Università di Pisa

Dipartimento di Informatica

Technical Report: TR-01-22

MuTACLP: A language for temporal reasoning with multiple theories

P. Baldan, P. Mancarella, A. Raffaetà and F. Turini

Dipartimento di Informatica, Università di Pisa

Corso Italia, 40, I-56125 Pisa, Italy

{baldan,p.mancarella,raffaeta,turini}@di.unipi.it

October 23, 2001

ADDRESS: Corso Italia 40, 56125 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

MuTACLP: A language for temporal reasoning with multiple theories

P. Baldan, P. Mancarella, A. Raffaetà and F. Turini

Dipartimento di Informatica, Università di Pisa

Corso Italia, 40, I-56125 Pisa, Italy

{baldan,p.mancarella,raffaeta,turini}@di.unipi.it

October 23, 2001

Abstract

In this paper we introduce MuTACLP, a knowledge representation language which provides facilities for modeling and handling temporal information, together with some basic operators for combining different temporal knowledge bases. The proposed approach stems from two separate lines of research: the general studies on meta-level operators on logic programs introduced by Brogi et al. [6, 8] and Temporal Annotated Constraint Logic Programming (TACLP) defined by Frühwirth [14]. In MuTACLP atoms are annotated with temporal information which are managed via a constraint theory, as in TACLP. Mechanisms for structuring programs and combining separate knowledge bases are provided through meta-level operators. The language is given two different and equivalent semantics, a top-down semantics which exploits meta-logic, and a bottom-up semantics based on an immediate consequence operator.

1 Introduction

Interest in research concerning the handling of temporal information has been growing steadily over the past two decades. On the one hand, much effort has been spent in developing extensions of logic languages capable to deal with time (see, e.g., [13, 35]). On the other hand, in the field of databases, many approaches have been proposed to extend existing data models, such as the *relational*, the *object-oriented* and the *deductive* models, to cope with temporal data (see, e.g., the books [45, 12] and references therein). Clearly these two strands of research are closely related, since temporal logic languages can provide solid theoretical foundations for temporal databases, and powerful knowledge representation and query languages for them [10, 16, 34]. Another basic motivation for our work is the need of mechanisms for combining pieces of knowledge which may be separated into various knowledge bases (e.g., distributed over the web), and thus which have to be merged together to reason with.

This paper aims at building a framework where temporal information can be naturally represented and handled, and, at the same time, knowledge can be separated and combined by means of meta-level composition operators. Concretely, we introduce a new language, called *MuTACLP*, which is based on Temporal Annotated Constraint Logic Programming (TACLP), a powerful framework defined by Frühwirth in [14], where temporal information and reasoning can be naturally formalized. Temporal information is represented by *temporal annotations* which say at what time(s) the formula to which they are attached is valid. Such annotations make time explicit but avoid the proliferation of temporal variables and quantifiers of the first-order approach. In this way, MuTACLP supports quantitative temporal reasoning and allows one to represent definite, indefinite and periodic temporal information, and to work both with time points and time periods (time intervals). Furthermore, as a mechanism for structuring programs and combining different knowledge sources, MuTACLP offers a set of program composition operators in the style of Brogi et al. [6, 8].

Concerning the semantical aspects, the use of meta-logic allows us to provide MuTACLP with a formal and, at the same time, executable top-down semantics based on a meta-interpreter. Furthermore the language is given a bottom-up semantics by introducing an immediate consequence operator which generalizes the operator for ordinary constraint logic programs. The two semantics are equivalent in the sense that the meta-interpreter can be proved sound and complete with respect to the semantics based on the immediate consequence operator.

An interesting aspect of MuTACLP is the fact that it integrates modularity and temporal reasoning, a feature which is not common to logical temporal languages (e.g., it is lacking in [1, 2, 9, 11, 14, 15, 20, 27]). Two exceptions are the language Temporal Datalog by Orgun [34] and the work on amalgamating knowledge bases by Subrahmanian [44]. Temporal Datalog introduces a concept of module, which, however, seems to be used as a means for defining new non-standard algebraic operators, rather than as a knowledge representation tool. On the other hand, the work on amalgamating knowledge bases offers a multi-theory framework, based on *annotated logics*, where temporal information can be handled, but only a limited interaction among the different knowledge sources is allowed: essentially a kind of message passing mechanism allows one to delegate the resolution of an atom to other databases.

In the database field, our approach is close to the paradigm of constraint databases [24, 26]. In fact, in MuTACLP the use of constraints allows one to model temporal information and to enable efficient implementations of the language. Moreover, from a deductive database perspective, each constraint logic program of our framework can be viewed as an enriched relational database where relations are represented partly intensionally and partly extensionally. The meta-level operators can then be considered as a means of constructing views by combining different databases in various ways.

The paper is organized as follows. Section 2 briefly introduces the program composition operators for combining logic theories of [6, 8] and their semantics.

Section 3, after reviewing the basics of constraint logic programming, introduces the language TACLP. Section 4 defines the new language MuTACLP, which integrates the basic ideas of TACLP with the composition operators on theories. In Section 5 the language MuTACLP is given a top-down semantics by means of a meta-interpreter and a bottom-up semantics based on an immediate consequence operator, and the two semantics are shown to be equivalent. Section 6 presents some examples to clarify the use of operators on theories and to show the expressive power and the knowledge representation capabilities of the language. Section 7 compares MuTACLP with some related approaches in the literature and, finally, Section 8 outlines our future research plans. Proofs of propositions and theorems are collected in the Appendix. An extended abstract of this paper has been presented at the International Workshop on Spatio-Temporal Data Models and Languages [32].

2 Operators for combining theories

Composition operators for logic programs have been thoroughly investigated in [6, 8], where both their meta-level and their bottom-up semantics are studied and compared. In order to illustrate the basic notions and ideas of such an approach this section describes the meta-level definition of the operators, which is simply obtained by adding new clauses to the well-known vanilla meta-interpreter for logic programs. The resulting meta-interpreter combines separate programs without actually building a new program. Its meaning is straightforward and, most importantly, the meta-logical definition shows that the multi-theory framework can be expressed from inside logic programming itself. We consider two operators to combine programs: union $\cup$ and intersection $\cap$. Then the so-called *program expressions* are built by starting from a set of *plain* programs, consisting of collections of clauses, and by repeatedly applying the composition operators. Formally, the language of *program expressions* Exp is defined by the following abstract syntax:

$$Exp ::= Pname \mid Exp \cup Exp \mid Exp \cap Exp$$

where $Pname$ is the syntactic category of constant names for plain programs.

Following [5], the two-argument predicate *demo* is used to represent provability. Namely, $demo(\mathcal{E}, G)$ means that the formula G is provable with respect to the program expression $\mathcal{E}$.

$$\begin{array}{rcl} demo(\mathcal{E}, empty). & & \\ demo(\mathcal{E}, (B_1, B_2)) & \leftarrow & demo(\mathcal{E}, B_1),\ demo(\mathcal{E}, B_2) \\ demo(\mathcal{E}, A) & \leftarrow & clause(\mathcal{E}, A, B),\ demo(\mathcal{E}, B) \end{array}$$

The unit clause states that the empty goal, represented by the constant symbol *empty*, is solved in any program expression $\mathcal{E}$. The second clause deals with conjunctive goals. It states that a conjunction (B_1, B_2) is solved in the program expression $\mathcal{E}$ if B_1 is solved in $\mathcal{E}$ and B_2 is solved in $\mathcal{E}$. Finally, the third clause deals with the case of atomic goal reduction. To solve an atomic goal A, a clause with head A is chosen from the program expression $\mathcal{E}$ and the body of the clause is recursively solved in $\mathcal{E}$.

We adopt the simple naming convention used in [28]. Object programs are named by constant symbols, denoted by capital letters like P and Q. Object level expressions are represented at the meta-level by themselves. In particular, object level variables are denoted by meta-level variables, according to the so-called *non-ground representation*. An object level program P is represented, at the meta-level, by a set of axioms of the kind $clause(P, A, B)$, one for each object level clause $A \leftarrow B$ in the program P.

Each program composition operator is represented at the meta-level by a functor, whose meaning is defined by adding new clauses to the above meta-interpreter.

$$\begin{array}{rcl} clause(\mathcal{E}_1 \cup \mathcal{E}_2, A, B) & \leftarrow & clause(\mathcal{E}_1, A, B) \\ clause(\mathcal{E}_1 \cup \mathcal{E}_2, A, B) & \leftarrow & clause(\mathcal{E}_2, A, B) \\ clause(\mathcal{E}_1 \cap \mathcal{E}_2, A, (B_1, B_2)) & \leftarrow & clause(\mathcal{E}_1, A, B_1), \\ & & clause(\mathcal{E}_2, A, B_2) \end{array}$$

The added clauses have a straightforward interpretation. Informally, union and intersection mirror two forms of cooperation among program expressions. In the case of union $\mathcal{E}_1 \cup \mathcal{E}_2$, whose meta-level implementation is defined by the first two clauses, either expression $\mathcal{E}_1$ or $\mathcal{E}_2$ may be used to perform a computation step. For instance, a clause $A \leftarrow B$ belongs to the meta-level representation of $\mathcal{P} \cup \mathcal{Q}$ if it belongs either to the meta-level representation of $\mathcal{P}$ or to the meta-level representation of $\mathcal{Q}$. In the case of intersection $\mathcal{E}_1 \cap \mathcal{E}_2$, both expressions must agree to perform a computation step. This is expressed by the third clause, which exploits the basic unification mechanism of logic programming and the non-ground representation of object level programs.

A program expression $\mathcal{E}$ can be queried by $demo(\mathcal{E}, G)$, where G is an object level goal.

3 Temporal Annotated CLP

In this section we first briefly recall the basic concepts of Constraint Logic Programming (CLP). Then we give an overview of Temporal Annotated CLP (TACLP), an extension of CLP suited to deal with time, which will be used as a basic language for plain programs in our multi-theory framework. The reader is referred to the survey of Jaffar and Maher [21] for a comprehensive introduction to the motivations, foundations, and applications of CLP languages, and to the recent work of Jaffar et al. [22] for the formal presentation of the semantics. A good reference for TACLP is Frühwirth's paper [14].

3.1 Constraint Logic Programming

A CLP language is completely determined by its constraint domain. A *constraint domain* $\mathcal{C}$ is a tuple $\langle S_\mathcal{C}, \mathcal{L}_\mathcal{C}, \mathcal{D}_\mathcal{C}, \mathcal{T}_\mathcal{C}, solv_\mathcal{C}\rangle$, where

- $S_\mathcal{C} = \langle \Sigma_\mathcal{C}, \Pi_\mathcal{C}\rangle$ is the constraint domain *signature*, comprising the function symbols $\Sigma_\mathcal{C}$ and the predicate symbols $\Pi_\mathcal{C}$.

- $\mathcal{L}_\mathcal{C}$ is the class of *constraints*, a set of first-order $S_\mathcal{C}$-formulae, denoted by C, possibly subscripted.
- $\mathcal{D}_\mathcal{C}$ is the *domain of computation*, a $S_\mathcal{C}$-structure which provides the intended interpretation of the constraints. The domain (or support) of $\mathcal{D}_\mathcal{C}$ is denoted by $D_\mathcal{C}$.
- $\mathcal{T}_\mathcal{C}$ is the *constraint theory*, a $S_\mathcal{C}$-theory describing the logical semantics of the constraints.
- $solv_\mathcal{C}$ is the *constraint solver*, a (computable) function which maps each formula in $\mathcal{L}_\mathcal{C}$ to either *true*, or *false*, or *unknown*, indicating that the formula is satisfiable, unsatisfiable or it cannot be told, respectively.

We assume that $\Pi_\mathcal{C}$ contains the predicate symbol "=", interpreted as identity in $\mathcal{D}_\mathcal{C}$. Furthermore we assume that $\mathcal{L}_\mathcal{C}$ contains all atoms constructed from "=", the always satisfiable constraint `true` and the unsatisfiable constraint `false`, and that $\mathcal{L}_\mathcal{C}$ is closed under variable renaming, existential quantification and conjunction. A *primitive* constraint is an atom of the form $p(t_1, \ldots, t_n)$ where p is a predicate in $\Pi_\mathcal{C}$ and $t_1, \ldots, t_n$ are terms on $\Sigma_\mathcal{C}$.

We assume that the solver does not take variable names into account. Also, the domain, the theory and the solver *agree* in the sense that $\mathcal{D}_\mathcal{C}$ is a model of $\mathcal{T}_\mathcal{C}$ and for every $C \in \mathcal{L}_\mathcal{C}$:

- $solv_\mathcal{C}(C) = true$ implies $\mathcal{T}_\mathcal{C} \models \exists C$, and
- $solv_\mathcal{C}(C) = false$ implies $\mathcal{T}_\mathcal{C} \models \neg\exists C$.

Example 1 (Real) The constraint domain *Real* has `<`, `<=`, `=`, `>=`, `>` as predicate symbols, `+`, `-`, `*`, `/` as function symbols and sequences of digits (possibly with a decimal point) as constant symbols. Examples of primitive constraints are `X + 3 <= Y * 1.1` and `X/2 > 10`. The domain of computation is the structure with reals as domain, and where the predicate symbols `<`, `<=`, `=`, `>=`, `>` and the function symbols `+`, `-`, `*`, `/` are interpreted as the usual relations and functions over reals. Finally, the theory $\mathcal{T}_{Real}$ is the theory of real closed fields.

A possible constraint solver is provided by the CLP($\mathcal{R}$) system [23], which relies on Gauss-Jordan elimination to handle linear constraints. Non-linear constraints are not taken into account by the solver (i.e., their evaluation is delayed) until they become linear.

Example 2 (Logic Programming) The constraint domain *Term* has $=$ as predicate symbol and strings of alphanumeric characters as function or constant symbols. The domain of computation of *Term* is the set *Tree* of *finite trees* (or, equivalently, of finite terms), while the theory $\mathcal{T}_{Term}$ is Clark's equality theory.

The interpretation of a constant is a tree with a single node labeled by the constant. The interpretation of an n-ary function symbol f is the function $f_{Tree} : Tree^n \rightarrow Tree$ mapping the trees $t_1, \ldots, t_n$ to a new tree with root labeled by f and with $t_1, \ldots, t_n$ as children.

A constraint solver is given by the *unification* algorithm. Then CLP(*Term*) coincides with logic programming.

For a given constraint domain $\mathcal{C}$, we denote by CLP($\mathcal{C}$) the CLP language based on $\mathcal{C}$. Our results are parametric to a language L in which all programs and queries under consideration are included. The set of function symbols in L, denoted by Σ_L, coincides with $\Sigma_{\mathcal{C}}$, while the set of predicate symbols Π_L includes $\Pi_{\mathcal{C}}$.

A *constraint logic program*, or simply a *program*, is a finite set of rules of the form:

$$A \leftarrow C_1, \ldots, C_n, B_1, \ldots, B_m$$

where A and $B_1, \ldots, B_m$ ($m \geq 0$) are atoms (whose predicate symbols are in Π_L but not in $\Pi_{\mathcal{C}}$), and $C_1, \ldots, C_n$ ($n \geq 0$) are primitive constraints[1] (A is called the *head* of the clause and $C_1, \ldots, C_n, B_1, \ldots, B_m$ the *body* of the clause). If $m = 0$ then the clause is called a *fact*. A query is a sequence of atoms and/or constraints.

Interpretations and Fixpoints.

A $\mathcal{C}$-interpretation for a CLP($\mathcal{C}$) program is an interpretation which agrees with $\mathcal{D}_{\mathcal{C}}$ on the interpretations of the symbols in $\mathcal{L}_{\mathcal{C}}$. Formally, a *$\mathcal{C}$-interpretation* I is a subset of $\mathcal{C}$-$base_L$, i.e. of the set

$$\{p(d_1, \ldots, d_n) \mid p \text{ predicate in } \Pi_L \setminus \Pi_{\mathcal{C}},\ d_1, \ldots, d_n \in D_{\mathcal{C}}\}.$$

Note that the meaning of primitive constraints is not specified, being fixed by $\mathcal{C}$.

The notions of $\mathcal{C}$-model and least $\mathcal{C}$-model are a natural extension of the corresponding logic programming concepts. A valuation σ is a function that maps variables into $D_{\mathcal{C}}$. A $\mathcal{C}$-ground instance A' of an atom A is obtained by applying a valuation σ to the atom, thus producing a construct of the form $p(a_1, \ldots, a_n)$ with $a_1, \ldots, a_n$ elements in $D_{\mathcal{C}}$. $\mathcal{C}$-ground instances of queries and clauses are defined in a similar way. We denote by $ground_{\mathcal{C}}(P)$ the set of $\mathcal{C}$-ground instances of clauses from P.

Finally the immediate consequence operator for a CLP($\mathcal{C}$) program P is a function $T_P^{\mathcal{C}} : \wp(\mathcal{C}\text{-}base_L) \rightarrow \wp(\mathcal{C}\text{-}base_L)$ defined as follows:

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

The operator $T_P^{\mathcal{C}}$ is continuous, and therefore it has a least fixpoint which can be computed as the least upper bound of the ω-chain $\{(T_P^{\mathcal{C}})^i\}_{i \geq 0}$ of the iterated applications of $T_P^{\mathcal{C}}$ starting from the empty set, i.e., $(T_P^{\mathcal{C}})^\omega = \bigcup_{i \in \mathbb{N}} (T_P^{\mathcal{C}})^i$.

[1] Constraints and atoms can be in any position inside the body of a clause, although, for the sake of simplicity, we will always assume that the sequence of constraints precedes the sequence of atoms.

3.2 Temporal Annotated Constraint Logic Programming

Temporal Annotated Constraint Logic Programming (TACLP), proposed by Frühwirth in [14, 38], has been shown to be a natural and powerful framework for formalizing temporal information and reasoning. In [14] TACLP is presented as an instance of annotated constraint logic (ACL) suited for reasoning about time. ACL, which can be seen as an extension of generalized annotated programs [25, 29], generalizes basic 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. An advantage of the languages in the ACL framework is that their 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.

We next summarize the syntax and semantics of TACLP. As mentioned above, TACLP is a constraint logic programming language where formulae can be annotated with temporal labels and where relations between these labels can be expressed by using constraints. In TACLP the choice of the temporal ontology is free. In this paper, we will consider the instance of TACLP where time points are totally ordered and labels involve convex, non-empty sets of time points. Moreover we will assume that only atomic formulae can be annotated and that clauses are negation free. With an abuse of notation, in the rest of the paper such a subset of the language will be referred to simply as TACLP.

Time can be discrete or dense. Time points are totally ordered by the relation $\leq$. We denote by D the set of time points and we suppose to have a set of operations (such as the binary operations $+, -$) to manage such points. 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 $r, s \in D$ and $0 \leq r \leq s \leq \infty$, which represents the convex, non-empty set of time points $\{t \mid r \leq t \leq s\}$[2]. 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 time points and on time periods. Let t be a time point and $J = [r, s]$ be a time period.

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

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

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

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

[2]The results we present 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.

$\texttt{in}$-annotated formula in terms of $\texttt{at}$ is:

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

The $\texttt{in}$ 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 ($\sqcup$).

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

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 α and the same formula holds with another annotation β then it holds with the least upper bound $\alpha \sqcup \beta$ of the two annotations.

Next, we introduce the *constraint theory for temporal annotations*. Recall that a constraint theory is a non-empty, consistent first order theory that axiomatizes the meaning of the constraints. Besides an axiomatization of the total order relation $\leq$ on the set of time points D, the constraint theory includes the following axioms defining the partial order on temporal annotations.

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

The first two axioms state that $\texttt{th}\,I$ and $\texttt{in}\,I$ are equivalent to $\texttt{at}\,t$ when the time period I consists of a single time point t.[3] 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). A consequence of the above axioms is

$$(\texttt{in}\,\texttt{th}\ \sqsubseteq) \qquad \texttt{in}\,[s_1,s_2] \sqsubseteq \texttt{th}\,[r_1,r_2] \Leftrightarrow s_1 \leq r_2, r_1 \leq s_2, s_1 \leq s_2, r_1 \leq r_2$$

i.e., an atom annotated by $\texttt{in}$ holds in any time period that overlaps with a time period where the atom holds throughout.

To summarize the above explanation, the axioms defining the partial order relation on annotations can be arranged in the following chain, where it is assumed that $r_1 \leq s_1,\ s_1 \leq s_2,\ s_2 \leq r_2$:

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

[3] Especially in dense time, one may disallow singleton periods and drop the two axioms. This restriction has no effects on the results we are presenting.

Before giving an axiomatization of the least upper bound $\sqcup$ on temporal annotations, let us recall that, as explained in [14], the least upper bound of two annotations always exists but sometimes it may be "too large". In fact, rule ($\sqcup$) is correct only if the lattice order ensures $A\,\alpha \wedge A\,\beta \wedge (\gamma = \alpha \sqcup \beta) \implies A\,\gamma$ whereas, in general, this is not true in our case. For instance, according to the lattice, $\mathtt{th}\,[1,2] \sqcup \mathtt{th}\,[4,5] = \mathtt{th}\,[1,5]$, but according to the definition of th-annotated formulae in terms of at , the conjunction $A\,\mathtt{th}\,[1,2] \wedge A\,\mathtt{th}\,[4,5]$ does not imply $A\,\mathtt{th}\,[1,5]$, since it does not express that $A\,\mathtt{at}\,3$ holds. From a theoretical point of view, this problem can be overcome by enriching the lattice of annotations with expressions involving $\sqcup$. In practice, it suffices to consider the least upper bound for time periods that produce another *different meaningful* time period. Concretely, one restricts to th annotations with overlapping time periods that do not include one another:

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

Summarizing, a constraint domain for *time points* is fixed where the signature includes suitable constants for time points, function symbols for operations on time points (e.g., $+, -, \ldots$) and the predicate symbol $\leq$, modeling the total order relation on time points. Such constraint domain is extended to a constraint domain $\mathcal{A}$ for handling *annotations*, by enriching the signature with function symbols $[\cdot,\cdot], \mathtt{at}, \mathtt{th}, \mathtt{in}, \sqcup$ and the predicate symbol $\sqsubseteq$, axiomatized as described above. Then, as for ordinary constraint logic programming, a TACLP language is determined by fixing a constraint domain $\mathcal{C}$, which is required to contain the constraint domain $\mathcal{A}$ for annotations. We denote by TACLP($\mathcal{C}$) the TACLP language based on $\mathcal{C}$. To lighten the notation, in the following, the "$\mathcal{C}$" will be often omitted.

The next definition introduces the clausal fragment of TACLP that can be used as an efficient temporal programming language.

Definition 3 *A* TACLP clause *is 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 constraints and the B_i's are atomic formulae. Constraints C_j cannot be annotated.

A TACLP program *is a finite set of TACLP clauses.*

4 Multi-theory Temporal Annotated Constraint Logic Programming

A first attempt to extend the multi-theory framework introduced in Section 2 to handle temporal information is presented in [31]. In that paper an object level program is a collection of annotated logic programming clauses, named by a constant symbol. An annotated clause is of the kind $A \leftarrow B_1, \ldots, B_n \;\Box\; [a, b]$ where the annotation $[a, b]$ represents the period of time in which the clause holds. The handling of time is hidden at the object level and it is managed at the meta-level by intersecting or joining the intervals associated with clauses. However, this approach is not completely satisfactory, in that it does not offer mechanisms for modeling indefinite temporal information and for handling periodic data. Moreover, some problems arise when we want to extract temporal information from the intervals at the object level.

To obtain a more expressive language, where in particular the mentioned deficiencies are overcome, in this paper we consider a multi-theory framework where object level programs are taken from Temporal Annotated Constraint Logic Programming (TACLP) and the composition operators are generalized to deal with temporal annotated constraint logic programs. The resulting language, called Multi-theory Temporal Annotated Constraint Logic Programming (MuTACLP for short), thus arises as a synthesis of the work on composition operators for logic programs and of TACLP. It can be thought of both as a language which enriches TACLP with high-level mechanisms for structuring programs and for combining separate knowledge bases, and as an extension of the language of program expressions with constraints and with time-representation mechanisms based on temporal annotations for atoms.

The language of program expressions remains formally the same as the one in Section 2. However now plain programs, named by the constant symbols in *Pname*, are TACLP programs as defined in Section 3.2.

Also the structure of the time domain remains unchanged, whereas, to deal with program composition, the constraint theory presented in Section 3.2 is enriched with the axiomatization of the greatest lower bound $\sqcap$ of two annotations:

$$(\mathtt{th}\sqcap)\quad \mathtt{th}[s_1,s_2]\sqcap\mathtt{th}[r_1,r_2]=\mathtt{th}[t_1,t_2] \Leftrightarrow s_1\le s_2, r_1\le r_2, t_1=max\{s_1,r_1\}, t_2=min\{s_2,r_2\}, t_1\le t_2$$

$$(\mathtt{th}\sqcap')\quad \mathtt{th}[s_1,s_2]\sqcap\mathtt{th}[r_1,r_2]=\mathtt{in}[t_2,t_1] \Leftrightarrow s_1\le s_2, r_1\le r_2, t_1=max\{s_1,r_1\}, t_2=min\{s_2,r_2\}, t_2< t_1$$

$$(\mathtt{th\,in}\sqcap)\quad \mathtt{th}[s_1,s_2]\sqcap\mathtt{in}[r_1,r_2]=\mathtt{in}[r_1,r_2] \Leftrightarrow s_1\le r_2, r_1\le s_2, s_1\le s_2, r_1\le r_2$$

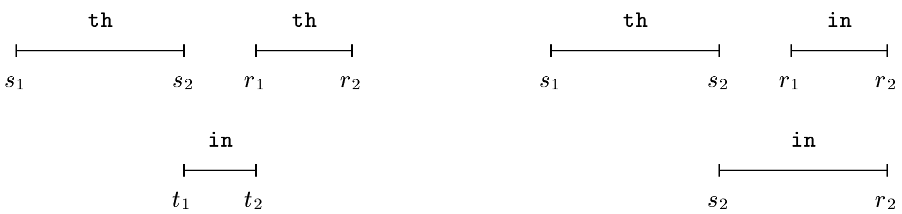
$$(\mathtt{th\,in}\sqcap')\quad \mathtt{th}[s_1,s_2]\sqcap\mathtt{in}[r_1,r_2]=\mathtt{in}[s_2,r_2] \Leftrightarrow s_1\le s_2, s_2< r_1, r_1\le r_2$$

$$(\mathtt{th\,in}\sqcap'')\quad \mathtt{th}[s_1,s_2]\sqcap\mathtt{in}[r_1,r_2]=\mathtt{in}[r_1,s_1] \Leftrightarrow r_1\le r_2, r_2< s_1, s_1\le s_2$$

$$(\mathtt{in}\sqcap)\quad \mathtt{in}[s_1,s_2]\sqcap\mathtt{in}[r_1,r_2]=\mathtt{in}[t_1,t_2] \Leftrightarrow s_1\le s_2, r_1\le r_2, t_1=min\{s_1,r_1\}, t_2=max\{s_2,r_2\}$$

Keeping in mind that annotations deal with time periods, i.e., *convex*, non-empty sets of time points, it is not difficult to verify that the axioms above indeed define the greatest lower bound with respect to the partial order relation $\sqsubseteq$. For instance the greatest lower bound of two th annotations, $\mathtt{th}[s_1,s_2]$ and $\mathtt{th}[r_1,r_2]$, can be:

- a $\mathtt{th}[t_1,t_2]$ annotation if $[r_1,r_2]$ and $[s_1,s_2]$ are overlapping intervals and $[t_1,t_2]$ is their (not empty) intersection (axiom $(\mathtt{th}\sqcap)$);
- an $\mathtt{in}[t_1,t_2]$ annotation, otherwise, where interval $[t_1,t_2]$ is the least convex set which intersects both $[s_1,s_2]$ and $[r_1,r_2]$ (axiom $(\mathtt{th}\sqcap')$, see Fig. 1.(a)).



(a) (b)

Figure 1: Greatest lower bound of annotations.

In all other cases the greatest lower bound is always an `in` annotation. For instance, as expressed by axiom (`th` `in` $\sqcap'$), the greatest lower bound of two annotations `th` $[s_1, s_2]$ and `in` $[r_1, r_2]$ with disjoint intervals is given by `in` $[s_2, r_2]$, where interval $[s_2, r_2]$ is the least convex set containing $[r_1, r_2]$ and intersecting $[s_1, s_2]$ (see Fig. 1.(b)). The greatest lower bound will play a basic role in the definition of the intersection operation over program expressions. Notice that in TACLP it is not needed since the problem of combining programs is not dealt with.

Finally, as in TACLP we still have, in addition to Modus Ponens, the inference rules ($\sqsubseteq$) and ($\sqcup$).

EXAMPLE 4 In a company there are some managers and a secretary who has to manage their meetings and appointments. During the day a manager can be busy if she/he is on a meeting or if she/he is not present in the office. This situation is modeled by the theory MANAGERS.

MANAGERS:

$busy(M)$ `th` $[T_1, T_2] \leftarrow$ $in\text{-}meeting(M)$ `th` $[T_1, T_2]$
$busy(M)$ `th` $[T_1, T_2] \leftarrow$ $out\text{-}of\text{-}office(M)$ `th` $[T_1, T_2]$

This theory is parametric with respect to the predicates *in-meeting* and *out-of-office* since the schedule of managers varies daily. The schedules are collected in a separate theory TODAY-SCHEDULE and, to know whether a manager is busy or not, such a theory is combined with MANAGERS by using the union operator.

For instance, suppose that the schedule for a given day is the following: Mr. Smith and Mr. Jones have a meeting at 9am lasting one hour. In the afternoon Mr. Smith goes out for lunch at 2pm and comes back at 3pm. The theory TODAY-SCHEDULE below represents such information.

TODAY-SCHEDULE:

$in\text{-}meeting(mrSmith)$ `th` $[9am, 10am]$.
$in\text{-}meeting(mrJones)$ `th` $[9am, 10am]$.
$out\text{-}of\text{-}office(mrSmith)$ `th` $[2pm, 3pm]$.

To know whether Mr. Smith is busy between 9:30am and 10:30am the secretary can ask for $busy(mrSmith)$ `in` $[9{:}30am, 10{:}30am]$. Since Mr. Smith is in a meeting from 9am till 10am, she will indeed obtain that Mr. Smith is busy. The considered query exploits indefinite information, because knowing that Mr. Smith is busy in one instant in $[9{:}30am, 10{:}30am]$ the secretary cannot schedule an appointment for him for that period.

EXAMPLE 5 At 10pm Tom was found dead in his house. The only hint is that the answering machine recorded some messages from 7pm up to 8pm. At a first glance, the doctor said Tom died one to two hours before. The detective made a further assumption: Tom did not answer the telephone so he could be already dead.

We collect all these hints and assumptions into three programs, HINTS, DOCTOR and DETECTIVE, in order not to mix firm facts with simple hypotheses that might change during the investigations.

HINTS: $\quad$ $\mathit{found}\ \mathtt{at}\ 10pm.$
$\mathit{ans\text{-}machine}\ \mathtt{th}\,[7pm, 8pm].$

DOCTOR: $\quad$ $\mathit{dead}\ \mathtt{in}\,[T-2{:}00, T-1{:}00]\ \leftarrow \mathit{found}\ \mathtt{at}\ T$

DETECTIVE: $\quad$ $\mathit{dead}\ \mathtt{in}\,[T_1, T_2]\ \leftarrow \mathit{ans\text{-}machine}\ \mathtt{th}\,[T_1, T_2]$

If we combine the hypotheses of the doctor and those of the detective we can extend the period of time in which Tom possibly died. The program expression DOCTOR $\cap$ DETECTIVE behaves as

$$\begin{aligned}\mathit{dead}\ \mathtt{in}\,[S_1, S_2]\ \leftarrow\ & \mathtt{in}\,[T-2{:}00, T-1{:}00] \sqcap \mathtt{in}\,[T_1, T_2] = \mathtt{in}\,[S_1, S_2],\\ & \mathit{found}\ \mathtt{at}\ T,\\ & \mathit{ans\text{-}machine}\ \mathtt{th}\,[T_1, T_2]\end{aligned}$$

The constraint $\mathtt{in}\,[T-2{:}00, T-1{:}00] \sqcap \mathtt{in}\,[T_1, T_2] = \mathtt{in}\,[S_1, S_2]$ determines the annotation $\mathtt{in}\,[S_1, S_2]$ in which Tom possibly died: according to axiom ($\mathtt{in}\,\sqcap$) the resulting interval is $S_1 = \mathit{min}\{T-2{:}00, T_1\}$ and $S_2 = \mathit{max}\{T-1{:}00, T_2\}$. In fact, according to the semantics defined in the next section, a consequence of the program expression

$$\text{HINTS} \cup (\text{DOCTOR} \cap \text{DETECTIVE})$$

is just $\mathit{dead}\ \mathtt{in}\,[7pm, 9pm]$ since the annotation $\mathtt{in}\,[7pm, 9pm]$ is the greatest lower bound of $\mathtt{in}\,[8pm, 9pm]$ and $\mathtt{in}\,[7pm, 8pm]$.

5 Semantics of MuTACLP

In this section we introduce an operational (top-down) semantics for the language MuTACLP by means of a meta-interpreter. Then we provide MuTACLP with a least fixpoint (bottom-up) semantics, based on the definition of an immediate consequence operator. Finally, the meta-interpreter for MuTACLP is proved sound and complete with respect to the least fixpoint semantics.

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

5.1 Meta-interpreter

The extended meta-interpreter is defined by the following clauses.

$$demo(\mathcal{E}, empty). \tag{1}$$

$$demo(\mathcal{E}, (B_1, B_2)) \leftarrow demo(\mathcal{E}, B_1), demo(\mathcal{E}, B_2) \tag{2}$$

$$\begin{aligned} demo(\mathcal{E}, A\,\mathtt{th}\,[T_1, T_2]) \leftarrow {} & S_1 \leq T_1, T_1 \leq T_2, T_2 \leq S_2, \\ & clause(\mathcal{E}, A\,\mathtt{th}\,[S_1, S_2], B), demo(\mathcal{E}, B) \end{aligned} \tag{3}$$

$$\begin{aligned} demo(\mathcal{E}, A\ \mathtt{th}\,[T_1, T_2]) \leftarrow {} & S_1 \leq T_1, T_1 < S_2, S_2 < T_2, \\ & clause(\mathcal{E}, A\ \mathtt{th}\,[S_1, S_2], B), demo(\mathcal{E}, B), \\ & demo(\mathcal{E}, A\ \mathtt{th}\,[S_2, T_2]) \end{aligned} \tag{4}$$

$$\begin{aligned} demo(\mathcal{E}, A\,\mathtt{in}\,[T_1, T_2]) \leftarrow {} & T_1 \leq S_2, S_1 \leq T_2, T_1 \leq T_2, \\ & clause(\mathcal{E}, A\,\mathtt{th}\,[S_1, S_2], B), demo(\mathcal{E}, B) \end{aligned} \tag{5}$$

$$\begin{aligned} demo(\mathcal{E}, A\,\mathtt{in}\,[T_1, T_2]) \leftarrow {} & T_1 \leq S_1, S_2 \leq T_2, \\ & clause(\mathcal{E}, A\,\mathtt{in}\,[S_1, S_2], B), demo(\mathcal{E}, B) \end{aligned} \tag{6}$$

$$demo(\mathcal{E}, C) \leftarrow constraint(C), C \tag{7}$$

$$clause(\mathcal{E}_1 \cup \mathcal{E}_2, A\,\alpha, B) \leftarrow clause(\mathcal{E}_1, A\,\alpha, B) \tag{8}$$

$$clause(\mathcal{E}_1 \cup \mathcal{E}_2, A\,\alpha, B) \leftarrow clause(\mathcal{E}_2, A\,\alpha, B) \tag{9}$$

$$\begin{aligned} clause(\mathcal{E}_1 \cap \mathcal{E}_2, A\,\gamma, (B_1, B_2)) \leftarrow {} & clause(\mathcal{E}_1, A\,\alpha, B_1), \\ & clause(\mathcal{E}_2, A\,\beta, B_2), \\ & \alpha \sqcap \beta = \gamma \end{aligned} \tag{10}$$

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

$$clause(P, A\,\alpha, B) \leftarrow S_1 \leq S_2 \tag{11}$$

where $\alpha = \mathtt{th}\,[S_1, S_2]$ or $\alpha = \mathtt{in}\,[S_1, S_2]$.

This meta-interpreter can be written in any CLP language that provides a suitable constraint solver for temporal annotations (see Section 3.2 for the corresponding constraint theory). A first difference with respect to the meta-interpreter in Section 2 is that our meta-interpreter handles *constraints* that can either occur explicitly in its clauses, e.g., the constraint $s_1 \leq t_1, t_1 \leq t_2, t_2 \leq s_2$ in clause (3), or can be produced by resolution steps. Constraints of the latter kind are 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 also rule ($\sqsubseteq$) and rule ($\sqcup$). This is the reason why the third clause for the predicate *demo* of the meta-interpreter in Section 2 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 $s_1 \leq t_1, t_1 \leq t_2, t_2 \leq s_2$. Clauses (5) and (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 (4). According to the discussion in Section 3.2, it is applicable only to th annotations involving 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., $demo(\mathcal{E}, A\,\mathtt{th}\,[s_2, t_2])$ 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 $[s_1, s_2]$, $[s_2, t_2]$ and their subintervals.

Finally, in the meta-level representation of object clauses, as expressed by clause (11), the constraint $s_1 \leq s_2$ is added to ensure that the head of the object clause has a well-formed, namely non-empty, annotation.

As far as the meta-level definition of the union and intersection operators is concerned, clauses implementing the union operation remain unchanged with respect to the original definition in Section 2, whereas in the clause implementing the intersection operation a constraint is added, which expresses the annotation for the derived atom. Informally, a clause $A\,\alpha \leftarrow B$, belonging to the intersection of two program expressions $\mathcal{E}_1$ and $\mathcal{E}_2$, is built by taking one clause instance from each program expression $\mathcal{E}_1$ and $\mathcal{E}_2$, such that the head atoms of the two clauses are unifiable. Let such instances of clauses be cl_1 and cl_2. Then B is the conjunction of the bodies of cl_1 and cl_2 and A is the unified atom labelled with the greatest lower bound of the annotations of the heads of cl_1 and cl_2.

The following example shows the usefulness of clause (4) to derive *new* temporal information according to the inference rule ($\sqcup$).

Example 6 Consider the databases DB1 and DB2 containing information about people working in two companies. Jim is a consultant and he works for the first company from January 1, 1995 to April 30, 1995 and for the second company from April 1, 1995 to September 15, 1995.

DB1: $\quad consultant(jim)\,\mathtt{th}\,[\mathit{Jan\ 1\ 1995}, \mathit{Apr\ 30\ 1995}]$.
DB2: $\quad consultant(jim)\,\mathtt{th}\,[\mathit{Apr\ 1\ 1995}, \mathit{Sep\ 15\ 1995}]$.

The period of time in which Jim works as a consultant can be obtained by querying the union of the above theories as follows:

$$demo(\mathrm{DB1} \cup \mathrm{DB2},\ consultant(jim)\ \mathtt{th}\,[T_1, T_2]).$$

By using clause (4), we can derive the interval $[Jan\ 1\ 1995, Sep\ 15\ 1995]$ (more precisely, the constraints $Jan\ 1\ 1995 \le T_1, T_1 < Apr\ 30\ 1995, Apr\ 30\ 1995 < T_2$, $T_2 \le Sep\ 15\ 1995$ are derived) that otherwise would never be generated. In fact, by applying clause (3) alone, we can prove only that Jim is a consultant in the intervals $[Jan\ 1\ 1995, Apr\ 30\ 1995]$ and $[Apr\ 1\ 1995, Sep\ 15\ 1995]$ (or in subintervals of them) separately.

5.2 Bottom-up semantics

To give a declarative meaning to program expressions, we define a "higher-order" semantics for MuTACLP. In fact, the results in [6] show that the least Herbrand model semantics of logic programs does not scale smoothly to program expressions. Fundamental properties of semantics, like compositionality and full abstraction, are definitely lost. Intuitively, the least Herbrand model semantics is not compositional since it identifies programs which have different meanings when combined with others. Actually, all the programs whose least Herbrand model is empty are identified with the empty program. For example, the programs

$$\{r \leftarrow s\} \qquad\qquad \{r \leftarrow q\}$$

are both denoted by the empty model, though they behave quite differently when composed with other programs (e.g., consider the union with $\{q.\}$).

Brogi et al. showed in [8] that defining as meaning of a program P the immediate consequence operator T_P itself (rather than the least fixpoint of T_P), one obtains a semantics which is compositional with respect to several interesting operations on programs, in particular $\cup$ and $\cap$.

Along the same line, the semantics of a MuTACLP program expression is taken to be the immediate consequence operator associated with it, i.e., a function from interpretations to interpretations. The immediate consequence operator of constraint logic programming is generalized to deal with temporal annotations by considering a kind of extended interpretations, which are basically sets of annotated elements of $\mathcal{C}\text{-}base_L$. More precisely, we first define a set of (semantical) annotations

$$Ann = \{\mathtt{th}\,[t_1, t_2], \mathtt{in}\,[t_1, t_2] \mid t_1, t_2 \ \textit{time points} \ \wedge \ \mathcal{D}_\mathcal{C} \models t_1 \le t_2\}$$

where $\mathcal{D}_\mathcal{C}$ is the $S_\mathcal{C}$-structure providing the intended interpretation of the constraints. Then the lattice of interpretations is defined as $(\wp(\mathcal{C}\text{-}base_L \times Ann), \subseteq)$ where $\subseteq$ is the usual set-theoretic inclusion. Finally the immediate consequence operator $\mathbb{T}^\mathcal{C}_\mathcal{E}$ for a program expression $\mathcal{E}$ is compositionally defined in terms of the immediate consequence operator for its sub-expressions.

Definition 7 (Bottom-up semantics) *Let $\mathcal{E}$ be a program expression, the function $\mathbb{T}^\mathcal{C}_\mathcal{E} : \wp(\mathcal{C}\text{-}base_L \times Ann) \to \wp(\mathcal{C}\text{-}base_L \times Ann)$ is defined as follows.*

- ($\mathcal{E}$ is a plain program P)
$\mathbb{T}^{\mathcal{C}}_{P}(I) =$

$$\left\{ (A,\alpha) \mid \begin{array}{l} (\alpha = \mathtt{th}[s_1,s_2] \ \vee \ \alpha = \mathtt{in}[s_1,s_2]), \\ A\ \alpha \leftarrow \bar{C}, B_1\alpha_1,\ldots,B_n\alpha_n \in ground_{\mathcal{C}}(P), \\ \{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq I, \\ \mathcal{D}_{\mathcal{C}} \models \bar{C}, \alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, s_1 \leq s_2 \end{array} \right\}$$
$$\cup$$
$$\left\{ (A,\mathtt{th}[s_1,r_2]) \mid \begin{array}{l} A\ \mathtt{th}[s_1,s_2] \leftarrow \bar{C}, B_1\alpha_1,\ldots,B_n\alpha_n \in ground_{\mathcal{C}}(P), \\ \{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq I,\ (A,\mathtt{th}[r_1,r_2]) \in I, \\ \mathcal{D}_{\mathcal{C}} \models \bar{C}, \alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, s_1 < r_1, r_1 \leq s_2, s_2 < r_2 \end{array} \right\}$$

where $\bar{C}$ *is a shortcut for* $C_1,\ldots,C_k$.

- ($\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$)
$\mathbb{T}^{\mathcal{C}}_{\mathcal{E}_1\cup\mathcal{E}_2}(I) = \mathbb{T}^{\mathcal{C}}_{\mathcal{E}_1}(I) \cup \mathbb{T}^{\mathcal{C}}_{\mathcal{E}_2}(I)$

- ($\mathcal{E} = \mathcal{E}_1 \cap \mathcal{E}_2$)
$\mathbb{T}^{\mathcal{C}}_{\mathcal{E}_1\cap\mathcal{E}_2}(I) = \mathbb{T}^{\mathcal{C}}_{\mathcal{E}_1}(I) \Cap \mathbb{T}^{\mathcal{C}}_{\mathcal{E}_2}(I)$

where $I_1 \Cap I_2 = \{(A,\gamma) \mid (A,\alpha) \in I_1,\ (A,\beta) \in I_2,\ \mathcal{D}_{\mathcal{C}} \models \alpha \sqcap \beta = \gamma\}$.

Observe that the definition above properly extends the standard definition of the immediate consequence operator for constraint logic programs (see Section 3.1). In fact, besides the usual Modus Ponens rule, it captures rule ($\sqcup$) (as expressed by the second set in the definition of $\mathbb{T}^{\mathcal{C}}_{P}$). Furthermore, also rule ($\sqsubseteq$) is taken into account 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 imply $B_i\,\alpha_i$, i.e., such that each β_i is an annotation stronger than α_i ($\mathcal{D}_{\mathcal{C}} \models \alpha_i \sqsubseteq \beta_i$). Notice that $\mathbb{T}^{\mathcal{C}}_{\mathcal{E}}(I)$ is not downward closed, namely, it is not true that if $(A,\alpha) \in \mathbb{T}^{\mathcal{C}}_{\mathcal{E}}(I)$ then for all (A,γ) such that $\mathcal{D}_{\mathcal{C}} \models \gamma \sqsubseteq \alpha$, we have $(A,\gamma) \in \mathbb{T}^{\mathcal{C}}_{\mathcal{E}}(I)$. The downward closure will be taken only after the fixpoint of $\mathbb{T}^{\mathcal{C}}_{\mathcal{E}}$ is computed. We will see that, nevertheless, no deductive capability is lost and rule ($\sqsubseteq$) is completely modeled.

The set of immediate consequences of a union of program expressions is the set-theoretic union of the immediate consequences of each program expression. Instead, an atom A labelled by γ is an immediate consequence of the intersection of two program expressions if A is a consequence of both program expressions, possibly with different annotations α and β, and the label γ is the greatest lower bound of the annotations α and β.

Let us formally define the downward closure of an interpretation.

Definition 8 (Downward closure) *The* downward closure *of an interpretation* $I \subseteq \mathcal{C}\text{-}base_L \times Ann$ *is defined as:*

$$\downarrow I = \{(A,\alpha) \mid (A,\beta) \in I,\ \mathcal{D}_{\mathcal{C}} \models \alpha \sqsubseteq \beta\}.$$

The next proposition sheds some more light on the semantics of the intersection operator, by showing that, when we apply the downward closure, the image of an interpretation through the operator $\mathbb{T}^{\mathcal{C}}_{\mathcal{E}_1 \cap \mathcal{E}_2}$ is the *set-theoretic intersection* of the images of the interpretation through the operators associated with $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively. This property supports the intuition that the program expressions have to agree at each computation step (see [8]).

PROPOSITION 9 *Let I_1 and I_2 be two interpretations. Then*

$$\downarrow (I_1 \Cap I_2) \;=\; (\downarrow I_1) \cap (\downarrow I_2).$$

The next theorem shows the continuity of the $\mathbb{T}^{\mathcal{C}}_{\mathcal{E}}$ operator over the lattice of interpretations.

THEOREM 10 (CONTINUITY) *For any program expression $\mathcal{E}$, the function $\mathbb{T}^{\mathcal{C}}_{\mathcal{E}}$ is continuous (over $(\wp(\mathcal{C}\text{-}base_L \times Ann), \subseteq)$).*

The fixpoint semantics for a program expression is now defined as the downward closure of the least fixpoint of $\mathbb{T}^{\mathcal{C}}_{\mathcal{E}}$ which, by continuity of $\mathbb{T}^{\mathcal{C}}_{\mathcal{E}}$, is determined as $\bigcup_{i \in \mathbb{N}} (\mathbb{T}^{\mathcal{C}}_{\mathcal{E}})^i$.

DEFINITION 11 (FIXPOINT SEMANTICS) *Let $\mathcal{E}$ be a program expression. The* fixpoint semantics *of $\mathcal{E}$ is defined as*

$$\mathbb{F}^{\mathcal{C}}(\mathcal{E}) = \downarrow (\mathbb{T}^{\mathcal{C}}_{\mathcal{E}})^{\omega}.$$

We remark that the downward closure is applied only once, after having computed the fixpoint of $\mathbb{T}^{\mathcal{C}}_{\mathcal{E}}$. However, it is easy to see that the closure is a continuous operator on the lattice of interpretations $\wp(\mathcal{C}\text{-}base_L \times Ann)$. Thus

$$\downarrow \left(\bigcup_{i \in \mathbb{N}} (\mathbb{T}^{\mathcal{C}}_{\mathcal{E}})^i \right) = \bigcup_{i \in \mathbb{N}} \downarrow (\mathbb{T}^{\mathcal{C}}_{\mathcal{E}})^i$$

showing that by taking the closure at each step we would have obtained the same set of consequences. Hence, as mentioned before, rule ($\sqsubseteq$) is completely captured.

5.3 Soundness and completeness

In the spirit of [6, 33] we define the semantics of the 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 $\mathcal{C}\text{-}base_L$ annotated with labels from Ann, which are the semantic counterpart of object level annotated atoms. We show that given a MuTACLP program expression $\mathcal{E}$ (object program) for any $A \in \mathcal{C}\text{-}base_L$ and any $\alpha \in Ann$, $demo(\mathcal{E}, A\,\alpha)$ is provable at the meta-level if and only if (A, α) is provable in the object program.

Theorem 12 (Soundness and completeness) *Let $\mathcal{E}$ be a program expression and let V be the meta-program containing the meta-level representation of the object level programs occurring in $\mathcal{E}$ and clauses (1)-(10). For any $A \in \mathcal{C}$-$base_L$ and $\alpha \in Ann$, the following statement holds:*

$$demo(\mathcal{E}, A\,\alpha) \in (T_V^{\mathcal{M}})^{\omega} \iff (A, \alpha) \in \mathbb{F}^{\mathcal{C}}(\mathcal{E}),$$

where $T_V^{\mathcal{M}}$ is the standard immediate consequence operator for CLP programs.

Note that V is a CLP($\mathcal{M}$) program where $\mathcal{M}$ is a multi-sorted constraint domain, including the constraint domain *Term*, presented in Example 2, and the constraint domain $\mathcal{C}$. It is worth observing that if C is a $\mathcal{C}$-ground instance of a constraint then $\mathcal{D}_{\mathcal{M}} \models C \Leftrightarrow \mathcal{D}_{\mathcal{C}} \models C$.

6 Some examples

This section is devoted to present examples which illustrate the use of annotations in the representation of temporal information and the structuring possibilities offered by the operators. First we describe applications of our framework in the field of legal reasoning. Then we show how the intersection operator can be employed to define a kind of valid-timeslice operator.

6.1 Applications to legal reasoning

Laws and regulations are naturally represented in separate theories and they are usually combined in ways that are necessarily more complex than a plain merging. Time is another crucial ingredient in the definition of laws and regulations, since, quite often, they refer to instants of time and, furthermore, their validity is restricted to a fixed period of time. This is especially true for laws and regulations which concern taxation and government budget related regulations in general.

British Nationality Act.

We start with a classical example in the field of legal reasoning [40], i.e. a small piece of the British Nationality Act. Simply partitioning the knowledge into separate programs and using the basic union operation, one can exploit the temporal information in an orderly way. Assume that *Jan 1 1955* is the commencement date of the law. Then statement

x obtains the British Nationality at time t
if x is born in U.K. at time t and
t is after commencement and
y is parent of x and
y is a British citizen at time t
or y is a British resident at time t

is modeled by the following program.

BNA:

$$\textit{get-citizenship(X)}\ \mathtt{at}\ T \leftarrow T \geq \textit{Jan 1 1955},\ \textit{born(X,uk)}\ \mathtt{at}\ T,\\ \textit{parent(Y,X)}\ \mathtt{at}\ T,\ \textit{british-citizen(Y)}\ \mathtt{at}\ T$$

$$\textit{get-citizenship(X)}\ \mathtt{at}\ T \leftarrow T \geq \textit{Jan 1 1955},\ \textit{born(X,uk)}\ \mathtt{at}\ T,\\ \textit{parent(Y,X)}\ \mathtt{at}\ T,\ \textit{british-resident(Y)}\ \mathtt{at}\ T$$

Now, the data for a single person, say John, can be encoded in a separate program.

JOHN:

$$\textit{born(john,uk)}\ \mathtt{at}\ \textit{Aug 10 1969}.\\ \textit{parent(bob,john)}\ \mathtt{th}\ [T,\infty] \leftarrow \textit{born(john,\_)}\ \mathtt{at}\ T\\ \textit{british-citizen(bob)}\ \mathtt{th}\ [\textit{Sept 6 1940},\infty].$$

Then, by means of the union operator, one can inquire about the citizenship of John, as follows

$$\textit{demo}(\text{BNA} \cup \text{JOHN},\ \textit{get-citizenship(john)}\ \mathtt{at}\ T)$$

obtaining as result $T = \textit{Aug 10 1969}$.

Movie tickets.

Since 1997, an Italian regulation for encouraging people to go to the cinema, states that on Wednesdays the ticket price is 8000 liras, whereas in the rest of the week it is 12000 liras. The situation can be modeled by the following theory BOXOFF.

BOXOFF:

$$\textit{ticket}(\textit{8000}, X)\ \mathtt{at}\ T \leftarrow T \geq \textit{Jan 1 1997}, \textit{wed}\ \mathtt{at}\ T\\ \textit{ticket}(\textit{12000}, X)\ \mathtt{at}\ T \leftarrow T \geq \textit{Jan 1 1997}, \textit{non\_wed}\ \mathtt{at}\ T$$

The constraint $T \geq \textit{Jan 1 1997}$ represents the validity of the clause, which holds from January 1, 1997 onwards.

The predicates *wed* and *non_wed* are defined in a separate theory DAYS, where w is assumed to be the last Wednesday of 1996.

DAYS:

$$\textit{wed}\ \mathtt{at}\ w.\\ \textit{wed}\ \mathtt{at}\ T+7 \leftarrow \textit{wed}\ \mathtt{at}\ T$$

$$\textit{non\_wed}\ \mathtt{th}\ [w+1, w+6].\\ \textit{non\_wed}\ \mathtt{at}\ T+7 \leftarrow \textit{non\_wed}\ \mathtt{at}\ T$$

Notice that, by means of recursive predicates one can easily express *periodic* temporal information. In the example, the definition of the predicate *wed* expresses the fact that a day is Wednesday if it is a date which is known to be Wednesday or it is a day coming seven days after a day proved to be Wednesday. The predicate *non_wed* is defined in an analogous way; in this case the unit clause states that all six consecutive days following a Wednesday are not Wednesdays.

Now, let us suppose that the owner of a cinema wants to increase the discount for young people on Wednesdays, establishing that the ticket price for people who are eighteen years old or younger is 6000 liras. By resorting to the intersection operation we can build a program expression that represents exactly the desired policy. We define three new programs CONS, DISC and AGE.

CONS: $\quad ticket(8000, X)\ \texttt{at}\ T \leftarrow Y > 18,\ age(X, Y)\ \texttt{at}\ T$
$\qquad\quad ticket(12000, X)\ \texttt{at}\ T.$

The above theory specifies how the predicate definitions in BOXOFF must change according to the new policy. In fact to get a 8000 liras ticket now a further constraint must be satisfied, namely the customer has to be older than eighteen years old. On the other hand, no further requirement is imposed to buy a 12000 liras ticket.

DISC: $\quad ticket(6000, X)\ \texttt{at}\ T \leftarrow a \leq 18,\ wed\ \texttt{at}\ T,\ age(p, a)\ \texttt{at}\ T$

The only clause in DISC states that a 6000 liras ticket can be bought on Wednesdays by a person who is eighteen years old or younger.

The programs CONS and DISC are parametric with respect to the predicate *age*, which is defined in a separate theory AGE.

AGE: $\quad age(X, Y)\ \texttt{at}\ T \leftarrow born(X)\ \texttt{at}\ T_1,\ year\text{-}diff(T_1, T, Y)$

At this point we can compose the above programs to obtain the program expression representing the new policy, namely

$$(\text{BOXOFF} \cap \text{CONS}) \cup \text{DISC} \cup \text{DAYS} \cup \text{AGE}.$$

Finally, in order to know how much is a ticket for a given person, the above program expression must be joined with a separate program containing the date of birth of the person. For instance, such program could be

TOM: $\quad born(tom)\ \texttt{at}\ May\ 7\ 1982.$

Then the answer to the query

$$demo(((\text{BOXOFF} \cap \text{CONS}) \cup \text{DISC} \cup \text{DAYS} \cup \text{TOM}),\\ ticket(X, tom)\ \texttt{at}\ May\ 20\ 1998)$$

is $X = 6000$ since *May 20 1998* is a Wednesday and Tom is sixteen years old.

Invim.

Invim was an Italian law dealing with paying taxes on real estate transactions. The original regulation, in force since January 1, 1950, requires time calculations, since the amount of taxes depends on the period of ownership of the real estate property. Furthermore, although the law has been abolished in 1992, it still applies but only for the period antecedent to 1992.

To see how our framework allows us to model the described situation let us first consider the program INVIM below, which contains a sketch of the original body of regulations.

INVIM:

$$\begin{aligned} \mathit{due(Amount,X,Prop)}\,\mathtt{th}\,[T_2,\infty] \leftarrow\ & T_2 \geq \mathit{Jan\ 1\ 1950},\ \mathit{buys(X,Prop)}\ \mathtt{at}\ T_1, \\ & \mathit{sells(X,Prop)}\ \mathtt{at}\ T_2, \\ & \mathit{compute(Amount,X,Prop,}T_1,T_2\mathit{)} \end{aligned}$$

$$\mathit{compute(Amount,X,Prop,}T_1,T_2\mathit{)} \leftarrow \ldots$$

To update the regulations in order to consider the decisions taken in 1992, as in the previous example we introduce two new theories. The first one includes a set of constraints on the applicability of the original regulations, while the second one is designed to embody regulations capable of handling the new situation.

CONSTRAINTS:

$$\begin{aligned} & \mathit{due(Amount,X,Prop)}\,\mathtt{th}\,[\mathit{Jan\ 1\ 1993},\infty] \leftarrow \\ & \qquad\qquad \mathit{sells(X,Prop)}\,\mathtt{in}\,[\mathit{Jan\ 1\ 1950},\mathit{Dec\ 31\ 1992}] \end{aligned}$$

$$\mathit{compute(Amount,X,Prop,}T_1,T_2\mathit{)}.$$

The first rule specifies that the relation *due* is computed, provided that the selling date is antecedent to December, 31 1992. The second rule specifies that the rules for *compute*, whatever number they are, and whatever complexity they have, carry on unconstrained to the new version of the regulation. It is important to notice that the design of the constraining theory can be done without taking care of the details (which may be quite complicated) embodied in the original law.

The theory which handles the case of a property bought before December 31, 1992 and sold after the first of January, 1993, is given below.

ADDITIONS:

$$\begin{aligned} \mathit{due(Amount,X,Prop)}\,\mathtt{th}\,[T_2,\infty] \leftarrow\ & T_2 \geq \mathit{Jan\ 1\ 1993},\ \mathit{buys(X,Prop)}\,\mathtt{at}\,T_1, \\ & \mathit{sells(X,Prop)}\,\mathtt{at}\,T_2, \\ & \mathit{compute(Amount,X,Prop,}T_1,\mathit{Dec\ 31\ 1992}) \end{aligned}$$

Now consider a separate theory representing the transactions regarding Mary, who bought an apartment on March 8, 1965 and sold it on July 2, 1997.

TRANS1:

$$\mathit{buys(mary,apt8)}\,\mathtt{at}\,\mathit{Mar\ 8\ 1965}.$$
$$\mathit{sells(mary,apt8)}\,\mathtt{at}\,\mathit{Jul\ 2\ 1997}.$$

The query

$$\mathit{demo}(\textsc{Invim} \cup \textsc{Trans1},\ \mathit{due(Amount,mary,apt8)}\,\mathtt{th}\,[\_,\_])$$

yields the amount, say 32.1, that Mary has to pay when selling the apartment according to the old regulations. On the other hand, the query

$$\begin{aligned} & \mathit{demo}(((\textsc{Invim} \cap \textsc{Constraints}) \cup \textsc{Additions}) \cup \textsc{Trans1}, \\ & \qquad \mathit{due(Amount,mary,apt8)}\,\mathtt{th}\,[\_,\_]) \end{aligned}$$

yields the amount, say 27.8, computed according to the new regulations. It is smaller than the previous one because taxes are computed only for the period from March 8, 1965 to December 31, 1992, by using the clause in the program ADDITIONS. The clause in INVIM $\cap$ CONSTRAINTS cannot be used since the condition regarding the selling date (*sells(X,Prop)* `in` [*Jan 1 1950*, *Dec 31 1992*]) does not hold.

In the transaction, represented by the program below, Paul buys the flat on January 1, 1995.

TRANS2:

buys(paul,apt9) `at` *Jan 1 1995*.
sells(paul,apt9) `at` *Sep 12 1998*.

demo(INVIM $\cup$ TRANS2, *due(Amount,paul,apt9)* `th` [_, _])

$Amount = 1.7$

demo(((INVIM $\cap$ CONSTRAINTS) $\cup$ ADDITIONS) $\cup$ TRANS2,
due(Amount,paul,apt9) `th` [_, _])

no

If we query the theory INVIM $\cup$ TRANS2 we will get that Paul must pay a certain amount of tax, say 1.7, while, according to the updated regulation, he must not pay the Invim tax because he bought and sold the flat after December 31, 1992. Indeed, the answer to the query computed with respect to the theory ((INVIM $\cap$ CONSTRAINTS) $\cup$ ADDITIONS) $\cup$ TRANS2 is *no*, i.e., no tax is due.

Summing up, the union operation can be used to obtain a larger set of clauses. We can join a program with another one to provide it with definitions of its undefined predicates (e.g., AGE provides a definition for the predicate *age* not defined in DISC and CONS) or alternatively to add new clauses for an existing predicate (e.g., DISC contains a new definition for the predicate *ticket* already defined in BOXOFF). On the other hand, the intersection operator provides a natural way of imposing constraints on existing programs (e.g., the program CONS constrains the definition of *ticket* given in BOXOFF). Such constraints affect not only the computation of a particular property, like the intersection operation defined by Brogi et al. [8], but also the temporal information in which the property holds.

The use of TACLP programs allows us to represent and reason on temporal information in a natural way. Since time is explicit, at the object level we can directly access the temporal information associated with atoms. Periodic information can be easily expressed by recursive predicates (see the predicates *wed* and *non-wed* in the theory DAYS). Indefinite temporal information can be represented by using `in` annotations. E.g., in the program ADDITIONS the `in` annotation is used to specify that a certain date is within a time period (*sell(X,Prop)* `in` [*Jan* 1 1950, *Dec* 31 1992]). This is a case in which it is not important to know the precise date but it is sufficient to have an information which delimits the time period in which it can occur.

6.2 Valid-timeslice operator

By exploiting the features of the intersection operator we can define an operator which eases the selection of information holding in a certain interval.

DEFINITION 13 *Let P be a plain program. For a ground interval $[t_1, t_2]$ we define*

$$P \Downarrow [t_1, t_2] = P \cap 1_P^{[t_1,t_2]}$$

where $1_P^{[t_1,t_2]}$ is a program which contains a fact "$p(X_1, \ldots, X_n)\,\mathtt{th}\,[t_1, t_2]$." for all p defined in P with arity n.

Intuitively the operator $\Downarrow$ selects only the clauses belonging to P that hold in $[t_1, t_2]$ or in a subinterval of $[t_1, t_2]$, and it restricts their validity time to such an interval. Therefore $\Downarrow$ allows us to create temporal views of programs, for instance $P \Downarrow [t, t]$ is the program P at time point t. Hence it acts as a *valid-timeslice operator* in the field of databases (see the glossary in [12]).

Consider again the Invim example of the previous section. The *whole* history of the regulation concerning Invim, can be represented by using the following program expression

$$(\textsc{Invim} \Downarrow [0, \mathit{Dec\ 31\ 1992}]) \cup ((\textsc{Invim} \cap \textsc{Constraints}) \cup \textsc{Additions})$$

By applying the operation $\Downarrow$, the validity of the clauses belonging to INVIM is restricted to the period from January 1, 1950 up to December 31, 1992, thus modeling the law before January 1, 1993. On the other hand, the program expression (INVIM $\cap$ CONSTRAINTS) $\cup$ ADDITIONS expresses the regulation in force since January 1, 1993, as we previously explained.

This example suggests how the operation $\Downarrow$ can be useful to model updates. Suppose that we want to represent that Frank is a research assistant in mathematics, and that, later, he is promoted becoming an assistant professor. In our formalism we can define a program FRANK that records the information associated with Frank as a research assistant.

FRANK:
$\quad$ *research_assistant(maths)* $\mathtt{th}\,[\mathit{Mar\ 8\ 1993}, \infty]$.

On March 1996 Frank becomes an assistant professor. In order to modify the information contained in the program FRANK, we build the following program expression:

$$(\textsc{Frank} \Downarrow [0, \mathit{Feb\ 29\ 1996}]) \cup \{\mathit{assistant\_prof(maths)}\,\mathtt{th}\,[\mathit{Mar\ 1\ 1996}, \infty].\}$$

where the second expression is an unnamed theory. Unnamed theories, which have not been discussed so far, can be represented by the following meta-level clause:

$$\mathit{clause}(\{X\,\alpha \leftarrow Y\}, X\,\alpha, Y) \leftarrow\ T_1 \leq T_2$$

where $\alpha = \mathtt{th}\,[T_1, T_2]$ or $\alpha = \mathtt{in}\,[T_1, T_2]$.

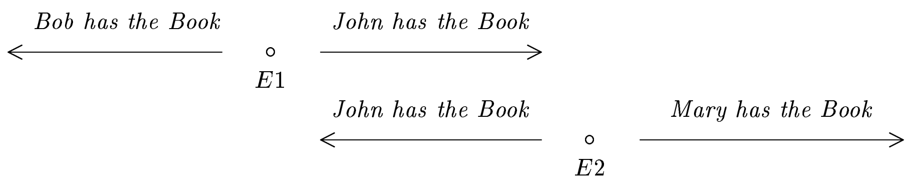
The described update resembles the addition and deletion of a ground atom. For instance in $\mathcal{LDL}$++ [46] an analogous change can be implemented by solving the goal $-research\_assistant(maths)$, $+assistant\_prof(maths)$. The advantage of our approach is that we do not change directly the clauses of a program, e.g. FRANK in the example, but we compose the old theory with a new one that represents the current situation. Therefore the state of the database before March 1, 1996 is preserved, thus maintaining the whole history. For instance, the first query below inquires the updated database before Frank's promotion whereas the second one shows how information in the database has been modified.

$$
\begin{array}{l}
demo((\text{FRANK} \Downarrow [0, \textit{Feb 29 1996}]) \cup \\
\qquad \{\textit{assistant\_prof(maths)}\ \texttt{th}\ [\textit{Mar 1 1996}, \infty].\}, \\
\qquad \textit{research\_assistant(X)}\ \texttt{at}\ \textit{Feb 23 1994}) \\
X = maths
\end{array}
$$

$$
\begin{array}{l}
demo((\text{FRANK} \Downarrow [0, \textit{Feb 29 1996}]) \cup \\
\qquad \{\textit{assistant\_prof(maths)}\ \texttt{th}\ [\textit{Mar 1 1996}, \infty].\}, \\
\qquad \textit{research\_assistant(X)}\ \texttt{at}\ \textit{Mar 12 1996}) \\
no.
\end{array}
$$

7 Related Work

Event Calculus by Kowalski and Sergot [27] has been the first attempt to cast into logic programming the rules for reasoning about time. In more details, Event Calculus is a treatment of time, based on the notion of event, in first-order classical logic augmented with negation as failure. It is closely related to Allen's interval temporal logic [3]. For example, let E1 be an event in which *Bob gives the Book to John* and let E2 be an event in which *John gives Mary the Book*. Assume that E2 occurs after E1. Given these event descriptions, we can deduce that there is a period started by the event E1 in which John possesses the book and that there is a period terminated by E1 in which Bob possesses the book. This situation is represented pictorially as follows:



A series of axioms for deducing the existence of time periods and the Start and End of each time period are given by using the *Holds* predicate.

$$\textit{Holds}(\textit{before}(e\ r))\ \textit{if}\ \ \textit{Terminates}(e\ r)$$

means that the relationship r holds in the time period *before(e r)* that denotes a time period terminated by the event e. *Holds(after(e r))* is defined in an analogous way. Event Calculus provides a natural treatment of valid time in databases, and it was extended in [42, 43] to include the concept of transaction time.

Therefore Event Calculus exploits the deductive power of logic and the computational power of logic programming as in our approach, but the modeling of time is different: events are the granularity of time chosen in Event Calculus, whereas we use time points and time periods. Furthermore no provision for multiple theories is given in Event Calculus.

Kifer and Subrahmanian in [25] introduce generalized annotated logic programs (GAPs), and show how Templog [1] and an interval based temporal logic can be translated into GAPs. The annotations used there correspond to the `th` annotations of MuTACLP. To implement the annotated logic language, the paper proposes to use "reductants", additional clauses which are derived from existing clauses to express all possible least upper bounds. The problem is that a finite program may generate infinitely many such reductants. Then a new kind of resolution for annotated logic programs, called "ca-resolution", is proposed in [29]. 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 in Section 5.1 which relies on recursion to collect the partial answers. However, in [29] the intermediate stages of the computation may not be sound with respect to the standard CLP semantics.

The paper [25] presents also two fixpoint semantics for GAPs, defined in terms of two different operators. The first operator, called T_P, is based on interpretations which associate with each element of the Herbrand Base of a program P a *set* of annotations which is an ideal, i.e., a set downward closed and closed under *finite* least upper bounds. For each atom A, 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 with each atom of the Herbrand Base a *single* annotation, obtained as the least upper bound of the set of annotations computed as in the previous case. Our fixpoint operator for MuTACLP works similarly to the T_P operator: at each step we take the closure 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 affect the set of derivable consequences. The main difference resides in the language: MuTACLP is an extension of CLP, which focuses on temporal aspects and provides mechanisms for combining programs, taking from GAP the basic ideas for handling annotations, whereas GAP is a general language with negation and arbitrary annotations but without constraints and multiple theories.

Our temporal annotations correspond to some of the predicates proposed by Galton in [18], which is a critical examination of Allen's classical work on a theory of action and time [3]. Galton accounts 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 MuTACLP'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. From the described correspondence it becomes clear that MuTACLP can be seen as reified FOL where annotated formulae, for example $born(john)\mathtt{at}\,t$, correspond to binary meta-relations between predicates and temporal information, for example $\mathtt{at}(born(john), t)$. But also, MuTACLP can be regarded as a modal logic, where the annotations are seen as parameterized modal operators, e.g., $born(john)$ $(\mathtt{at}\, t)$.

Our temporal annotations also correspond to some *temporal characteristics* in the ChronoBase data model [41]. Such a model allows for the representation of a wide variety of temporal phenomena in a temporal database which cannot be expressed by using only `th` and `in` annotations. However, this model is an extension of the relational data model and, differently from our model, it is not rule-based. An interesting line of research could be to investigate the possibility of enriching the set of annotations in order to capture some other temporal characteristics, like a property that holds in an interval but not in its subintervals, still maintaining a simple and clear semantics.

In [9], 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. 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, MuTACLP inherits an efficient standard constraint-based implementation of annotations from the TACLP framework.

As far as the multi-theory setting is concerned, i.e. the possibility offered by MuTACLP to structure and compose (temporal) knowledge, there are few logic-based approaches providing the user with these tools. One is Temporal Datalog [34], an extension of Datalog based on a simple temporal logic with two temporal operators, namely *first* and *next*. Temporal Datalog introduces a notion of module, which however does not seem to be used as a knowledge representation tool but rather to define new non-standard algebraic operators. In fact, to query a temporal Datalog program, Orgun proposes a "point-wise extension" of the relational algebra upon the set of natural numbers, called TRA-algebra. Then he provides a mechanism for specifying generic modules, called temporal modules, which are parametric Temporal Datalog programs, with a number of input predicates (parameters) and an output predicate. A module can be then regarded as an operator which, given a temporal relation, returns a temporal relation. Thus temporal modules are indeed used as operators of TRA, through which one has access to the use of recursion, arithmetic predicates and temporal operators.

A multi-theory framework in which temporal information can be handled, based on *annotated logics*, is proposed by Subrahmanian in [44]. This is a very general framework aimed at amalgamating multiple knowledge bases which can also contain temporal information. The knowledge bases are GAPs [25] and temporal information is modeled by using an appropriate lattice of annotations. In order to integrate these programs, a so called *Mediatory Database* is given, which is a GAP having clauses of the form

$$A_0 : [m, \mu] \leftarrow A_1 : [D_1, \mu_1], \ldots, A_n : [D_n, \mu_n]$$

where each D_i is a set of database names. Intuitively, a ground instance of a clause in the mediator can be interpreted as follows: "If the databases in set D_i, $1 \leq i \leq n$, (jointly) imply that the truth value of A_i is at least μ_i, then the mediator will conclude that the truth value of A_0 is at least μ". Essentially the fundamental mechanism provided to combine knowledge bases is a kind of message passing. Roughly speaking, the resolution of an atom $A_i : [D_i, \mu_i]$ is delegated to different databases, specified by the set D_i of database names, and the annotation μ_i is obtained by considering the least upper bounds of the annotations of each A_i computed in the distinct databases. Our approach is quite different because the meta-level composition operators allow us to access not only to the relation defined by a predicate but also to the definition of the predicate. For instance $P \cup Q$ is equivalent to a program whose clauses are the union of the clauses of P and Q and thus the information which can be derived from $P \cup Q$ is greater than the union of what we can derive from P and Q separately.

8 Conclusion

In this paper we have introduced MuTACLP, a language which joins the advantages of TACLP in handling temporal information with the ability to structure and compose programs. The proposed framework allows one to deal with time points and time periods and to model definite, indefinite and periodic temporal information, which can be distributed among different theories. Representing knowledge in separate programs naturally leads to use knowledge from different sources; information can be stored at different sites and combined in a modular way by employing the meta-level operators. This modular approach also favors the *reuse* of the knowledge encoded in the programs for future applications.

The language MuTACLP has been given a top-down semantics by means of a meta-interepreter and a bottom-up semantics based on an immediate consequence operator. Concerning the bottom-up semantics, it would be interesting to investigate on different definitions of the immediate consequence operator, for instance by considering an operator similar to the function R_P for generalized annotated programs [25]. The domain of interpretations considered in this paper is, in a certain sense, unstructured: interpretations are general sets of annotated atoms and the order, which is simply subset inclusion, does not take into account the order on annotations. Alternative solutions, based on different notions of interpretation, may consider more abstract domains. These domains can be obtained by endowing $\mathcal{C}\text{-}base_L \times Ann$ with the product order (induced by the identity relation on $\mathcal{C}\text{-}base_L$ and the order on Ann) and then by taking as elements of the domain (i.e. as interpretations) only those subsets of annotated atoms that satisfy some closure properties with respect to such an order. For instance, one can require "downward-closedness", which amounts to including subsumption in the immediate consequence 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.

In [14] the language TACLP is presented as an instance of annotated constraint logic (ACL) for reasoning about time. Similarly, we could have first introduced a Multi-theory Annotated Constraint Logic (MuACL in brief), viewing MuTACLP as an instance of MuACL. To define MuACL the constructions described in this paper should be generalized by using, as basic language for plain programs, the more general paradigm of ACL where atoms can be labelled by a general class of annotations. In defining MuACL we should require that the class of annotations forms a lattice, in order to have both upper bounds and lower bounds (the latter are necessary for the definition of the intersection operator). Indeed, it is not difficult to see that, under the assumption that only atoms can be annotated and clauses are free of negation, both the meta-interpreter and the immediate consequence operator smootly generalize to deal with general annotations.

Another interesting topic for future investigation is the treatment of negation. In the line of Frühwirth, a possible solution consists of embodying the "negation by default" of logic programming into MuTACLP by exploiting the logical equalities proved in [14]:

$$((\neg A)\,\mathtt{th}\,I) \;\Leftrightarrow\; \neg(A\,\mathtt{in}\,I) \qquad\qquad ((\neg A)\,\mathtt{in}\,I) \;\Leftrightarrow\; \neg(A\,\mathtt{th}\,I)$$

Consequently, the meta-interpreter is extended with two clauses which use such equalities:

$$\begin{array}{l} demo(\mathcal{E}, (\neg A)\,\mathtt{th}\,I) \leftarrow \neg demo(\mathcal{E}, A\ \mathtt{in}\,I) \\ demo(\mathcal{E}, (\neg A)\,\mathtt{in}\,I) \leftarrow \neg demo(\mathcal{E}, A\ \mathtt{th}\,I) \end{array}$$

However the interaction between negation by default and program composition operations is still to be fully understood. Some results on the semantic interactions between operations and negation by default are presented in [7], where, nevertheless, the handling of time is not considered.

Furthermore, it is worth noticing that in this paper we have implicitly assumed that the same unit for time is used in different programs, i.e. we have not dealt with different time *granularities*. The ability to cope with different granularities (e.g. seconds, days, etc.) is particularly relevant to support interoperability among systems. A simple way to handle this feature, is by introducing in MuTACLP a notion of *time unit* and a set of conversion predicates which transform time points into the chosen time unit (see, e.g., [4]).

We finally observe that in MuTACLP also spatial data can be naturally modelled. In fact, in the style of the constraint databases approaches (see, e.g., [24, 36, 19]) spatial data can be represented by using constraints. The facilities to handle time offered by MuTACLP allows one to easily establish spatio-temporal correlations, for instance time-varying areas, or, more generally, moving objects, supporting either discrete or continuous changes (see [37, 30, 39]).

Acknowledgments: This work has been partially supported by Esprit Working Group 28115 - DeduGIS.

References

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

[2] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

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

[4] C. Bettini, X. S. Wang, and S. Jajodia. An architecture for supporting interoperability among temporal databases. In *[12]*, pages 36–55.

[5] K.A. Bowen and R.A. Kowalski. Amalgamating language and metalanguage in logic programming. In K. L. Clark and S.-A. Tarnlund, editors, *Logic programming*, volume 16 of *APIC studies in data processing*, pages 153–172. Academic Press, 1982.

[6] A. Brogi. *Program Construction in Computational Logic.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 1993.

[7] A. Brogi, S. Contiero, and F. Turini. Programming by combining general logic programs. *Journal of Logic and Computation*, 9(1):7–24, 1999.

[8] A. Brogi, P. Mancarella, D. Pedreschi, and F. Turini. Modular logic programming. *ACM Transactions on Programming Languages and Systems*, 16(4):1361–1398, 1994.

[9] C. Brzoska. Temporal Logic Programming with Metric and Past Operators. In *[13]*, pages 21–39.

[10] 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.

[11] J. Chomicki and T. Imielinski. Temporal Deductive Databases and Infinite Objects. In *Proceedings of ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 61–73, 1988.

[12] O. Etzion, S. Jajodia, and S. Sripada, editors. *Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science.* Springer, 1998.

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

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

[15] D. M. Gabbay. Modal and temporal logic programming. In *[17]*, pages 197–237.

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

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

[18] A. Galton. A Critical Examination of Allen's Theory of Action and Time. *Artificial Intelligence*, 42:159–188, 1990.

[19] S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE system for complex spatial queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, pages 213–224, 1998.

[20] T. Hrycej. A temporal extension of Prolog. *Journal of Logic Programming*, 15(1& 2):113–145, 1993.

[21] J. Jaffar and M.J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19 & 20:503–582, 1994.

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

[23] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP(R) Language and System. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[24] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.

[25] M. Kifer and V.S. Subrahmanian. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming*, 12:335–367, 1992.

[26] M. Koubarakis. Database models for infinite and indefinite temporal information. *Information Systems*, 19(2):141–173, 1994.

[27] R. A. Kowalski and M.J. Sergot. A Logic-based Calculus of Events. *New Generation Computing*, 4(1):67–95, 1986.

[28] R.A. Kowalski and J.S. Kim. A metalogic programming approach to multi-agent knowledge and belief. In *Artificial Intelligence and Mathematical Theory of Computation*. Academic Press, 1991.

[29] S.M. Leach and J.J. Lu. Computing Annotated Logic Programs. In *Proceedings of the eleventh International Conference on Logic Programming*, pages 257–271, 1994.

[30] P. Mancarella, G. Nerbini, A. Raffaetà, and F. Turini. MuTACLP: A language for declarative GIS analysis. In *Proceedings of the Sixth International Conference on Rules and Objects in Databases (DOOD2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 1002–1016. Springer, 2000.

[31] 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.

[32] P. Mancarella, A. Raffaetà, and F. Turini. Temporal Annotated Constraint Logic Programming with Multiple Theories. In *Tenth International Workshop on Database and Expert Systems Applications*, pages 501–508. IEEE Computer Society Press, 1999.

[33] B. Martens and D. De Schreye. Why Untyped Nonground Metaprogramming Is Not (Much Of) A Problem. *Journal of Logic Programming*, 22(1):47–99, 1995.

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

[35] 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. Springer, 1994.

[36] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proceedings of the 13th ACM Symposium on Principles of Database Systems*, pages 279–288, 1994.

[37] A. Raffaetà. *Spatio-temporal knowledge bases in a constraint logic programming framework with multiple theories*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2000.

[38] A. Raffaetà and T. Frühwirth. Semantics for Temporal Annotated Constraint Logic Programming. In *Labelled Deduction*, volume 17 of *Applied Logic Series*, pages 215–243. Kluwer Academic, 2000.

[39] A. Raffaetà and C. Renso. Temporal Reasoning in Geographical Information Systems. In *International Workshop on Advanced Spatial Data Management (DEXA Workshop)*, pages 899–905. IEEE Computer Society Press, 2000.

[40] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The British Nationality Act as a logic program. *Communications of the ACM*, 29(5):370–386, 1986.

[41] S. Sripada and P. Möller. The Generalized ChronoBase Temporal Data Model. In *Meta-logics and Logic Programming*, pages 310–335. MIT Press, 1995.

[42] S.M. Sripada. A logical framework for temporal deductive databases. In *Proceedings of the Very Large Databases Conference*, pages 171–182, 1988.

[43] S.M. Sripada. *Temporal Reasoning in Deductive Databases*. PhD thesis, Department of Computing Imperial College of Science & Technology, 1991.

[44] V. S. Subrahmanian. Amalgamating Knowledge Bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.

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

[46] C. Zaniolo, N. Arni, and K. Ong. Negation and aggregates in recursive rules: The LDL++Approach. In *International conference on Deductive and Object-Oriented Databases (DOOD'93)*, volume 760 of *Lecture Notes in Computer Science*. Springer, 1993.

Appendix: Proofs

Proposition 9 Let I_1 and I_2 be two interpretations. Then

$$\downarrow (I_1 \Cap I_2) \ = \downarrow I_1 \bigcap \downarrow I_2.$$

Proof . Assume $(A,\alpha) \in\downarrow (I_1 \Cap I_2)$. By definition of downward closure there exists γ such that $(A,\gamma) \in I_1 \Cap I_2$ and $\mathcal{D_C} \models \alpha \sqsubseteq \gamma$. By definition of $\Cap$ there exist β and β' such that $(A,\beta) \in I_1$ and $(A,\beta') \in I_2$ and $\mathcal{D_C} \models \beta \sqcap \beta' = \gamma$. Therefore $\mathcal{D_C} \models \alpha \sqsubseteq \beta, \alpha \sqsubseteq \beta'$, by definition of downward closure we conclude $(A,\alpha) \in\downarrow I_1$ and $(A,\alpha) \in\downarrow I_2$, i.e., $(A,\alpha) \in\downarrow I_1 \bigcap \downarrow I_2$.

Vice versa assume $(A,\alpha) \in\downarrow I_1 \cap \downarrow I_2$. By definition of set-theoretic intersection and downward closure there exist β and β' such that $\mathcal{D_C} \models \alpha \sqsubseteq \beta, \alpha \sqsubseteq \beta'$ and $(A,\beta) \in I_1$ and $(A,\beta') \in I_2$. By definition of $\Cap$, $(A,\gamma) \in I_1 \Cap I_2$ and $\mathcal{D_C} \models \beta \sqcap \beta' = \gamma$. By property of the greatest lower bound $\mathcal{D_C} \models \alpha \sqsubseteq \beta \sqcap \beta'$, hence $(A,\alpha) \in\downarrow (I_1 \Cap I_2)$.

Theorem 10 Let $\mathcal{E}$ be a program expression. The function $\mathbb{T}^{\mathcal{C}}_{\mathcal{E}}$ is continuous (on $(\wp(\mathcal{C}\text{-}base_L \times Ann), \subseteq)$).

Proof . Let $\{I_i\}_{i\in\mathbb{N}}$ be a chain in $(\wp(\mathcal{C}\text{-}base_L \times Ann), \subseteq)$, i.e., $I_0 \subseteq I_1 \subseteq \ldots \subseteq I_i \ldots$. Then we have to prove

$$(A,\alpha) \in \mathbb{T}^{\mathcal{C}}_{\mathcal{E}}\left(\bigcup_{i\in\mathbb{N}} I_i\right) \Longleftrightarrow (A,\alpha) \in \bigcup_{i\in\mathbb{N}} \mathbb{T}^{\mathcal{C}}_{\mathcal{E}}(I_i).$$

The proof is by structural induction of $\mathcal{E}$.

($\mathcal{E}$ is a plain program P).

$(A,\alpha) \in \mathbb{T}^{\mathcal{C}}_{P}(\bigcup_{i\in\mathbb{N}} I_i)$
$\Longleftrightarrow$ {definition of $\mathbb{T}^{\mathcal{C}}_{P}$}
$((\alpha = \mathtt{th}[s_1,s_2] \ \vee \ \alpha = \mathtt{in}[s_1,s_2]) \wedge$
$A\ \alpha \leftarrow C_1,\ldots,C_k,B_1\alpha_1,\ldots,B_n\alpha_n \in ground_{\mathcal{C}}(P) \ \wedge$
$\{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq \bigcup_{i\in\mathbb{N}} I_i \ \wedge$
$\mathcal{D_C} \models C_1,\ldots,C_k,\alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, s_1 \le s_2) \ \vee$
$(\alpha = \mathtt{th}[s_1,r_2] \ \wedge \ A\ \mathtt{th}[s_1,s_2] \leftarrow C_1,\ldots,C_k,B_1\alpha_1,\ldots,B_n\alpha_n \in ground_{\mathcal{C}}(P) \ \wedge$
$\{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq \bigcup_{i\in\mathbb{N}} I_i \ \wedge \ (A,\mathtt{th}[r_1,r_2]) \in \bigcup_{i\in\mathbb{N}} I_i \ \wedge$
$\mathcal{D_C} \models C_1,\ldots,C_k,\alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, s_1 < r_1, r_1 \le s_2, s_2 < r_2)$
$\Longleftrightarrow$ {property of set-theoretic union and $\{I_i\}_{i\in\mathbb{N}}$ is a chain. Notice that for ($\Longrightarrow$) j can be any element of the set $\{k \mid (B_i,\beta_i) \in I_k, i = 1,\ldots,n\}$ which is clearly not empty}
$((\alpha = \mathtt{th}[s_1,s_2] \ \vee \ \mathtt{in}[s_1,s_2]) \wedge$
$A\ \alpha \leftarrow C_1,\ldots,C_k,B_1\alpha_1,\ldots,B_n\alpha_n \in ground_{\mathcal{C}}(P) \ \wedge$
$\{(B_1,\beta_1),\ldots,(B_n,\beta_n)\} \subseteq I_j \ \wedge$
$\mathcal{D_C} \models C_1,\ldots,C_k,\alpha_1 \sqsubseteq \beta_1,\ldots,\alpha_n \sqsubseteq \beta_n, s_1 \le s_2) \ \vee$

$(\alpha = \mathtt{th}[s_1, r_2] \;\wedge\; A\ \mathtt{th}[s_1, s_2] \leftarrow C_1, \ldots, C_k, B_1\alpha_1, \ldots, B_n\alpha_n \in ground_{\mathcal{C}}(P)$
$\wedge\ \{(B_1, \beta_1), \ldots, (B_n, \beta_n)\} \subseteq I_j \;\wedge\; (A, \mathtt{th}[r_1, r_2]) \in I_j \;\wedge$
$\mathcal{D}_{\mathcal{C}} \models C_1, \ldots, C_k, \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, s_1 < r_1, r_1 \leq s_2, s_2 < r_2)$
$\iff$ {definition of $\mathbb{T}^{\mathcal{C}}_P$}
$(A, \alpha) \in \mathbb{T}^{\mathcal{C}}_P(I_j)$
$\iff$ {set-theoretic union}
$(A, \alpha) \in \bigcup_{i\in\mathbb{N}} \mathbb{T}^{\mathcal{C}}_P(I_i)$

$(\mathcal{E} = \mathcal{Q} \cup \mathcal{R})$.

$(A, \alpha) \in \mathbb{T}^{\mathcal{C}}_{\mathcal{Q}\cup\mathcal{R}}(\bigcup_{i\in\mathbb{N}} I_i)$
$\iff$ {definition of $\mathbb{T}^{\mathcal{C}}_{\mathcal{Q}\cup\mathcal{R}}$}
$(A, \alpha) \in \mathbb{T}^{\mathcal{C}}_{\mathcal{Q}}(\bigcup_{i\in\mathbb{N}} I_i) \cup \mathbb{T}^{\mathcal{C}}_{\mathcal{R}}(\bigcup_{i\in\mathbb{N}} I_i)$
$\iff$ {inductive hypothesis}
$(A, \alpha) \in \left(\bigcup_{i\in\mathbb{N}} \mathbb{T}^{\mathcal{C}}_{\mathcal{Q}}(I_i)\right) \cup \left(\bigcup_{i\in\mathbb{N}} \mathbb{T}^{\mathcal{C}}_{\mathcal{R}}(I_i)\right)$
$\iff$ {properties of union}
$(A, \alpha) \in \bigcup_{i\in\mathbb{N}} \left(\mathbb{T}^{\mathcal{C}}_{\mathcal{Q}}(I_i) \cup \mathbb{T}^{\mathcal{C}}_{\mathcal{R}}(I_i)\right)$
$\iff$ {definition of $\mathbb{T}^{\mathcal{C}}_{\mathcal{Q}\cup\mathcal{R}}$}
$(A, \alpha) \in \bigcup_{i\in\mathbb{N}} \mathbb{T}^{\mathcal{C}}_{\mathcal{Q}\cup\mathcal{R}}(I_i)$

$(\mathcal{E} = \mathcal{Q} \cap \mathcal{R})$.

$(A, \alpha) \in \mathbb{T}^{\mathcal{C}}_{\mathcal{Q}\cap\mathcal{R}}(\bigcup_{i\in\mathbb{N}} I_i)$
$\iff$ {definition of $\mathbb{T}^{\mathcal{C}}_{\mathcal{Q}\cap\mathcal{R}}$}
$(A, \alpha) \in \mathbb{T}^{\mathcal{C}}_{\mathcal{Q}}(\bigcup_{i\in\mathbb{N}} I_i) \Cap \mathbb{T}^{\mathcal{C}}_{\mathcal{R}}(\bigcup_{i\in\mathbb{N}} I_i)$
$\iff$ {inductive hypothesis}
$(A, \alpha) \in \left(\bigcup_{i\in\mathbb{N}} \mathbb{T}^{\mathcal{C}}_{\mathcal{Q}}(I_i)\right) \Cap \left(\bigcup_{i\in\mathbb{N}} \mathbb{T}^{\mathcal{C}}_{\mathcal{R}}(I_i)\right)$
$\iff$ {definition of $\Cap$ and monotonicity of $\mathbb{T}^{\mathcal{C}}$}
$(A, \alpha) \in \bigcup_{i\in\mathbb{N}} \left(\mathbb{T}^{\mathcal{C}}_{\mathcal{Q}}(I_i) \Cap \mathbb{T}^{\mathcal{C}}_{\mathcal{R}}(I_i)\right)$
$\iff$ {definition of $\mathbb{T}^{\mathcal{C}}_{\mathcal{Q}\cap\mathcal{R}}$}
$(A, \alpha) \in \bigcup_{i\in\mathbb{N}} \mathbb{T}^{\mathcal{C}}_{\mathcal{Q}\cap\mathcal{R}}(I_i)$

Soundness and completeness

This section presents the proofs of the soundness and completeness results for MuTACLP meta-interpreter. We first fix some notational conventions. In the following we will denote by $\mathcal{E}$, $\mathcal{N}$, $\mathcal{R}$ and $\mathcal{Q}$ generic program expressions, and by $\mathcal{C}$ the fixed constraint domain where the constraints of object programs are interpreted. Let $\mathcal{M}$ be the fixed constraint domain, where the constraints of the meta-interpreter defined in Section 5.1 are interpreted. We denote by A, B elements of $\mathcal{C}$-base_L, with α, β, γ annotations in Ann and by C a $\mathcal{C}$-ground instance of a constraint. All symbols may have subscripts. In the following for simplicity we will drop the reference to $\mathcal{C}$ and $\mathcal{M}$ in the name of the immediate consequence operators. Moreover we refer to the program containing the meta-level representation of object level programs and clauses (1)-(10) as "the meta-program V *corresponding* to a program expression".

We will say that an interpretation $I \subseteq \mathcal{C}\text{-}base_L \times Ann$ *satisfies* the body of a $\mathcal{C}$-ground instance $A\,\alpha \leftarrow C_1, \ldots, C_k, B_1\alpha_1, \ldots, B_n\alpha_n$ of a clause, or in symbols $I \models C_1, \ldots, C_k, B_1\alpha_1, \ldots, B_n\alpha_n$, if

1. $\mathcal{D}_\mathcal{C} \models C_1, \ldots, C_k$ and
2. there are annotations $\beta_1, \ldots, \beta_n$ such that $\{(B_1, \beta_1), \ldots, (B_n, \beta_n)\} \subseteq I$ and $\mathcal{D}_\mathcal{C} \models \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n$.

Furthermore, will often denote a sequence $C_1, \ldots, C_k$ of $\mathcal{C}$-ground instances of constraints by $\bar{C}$, while a sequence $B_1\alpha_1, \ldots, B_n\alpha_n$ of annotated atoms in $\mathcal{C}\text{-}base_L \times Ann$ will be denoted by $\bar{B}$. For example, with this convention a clause of the kind $A\,\alpha \leftarrow C_1, \ldots, C_k, B_1\alpha_1, \ldots, B_n\alpha_n$ will be written as $A\,\alpha \leftarrow \bar{C}, \bar{B}$, and, similarly, in the meta-level representation, we will write $clause(\mathcal{E}, A\,\alpha, (\bar{C}, \bar{B}))$ in place of $clause(\mathcal{E}, A\,\alpha, (C_1, \ldots, C_k, B_1\alpha_1, \ldots, B_n\alpha_n))$.

Soundness.

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

LEMMA 14 *Let $\mathcal{E}$ be a program expression and let V be the corresponding meta-interpreter. For any $B_1\,\alpha_1, \ldots, B_n\,\alpha_n$ with $B_i \in \mathcal{C}\text{-}base_L$ and $\alpha_i \in Ann$ and for any $C_1, \ldots, C_k$, with C_i a $\mathcal{C}$-ground instance of a constraint, we have:*

$$\begin{aligned} &\textit{For all } h \quad demo(\mathcal{E}, (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_V^h \\ &\qquad \Longrightarrow \{demo(\mathcal{E}, B_1\,\alpha_1), \ldots, demo(\mathcal{E}, B_n\,\alpha_n)\} \subseteq T_V^h \ \wedge \ \mathcal{D}_\mathcal{C} \models C_1, \ldots, C_k. \end{aligned}$$

Proof . The result easily follows from the definition of T_V and from clauses (2) and (7) of the meta-interpreter. We next present the detailed proof.

Notice that if $n = 1$ and $k = 0$ the proof is immediate and if $n = 0$ and $k = 1$ the claim trivially follows by the definition of T_V and clause (7).

For $n + k \geq 2$, the proof is by induction on h.
(Base case). Trivial since $T_V^0 = \emptyset$.

(Inductive case). Assume that

$$\begin{aligned} &demo(\mathcal{E}, (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_V^h \\ &\qquad \Longrightarrow \{demo(\mathcal{E}, B_1\,\alpha_1), \ldots, demo(\mathcal{E}, B_n\,\alpha_n)\} \subseteq T_V^h \ \wedge \ \mathcal{D}_\mathcal{C} \models C_1, \ldots, C_k \end{aligned}$$

Then:

$$\begin{aligned} &demo(\mathcal{E}, (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_V^{h+1} \\ \Longleftrightarrow &\{\text{definition of } T_V^i\} \\ &demo(\mathcal{E}, (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_V(T_V^h) \end{aligned}$$

$\iff$ {definition of T_V, clause (2) and
$(G_1, \ldots, G_{n+k}) = (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)$}
$\exists\, 1 \le r < n+k:\ \{demo(\mathcal{E}, (G_1, \ldots, G_r)), demo(\mathcal{E}, (G_{r+1}, \ldots, G_{n+k}))\} \subseteq T_V^h$
$\Longrightarrow$ {inductive hypothesis or previous remark if $r = 1$ or $r = n+k-1$}
$\{demo(\mathcal{E}, G_i) \mid 1 \le i \le r,\ G_i$ is an annotated atom$\} \subseteq T_V^h\ \wedge$
$\mathcal{D}_\mathcal{C} \models G_{m_1}, \ldots, G_{m_i}$ with $G_{m_1}, \ldots, G_{m_i}$ the constraints in $\{G_1, \ldots, G_r\}\ \wedge$
$\{demo(\mathcal{E}, G_i) \mid r+1 \le i \le n+k,\ G_i$ is an annotated atom$\} \subseteq T_V^h\ \wedge$
$\mathcal{D}_\mathcal{C} \models G_{l_1}, \ldots, G_{l_j}$ with $G_{l_1}, \ldots, G_{l_j}$ the constraints in $\{G_{r+1}, \ldots, G_{n+k}\}$
$\Longrightarrow$ {monotonicity of T_V}
$\{demo(\mathcal{E}, B_1\,\alpha_1), \ldots, demo(\mathcal{E}, B_n\,\alpha_n)\} \subseteq T_V^{h+1}\ \wedge\ \mathcal{D}_\mathcal{C} \models C_1, \ldots, C_k$

The next two lemmata relate the clauses computed from a program expression $\mathcal{E}$ at the meta-level, called "virtual clauses", with the set of consequences of $\mathcal{E}$. The first lemma states that whenever we can find a virtual clause computed from $\mathcal{E}$ whose body is satisfied by I, the head $A\,\alpha$ of the clause is a consequence of the program expression $\mathcal{E}$. The second one shows how the head of a virtual clause can be "joined" with an already existing annotated atom in order to obtain an atom with a larger `th` annotation.

Lemma 15 (Virtual Clauses Lemma 1) *Let $\mathcal{E}$ be a program expression and V be the corresponding meta-interpreter. For any sequence $\bar{C}$ of $\mathcal{C}$-ground instances of constraints, for any $A\,\alpha$, $\bar{B}$ in $\mathcal{C}$-base$_L \times Ann$ and any interpretation $I \subseteq \mathcal{C}$-base$_L \times Ann$, we have:*

$$clause(\mathcal{E}, A\,\alpha, (\bar{C}, \bar{B})) \in T_V^\omega\ \wedge\ I \models \bar{C}, \bar{B} \qquad \Longrightarrow \qquad (A, \alpha) \in \mathbb{T}_\mathcal{E}(I).$$

Proof. The proof is by structural induction on $\mathcal{E}$.

($\mathcal{E}$ is a plain program P).

$clause(P, A\,\alpha, (\bar{C}, \bar{B})) \in T_V^\omega\ \wedge\ I \models \bar{C}, \bar{B}$
$\Longrightarrow$ {definition of T_V and clause (11)}
$(\alpha = \mathtt{th}[s_1, s_2]\ \vee \alpha = \mathtt{in}[s_1, s_2])\ \wedge$
$A\,\alpha \leftarrow \bar{C}, \bar{B} \in ground_\mathcal{C}(P)\ \wedge\ I \models \bar{C}, \bar{B}, s_1 \le s_2$
$\Longrightarrow$ {definition of $\mathbb{T}_P$}
$(A, \alpha) \in \mathbb{T}_P(I)$

($\mathcal{E} = \mathcal{Q} \cup \mathcal{R}$).

$clause(\mathcal{Q} \cup \mathcal{R}, A\,\alpha, (\bar{C}, \bar{B})) \in T_V^\omega\ \wedge\ I \models \bar{C}, \bar{B}$
$\Longrightarrow$ {definition of T_V and clauses (8) and (9)}
$(clause(\mathcal{Q}, A\,\alpha, (\bar{C}, \bar{B})) \in T_V^\omega\ \wedge\ I \models \bar{C}, \bar{B}\ \vee$
$(clause(\mathcal{R}, A\,\alpha, (\bar{C}, \bar{B})) \in T_V^\omega\ \wedge I \models \bar{C}, \bar{B}$
$\Longrightarrow$ {inductive hypothesis}
$(A, \alpha) \in \mathbb{T}_\mathcal{Q}(I)\ \vee\ (A, \alpha) \in \mathbb{T}_\mathcal{R}(I)$
$\Longrightarrow$ {definition of $\mathbb{T}_{\mathcal{Q} \cup \mathcal{R}}$ }
$(A, \alpha) \in \mathbb{T}_{\mathcal{Q} \cup \mathcal{R}}(I)$

($\mathcal{E} = \mathcal{Q} \cap \mathcal{R}$).

$$
\begin{array}{l}
\quad clause(\mathcal{Q} \cap \mathcal{R}, A\,\alpha, (\bar{C}, \bar{B})) \in T_V^\omega \;\wedge\; I \models \bar{C}, \bar{B} \\
\Longrightarrow \{(\bar{C}', \bar{B}') \text{ and } (\bar{C}'', \bar{B}'') \text{ partition of } (\bar{C}, \bar{B}), \text{ definition of } T_V \text{ and clause (10)}\} \\
\quad clause(\mathcal{Q}, A\,\beta, (\bar{C}', \bar{B}')) \in T_V^\omega \;\wedge\; clause(\mathcal{R}, A\,\gamma, (\bar{C}'', \bar{B}'')) \in T_V^\omega \;\wedge \\
\quad I \models \bar{C}', \bar{B}' \;\wedge\; I \models \bar{C}'', \bar{B}'' \;\wedge\; \mathcal{D}_\mathcal{C} \models \beta \sqcap \gamma = \alpha \\
\Longrightarrow \{\text{inductive hypothesis}\} \\
\quad (A, \beta) \in \mathbb{T}_\mathcal{Q}(I) \;\wedge\; (A, \gamma) \in \mathbb{T}_\mathcal{R}(I) \;\wedge\; \mathcal{D}_\mathcal{C} \models \beta \sqcap \gamma = \alpha \\
\Longrightarrow \{\text{definition of } \Cap\} \\
\quad (A, \alpha) \in \mathbb{T}_{\mathcal{Q} \cap \mathcal{R}}(I)
\end{array}
$$

Lemma 16 (Virtual Clauses Lemma 2) *Let $\mathcal{E}$ be a program expression and V be the corresponding meta-program. For any $A\,\mathtt{th}[s_1, s_2]$, $A\,\mathtt{th}[r_1, r_2]$, $\bar{B}$ in $\mathcal{C}$-base$_L \times Ann$, for any sequence $\bar{C}$ of $\mathcal{C}$-ground instances of constraints, and any interpretation $I \subseteq \mathcal{C}$-base$_L \times Ann$, the following statement holds:*

$$
\begin{array}{l}
clause(\mathcal{E}, A\,\mathtt{th}[s_1, s_2], (\bar{C}, \bar{B})) \in T_V^\omega \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge \\
(A, \mathtt{th}[r_1, r_2]) \in I \;\wedge\; \mathcal{D}_\mathcal{C} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2 \\
\qquad\qquad\quad \Longrightarrow \quad (A, \mathtt{th}[s_1, r_2]) \in \mathbb{T}_\mathcal{E}(I).
\end{array}
$$

Proof . The proof is by structural induction on $\mathcal{E}$ and it is very similar to the proof of Lemma 15.

($\mathcal{E}$ is a plain program P).

$$
\begin{array}{l}
\quad clause(P, A\,\mathtt{th}[s_1, s_2], (\bar{C}, \bar{B})) \in T_V^\omega \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; (A, \mathtt{th}[r_1, r_2]) \in I \;\wedge \\
\quad \mathcal{D}_\mathcal{C} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2 \\
\Longrightarrow \{P \text{ is a plain program and definition of } T_V \text{ and clause (11)}\} \\
\quad A\,\mathtt{th}[s_1, s_2] \leftarrow \bar{C}, \bar{B} \in ground_\mathcal{C}(P) \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; (A, \mathtt{th}[r_1, r_2]) \in I \;\wedge \\
\quad \mathcal{D}_\mathcal{C} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2 \\
\Longrightarrow \{\text{definition of } \mathbb{T}_P\} \\
\quad (A, \mathtt{th}[s_1, r_2]) \in \mathbb{T}_P(I)
\end{array}
$$

($\mathcal{E} = \mathcal{Q} \cup \mathcal{R}$).

$$
\begin{array}{l}
\quad clause(\mathcal{Q} \cup \mathcal{R}, A\,\mathtt{th}[s_1, s_2], (\bar{C}, \bar{B})) \in T_V^\omega \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; (A, \mathtt{th}[r_1, r_2]) \in I \;\wedge \\
\quad \mathcal{D}_\mathcal{C} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2 \\
\Longrightarrow \{\text{definition of } T_V \text{ and clauses (8) and (9)}\} \\
\quad (clause(\mathcal{Q}, A\,\mathtt{th}[s_1, s_2], (\bar{C}, \bar{B})) \in T_V^\omega \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; (A, \mathtt{th}[r_1, r_2]) \in I \;\wedge \\
\quad \mathcal{D}_\mathcal{C} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2) \;\vee \\
\quad (clause(\mathcal{R}, A\,\mathtt{th}[s_1, s_2], (\bar{C}, \bar{B})) \in T_V^\omega \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; (A, \mathtt{th}[r_1, r_2]) \in I \;\wedge \\
\quad \mathcal{D}_\mathcal{C} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2) \\
\Longrightarrow \{\text{inductive hypothesis}\} \\
\quad (A, \mathtt{th}[s_1, r_2]) \in \mathbb{T}_\mathcal{Q}(I) \vee\; (A, \mathtt{th}[s_1, r_2]) \in \mathbb{T}_\mathcal{R}(I) \\
\Longrightarrow \{\text{definition of } \mathbb{T}_{\mathcal{Q} \cup \mathcal{R}}\;\} \\
\quad (A, \mathtt{th}[s_1, r_2]) \in \mathbb{T}_{\mathcal{Q} \cup \mathcal{R}}(I)
\end{array}
$$

($\mathcal{E} = \mathcal{Q} \cap \mathcal{R}$).

$clause(\mathcal{Q} \cap \mathcal{R}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})) \in T_V^\omega \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; (A, \mathtt{th}\,[r_1, r_2]) \in I \;\wedge$
$\mathcal{D_C} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2$
$\Longrightarrow$ {$(\bar{C}', \bar{B}')$ and $(\bar{C}'', \bar{B}'')$ partition of $(\bar{B}, \bar{C})$, definition of T_V, axiom ($\mathtt{th}\sqcap$) and clause (10)}
$clause(\mathcal{Q}, A\,\mathtt{th}\,[t_1, t_2], (\bar{C}', \bar{B}')) \in T_V^\omega \;\wedge\; clause(\mathcal{R}, A\,\mathtt{th}\,[w_1, w_2], (\bar{C}'', \bar{B}'')) \in T_V^\omega$
$\wedge\; s_1 = max\{t_1, w_1\} \;\wedge\; s_2 = min\{t_2, w_2\} \;\wedge\; I \models \bar{C}', \bar{B}' \;\wedge\; I \models \bar{C}'', \bar{B}'' \;\wedge$
$(A, \mathtt{th}\,[r_1, r_2]) \in I \;\wedge\; \mathcal{D_C} \models s_1 < r_1, r_1 \leq s_2, s_2 < r_2$

We distinguish three cases with respect to the relation between w_2, t_2 and r_2.

- $\mathcal{D_C} \models t_2 < r_2, w_2 < r_2$.
  Since $\mathcal{D_C} \models t_1 < r_1, r_1 \leq t_2, t_2 < r_2$ and $\mathcal{D_C} \models w_1 < r_1, r_1 \leq w_2, w_2 < r_2$, by inductive hypothesis we have $(A, \mathtt{th}\,[t_1, r_2]) \in \mathbb{T}_\mathcal{Q}(I) \;\wedge\; (A, \mathtt{th}\,[w_1, r_2]) \in \mathbb{T}_\mathcal{R}(I)$. Recalling that $s_1 = max\{t_1, w_1\}$ and $\mathcal{D_C} \models s_1 < r_2$, by definition of $\Cap$ we obtain $(A, \mathtt{th}\,[s_1, r_2]) \in \mathbb{T}_{\mathcal{Q}\cap\mathcal{R}}(I)$.

- $\mathcal{D_C} \models t_2 < r_2, r_2 \leq w_2$.
  Since $\mathcal{D_C} \models t_1 < r_1, r_1 \leq t_2, t_2 < r_2$, by inductive hypothesis on $\mathcal{Q}$, we have that $(A, \mathtt{th}\,[t_1, r_2]) \in \mathbb{T}_\mathcal{Q}(I)$. Furthermore, Lemma 15 applied to $\mathcal{R}$ ensures that $(A, \mathtt{th}\,[w_1, w_2]) \in \mathbb{T}_\mathcal{R}(I)$. Since $s_1 = max\{t_1, w_1\}$ and $\mathcal{D_C} \models r_2 \leq w_2, s_1 < r_2$, by definition of $\Cap$ we can conclude $(A, \mathtt{th}\,[s_1, r_2]) \in \mathbb{T}_{\mathcal{Q}\cap\mathcal{R}}(I)$.

- $\mathcal{D_C} \models r_2 \leq t_2, w_2 < r_2$.
  By symmetry from the previous case.

The analyzed cases are exhaustive: $\mathcal{D_C} \not\models r_2 \leq t_2, r_2 \leq w_2$ since $\mathcal{D_C} \models s_2 < r_2$ and $s_2 = min\{t_2, w_2\}$.

Now, the soundness of the meta-interpreter can be proved by showing that if an annotated atom $A\,\alpha$ is provable at the meta-level from the program expression $\mathcal{E}$ then $A\,\gamma$ is a consequence of $\mathcal{E}$ for some γ such that $A\,\gamma \Rightarrow A\,\alpha$, i.e., the annotation α is less or equal to γ.

THEOREM 17 (SOUNDNESS) *Let $\mathcal{E}$ be a program expression and let V be the corresponding meta-program. For any $A\,\alpha$ with $A \in \mathcal{C}$-$base_L$ and $\alpha \in Ann$, the following statement holds:*

$$demo(\mathcal{E}, A\,\alpha) \in T_V^\omega \qquad \Longrightarrow \qquad (A, \alpha) \in \mathbb{F}^\mathcal{C}(\mathcal{E}).$$

Proof . We first show that for all h

$$demo(\mathcal{E}, A\,\alpha) \in T_V^h \qquad \Longrightarrow \qquad \exists\gamma : \; (A, \gamma) \in \mathbb{T}_\mathcal{E}^\omega \;\wedge\; \mathcal{D_C} \models \alpha \sqsubseteq \gamma. \tag{12}$$

The proof is by induction on h.
(Base case). Trivial since $T_V^0 = \emptyset$.

(Inductive case). Assume that

$$demo(\mathcal{E}, A\,\alpha) \in T_V^h \qquad \Longrightarrow \qquad \exists\gamma : \; (A, \gamma) \in \mathbb{T}_\mathcal{E}^\omega \;\wedge\; \mathcal{D_C} \models \alpha \sqsubseteq \gamma.$$

Then:

$$
\begin{array}{l}
\quad demo(\mathcal{E}, A\,\alpha) \in T_V^{h+1} \\
\iff \{\text{definition of } T_V^i\} \\
\quad demo(\mathcal{E}, A\,\alpha) \in T_V(T_V^h)
\end{array}
$$

We have four cases corresponding to clauses (3), (4), (5) and (6). We only show the cases related to clause (3) and (4) since the others are proved in an analogous way.

$$
\begin{array}{l}
\text{(clause (3))} \quad \{\alpha = \mathtt{th}\,[t_1, t_2], \text{ definition of } T_V \text{ and clause (3)}\} \\
\quad \{clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})), demo(\mathcal{E}, (\bar{C}, \bar{B}))\} \subseteq T_V^h \,\wedge \\
\quad \mathcal{D}_\mathcal{C} \models s_1 \le t_1, t_2 \le s_2, t_1 \le t_2 \\
\Longrightarrow \{\text{Lemma 14 and } (\bar{C}, \bar{B}) = (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)\} \\
\quad clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_V^h \,\wedge \\
\quad \{demo(\mathcal{E}, B_1\,\alpha_1), \ldots, demo(\mathcal{E}, B_n\,\alpha_n)\} \subseteq T_V^h \,\wedge \\
\quad \mathcal{D}_\mathcal{C} \models C_1, \ldots, C_k \;\wedge \mathcal{D}_\mathcal{C} \models s_1 \le t_1, t_2 \le s_2, t_1 \le t_2 \\
\Longrightarrow \{\text{inductive hypothesis}\} \\
\quad \exists \beta_1, \ldots, \beta_n :\; clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_V^h \,\wedge \\
\quad \{(B_1, \beta_1), \ldots, (B_n, \beta_n)\} \subseteq \mathbb{T}_\mathcal{E}^\omega \;\wedge\; \mathcal{D}_\mathcal{C} \models \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n \,\wedge \\
\quad \mathcal{D}_\mathcal{C} \models C_1, \ldots, C_k \;\wedge\; \mathcal{D}_\mathcal{C} \models s_1 \le t_1, t_2 \le s_2, t_1 \le t_2 \\
\Longrightarrow \{T_V^\omega = \bigcup_{i \in \mathbb{N}} T_V^i\} \\
\quad clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_V^\omega \,\wedge \\
\quad \{(B_1, \beta_1), \ldots, (B_n, \beta_n)\} \subseteq \mathbb{T}_\mathcal{E}^\omega \;\wedge\; \mathcal{D}_\mathcal{C} \models \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n \,\wedge \\
\quad \mathcal{D}_\mathcal{C} \models C_1, \ldots, C_k \;\wedge\; \mathcal{D}_\mathcal{C} \models s_1 \le t_1, t_2 \le s_2, t_1 \le t_2 \\
\Longrightarrow \{\text{Lemma 15}\} \\
\quad (A, \mathtt{th}\,[s_1, s_2]) \in \mathbb{T}_\mathcal{E}(\mathbb{T}_\mathcal{E}^\omega) \;\wedge\; \mathcal{D}_\mathcal{C} \models s_1 \le t_1, t_2 \le s_2, t_1 \le t_2 \\
\Longrightarrow \{\mathbb{T}_\mathcal{E}^\omega \text{ is a fixpoint of } \mathbb{T}_\mathcal{E} \text{ and } \mathcal{D}_\mathcal{C} \models s_1 \le t_1, t_2 \le s_2, t_1 \le t_2\} \\
\quad (A, \mathtt{th}\,[s_1, s_2]) \in \mathbb{T}_\mathcal{E}^\omega \;\wedge\; \mathcal{D}_\mathcal{C} \models \mathtt{th}\,[t_1, t_2] \sqsubseteq \mathtt{th}\,[s_1, s_2]
\end{array}
$$

$$
\begin{array}{l}
\text{(clause (4))} \quad \{\alpha = \mathtt{th}\,[t_1, t_2], \text{ definition of } T_V \text{ and clause (4)}\} \\
\quad \{clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})), demo(\mathcal{E}, (\bar{C}, \bar{B})), demo(\mathcal{E}, A\,\mathtt{th}\,[s_2, t_2])\} \subseteq T_V^h \\
\quad \wedge\, \mathcal{D}_\mathcal{C} \models s_1 \le t_1, t_1 < s_2, s_2 < t_2 \\
\Longrightarrow \{\text{Lemma 14 and } (\bar{C}, \bar{B}) = (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)\} \\
\quad clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_V^h \,\wedge \\
\quad \{demo(\mathcal{E}, B_1\,\alpha_1), \ldots, demo(\mathcal{E}, B_n\,\alpha_n), demo(\mathcal{E}, A\,\mathtt{th}\,[s_2, t_2])\} \subseteq T_V^h \,\wedge \\
\quad \mathcal{D}_\mathcal{C} \models C_1, \ldots, C_k \;\wedge\; \mathcal{D}_\mathcal{C} \models s_1 \le t_1, t_1 < s_2, s_2 < t_2 \\
\Longrightarrow \{\text{inductive hypothesis}\} \\
\quad \exists \beta, \beta_1, \ldots, \beta_n :\; clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (C_1, \ldots, C_k, B_1\,\alpha_1, \ldots, B_n\,\alpha_n)) \in T_V^h \,\wedge \\
\quad \{(B_1, \beta_1), \ldots, (B_n, \beta_n), (A, \beta)\} \subseteq \mathbb{T}_\mathcal{E}^\omega \,\wedge \\
\quad \mathcal{D}_\mathcal{C} \models \alpha_1 \sqsubseteq \beta_1, \ldots, \alpha_n \sqsubseteq \beta_n, \mathtt{th}\,[s_2, t_2] \sqsubseteq \beta \,\wedge \\
\quad \mathcal{D}_\mathcal{C} \models C_1, \ldots, C_k \;\wedge\; \mathcal{D}_\mathcal{C} \models s_1 \le t_1, t_1 < s_2, s_2 < t_2.
\end{array}
$$

Since $\mathcal{D}_\mathcal{C} \models \mathtt{th}\,[s_2, t_2] \sqsubseteq \beta$ then $\beta = \mathtt{th}\,[w_1, w_2]$ with $\mathcal{D}_\mathcal{C} \models w_1 \le s_2, t_2 \le w_2$. Hence we distinguish two cases according to the relation between w_1 and s_1.

- $\mathcal{D}_\mathcal{C} \models w_1 \le s_1$.
  In this case we immediately conclude because $\mathcal{D}_\mathcal{C} \models \mathtt{th}\,[t_1, t_2] \sqsubseteq \mathtt{th}\,[w_1, w_2]$, and thus $(A, \mathtt{th}\,[w_1, w_2]) \in \mathbb{T}_\mathcal{E}^\omega \;\wedge\; \mathcal{D}_\mathcal{C} \models \mathtt{th}\,[t_1, t_2] \sqsubseteq \mathtt{th}\,[w_1, w_2]$.

- $\mathcal{D}_\mathcal{C} \models s_1 < w_1$.
  In this case $clause(\mathcal{E}, A\mathtt{th}[s_1, s_2], (C_1, \dots, C_k, B_1\alpha_1, \dots, B_n\alpha_n)) \in T_V^\omega$, since $T_V^\omega = \bigcup_{i \in \mathbb{N}} T_V^i$. Moreover, from $\mathcal{D}_\mathcal{C} \models s_1 < w_1, w_1 \leq s_2, s_2 < t_2, t_2 \leq w_2$, by Lemma 16 we obtain $(A, \mathtt{th}[s_1, w_2]) \in \mathbb{T}_\mathcal{E}(\mathbb{T}_\mathcal{E}^\omega)$. Since $\mathbb{T}_\mathcal{E}^\omega$ is a fixpoint of $\mathbb{T}_\mathcal{E}$ and $\mathcal{D}_\mathcal{C} \models s_1 \leq t_1, t_2 \leq w_2$ we can conclude $(A, \mathtt{th}[s_1, w_2]) \in \mathbb{T}_\mathcal{E}^\omega$ and $\mathcal{D}_\mathcal{C} \models \mathtt{th}[t_1, t_2] \sqsubseteq \mathtt{th}[s_1, w_2]$.

We are finally able to prove the soundness of the meta-interpreter with respect to the least fixpoint semantics.

$$
\begin{array}{l}
\quad demo(\mathcal{E}, A\,\alpha) \in T_V^\omega \\
\Longrightarrow \{T_V^\omega = \bigcup_{i \in \mathbb{N}} T_V^i\} \\
\quad \exists h : \; demo(\mathcal{E}, A\,\alpha) \in T_V^h \\
\Longrightarrow \{\text{Statement (12)}\} \\
\quad \exists \beta : (A, \beta) \in \mathbb{T}_\mathcal{E}^\omega \;\wedge\; \mathcal{D}_\mathcal{C} \models \alpha \sqsubseteq \beta \\
\Longrightarrow \{\text{definition of } \mathbb{F}^\mathcal{C}\} \\
(A, \alpha) \in \mathbb{F}^\mathcal{C}(\mathcal{E}).
\end{array}
$$

Completeness.

We first need a lemma stating that if an annotated atom $A\,\alpha$ is provable at the meta-level in a program expression $\mathcal{E}$ then we can prove at the meta-level the same atom A with any other "weaker" annotation (namely $A\,\gamma$, with $\gamma \sqsubseteq \alpha$).

Lemma 18 *Let $\mathcal{E}$ be a program expression and V be the corresponding meta-program. For any $A \in \mathcal{C}\text{-}base_L$ and $\alpha \in Ann$, the following statement holds:*

$$demo(\mathcal{E}, A\,\alpha) \in T_V^\omega \quad \Longrightarrow \quad \{demo(\mathcal{E}, A\,\gamma) \mid \gamma \in Ann, \mathcal{D}_\mathcal{C} \models \gamma \sqsubseteq \alpha\} \subseteq T_V^\omega.$$

Proof . We first show that for all h

$$demo(\mathcal{E}, A\alpha) \in T_V^h \Longrightarrow \{demo(\mathcal{E}, A\gamma) \mid \gamma \in Ann, \mathcal{D}_\mathcal{C} \models \gamma \sqsubseteq \alpha\} \subseteq T_V^h \tag{13}$$

The proof is by induction on h.
(Base case). Trivial since $T_V^0 = \emptyset$.

(Inductive case). Assume that

$$demo(\mathcal{E}, A\,\alpha) \in T_V^h \quad \Longrightarrow \quad \{demo(\mathcal{E}, A\,\gamma) \mid \gamma \in Ann, \mathcal{D}_\mathcal{C} \models \gamma \sqsubseteq \alpha\} \subseteq T_V^h.$$

Then:

$$
\begin{array}{l}
\quad demo(\mathcal{E}, A\,\alpha) \in T_V^{h+1} \\
\Longleftrightarrow \{\text{definition of } T_V^i\} \\
\quad demo(\mathcal{E}, A\,\alpha) \in T_V(T_V^h)
\end{array}
$$

We have four cases corresponding to clauses (3), (4), (5) and (6) of the meta-interpreter.

(clause (3)) $\{\alpha = \mathtt{th}\,[t_1, t_2]$, definition of T_V and clause (3)$\}$
$\{clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})), demo(\mathcal{E}, (\bar{C}, \bar{B}))\} \subseteq T_V^h \wedge$
$\mathcal{D}_\mathcal{C} \models s_1 \leq t_1, t_2 \leq s_2, t_1 \leq t_2$

The annotations β less than $\mathtt{th}\,[t_1, t_2]$, namely such that $\mathcal{D}_\mathcal{C} \models \beta \sqsubseteq \mathtt{th}\,[t_1, t_2]$, are of two kinds:

- $\beta = \mathtt{th}\,[w_1, w_2]$ with $\mathcal{D}_\mathcal{C} \models t_1 \leq w_1, w_1 \leq w_2, w_2 \leq t_2$. In this case we have $\mathcal{D}_\mathcal{C} \models s_1 \leq w_1, w_1 \leq w_2, w_2 \leq s_2$ and by using definition T_V, clause (3) and $\{clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})), demo(\mathcal{E}, (\bar{C}, \bar{B}))\} \subseteq T_V^h$ we can conclude $demo(\mathcal{E}, A\,\mathtt{th}\,[w_1, w_2]) \in T_V(T_V^h) = T_V^{h+1}$.
- $\beta = \mathtt{in}\,[w_1, w_2]$ with $\mathcal{D}_\mathcal{C} \models w_1 \leq t_2, t_1 \leq w_2, w_1 \leq w_2$. We reason as in the previous case, but using clause (5) and the fact that $\mathcal{D}_\mathcal{C} \models w_1 \leq t_2, t_2 \leq s_2, s_1 \leq w_2, w_1 \leq w_2$. Hence we can conclude $demo(\mathcal{E}, A\,\mathtt{in}\,[w_1, w_2]) \in T_V^{h+1}$.

(clause (4)) $\{\alpha = \mathtt{th}\,[t_1, t_2]$, definition of T_V and clause (4)$\}$
$\{clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})), demo(\mathcal{E}, (\bar{C}, \bar{B})), demo(\mathcal{E}, A\,\mathtt{th}\,[s_2, t_2])\} \subseteq T_V^h$
$\wedge\ \mathcal{D}_\mathcal{C} \models s_1 \leq t_1, t_1 < s_2, s_2 < t_2$

The annotations β less than $\mathtt{th}\,[t_1, t_2]$ are of two kinds:

- $\beta = \mathtt{th}\,[w_1, w_2]$ with $\mathcal{D}_\mathcal{C} \models t_1 \leq w_1, w_1 \leq w_2, w_2 \leq t_2$.
  We distinguish three cases according to the relation of s_2 with w_1 and w_2:
  - $\mathcal{D}_\mathcal{C} \models w_2 \leq s_2$.
    In this case $\mathcal{D}_\mathcal{C} \models s_1 \leq w_1, w_1 \leq w_2, w_2 \leq s_2$. By using the definition of T_V, clause (3) and the fact that $clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})) \in T_V^h$ and $demo(\mathcal{E}, (\bar{C}, \bar{B})) \in T_V^h$ we conclude $demo(\mathcal{E}, A\,\mathtt{th}\,[w_1, w_2]) \in T_V^{h+1}$.
  - $\mathcal{D}_\mathcal{C} \models s_2 \leq w_1, s_2 < w_2$.
    In this case $\mathcal{D}_\mathcal{C} \models \mathtt{th}\,[w_1, w_2] \sqsubseteq \mathtt{th}\,[s_2, t_2]$. Furthermore, recalling that $demo(\mathcal{E}, A\,\mathtt{th}\,[s_2, t_2]) \in T_V^h$, by inductive hypothesis we can deduce that $demo(\mathcal{E}, A\,\mathtt{th}\,[w_1, w_2]) \in T_V^h$. By monotonicity of T_V we conclude.
  - $\mathcal{D}_\mathcal{C} \models w_1 < s_2, s_2 < w_2$.
    In this case $\mathcal{D}_\mathcal{C} \models s_1 \leq w_1, w_1 < s_2, s_2 < w_2, w_2 \leq t_2$. Moreover, since $demo(\mathcal{E}, A\,\mathtt{th}\,[s_2, t_2]) \in T_V^h$, by inductive hypothesis we deduce that $demo(\mathcal{E}, A\,\mathtt{th}\,[s_2, w_2]) \in T_V^h$ and by using the definition of T_V, clause (4) and $\{clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})), demo(\mathcal{E}, (\bar{C}, \bar{B}))\} \subseteq T_V^h$ we conclude that $demo(\mathcal{E}, A\,\mathtt{th}\,[w_1, w_2]) \in T_V^{h+1}$.
- $\beta = \mathtt{in}\,[w_1, w_2]$ with $\mathcal{D}_\mathcal{C} \models w_1 \leq t_2, t_1 \leq w_2, w_1 \leq w_2$.
  We distinguish two subcases:
  - $[w_1, w_2] \cap [s_1, s_2] \neq \emptyset$.
    In this case $\mathcal{D}_\mathcal{C} \models w_1 \leq s_2, s_1 \leq w_2, w_1 \leq w_2$. Observing that we have $\{clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})), demo(\mathcal{E}, (\bar{C}, \bar{B}))\} \subseteq T_V^h$, by using the definition of T_V and clause (5) we conclude $demo(\mathcal{E}, A\,\mathtt{in}\,[w_1, w_2]) \in T_V^{h+1}$.

– $[w_1, w_2] \cap [s_2, t_2] \neq \emptyset$.
In this case $\mathcal{D}_\mathcal{C} \models \mathtt{in}\,[w_1, w_2] \sqsubseteq \mathtt{th}\,[s_2, t_2]$. Since $demo(\mathcal{E}, A\,\mathtt{th}\,[s_2, t_2]) \in T_V^h$, by inductive hypothesis we have that $demo(\mathcal{E}, A\,\mathtt{in}\,[w_1, w_2]) \in T_V^h$. By monotonicity of T_V we conclude.

(clause (5)) $\{\alpha = \mathtt{in}\,[t_1, t_2]$, definition of T_V and clause (5)$\}$
$\{clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})), demo(\mathcal{E}, (\bar{C}, \bar{B}))\} \subseteq T_V^h \wedge$
$\mathcal{D}_\mathcal{C} \models t_1 \leq s_2, s_1 \leq t_2, t_1 \leq t_2$

If $\mathcal{D}_\mathcal{C} \models \gamma \sqsubseteq \mathtt{in}\,[t_1, t_2]$ then $\gamma = \mathtt{in}\,[w_1, w_2]$ with $\mathcal{D}_\mathcal{C} \models w_1 \leq t_1, t_2 \leq w_2$. Hence we have $\mathcal{D}_\mathcal{C} \models w_1 \leq s_2, s_1 \leq w_2, w_1 \leq w_2$ and by using definition T_V, clause (5) and the fact that $\{clause(\mathcal{E}, A\,\mathtt{th}\,[s_1, s_2], (\bar{C}, \bar{B})), demo(\mathcal{E}, (\bar{C}, \bar{B}))\} \subseteq T_V^h$ we can conclude that $demo(\mathcal{E}, A\,\mathtt{in}\,[w_1, w_2]) \in T_V^{h+1}$.

(clause (6)) $\{\alpha = \mathtt{in}\,[t_1, t_2]\}$

Reasoning as in the previous case, by using the definition of T_V and clause (6), we can conclude. Now the completeness result for MuTACLP meta-interpreter basically relies on two technical lemmata (Lemma 23 and Lemma 25). Roughly speaking they assert that when th and in annotated atoms are derivable from an interpretation I by using the $\mathbb{T}_\mathcal{E}$ operator then we can find corresponding virtual clauses in the program expression $\mathcal{E}$ which permit to derive the same or greater information.

Let us first introduce some preliminary notions and results.

Definition 19 (covering) *A* covering *for a* th*-annotation* $\mathtt{th}\,[t_1, t_2]$ *is a sequence of annotations* $\{\mathtt{th}\,[t_1^i, t_2^i]\}_{i \in \{1,\ldots,n\}}$, *such that* $\mathcal{D}_\mathcal{C} \models \mathtt{th}\,[t_1, t_2] \sqsubseteq \mathtt{th}\,[t_1^1, t_n^2]$ *and for any* $i \in \{1, \ldots, n\}$

$$\mathcal{D}_\mathcal{C} \models t_1^i \leq t_2^i,\ t_1^{i+1} \leq t_2^i,\ t_1^i < t_1^{i+1}.$$

In words, a covering of a th annotation $\mathtt{th}\,[t_1, t_2]$ is a sequence of annotations $\{\mathtt{th}\,[t_1^i, t_2^i]\}_{i \in \{1,\ldots,n\}}$ such that each of the intervals overlaps with its successor, and the union of such intervals includes $[t_1, t_2]$. The next simple lemma observes that, given two annotations and a covering for each of them, we can always build a covering for their greatest lower bound.

Lemma 20 *Let* $\mathtt{th}\,[t_1, t_2]$ *and* $\mathtt{th}\,[s_1, s_2]$ *be annotations and* $\mathtt{th}\,[w_1, w_2] = \mathtt{th}\,[t_1, t_2] \sqcap \mathtt{th}\,[s_1, s_2]$. *Let* $\{\mathtt{th}\,[t_1^i, t_2^i]\}_{i \in \{1,\ldots,n\}}$ *and* $\{\mathtt{th}\,[s_1^j, s_2^j]\}_{j \in \{1,\ldots,m\}}$ *be coverings for* $\mathtt{th}\,[t_1, t_2]$ *and* $\mathtt{th}\,[s_1, s_2]$, *respectively. Then a covering for* $\mathtt{th}\,[w_1, w_2]$ *can be extracted from*

$$\{\mathtt{th}\,[t_1^i, t_2^i] \sqcap \mathtt{th}\,[s_1^j, s_2^j] \mid i \in \{1, \ldots n\}\ \wedge\ j \in \{1, \ldots, m\}\}.$$

In the hypothesis of the previous lemma $[w_1, w_2] = [t_1, t_2] \cap [s_1, s_2]$. Thus the result of the lemma is simply a consequence of the distributivity of set-theoretical intersection with respect to union.

DEFINITION 21 *Let $\mathcal{E}$ be a program expression, let V be the corresponding meta-program and let $I \subseteq \mathcal{C}\text{-}base_L \times Ann$ be an interpretation. Given an annotated atom $(A, \mathtt{th}[t_1, t_2]) \in \mathcal{C}\text{-}base_L \times Ann$, an $(\mathcal{E}, I)$-*set for $(A, \mathtt{th}[t_1, t_2])$ *is a set*

$$\{clause(\mathcal{E}, A\,\mathtt{th}[t_1^i, t_2^i], (\bar{C}^i, \bar{B}^i))\}_{i \in \{1,\dots,n\}} \subseteq T_V^\omega$$

such that

1. *$\{\mathtt{th}[t_1^i, t_2^i]\}_{i \in \{1,\dots,n\}}$ is a covering of $\mathtt{th}[t_1, t_2]$, and*
2. *for $i \in \{1, \dots, n\}$, $I \models \bar{C}^i, \bar{B}^i$.*

An interpretation $I \subseteq \mathcal{C}\text{-}base_L \times Ann$ is called $\mathtt{th}$-closed with respect to *$\mathcal{E}$ (or $\mathcal{E}$-closed, for short) if there is an $(\mathcal{E}, I)$-set for every annotated atom $(A, \mathtt{th}[t_1, t_2]) \in I$.*

The next lemma presents some properties of the notion of $\mathcal{E}$-closedness, which essentially state that the property of being $\mathcal{E}$-closed is invariant with respect to some obvious algebraic transformations of the program expression $\mathcal{E}$. The proofs are trivial and thus are omitted.

LEMMA 22 *Let $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{N}$ be program expressions and let I be an interpretation. Then the following properties hold, where $op \in \{\cup, \cap\}$*

1. *I is $(\mathcal{E}\, op\, \mathcal{E})$-closed iff I is $\mathcal{E}$-closed;*
2. *I is $(\mathcal{E}\, op\, \mathcal{R})$-closed iff I is $(\mathcal{R}\, op\, \mathcal{E})$-closed;*
3. *I is $((\mathcal{E}\, op\, \mathcal{R})\, op\, \mathcal{N})$-closed iff I is $\mathcal{E}\, op\, (\mathcal{R}\, op\, \mathcal{N})$-closed;*
4. *if I is $\mathcal{E}$-closed then I is $(\mathcal{E} \cup \mathcal{R})$-closed;*
5. *if I is $(\mathcal{E} \cap \mathcal{R})$-closed then I is $\mathcal{E}$-closed;*
6. *I is $((\mathcal{E} \cap \mathcal{R}) \cup \mathcal{N})$-closed iff I is $((\mathcal{E} \cup \mathcal{N}) \cap (\mathcal{R} \cup \mathcal{N}))$-closed.*

We next prove that if we apply the $\mathbb{T}_\mathcal{E}$ operator to an $\mathcal{E}$-closed interpretation, then for any derived $\mathtt{th}$-annotated atom there exists an $(\mathcal{E}, I)$-set (see Definition 21). This result represents a basic step towards the completeness proof. In fact, it tells us that starting from the empty interpretation, which is obviously $\mathcal{E}$-closed, and iterating the $\mathbb{T}_\mathcal{E}$ then we get, step after step, $\mathtt{th}$-annotated atoms which can be also derived from the virtual clauses of the program expression at hand. For technical reasons, to make the induction work, we need a slightly stronger property.

LEMMA 23 *Let $\mathcal{E}$ and $\mathcal{Q}$ be program expressions, let V be the corresponding meta-program[4] and let $I \subseteq \mathcal{C}\text{-}base_L \times Ann$ be an $(\mathcal{E} \cup \mathcal{Q})$-closed interpretation. Then for any atom $(A, \mathtt{th}[t_1, t_2]) \in \mathbb{T}_\mathcal{E}(I)$ there exists an $(\mathcal{E} \cup \mathcal{Q}, I)$-set.*

[4]The meta-program contains the meta-level representation of the plain programs in $\mathcal{E}$ and $\mathcal{Q}$.

Proof . The proof is done by structural induction on the program expression $\mathcal{E}$.

($\mathcal{E}$ is a plain program P).
Let $(A, \mathtt{th}\,[t_1, t_2]) \in \mathbb{T}_P(I)$. By definition of $\mathbb{T}_P$ we have two possibilities

1. $A\,\mathtt{th}\,[t_1, t_2] \leftarrow \bar{C}, \bar{B} \in ground_{\mathcal{C}}(P) \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; \mathcal{D}_{\mathcal{C}} \models t_1 \leq t_2$
   Hence, by definition of the meta-level representation of plain programs, according to clause (11), we have that
   $$clause(P, A\,\mathtt{th}\,[t_1, t_2], (\bar{C}, \bar{B})) \in T_V^\omega$$
   Hence, by clause (8), $clause(P \cup \mathcal{Q}, A\,\mathtt{th}\,[t_1, t_2], (\bar{C}, \bar{B})) \in T_V^\omega$ and thus this single virtual clause is a $(P \cup \mathcal{Q}, I)$-set for $(A, \mathtt{th}\,[t_1, t_2])$.

2. $A\,\mathtt{th}\,[t_1, t_2'] \leftarrow \bar{C}, \bar{B} \in ground_{\mathcal{C}}(P) \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; (A, \mathtt{th}\,[t_1', t_2]) \in I \;\wedge$
   $\mathcal{D}_{\mathcal{C}} \models t_1 < t_1', t_1' \leq t_2', t_2' < t_2$
   Again, by definition of the meta-level representation according to clause (11), definition of T_V and, observing that $\mathcal{D}_{\mathcal{C}} \models t_1 \leq t_2'$ we obtain that
   $$clause(P, A\,\mathtt{th}\,[t_1, t_2'], (\bar{C}, \bar{B})) \in T_V^\omega$$
   and therefore, by definition of the union operator (clause (8) in the meta-interpreter)
   $$clause(P \cup \mathcal{Q}, A\,\mathtt{th}\,[t_1, t_2'], (\bar{C}, \bar{B})) \in T_V^\omega \tag{14}$$
   Furthermore, since by hypothesis I is $(P \cup \mathcal{Q})$-closed, there is a $(P \cup \mathcal{Q}, I)$-set for $(A, \mathtt{th}\,[t_1', t_2])$
   $$\{clause(P \cup \mathcal{Q}, A\,\mathtt{th}\,[t_1^i, t_2^i], (\bar{C}^i, \bar{B}^i))\}_{i \in \{1,\ldots,n\}} \subseteq T_V^\omega \tag{15}$$
   Now, if $\mathcal{D}_{\mathcal{C}} \models t_1^1 \leq t_1$ we can conclude because (15) is a $(P \cup \mathcal{Q}, I)$-set for $A\,\mathtt{th}\,[t_1, t_2]$. If instead $\mathcal{D}_{\mathcal{C}} \models t_1 < t_1^1, t_1^1 \leq t_1', t_1' \leq t_2'$ then it is immediate to conclude that adding the clause in (14) to the $(P \cup \mathcal{Q})$-set for $(A, \mathtt{th}\,[t_1', t_2])$ in (15), we can obtain a $(P \cup \mathcal{Q}, I)$-set for $(A, \mathtt{th}\,[t_1, t_2])$.

($\mathcal{E} = \mathcal{R} \cup \mathcal{N}$)
Let $(A, \mathtt{th}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{E}}(I) = \mathbb{T}_{\mathcal{R}}(I) \cup \mathbb{T}_{\mathcal{N}}(I)$. There are two possible cases:

1. $(A, \mathtt{th}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{R}}(I)$.
   Since by hypothesis I is $((\mathcal{R} \cup \mathcal{N}) \cup \mathcal{Q})$-closed, by point (3) of Lemma 22, I is $(\mathcal{R} \cup (\mathcal{N} \cup \mathcal{Q}))$-closed, and therefore, by inductive hypothesis, there exists a $(\mathcal{R} \cup (\mathcal{N} \cup \mathcal{Q}), I)$-set for $(A, \mathtt{th}\,[t_1, t_2])$, which is trivially a $((\mathcal{R} \cup \mathcal{N}) \cup \mathcal{Q}), I)$-set.

2. $(A, \mathtt{th}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{N}}(I)$.
   By points (2) and (3) of Lemma 22, the interpretation I is $(\mathcal{N} \cup (\mathcal{R} \cup \mathcal{Q}))$-closed. Thus, by inductive hypothesis, there is a $(\mathcal{N} \cup (\mathcal{R} \cup \mathcal{Q}), I)$-set for $(A, \mathtt{th}\,[t_1, t_2])$, which is obviously a $((\mathcal{R} \cup \mathcal{N}) \cup \mathcal{Q}, I)$-set.

$(\mathcal{E} = \mathcal{R} \cap \mathcal{N})$
Consider any annotated atom $(A, \mathtt{th}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{E}}(I) = \mathbb{T}_{\mathcal{R}}(I) \Cap \mathbb{T}_{\mathcal{N}}(I)$. By definition of the operator $\Cap$ there are $(A, \mathtt{th}\,[s_1, s_2]) \in \mathbb{T}_{\mathcal{R}}(I)$ and $(A, \mathtt{th}\,[w_1, w_2]) \in \mathbb{T}_{\mathcal{N}}(I)$ such that

$$\mathtt{th}\,[t_1, t_2] = \mathtt{th}\,[s_1, s_2] \sqcap \mathtt{th}\,[w_1, w_2]$$

By hypothesis, the interpretation I is $((\mathcal{R} \cap \mathcal{N}) \cup \mathcal{Q})$-closed and thus, by points (6) and (5) of Lemma 22, I is also $(\mathcal{R} \cup \mathcal{Q})$-closed. Therefore, by inductive hypothesis, we can find a $(\mathcal{R} \cup \mathcal{Q}, I)$-set for $(A, \mathtt{th}\,[s_1, s_2])$:

$$\{\mathit{clause}(\mathcal{R} \cup \mathcal{Q}, A\,\mathtt{th}\,[s_1^i, s_2^i], (\bar{C}_R^i, \bar{B}_R^i))\}_{i \in \{1,\ldots,n\}}$$

and, similarly, since I is also $(\mathcal{N} \cup \mathcal{Q})$-closed, we can find an $(\mathcal{N} \cup \mathcal{Q}, I)$-set for $(A, \mathtt{th}\,[w_1, w_2])$:

$$\{\mathit{clause}(\mathcal{N} \cup \mathcal{Q}, A\,\mathtt{th}\,[w_1^j, w_2^j], (\bar{C}_N^j, \bar{B}_N^j))\}_{j \in \{1,\ldots,m\}} T_V^{\omega}$$

By Lemma 20 there exists a covering $\{\mathtt{th}\,[t_1^k, t_2^k]\}_{k \in \{1,\ldots,l\}}$ for $\mathtt{th}\,[t_1, t_2]$, such that for any k there are i_k and j_k such that

$$\mathtt{th}\,[t_1^k, t_2^k] = \mathtt{th}\,[s_1^{i_k}, s_2^{i_k}] \sqcap \mathtt{th}\,[w_1^{j_k}, w_2^{j_k}]$$

We can finally conclude that the above covering gives rise to a $((\mathcal{R} \cap \mathcal{N}) \cup \mathcal{Q}, I)$-set for $(A, \mathtt{th}\,[t_1, t_2])$. In fact for each of the annotations $\mathtt{th}\,[t_1^k, t_2^k]$ for $k \in \{1, \ldots, l\}$ there are three possibilities:

1. $\mathit{clause}(\mathcal{R}, A\,\mathtt{th}\,[s_1^{i_k}, s_2^{i_k}], (\bar{C}_R^{i_k}, \bar{B}_R^{i_k})) \in T_V^{\omega} \wedge$
   $\mathit{clause}(\mathcal{N}, A\,\mathtt{th}\,[w_1^{j_k}, w_2^{j_k}], (\bar{C}_N^{j_k}, \bar{B}_N^{j_k})) \in T_V^{\omega}$

   In this case, by definition of intersection operator (clause (10) in the meta-interpreter), we have that

   $$\mathit{clause}(\mathcal{R} \cap \mathcal{N}, A\,\mathtt{th}\,[t_1^k, t_2^k], (\bar{C}_R^{i_k}, \bar{C}_N^{j_k}, \bar{B}_R^{i_k}, \bar{B}_N^{j_k})) \in T_V^{\omega}$$

   and therefore $\mathit{clause}((\mathcal{R} \cap \mathcal{N}) \cup \mathcal{Q}, A\,\mathtt{th}\,[t_1^k, t_2^k], (\bar{C}_R^{i_k}, \bar{C}_N^{j_k}, \bar{B}_R^{i_k}, \bar{B}_N^{j_k})) \in T_V^{\omega}$, by definition of union operator (clause (8)). Moreover, by construction, the body of the clause is satisfied by I, namely $I \models \bar{C}_R^{i_k}, \bar{C}_N^{j_k}, \bar{B}_R^{i_k}, \bar{B}_N^{j_k}$.

2. $\mathit{clause}(\mathcal{Q}, A\,\mathtt{th}\,[s_1^{i_k}, s_2^{i_k}], (\bar{C}_R^{i_k}, \bar{B}_R^{i_k})) \in T_V^{\omega}$

   In this case by definition of union operator (clause (9)), we have that

   $$\mathit{clause}((\mathcal{R} \cap \mathcal{N}) \cup \mathcal{Q}, A\,\mathtt{th}\,[s_1^{i_k}, s_2^{i_k}], (\bar{C}_R^{i_k}, \bar{B}_R^{i_k})) \in T_V^{\omega}$$

   and, by construction, $I \models \bar{C}_R^{i_k}, \bar{B}_R^{i_k}$.

3. $\mathit{clause}(\mathcal{Q}, A\,\mathtt{th}\,[w_1^{j_k}, w_2^{j_k}], (\bar{C}_N^{j_k}, \bar{B}_N^{j_k})) \in T_V^{\omega}$

   As in the previous point we have that

   $$\mathit{clause}((\mathcal{R} \cap \mathcal{N}) \cup \mathcal{Q}, A\,\mathtt{th}\,[w_1^{j_k}, w_2^{j_k}], (\bar{C}_N^{j_k}, \bar{B}_N^{j_k})) \in T_V^{\omega}$$

   and, by construction, $I \models \bar{C}_N^{j_k}, \bar{B}_N^{j_k}$.

Corollary 24 *Let $\mathcal{E}$ be any program expression and let V be the corresponding meta-program. Then for any $h \in \mathbb{N}$ the interpretation $\mathbb{T}^h_{\mathcal{E}}$ is $\mathcal{E}$-closed. Therefore $\mathbb{T}^\omega_{\mathcal{E}}$ is $\mathcal{E}$-closed.*

Proof . The fact that $\mathbb{T}^h_{\mathcal{E}}$ is $\mathcal{E}$-closed for each $h \in \mathbb{N}$ is proved by induction on h.

(Base case). Obvious, since $\mathbb{T}^0_{\mathcal{E}} = \emptyset$ is trivially $\mathcal{E}$-closed.

(Inductive case). Assume that $\mathbb{T}^h_{\mathcal{E}}$ is $\mathcal{E}$-closed. By Lemma 23 and point (1) of Lemma 22 for any atom $(A, \mathtt{th}\,[t_1, t_2]) \in \mathbb{T}^{h+1}_{\mathcal{E}} = \mathbb{T}_{\mathcal{E}}(\mathbb{T}^h_{\mathcal{E}})$ there exists a $(\mathcal{E}, \mathbb{T}^h_{\mathcal{E}})$-set. But since $\mathbb{T}^h_{\mathcal{E}} \subseteq \mathbb{T}^{h+1}_{\mathcal{E}}$ we conclude that there is also a $(\mathcal{E}, \mathbb{T}^{h+1}_{\mathcal{E}})$-set for $(A, \mathtt{th}\,[t_1, t_2])$. Therefore $\mathbb{T}^{h+1}_{\mathcal{E}}$ is $\mathcal{E}$-closed.

Since the union of $\mathcal{E}$-closed interpretations is clearly $\mathcal{E}$-closed, we immediately deduce also the final statement of the corollary.

Another technical lemma is needed for dealing with the $\mathtt{in}$ annotations, which comes in pair with Lemma 23.

Lemma 25 *Let $\mathcal{E}$ be a program expression, let V be the corresponding meta-program and let I be any $\mathcal{E}$-closed interpretation. For any atom $(A, \mathtt{in}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{E}}(I)$ we have*

$$\mathit{clause}(\mathcal{E}, A\,\alpha, (\bar{C}, \bar{B})) \in T^\omega_V \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; \mathcal{D}_{\mathcal{C}} \models \mathtt{in}\,[t_1, t_2] \sqsubseteq \alpha.$$

Proof . The proof is done by induction on the structure of $\mathcal{E}$.

($\mathcal{E}$ is a plain program P)
Let $(A, \mathtt{in}\,[t_1, t_2]) \in \mathbb{T}_P(I)$. By definition of $\mathbb{T}_P$

$$A\,\mathtt{in}\,[t_1, t_2] \leftarrow \bar{C}, \bar{B} \in \mathit{ground}_{\mathcal{C}}(P) \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; \mathcal{D}_{\mathcal{C}} \models t_1 \leq t_2$$

By the meta-level representation of plain programs, according to clause (11), we have

$$\mathit{clause}(P, A\,\mathtt{in}\,[t_1, t_2], (\bar{C}, \bar{B})) \in T^\omega_V$$

Recalling that $I \models \bar{C}, \bar{B}$, and observing that $\mathcal{D}_{\mathcal{C}} \models \mathtt{in}\,[t_1, t_2] \sqsubseteq \mathtt{in}\,[t_1, t_2]$ we conclude.

($\mathcal{E} = \mathcal{R} \cup \mathcal{N}$)
Let $(A, \mathtt{in}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{R}\cup\mathcal{N}}(I) = \mathbb{T}_{\mathcal{R}}(I) \cup \mathbb{T}_{\mathcal{N}}(I)$. Therefore $(A, \mathtt{in}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{R}}(I)$ or $(A, \mathtt{in}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{N}}(I)$. In the first case, by inductive hypothesis

$$\mathit{clause}(\mathcal{R}, A\,\alpha, (\bar{C}, \bar{B})) \in T^\omega_V \;\wedge\; I \models \bar{C}, \bar{B} \;\wedge\; \mathcal{D}_{\mathcal{C}} \models \mathtt{in}\,[t_1, t_2] \sqsubseteq \alpha$$

Since, by clause (8) in the meta-interpreter, $\mathit{clause}(\mathcal{R} \cup \mathcal{N}, A\,\alpha, (\bar{C}, \bar{B})) \in T^\omega_V$, we can conclude. If instead $(A, \mathtt{in}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{N}}(I)$, a symmetric reasoning allows us to reach the desired conclusion.

$(\mathcal{E} = \mathcal{R} \cap \mathcal{N})$
Let $(A, \mathtt{in}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{R}\cap\mathcal{N}}(I) = \mathbb{T}_{\mathcal{R}}(I) \Cap \mathbb{T}_{\mathcal{N}}(I)$. By definition of the operator $\Cap$ there are $(A, \beta) \in \mathbb{T}_{\mathcal{R}}(I)$ and $(A, \gamma) \in \mathbb{T}_{\mathcal{N}}(I)$ such that

$$\mathtt{in}\,[t_1, t_2] = \beta \sqcap \gamma.$$

Several cases may arise corresponding to the axioms $(\mathtt{th}\,\sqcap')$, $(\mathtt{th}\,\mathtt{in}\,\sqcap)$, $(\mathtt{th}\,\mathtt{in}\,\sqcap')$, $(\mathtt{th}\,\mathtt{in}\,\sqcap'')$, $(\mathtt{in}\,\sqcap)$ defining the greatest lower bound $\sqcap$. We next analyse only the first and the last cases, since the others can be proved analogously.

- $(\mathtt{th}\,\sqcap')$
  In this case $\beta = \mathtt{th}\,[s_1, s_2]$, $\gamma = \mathtt{th}\,[r_1, r_2]$, $t_1 = min\{s_2, r_2\}$, $t_2 = max\{s_1, r_1\}$, $\mathcal{D}_{\mathcal{C}} \models t_1 < t_2$.

  Since by hypothesis I is $(\mathcal{R} \cap \mathcal{N})$-closed, by using point (5) of Lemma 22, I is $\mathcal{R}$-closed and thus $(\mathcal{R} \cup \mathcal{R})$-closed, by point (1) of the same lemma. Hence, recalling that $(A, \mathtt{th}\,[s_1, s_2]) \in \mathbb{T}_{\mathcal{R}}(I)$, we can apply Lemma 23 to deduce that there exists an $(\mathcal{R} \cup \mathcal{R}, I)$-set, or equivalently an $(\mathcal{R}, I)$-set for the atom $(A, \mathtt{th}\,[s_1, s_2])$. In other words we can find

  $$\{clause(\mathcal{R}, A\,\mathtt{th}\,[s_1^i, s_2^i], (\bar{C}_R^i, \bar{B}_R^i))\}_{i\in\{1,\ldots,n\}} \subseteq T_V^{\omega}$$

  such that $I \models \bar{C}_R^i, \bar{B}_R^i$ and $\{\mathtt{th}\,[s_1^i, s_2^i]\}_{i\in\{1,\ldots,n\}}$ is a covering of $\mathtt{th}\,[s_1, s_2]$.

  Reasoning in a similar way on $(A\,\mathtt{th}\,[r_1, r_2]) \in \mathbb{T}_{\mathcal{N}}(I)$, we find

  $$\{clause(\mathcal{N}, A\,\mathtt{th}\,[r_1^j, r_2^j], (\bar{C}_N^j, \bar{B}_N^j))\}_{j\in\{1,\ldots,m\}} \subseteq T_V^{\omega}$$

  such that $I \models \bar{C}_N^j, \bar{B}_N^j$ and $\{\mathtt{th}\,[r_1^j, r_2^j]\}_{j\in\{1,\ldots,m\}}$ is a covering of $\mathtt{th}\,[r_1, r_2]$.

  Let $\delta = \mathtt{th}\,[s_1^n, s_2^n] \sqcap \mathtt{th}\,[r_1^1, r_2^1]$ if $\mathcal{D}_{\mathcal{C}} \models s_1 < r_1$ otherwise $\delta = \mathtt{th}\,[s_1^1, s_2^1] \sqcap \mathtt{th}\,[r_1^m, r_2^m]$, then it is easy to check that $\mathcal{D}_{\mathcal{C}} \models \mathtt{in}\,[t_1, t_2] \sqsubseteq \delta$. Furthermore, by definition of the intersection operator (clause (10) in the meta-interpreter) we have

  $$clause(\mathcal{R} \cap \mathcal{N}, A\,\delta, (\bar{C}_R^n, \bar{C}_N^m, \bar{B}_R^n, \bar{B}_N^m)) \in T_V^{\omega}$$

  and by construction $I \models \bar{C}_R^n, \bar{C}_N^m, \bar{B}_R^n, \bar{B}_N^m$.

- $(\mathtt{in}\,\sqcap)$
  In this case $\beta = \mathtt{in}\,[s_1, s_2]$, $\gamma = \mathtt{in}\,[r_1, r_2]$, $t_1 = min\{s_1, r_1\}$, $t_2 = max\{s_2, r_2\}$, $\mathcal{D}_{\mathcal{C}} \models s_1 \le s_2, r_1 \le r_2$.

  By inductive hypothesis we have that

  $$clause(\mathcal{R}, A\,\alpha', (\bar{C}', \bar{B}')) \in T_V^{\omega} \;\wedge\; I \models \bar{C}', \bar{B}' \;\wedge\; \mathcal{D}_{\mathcal{C}} \models \mathtt{in}\,[s_1, s_2] \sqsubseteq \alpha'$$

  and

  $$clause(\mathcal{N}, A\,\alpha'', (\bar{C}'', \bar{B}'')) \in T_V^{\omega} \;\wedge\; I \models \bar{C}'', \bar{B}'' \;\wedge\; \mathcal{D}_{\mathcal{C}} \models \mathtt{in}\,[r_1, r_2] \sqsubseteq \alpha''$$

  Let $\delta = \alpha' \sqcap \alpha''$. Since $\mathcal{D}_{\mathcal{C}} \models \mathtt{in}\,[s_1, s_2] \sqsubseteq \alpha'$, $\mathtt{in}\,[r_1, r_2] \sqsubseteq \alpha''$, by the properties of the greatest lower bound $\mathcal{D}_{\mathcal{C}} \models \mathtt{in}\,[t_1, t_2] \sqsubseteq \delta$. Then, by definition of the intersection operator (clause (10) in the meta-interpreter), we can conclude that

$$clause(\mathcal{R} \cap \mathcal{N}, A\,\delta, (\bar{C}', \bar{C}'', \bar{B}', \bar{B}'')) \in T_V^{\omega}$$

where $I \models \bar{C}', \bar{C}'', \bar{B}', \bar{B}''$ and $\mathcal{D_C} \models \mathtt{in}\,[t_1, t_2] \sqsubseteq \delta$, as desired.

Now we can prove the completeness of the meta-interpreter with respect to the least fixpoint semantics.

THEOREM 26 (COMPLETENESS) *Let $\mathcal{E}$ be a program expression and V be the corresponding meta-program. For any $A \in \mathcal{C}\text{-}base_L$ and $\alpha \in Ann$ the following statement holds:*

$$(A, \alpha) \in \mathbb{F}^{\mathcal{C}}(\mathcal{E}) \quad \Longrightarrow \quad demo(\mathcal{E}, A\,\alpha) \in T_V^{\omega}.$$

Proof . We first show that for all h

$$(A, \alpha) \in \mathbb{T}_{\mathcal{E}}^{h} \qquad \Longrightarrow \qquad demo(\mathcal{E}, A\,\alpha) \in T_V^{\omega}. \tag{16}$$

The proof is by induction on h.
(Base case). Trivial since $\mathbb{T}_{\mathcal{E}}^{0} = \emptyset$.
(Inductive case). Assume that

$$(A, \alpha) \in \mathbb{T}_{\mathcal{E}}^{h} \quad \Longrightarrow \quad demo(\mathcal{E}, A\,\alpha) \in T_V^{\omega}.$$

Observe that, under the above assumption,

$$\mathbb{T}_{\mathcal{E}}^{h} \models \bar{C}, \bar{B} \quad \Rightarrow \quad demo(\mathcal{E}, (\bar{C}, \bar{B})) \in T_V^{\omega}. \tag{17}$$

In fact let $\bar{C} = C_1, \ldots, C_k$ and $\bar{B} = B_1\alpha_1, \ldots, B_n\alpha_n$. Then the notation $\mathbb{T}_{\mathcal{E}}^{h} \models \bar{C}$ amounts to say that for each i, $\mathcal{D_C} \models C_i$ and thus $demo(\mathcal{E}, C_i) \in T_V^{\omega}$, by definition of T_V and clause (7). Furthermore $\mathbb{T}_{\mathcal{E}}^{h} \models \bar{B}$ means that for each i, $(B_i, \beta_i) \in \mathbb{T}_{\mathcal{E}}^{h}$ and $\mathcal{D_C} \models \alpha_i \sqsubseteq \beta_i$. Hence by inductive hypothesis $demo(\mathcal{E}, B_i\,\beta_i) \in T_V^{\omega}$ and thus, by Lemma 18, $demo(\mathcal{E}, B_i\,\alpha_i) \in T_V^{\omega}$. By several applications of clause (2) in the meta-interpreter we finally deduce $demo(\mathcal{E}, (\bar{B}, \bar{C})) \in T_V^{\omega}$.

It is convenient to treat separately the cases of `th` and `in` annotations. If we assume that $\alpha = \mathtt{th}\,[t_1, t_2]$, then

$(A, \mathtt{th}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{E}}^{h+1}$
$\Longleftrightarrow$ {definition of $\mathbb{T}_{\mathcal{E}}^{i}$}
$(A, \mathtt{th}\,[t_1, t_2]) \in \mathbb{T}_{\mathcal{E}}(\mathbb{T}_{\mathcal{E}}^{h})$
$\Longrightarrow$ {Lemma 23 and $\mathbb{T}_{\mathcal{E}}^{h}$ is $\mathcal{E}$-closed by Corollary 24}
$\{clause(\mathcal{E}, A\,\mathtt{th}\,[t_1^i, t_2^i], (\bar{C}^i, \bar{B}^i))\}_{i \in \{1,\ldots,n\}} \subseteq T_V^{\omega}\ \wedge$
$\mathbb{T}_{\mathcal{E}}^{h} \models \bar{C}^i, \bar{B}^i$ for $i \in \{1, \ldots, n\} \wedge \{\mathtt{th}\,[t_1^i, t_2^i]\}_{i \in \{1,\ldots,n\}}$ covering of $\mathtt{th}\,[t_1, t_2]$
$\Longrightarrow$ {previous remark (17)}
$\{clause(\mathcal{E}, A\,\mathtt{th}\,[t_1^i, t_2^i], (\bar{C}^i, \bar{B}^i))\}_{i \in \{1,\ldots,n\}} \subseteq T_V^{\omega}\ \wedge$
$demo(\mathcal{E}, (\bar{C}^i, \bar{B}^i)) \in T_V^{\omega}$ for $i \in \{1, \ldots, n\} \wedge$
$\{\mathtt{th}\,[t_1^i, t_2^i]\}_{i \in \{1,\ldots,n\}}$ covering of $\mathtt{th}\,[t_1, t_2]$

$\Longrightarrow$ {definition of T_V, clause (3) and T_V^ω is a fixpoint of T_V}
$demo(\mathcal{E}, A\,\mathtt{th}\,[t_1^n, t_2^n]) \in T_V^\omega \wedge$
$\{clause(\mathcal{E}, A\,\mathtt{th}\,[t_1^i, t_2^i], (\bar{C}^i, \bar{B}^i))\}_{i\in\{1,\ldots,n-1\}} \subseteq T_V^\omega \wedge$
$demo(\mathcal{E}, (\bar{C}^i, \bar{B}^i)) \in T_V^\omega$ for $i \in \{1, \ldots, n-1\} \wedge$
$\{\mathtt{th}\,[t_1^i, t_2^i]\}_{i\in\{1,\ldots,n\}}$ covering of $\mathtt{th}\,[t_1, t_2]$
$\Longrightarrow$ {definition of T_V, clause (4), Lemma 18 and T_V^ω is a fixpoint of T_V}
$demo(\mathcal{E}, A\,\mathtt{th}\,[t_1^{n-1}, t_2^n]) \wedge \{clause(\mathcal{E}, A\,\mathtt{th}\,[t_1^i, t_2^i], (\bar{C}^i, \bar{B}^i))\}_{i\in\{1,\ldots,n-2\}} \subseteq T_V^\omega$
$\wedge\ demo(\mathcal{E}, (\bar{C}^i, \bar{B}^i)) \in T_V^\omega$ for $i \in \{1, \ldots, n-2\} \wedge$
$\{\mathtt{th}\,[t_1^i, t_2^i]\}_{i\in\{1,\ldots,n\}}$ covering of $\mathtt{th}\,[t_1, t_2]$
$\Longrightarrow$ {by exploiting several times clause (4) as above}
$demo(\mathcal{E}, A\,\mathtt{th}\,[t_1^1, t_2^n]) \ \wedge\ \{\mathtt{th}\,[t_1^i, t_2^i]\}_{i\in\{1,\ldots,n\}}$ covering of $\mathtt{th}\,[t_1, t_2]$
$\Longrightarrow$ {by definition of covering $\mathcal{D}_\mathcal{C} \models \mathtt{th}\,[t_1, t_2] \sqsubseteq \mathtt{th}\,[t_1^1, t_2^n]$ and Lemma 18}
$demo(\mathcal{E}, A\,\mathtt{th}\,[t_1, t_2]) \in T_V^\omega$

Instead, if $\alpha = \mathtt{in}\,[t_1, t_2]$, then

$(A, \mathtt{in}\,[t_1, t_2]) \in \mathbb{T}_\mathcal{E}^{h+1}$
$\Longleftrightarrow$ {definition of $\mathbb{T}_\mathcal{E}^i$}
$(A, \mathtt{in}\,[t_1, t_2]) \in \mathbb{T}_\mathcal{E}(\mathbb{T}_\mathcal{E}^h)$
$\Longrightarrow$ {Lemma 25}
$clause(\mathcal{E}, A\,\beta, (\bar{C}, \bar{B})) \in T_V^\omega \ \wedge\ \mathbb{T}_\mathcal{E}^h \models \bar{C}, \bar{B} \ \wedge\ \mathcal{D}_\mathcal{C} \models \mathtt{in}\,[t_1, t_2] \sqsubseteq \beta$
$\Longrightarrow$ {previous remark (17)}
$clause(\mathcal{E}, A\,\beta, (\bar{C}, \bar{B})) \in T_V^\omega \ \wedge\ demo(\mathcal{E}, (\bar{C}, \bar{B})) \in T_V^\omega \ \wedge \mathcal{D}_\mathcal{C} \models \mathtt{in}\,[t_1, t_2] \sqsubseteq \beta$
$\Longrightarrow$ {clause (3) or (6), and T_V^ω is a fixpoint of T_V}
$demo(\mathcal{E}, A\,\beta) \in T_V^\omega \ \wedge\ \mathcal{D}_\mathcal{C} \models \mathtt{in}\,[t_1, t_2] \sqsubseteq \beta$
$\Longrightarrow$ {Lemma 18}
$demo(\mathcal{E}, A\,\mathtt{in}\,[t_1, t_2]) \in T_V^\omega$

We now prove the completeness of the meta-interpreter of the program expressions with respect to the least fixpoint semantics.

$(A, \alpha) \in \mathbb{F}^\mathcal{C}(\mathcal{E})$
$\Longrightarrow$ {definition of $\mathbb{F}^\mathcal{C}(\mathcal{E})$}
$\exists \gamma \in Ann:\ (A, \gamma) \in \mathbb{T}_\mathcal{E}^\omega \ \wedge\ \mathcal{D}_\mathcal{C} \models \alpha \sqsubseteq \gamma$
$\Longrightarrow$ $\{\mathbb{T}_\mathcal{E}^\omega = \bigcup_{i\in\mathbb{N}} \mathbb{T}_\mathcal{E}^i\}$
$\exists h:\ (A, \gamma) \in \mathbb{T}_\mathcal{E}^h \ \wedge\ \mathcal{D}_\mathcal{C} \models \alpha \sqsubseteq \gamma$
$\Longrightarrow$ {statement (16)}
$demo(\mathcal{E}, A\,\gamma) \in T_V^\omega \ \wedge\ \mathcal{D}_\mathcal{C} \models \alpha \sqsubseteq \gamma$
$\Longrightarrow$ {Lemma 18}
$demo(\mathcal{E}, A\,\alpha) \in T_V^\omega$