots,demo(C_k)\} \subseteq T_{\mathcal{V}}^{\omega} \wedge$
$\mathcal{A} \models t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2)$
$\Longrightarrow$ {definition of $T_{\mathcal{V}}$ and clause (1.2) used $n+k-1$ times and $T_{\mathcal{V}}^{\omega}$ is a fixpoint of $T_{\mathcal{V}}$}
$(clause(A\,\alpha,(C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge$
$demo((C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega}) \vee$
$(\alpha = \mathtt{th}\,[s_1,r_2] \wedge clause(A\,\mathtt{th}\,[s_1,s_2],(C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge$
$demo((C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge demo(A\,\mathtt{th}\,[r_1,r_2]) \in T_{\mathcal{V}}^{\omega} \wedge$
$\mathcal{A} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2) \vee$
$(\alpha = \mathtt{in}\,[t_1,t_2] \wedge clause(A\,\mathtt{th}\,[s_1,s_2],(C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge$
$demo((C_1,\ldots,C_k,B_1\,\alpha_1,\ldots,B_n\,\alpha_n)) \in T_{\mathcal{V}}^{\omega} \wedge \mathcal{A} \models t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2)$

Now by exploiting the fact that $T_{\mathcal{V}}^{\omega}$ is a fixpoint of $T_{\mathcal{V}}$ we have:

- by the first disjunct of the formula we conclude $demo(A\,\alpha) \in T_{\mathcal{V}}^{\omega}$ using respectively clause (1.3) if α is a `th` annotation and clause (1.6) if α is an `in` annotation.
- By the second disjunct of the formula and Lemma 10, since $\mathcal{A} \models \mathtt{th}\,[s_2,r_2] \sqsubseteq \mathtt{th}\,[r_1,r_2]$ we have $demo(A\,\mathtt{th}\,[s_2,r_2]) \in T_{\mathcal{V}}^{\omega}$. Now $\mathcal{A} \models s_1 \leq s_1, s_1 < s_2, s_2 < r_2$ then by using clause (1.4) we have $demo(A\,\mathtt{th}\,[s_1,r_2]) \in T_{\mathcal{V}}^{\omega}$.
- By the third disjunct of the formula, by using clause (1.5) we can conclude that $demo(A\,\mathtt{in}\,[t_1,t_2]) \in T_{\mathcal{V}}^{\omega}$.

Hence $demo(A\,\alpha) \in T_{\mathcal{V}}^{\omega}$.

We are now able to conclude the proof of completeness.

$(A,\alpha) \in \mathcal{F}^{\omega}(P)$
$\Longrightarrow$ {definition of $\mathcal{F}^{\omega}(P)$}
$\exists \gamma \in Ann:\ (A,\gamma) \in \mathcal{T}_P^{\omega} \wedge \mathcal{A} \models \alpha \sqsubseteq \gamma$
$\Longrightarrow$ {definition of $\mathcal{T}_P^{\omega}$}
$\exists h:\ (A,\gamma) \in \mathcal{T}_P^{h} \wedge \mathcal{A} \models \alpha \sqsubseteq \gamma$
$\Longrightarrow$ {statement (1.11)}
$demo(A\,\gamma) \in T_{\mathcal{V}}^{\omega} \wedge \mathcal{A} \models \alpha \sqsubseteq \gamma$
$\Longrightarrow$ {Lemma 10}

$demo(A\,\alpha) \in T_{\mathcal{V}}^{\omega}$

□

4. RELATED WORK

One of the first temporal logic programming languages was Templog [1]. In [26], Templog and an interval based temporal logic are translated into annotated logic programs. The annotations used there correspond to the `th` annotations of TACLP. To implement the annotated logic language, the paper proposed to use "reductants", additional clauses which are derived from existing clauses to express all possible least upper bounds. The problem was that a finite program may generate infinitely many such reductants. Then, "ca-resolution" for annotated logic programs was proposed [27]. The idea is to compute dynamically and incrementally the least upper bounds by collecting partial answers. Operationally this is similar to the meta-interpreter presented here which relies on recursion to collect the partial answers. However, in [27] the intermediate stages of the computation are not sound with respect to the standard CLP semantics.

Moreover, in [26] two fixpoint semantics are presented for generalized annotated programs (GAP). The first one, called T_P, is based on interpretations which associate to each element of the Herbrand Base of the program P a *set* of annotations which is an ideal, i.e., a set downward closed and closed with respect to *finite* least upper bounds. The computed ideal is the least one containing the annotations α of annotated atoms $A\,\alpha$ which are heads of (instances of) clauses whose body holds in the interpretation. The other one R_P is based on interpretations which associate to each atom of the Herbrand Base a *single* annotation which is the least upper bound of the set of annotations computed as in the previous case. Our fixpoint operator for TACLP works similarly to the T_P operator: at each step we close with respect to (representable) finite least upper bounds, and, although we perform the downward closure only at the end of the computation, this does not reduce the set of derivable consequences. The main difference resides in the language: TACLP is an extension of CLP, taking from GAP the handling of annotations, which focuses on the temporal aspects, whereas GAP is a general language with negation and arbitrary annotations but without constraints.

Our temporal annotations correspond to some of the predicates proposed by Galton in [22], which is a critical examination of Allen's classical work on a theory of action and time [2]. Galton provides for both time points and time periods in dense linear time. Assuming that the intervals I are not singletons, Galton's predicate *holds-in(A,I)* can be mapped into TACLP's A `in` I, *holds-on(A,I)* into A `th` I, and *holds-at(A,t)* into A `at` t, where A is an atomic formula.

From this mapping it becomes clear that TACLP can be seen as reified FOL where annotated formulae, e.g. $born(john)$ `at` t, correspond to binary meta-relations between predicates and temporal information, e.g. `at`$(born(john), t)$. But also, TACLP can be regarded as a modal logic, where the annotations are seen as parameterized modal operators, e.g. $born(john)$ (`at` t).

In [8], a powerful temporal logic named MTL (tense logic extended by parameterized temporal operators) is translated into first order constraint logic. The resulting language subsumes Templog, as does TACLP. The parameterized temporal operators of MTL correspond to the temporal annotations of TACLP. The constraint theory of MTL is rather complex as it involves quantified variables and implication, whose treatment goes beyond standard CLP implementations. On the other hand, TACLP inherits an efficient standard constraint-based implementation of annotations from the ACL framework.

The ACL framework shares ideas with the work in [10]. There, a proof system for a large class of (propositional and) quantified modal logics is formalized. Soundness, completeness, and normalization can be proved uniformly for every logic in the class. The class is modular both with respect to properties of the accessibility relation in the Kripke frame and leads to a simple implementation of a modal logic theorem prover in standard logical frameworks. Obviously, this approach is motivated by the same ideas that motivated ACL, though on a much wider and thus more abstract class of languages. What [10] calls *base logic*, corresponds to FOL in ACL, what is called *relational theory*, corresponds to the constraint theory of ACL. The relational theory defines the accessibility relation in the Kripke frame by (Horn) rules, while the constraint theory can be any FOL theory, as long as there are efficient algorithms to implement it. Already in [12] it has been argued that the accessibility relation can be regarded as constraint and the associated axioms as constraint theory.

In [5], translations between *signed logic* and FOL are investigated. Annotated logic [26] can be embedded in signed logic. Both formalisms work with labels whose structure forms a lattice. The two rules of regular unit resolution for signed logic correspond exactly to the inference rules of constraint annotated logics. In [5], the lattice structure is encoded by giving FOL clauses for each pair of lattice elements (i.e., labels). This is only feasible for finite lattices. In ACL, the use of a constraint theory also allows for infinite lattices using a small number of axioms.

5. CONCLUSIONS

We investigated semantics of a considerable subset of the language TACLP that allows us to reason about qualitative and quantitative, definite and indefinite temporal information using time points and time periods. Its expressive power has been illustrated with some non-trivial examples.

We defined the operational (top-down) semantics of TACLP by presenting a meta-interpreter for it. Then we provided TACLP for the first time with a fixpoint (bottom-up) semantics, based on the definition of an immediate consequence operator. We proved that the meta-interpreter is sound and complete with respect to the bottom-up semantics. As future work it would be interesting to investigate operators similar to the function R_P defined in [26], and adapt our approach to the bottom-up semantics to the general framework of annotated constraint logic.

Moreover in this paper, we considered the subset of TACLP, where time points are totally ordered, sets of time points are convex and non-empty, and only atomic formulae can be annotated. Furthermore clauses are free of negation. In general, in TACLP arbitrary formulae can be annotated. In some cases, as shown in [15], the annotations can be pushed inside disjunctions, conjunctions and negation. This means that the omission of negation is the main restriction of the current work. Consequently, we want to investigate next if and how the proofs relating the operational and fixpoint semantics can be adapted to deal with negation.

Finally, there are many applications that need the ability of storing and manipulating geometric and temporal data, such as geographic information systems (GIS), geometric modeling systems (CAD), and temporal databases. In such applications space and time are often closely interconnected: much information which is referenced to space is also referenced to time. Therefore another interesting direction for future research is the representation and handling of spatio-temporal data using annotations.

Acknowledgments

We thank Paolo Baldan and Roberta Gori for their useful comments and suggestions.

References

[1] M. Abadi and Z. Manna. Temporal logic programming. In *Journal of Symbolic Computation*, volume 8, pages 277–295, 1989.

[2] J.F. Allen. Towards a general theory of action and time. In *Artificial Intelligence*, volume 23, pages 123–154, 1984.

[3] K. Apt and F. Turini, editors. *Meta-logics and Logic Programming*. MIT Press, London, 1995.

[4] M. Baudinet, J. Chomicki, and P. Wolper. Temporal Deductive Databases. In *[38]*, pages 294–320.

[5] B. Beckert, R. Hähnle, and F. Manyá. Transformations between signed and classical clause logic. In *LD'98 The First International Workshop on Labelled Deduction*, Freiburg, Germany, September 1998. Albert-Ludwigs-Universität.

[6] M. Böhlen and R. Marti. On the Completeness of Temporal Database Query Languages. In *Temporal Logic: Proceedings of the First International Conference, ICTL'94*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 283–300, 1994.

[7] A. Brogi and F. Turini. Meta-Logic for Program Composition: Semantics Issues. In K. Apt and F. Turini, editors, *Meta-Logics and Logic Programming*, pages 83–109, 1995.

[8] C. Brzoska. Temporal logic programming with metric and past operators. In *[11]*, pages 21–39.

[9] J. Chomicki. Temporal Query Languages: A Survey. In *Temporal Logic: Proceedings of the First International Conference, ICTL'94*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 506–534. Springer Verlag, 1994.

[10] S. Matthews D. Basin and L. Viganò. Labelled modal logics: Quantifiers. *Journal of Logic, Language, and Information*, 7(3), 1998.

[11] M. Fisher and R. Owens, editors. *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1995.

[12] A. M. Frisch and R. B. Scherl. A general framework for modal deduction. In *Proceedings 2nd KR '91*, pages 196–207. Morgan Kaufmann, 1991.

[13] T. Frühwirth. Temporal logic and annotated constraint logic programming. In *[11]*, pages 58–68.

[14] T. Frühwirth. Annotated constraint logic programming applied to temporal reasoning. In *Programming Language Implementation and Logic Programming (PLILP)*, volume 844 of *Lecture Notes in Computer Science*, pages 230–243. Springer Verlag, 1994.

[15] T. Frühwirth. Temporal Annotated Constraint Logic Programming. *Journal of Symbolic Computation*, 22:555–583, 1996.

[16] T. Frühwirth and S. Abdennadher. *Constraint-Programmierung: Grundlagen und Anwendungen*. Springer, Berlin, 1997.

[17] D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic*. Clarendon Press, Oxford, 1994.

[18] D. M. Gabbay. Modal and temporal logic programming. In A. Galton, editor, *Temporal Logics and Their Applications*, pages 197–237. Academic Press, 1987.

[19] D. M. Gabbay. *Labelled deductive systems : volume 1*, volume 33 of *Oxford logic guides*. Clarendon Press, Oxford, 1996.

[20] D.M. Gabbay and P. McBrien. Temporal Logic & Historical Databases. In *Proceedings of the Seventeenth International Conference on Very Large Databases*, pages 423–430, September 1991.

[21] A. Galton, editor. *Temporal Logics and Their Applications*. Academic Press, 1987.

[22] A. Galton. A critical examination of allen's theory of action and time. *Artificial Intelligence*, 42:159–188, 1990.

[23] P. Hill and J.W. Lloyd. Analysis of Metaprograms. In H.D. Abramson and M.H. Rogers, editors, *Metaprogramming in Logic Programming*, pages 23–52. The MIT Press, 1989.

[24] J. Jaffar and J. L. Lassez. Constraint Logic Programming. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[25] J. Jaffar and M.J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19 & 20:503–582, May 1994.

[26] M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.

[27] S.M. Leach and J.J. Lu. Computing annotated logic programs. In *Proceedings of the eleventh ICLP*, pages 257–271, 1994.

[28] P. Mancarella, A. Raffaetà, and F. Turini. Knowledge Representation with Multiple Logical Theories and Time. *Journal of Experimental and Theoretical Artificial Intelligence*, 11:47–76, 1999.

[29] K. Marriott, J. Jaffar, M.J. Maher, and P.J. Stuckey. The Semantics of Constraint Logic Programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.

[30] K. Marriott and P. J. Stuckey. *Programming with Constraints*. MIT Press, USA, 1998.

[31] B. Martens and D. De Schreye. Why untyped nonground metaprogramming is not (much of) A problem. *Journal of Logic Programming*, 22(1):47–99, January 1995.

[32] B. Moszkowski. *Execution Temporal Logic Programs*. Cambridge University Press, 1986.

[33] M. A. Orgun. On temporal deductive databases. *Computational Intelligence*, 12(2):235–259, May 1996.

[34] M. A. Orgun and W. Ma. An Overview of Temporal and Modal Logic Programming. In *Temporal Logic: Proceedings of the First International Conference, ICTL'94*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 445–479, 1994.

[35] J. Singer. Constraint-Based Temporal Logic Programming. BSc. dissertation, Department of Artificial Intelligence, University of Edinburgh, May 1996.

[36] R. Snodgrass. Temporal Databases. In *Proceedings of the International Conference on GIS - From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 22–64, 1992.

[37] L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, 1986.

[38] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass editors. *Temporal Databases: Theory, Design, and Implementation.* Benjamin/Cummings, 1993.

[39] J. F. A. K. van Benthem. *The logic of time: a model-theoretic investigation into the varieties of temporal ontology and temporal discourse*, volume 156 of *Synthese Library*. Reidel, Dordrecht, 1983.