TWO 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
www.pst.informatik.uni-muenchen.de/~fruehwir/

We investigate the 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. TACLP is given two different kinds of semantics, an operational one based on meta-logic (top-down semantics) and a fixpoint one based on an immediate consequence operator (bottom-up 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 [1,2,3], temporal programming languages [4,5,6,7,8,9,10,11] or temporal databases [12,13,14,15,16,17,18]. 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 [19]. 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 [20]. Based on this framework and on constraint logic programming concepts [21,22,23,24,25], the family of temporal annotated constraint logic programming (TACLP) languages has been developed in [26,27,11,28,29].

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 [11] TACLP is presented as an instance of annotated constraint logic (ACL) for reasoning about time. ACL is a generalization of generalized annotated programs [20,30], 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 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*) [21,22,23,24,25] is an extension of logic programming, where in addition to ordinary 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 full, revised paper of this article contains soundness and completeness proofs relating the two semantics presented here [31].

The paper is organized as follows. Section 2 introduces the TACLP framework. Section 3 defines the two semantics for TACLP. Section 4 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 [11].

An *annotated formula* is of the form $A\,\alpha$ where A is a first order 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 set of time points.

(`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 set 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 set 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 ($\sqcup$). 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. These three inference rules can be merged into a single rule called *A-resolution*:

$$\frac{A\ \alpha \qquad B \qquad (A\ \beta \leftarrow B) \qquad \gamma \sqsubseteq (\alpha \sqcup \beta)}{A\ \gamma} \qquad \textit{(A-Resolution)}$$

Time can be discrete or dense. Time points are totally ordered by the relation $\leq$. We call the set of time points D. 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 $\{t \mid r \leq t \leq s\}$. Thus the interval $[0, \infty]$ denotes the whole time line.

The *constraint theory for temporal annotations* over time points and time periods contains an axiomatization of the total order relation $\leq$ on D and the following axioms defining the partial order on temporal annotations:

$$
\begin{array}{ll}
(\mathtt{at}\,\mathtt{th}) & \mathtt{at}\, t = \mathtt{th}\,[t, t] \\
(\mathtt{at}\,\mathtt{in}) & \mathtt{at}\, t = \mathtt{in}\,[t, t] \\
(\mathtt{th}\ \sqsubseteq) & \mathtt{th}\,[s_1, s_2] \sqsubseteq \mathtt{th}\,[r_1, r_2] \Leftrightarrow r_1 \leq s_1,\ s_1 \leq s_2,\ s_2 \leq r_2 \\
(\mathtt{in}\ \sqsubseteq) & \mathtt{in}\,[r_1, r_2] \sqsubseteq \mathtt{in}\,[s_1, s_2] \Leftrightarrow r_1 \leq s_1,\ s_1 \leq s_2,\ s_2 \leq r_2
\end{array}
$$

The first two axioms state that $\mathtt{th}\, I$ and $\mathtt{in}\, I$ are equivalent to $\mathtt{at}\, t$ when the time period I consists of a single time point t. 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 (($\mathtt{th}\ \sqsubseteq$) 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 (($\mathtt{in}\ \sqsubseteq$) axiom).

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

$$
\begin{array}{c}
\mathtt{in}\,[r_1, r_2] \sqsubseteq \mathtt{in}\,[s_1, s_2] \sqsubseteq \mathtt{in}\,[s_1, s_1] = \mathtt{at}\, s_1 = \\
= \mathtt{th}\,[s_1, s_1] \sqsubseteq \mathtt{th}\,[s_1, s_2] \sqsubseteq \mathtt{th}\,[r_1, r_2]
\end{array}
$$

Now we axiomatize the least upper bound $\sqcup$ of temporal annotations over time points and time periods. As explained in [11], the annotations are not closed under $\sqcup$. From a theoretical point of view, this problem can be overcome via a closure operation which includes in the lattice expressions with $\sqcup$. In practice, it suffices to consider the least upper bound for time periods that produce a different time period. Therefore we can restrict ourselves to $\mathtt{th}$ annotations with overlapping time periods that do not include one another:

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

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 ACL clauses. A *TACLP clause* is a TACLP formula of the form:

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

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

In [28] 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 1 *We model information about the growth of trees.*

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

   $$sprouts(Tree1)\ \mathtt{at}\ 3.5.$$

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 $h = (t - s) \times r$.*

   $$\begin{aligned} &height(tree, h)\ \mathtt{at}\ t \leftarrow \\ &\qquad h = (t - s) \times r, \\ &\qquad tree\_type(tree, type), growth\_rate(type, r), \\ &\qquad sprouts(tree)\ \mathtt{at}\ s \end{aligned}$$

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

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

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

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

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

The query $mature(Tree1)\ \mathtt{th}\ [t_1, t_2]$ *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.

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 $\mathtt{th}\,[t,t]$ by exploiting the ($\mathtt{at}\,\mathtt{th}$) 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 $\mathtt{th}\,[0,\infty]$. Constraints stay unchanged.

3.1 Operational Semantics via Meta-Interpreter

The *vanilla* meta-interpreter [32] 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.

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

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:

$$demo(Empty). \tag{1}$$

$$demo((b_1,b_2)) \leftarrow demo(b_1), demo(b_2) \tag{2}$$

$$\begin{aligned} demo(a\ \mathtt{th}\,[t_1,t_2]) \leftarrow\ & s_1 \le t_1, t_2 \le s_2, t_1 \le t_2,\\ & clause(a\ \mathtt{th}\,[s_1,s_2], b), demo(b) \end{aligned} \tag{3}$$

$$demo(a\ \mathtt{th}\,[t_1, t_2]) \leftarrow s_1 \leq t_1, t_1 < s_2, s_2 < t_2,$$
$$clause(a\ \mathtt{th}\,[s_1, s_2], b), demo(b), demo(a\ \mathtt{th}\,[s_2, t_2]) \tag{4}$$

$$demo(a\ \mathtt{in}\,[t_1, t_2]) \leftarrow t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2,$$
$$clause(a\ \mathtt{th}\,[s_1, s_2], b), demo(b) \tag{5}$$

$$demo(a\ \mathtt{in}\,[t_1, t_2]) \leftarrow t_1 \leq s_1, s_2 \leq t_2,$$
$$clause(a\ \mathtt{in}\,[s_1, s_2], b), demo(b) \tag{6}$$

$$demo(c) \leftarrow constraint(c), c \tag{7}$$

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

$$clause(A\,\alpha, B) \leftarrow t_1 \leq t_2. \tag{8}$$

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 (3), or can come from the resolution steps. The latter kind of constraints is managed by clause (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 the more powerful A-resolution rule, which is the combination of Modus Ponens itself with rule ($\sqsubseteq$) and rule ($\sqcup$). This is the reason why the third *demo* clause of the vanilla meta-interpreter is now split into four clauses. Clauses (3), (5) and (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 (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 (5) and (6) are built in an analogous way by exploiting axioms ($\mathtt{in}\,\mathtt{th}\ \sqsubseteq$) and ($\mathtt{in}\ \sqsubseteq$), respectively.
Rule ($\sqcup$) is implemented by clause (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 (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 $\mathtt{th}\, [t_1, t_2] \sqsubseteq \mathtt{th}\, [s_1, t_2]$. 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 (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 2 *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 (4), we can 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 (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.*

In [11] a compiler for TACLP has been defined by means of a compilation function comp which translates an annotated formula into its *CLP* form. The essential step is the inclusion of the temporal annotation of an atom in the corresponding predicate as an extra-argument.

$$\mathsf{comp}(p(t_1, \ldots, t_n)\, \alpha) = p(t_1, \ldots, t_n, \alpha).$$

Now we can basically read off the other rules of the translation function comp directly from the meta-interpreter defined in the previous section.

A constraint C is compiled into itself, i.e., $\mathsf{comp}(C) = C$, and a conjunction of formulae is compiled into the conjunction of the compiled version of such formulae, i.e. $\mathsf{comp}(B_1, B_2) = \mathsf{comp}(B_1), \mathsf{comp}(B_2)$.

Finally, the compilation of a program clause is defined in the following way:

- for each clause of the form $A\, \mathtt{th}\, [s_1, s_2] \leftarrow B$ the compiler generates three clauses

– $\mathsf{comp}(A\,\mathtt{th}\,[t_1,t_2]) \leftarrow s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2,\ \mathsf{comp}(B)$

– $\mathsf{comp}(A\,\mathtt{th}\,[t_1,t_2]) \leftarrow s_1 \leq t_1, t_1 < s_2, s_2 < t_2,\ \mathsf{comp}(B),$
$\mathsf{comp}(A\,\mathtt{th}\,[s_2,t_2])$

– $\mathsf{comp}(A\,\mathtt{in}\,[t_1,t_2]) \leftarrow t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2, s_1 \leq s_2,\ \mathsf{comp}(B)$

- for each clause of the form $A\,\mathtt{in}\,[s_1,s_2] \leftarrow B$ the compiler generates a clause

  – $\mathsf{comp}(A\,\mathtt{in}\,[t_1,t_2]) \leftarrow t_1 \leq s_1, s_1 \leq s_2, s_2 \leq t_2,\ \mathsf{comp}(B)$

The result of the compilation is a standard *CLP* program.

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 $\wp(\mathcal{A}\text{-}base^{a} \times Ann)$ with set-theoretic inclusion, disregarding the partial order structure of the set of annotations Ann. Alternative solutions (as for generalized annotated programs in [20]) may consider a more abstract domain, which is obtained by endowing $\mathcal{A}\text{-}base \times Ann$ with the product order (induced by the discrete order on $\mathcal{A}\text{-}base$ 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

[a] 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.

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.

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 $\mathcal{A}\text{-}base_P$, 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 [23] for a $CLP(\mathcal{A})$ program P is a function $T_P^{\mathcal{A}} : \wp(\mathcal{A}\text{-}base_P) \to \wp(\mathcal{A}\text{-}base_P)$ 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 [23], 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.[b] 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 $(\wp(\mathcal{A}\text{-}base_P \times Ann), \subseteq)$ where $\wp$ is the powerset operator and $\subseteq$ is the usual relation of set-theoretic inclusion.

Definition 1 *Let P be a TACLP program, the function $\mathcal{T}_P^{\mathcal{A}} : \wp(\mathcal{A}\text{-}base_P \times Ann) \to \wp(\mathcal{A}\text{-}base_P \times Ann)$ is defined as follows.*

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

$\mathcal{T}_P^{\mathcal{A}}(I) =$

$$\left\{ (A,\alpha) \mid \begin{array}{l} (\alpha = \mathtt{th}[s_1,s_2] \ \vee \ \alpha = \mathtt{in}[s_1,s_2]) \\ A \ \alpha \leftarrow C_1,\ldots,C_k, B_1\alpha_1,\ldots,B_n\alpha_n \in \mathit{ground}_{\mathcal{A}}(P), \\ \{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq I, \\ \mathcal{A} \models C_1,\ldots,C_k, \alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, s_1 \leq s_2 \end{array} \right\}$$

$\cup$

$$\left\{ (A,\mathtt{th}[s_1,r_2]) \mid \begin{array}{l} A \ \mathtt{th}[s_1,s_2] \leftarrow C_1,\ldots,C_k, B_1\alpha_1,\ldots,B_n\alpha_n \in \mathit{ground}_{\mathcal{A}}(P), \\ \{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq I, \ (A,\mathtt{th}[r_1,r_2]) \in I, \\ \mathcal{A} \models C_1,\ldots,C_k, \alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, s_1 < r_1, r_1 \leq s_2, \\ \qquad s_2 < r_2 \end{array} \right\}$$

$\cup$

$$\left\{ (A,\mathtt{in}[t_1,t_2]) \mid \begin{array}{l} A \ \mathtt{th}[s_1,s_2] \leftarrow C_1,\ldots,C_k, B_1\alpha_1,\ldots,B_n\alpha_n \in \mathit{ground}_{\mathcal{A}}(P), \\ \{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq I, \\ \mathcal{A} \models C_1,\ldots,C_k, \alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, t_1 \leq s_2, s_1 \leq t_2, \\ \qquad t_1 \leq t_2, s_1 \leq s_2 \end{array} \right\}$$

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 A-resolution rule which combines Modus Ponens with $(\sqcup)$ and $(\sqsubseteq)$ rules 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 1 (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 full version of the paper [31]. $\square$

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 1 is the least upper bound of the chain $\{(\mathcal{T}_P^{\mathcal{A}})^i\}_{i \geq 0}$.

Definition 2 *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 $(\mathcal{T}_P^{\mathcal{A}})^{\omega} = \bigcup_{i \geq 0} (\mathcal{T}_P^{\mathcal{A}})^i$.

4 Related Work

In [20], Templog [8] 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 [30]. 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 [30] the intermediate stages of the computation are not sound with respect to the standard *CLP* semantics.

Moreover, in [20] two fixpoint semantics, defined in terms of two different operators, are presented for generalized annotated programs (GAP). The first operator, 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 operator 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 [33], which is a critical examination of Allen's classical work on a theory of action and time [34]. 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 $A\,\mathtt{in}\,I$, *holds-on(A,I)* into $A\,\mathtt{th}\,I$, and *holds-at(A,t)* into $A\,\mathtt{at}\,t$, where A is an atomic formula.

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. We defined the operational (top-down) semantics of TACLP by presenting a meta-interpreter for it. Then we provided TACLP with a fixpoint (bottom-up) semantics, based on the definition of an immediate consequence operator.

Here 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 [11], 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 how the semantics can be adapted to deal with negation.

Acknowledgments

We thank Paolo Baldan and Roberta Gori for their useful comments and suggestions. This work has been partially supported by Esprit Working Group 28115 - DeduGIS.

References

1. 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.
2. A. Galton, editor. *Temporal Logics and Their Applications*. Academic Press, 1987.
3. D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic*. Clarendon Press, Oxford, 1994.

4. B. Moszkowski. *Execution Temporal Logic Programs.* Cambridge University Press, 1986.
5. 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.
6. D. M. Gabbay. Modal and temporal logic programming. In A. Galton, editor, *Temporal Logics and Their Applications*, pages 197–237. Academic Press, 1987.
7. W.W. Wadge. Tense Logic Programming: a Respectable Alternative. In *Proceedings of the 1988 International Symposium on Lucid and Intensional Programming*, pages 26–32, 1988.
8. M. Abadi and Z. Manna. Temporal logic programming. In *Journal of Symbolic Computation*, volume 8, pages 277–295, 1989.
9. C. Brzoska. Temporal logic programming with metric and past operators. In [10], pages 21–39, 1995.
10. M. Fisher and R. Owens, editors. *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence.* Springer, 1995.
11. T. Frühwirth. Temporal Annotated Constraint Logic Programming. *Journal of Symbolic Computation*, 22:555–583, 1996.
12. 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.
13. A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass editors. *Temporal Databases: Theory, Design, and Implementation.* Benjamin/Cummings, 1993.
14. M. Baudinet, J. Chomicki, and P. Wolper. Temporal Deductive Databases. In [13], pages 294–320. 1993.
15. 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.
16. 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, 1994.
17. 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.
18. M. A. Orgun. On temporal deductive databases. *Computational Intelligence*, 12(2):235–259, May 1996.

19. D. M. Gabbay. *Labelled deductive systems : volume 1*, volume 33 of *Oxford logic guides*. Clarendon Press, Oxford, 1996.
20. M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
21. 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.
22. J. Jaffar and M.J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19 & 20:503–582, May 1994.
23. 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.
24. K. Marriott and P. J. Stuckey. *Programming with Constraints*. MIT Press, USA, 1998.
25. T. Frühwirth and S. Abdennadher. *Constraint-Programmierung: Grundlagen und Anwendungen*. Springer, Berlin, 1997.
26. 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.
27. T. Frühwirth. Temporal logic and annotated constraint logic programming. In [10], pages 58–68, 1995.
28. J. Singer. Constraint-Based Temporal Logic Programming. BSc. dissertation, Department of Artificial Intelligence, Univ. of Edinburgh, 1996.
29. P. Mancarella, A. Raffaetà, and F. Turini. Temporal Annotated Constraint Logic Programming with Multiple Theories. In *Proceedings of the Tenth International Workshop on Database and Expert Systems Applications*, pages 501–508. IEEE Computer Society Press, 1999.
30. S.M. Leach and J.J. Lu. Computing annotated logic programs. In *Proceedings of the eleventh ICLP*, pages 257–271, 1994.
31. A. Raffaetà and T. Frühwirth. Semantics for Temporal Annotated Constraint Logic Programming. In D. Basin, M. D'Agostino, D. Gabbay, S. Matthews and L. Vigano, editors, *Labelled Deduction*, Applied Logic Series, Kluwer Academic Publishers. To appear.
32. L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, 1986.
33. A. Galton. A critical examination of allen's theory of action and time. *Artificial Intelligence*, 42:159–188, 1990.
34. J.F. Allen. Towards a general theory of action and time. In *Artificial Intelligence*, volume 23, pages 123–154, 1984.