A New Unification Method for Temporal Reasoning with Constraints

Eddie Schwalb

Information and Computer Science
University of California, Irvine
eschwalb@ics.uci.edu

Abstract

In this work we consider using logic programs to perform temporal reasoning. We identify some difficulties of combining constraint propagation and generalized resolution when temporal information is represented using tokens. We show that standard top-down evaluation (i.e. resolution) is incomplete due to the inability to unify constraints and ground terms. We present some syntactic restrictions that enable temporal resolution. Under these restrictions, we propose a new unification method composed of constraint unification and token fusion algorithms. Incorporating them within a generalized resolution scheme render it complete.

Introduction

A significant body of knowledge on the utility of introducing constraints into logic programming was accumulated in the Constraint Logic Programming (CLP) literature [14, 7]. To introduce time and temporal constraints into logic programming, three components need to be defined: (i) a temporal qualification method, (ii) a theory of temporal incidence and (iii) a temporal constraint domain. Temporal qualification is the method in which non-temporal sentences are qualified with time. For example, to specify that the formula $A \rightarrow B$ holds between the time points t_1, t_2 we could use reification to write $\texttt{Holds}(A \rightarrow B,\ t_1, t_2)$. Temporal incidence is the method in which the properties of the atoms are described. For example, part of a temporal incidence theory is the homogeneity axiom which specifies that if $\texttt{Holds}(A \rightarrow B,\ t_1, t_2)$ is *true* then $A \rightarrow B$ holds for every point inside the interval $[t_1, t_2]$. The temporal constraint domain specifies the class of atomic constraints being used together with axioms (or tables) describing their semantics. For example, we could use constraints such as $X < Y$ and the transitivity axiom $(X < Y) \wedge (Y < Z) \Rightarrow (X < Z)$.

Tokens are used to qualify propositions with time. For example, to state that $A \rightarrow B$ holds throughout an interval $[t_1, t_2]$ we introduce a token $\texttt{k}$ associated with this interval and write the conjunction $\texttt{Holds}(A \rightarrow B,\ \texttt{k}) \wedge \texttt{begin(k)=t}_1 \wedge \texttt{end(k)=t}_2$. Although there is no single dominant temporal qualification method, time tokens were shown to be advantageous from both syntactic and computational points of view [4, 15]. The use of tokens enables accommodating binary temporal constraints and allows representing infinite periodic temporal patterns while maintaining its decidability and finite representability [12].

In this paper we identify and address two difficulties that arise from the use of tokens as the temporal qualification method. To illustrate these difficulties, consider formalizing the statement "*The medicine needs to be taken every 8 hours during five days*". Let T be the only temporal variable used and let `medicine(T)` evaluate to *true* iff the medicine is taken at time point T. We could represent the example statement by the program:

```
medicine(0).
medicine(T+8) :- medicine(T), T∈[0,120].
```

Queries on programs in similar languages, that do not support a token based qualification method, can be answered correctly using standard resolution algorithms. However, as we demonstrate in this paper, the use of tokens introduces some problems. For example, using K as the only token variable and $\texttt{k}_0$ as the only token constant, an equivalent program is as follows:

```
time(k0)=0.
time(next(K))-time(K)=8.
medicine(k0).
medicine(next(K)) :-
        medicine(K), time(K)∈[0,120].
```

where `next` is a function mapping tokens to tokens. One of the advantages of this Token-Datalog formulation is that the temporal constraint `time(next(K))-time(K)=8` can be easily replaced with a non-deterministic constraint `time(next(K))-time(K)∈[7,9]`. The syntax and semantics of Token-Datalog will be clarified below.

Consider the query "*Is the medicine taken at time point 8 ?*" which can be formalized as follows:

```
time(k1)=8.
:- medicine(k1).
```

Top-Down evaluation of this query proceeds as follows: It begins by resolving the negation of the *query* literal

$\neg$medicine(k_1) with the rule

```
medicine(next(K)) :-  medicine(K),
                      time(K)∈[0,120].
```

Because medicine(next(K)) is in the head (i.e. not negated), an attempt is made to unify it with $\neg$medicine(k_1). The unification succeeds with the substitution θ_1 = (next(K)/k_1) and the resolvant is

```
:- medicine(K), time(K)∈[0,120].
```

This resolvant consists of atoms that are still not ground, and K can be assigned values such that K$\neq$next(k_1). This is undesirable because the relationship between the terms T_1 =next(K) and T_2 =K is lost. We therefore express this relation explicitly by adding the substitution θ_2=(K/next^{-1}(k_1)) and deriving the resolvant:

```
:-  medicine(next⁻¹(k1)),
    time(next⁻¹(k1))∈[0,120].
```

The next inference step involves *temporal constraint propagation.* The fact time(next(K))-time(K)=8 entails time(k_1)-time(next^{-1}(k_1))=8. From the fact time(k_1)=8 we can infer that time(next^{-1}(k_1)) =0 and that time(k_0)=time(next^{-1}(k_1)).

At this point, we make our *key observation.* We need to resolve again but we cannot unify the term next^{-1}(k_1) with any other term, because it is already ground. To arrive at the correct answer and infer that the medicine is given at k_1, we need to substitute θ_3 = (next^{-1}(k_1)/k_0). We call this non-standard operation of unifying (or substituting) two ground token terms *token fusion.* After the fusion we obtain two new subgoals:

```
:- medicine(k0), time(k0)∈[0,120].
```

medicine(k_0) is given as a fact. Because time(k_0)=0 is also a fact we know that the literal time(k_0)$\in$[0,120] is entailed. This completes the proof for the query :- medicine(k_1). The substitution used in answering this query is

$$\begin{aligned}\theta &= \theta_1\theta_2\theta_3 \\ &= (\text{next}(K)/k_1)(K/\text{next}^{-1}(k_1))(\text{next}^{-1}(k_1)/k_0)\end{aligned}$$

where θ_3 is *token fusion.*

Scope and Approach

First order logic programs are our starting point. We focus on identifying syntactic restrictions that render temporal resolution meaningful and feasible. Within this framework, we describe the unification algorithms that are components of the temporal resolution operation sought. Two important issues are left open: an ordering strategy for the generation of substitutions and a search strategy for applying the temporal resolution operation. Furthermore, extending CLP with a new temporal constraint domain is outside the scope of this paper.

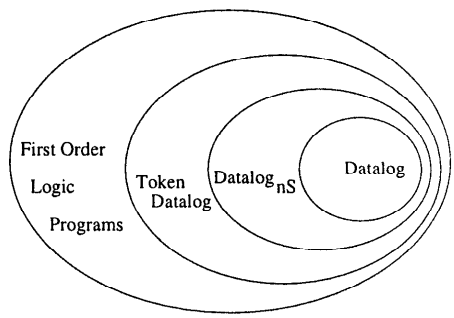

Figure 1: The hierarchy of languages.

The syntactic restrictions described in this paper define a new language. Due to its resemblance to Datalog$_{nS}$ [2, 3], we call it Token-Datalog. First Order Logic Programs subsume Token-Datalog programs, which subsume Datalog$_{nS}$ programs, which subsume Datalog programs (see Figure 1). Token-Datalog (i) uses a token based qualification method [15], (ii) its theory of temporal incidence includes the homogeneity axiom [13, 6, 16] and (iii) it supports four kinds of temporal constraints described in the preliminaries section.

The degree in which the results presented in this paper can be generalized is described in detail for each result separately. It is important to note, though, that Token-Datalog is powerful enough to represent and reason about (i) temporal relational algebra in which the end-points of the validity and transaction intervals are variables that relate to each other via constraints, and (ii) VLSI circuit verification tasks that involve combining temporal and combinatorial constraints such as "two signals are always the same" or "if signal a is received then eventually signal b will become equal to signal a (i.e. synchronize)".

The organization of this paper is as follows: The preliminaries section presents temporal constraints and the syntactic restrictions defining the Token-Datalog language. Subsequently, one section is devoted to constraint unification and one section for token fusion. The completeness section examines the combination of both and, finally, there is a concluding section.

Preliminaries

Temporal Constraints

A well known model for describing temporal relations between time points and intervals is Temporal Constraint Satisfaction Problems (TCSP). It involves a set of point and interval variables together with a set of binary constraints over them. The constraint C_{ij} between a pair of variables X_i, X_j is described by specifying a set of allowed relations, namely $C_{ij} \stackrel{\text{def}}{=} C(X_i, X_j) = (X_i\ r_1\ X_j) \vee \cdots \vee (X_i\ r_k\ X_j)$ denoted $C_{ij} = \{r_1, \ldots, r_k\}$ or $X_i\{r_1, \ldots, r_k\}X_j$.

Temporal constraints are classified according to the class the relations $r_1, \ldots r_k$ belong to. There are four classes:

1. [17] **qualitative point-point** constraints over a pair of point variables X_i, X_j where $r_i \in \{<, =, >\}$.
2. [10] **metric point-point** constraints over a pair of point variables X_i, X_j where $r_i \equiv X_j - X_i \in [a, b]$ where a, b are constants.
3. [9] **qualitative point-interval** constraints between a point variable and an interval variable, where $r_i \in$ { **before, starts, during, finishes, after** }.
4. [1] **qualitative interval-interval** constraints over pairs of intervals where

$$r_i \in \left\{ \begin{array}{c} \textbf{before, after, meets, met-by,} \\ \textbf{overlaps, overlaps-by, during, contains,} \\ \textbf{starts, started-by,} \\ \textbf{finishes, finished-by, equals} \end{array} \right\}$$

A *singleton labeling* of a TCSP is a selection of a single disjunct from every constraint. Its satisfiability can be decided in $O(n^3)$ where n is the number of variables (not labels). A *solution* is a satisfiable singleton labeling and a TCSP is *consistent* iff it has at least one solution. A disjunct $r \in C_{ij}$ is *feasible* iff replacing C_{ij} with r results in a consistent TCSP. A constraint C_{ij} is *minimal* iff all the disjuncts in C_{ij} are feasible, and all the feasible disjuncts are in C_{ij}. The *minimal TCSP* is such that all the constraints are minimal.

A TCSP $\mathcal{C}$ entails a constraint C, denoted $\mathcal{C} \models C$, iff C specifies a pair of variables in $\mathcal{C}$ and C is satisfied by all solutions of $\mathcal{C}$, namely $\mathcal{C} \models C$ iff $\mathcal{C} \wedge \neg C$ is inconsistent.

In general, deciding consistency and computing the minimal TCSP is intractable. However there are numerous subclasses for which these tasks are tractable. Moreover, there are numerous approximation algorithms with various levels of efficiency and effectiveness [1, 10, 9, 8, 5, 11].

Token-Datalog

Our starting point is the language of sorted first order logic programs. We start with three sorts: data, token and constraint. The *data* sort is inherited from Datalog and has a finite domain. The *token* sort is introduced to support token based temporal qualification. The *constraint* sort is introduced to support temporal constraints. Both constraint and token sorts are diversified into *point* and *interval* sorts. Rules are of the form `H :- B1,...,Bk` where `H` is the head and `B1,...,Bk` is the body. There are several syntactic restrictions that are necessary for the soundness and completeness of the new unification algorithms.

1. **Rules** are *range restricted*, namely all non-constraint (i.e. token or data) variables must appear in the body of some rule or equated to a variable in the body of some rule.
2. **Predicates** are classified into two types: (i) *constraint predicates* (also called temporal relations) whose arguments are constraint terms only and (ii) *non-constraint predicates* whose arguments *exclude* constraint terms and take at most one token term.
3. **Functions** are restricted to take at most one token term and map to a token term of the same type (i.e. point↦point and interval↦interval).
4. **Constraint terms** are of the form `f(K)` where `K` is a token term and `f` is a function mapping token terms into constraint terms. To represent both time points and intervals, four special purpose functions are used: `time`, `interval`, `begin` and `end`. The function `time` maps *point token* terms to *point constraint* terms (i.e. point variables). The function `interval` maps *interval token* terms to *interval constraint* terms (i.e. interval variables). The functions `begin` and `end` map *interval token* terms to *point constraint terms* such that `begin(K)` and `end(K)` are the beginning and end points of the interval specified by `interval(K)`.

The implications of lifting each of these restrictions are given separately for each of the results presented in this paper.

Example 1 : Let `X` be a data variable with the domain {`Sat, Sun`}, let `K` be a token variable, let k_0 be a token constant and let `next` be a function mapping interval tokens to interval tokens. The variable `X` and the constant `Sun` are *data terms*. `K` and `next(K)` are *interval token terms* while `begin(K)` and `end(K)` are *point constraint terms*. The term `next(interval(K))` is illegal because the function `next` takes token terms as arguments.

Tokens are used to qualify propositions with time. For example, to state that $A \rightarrow B$ holds throughout an interval $[t_1, t_2]$ we use the conjunction `Holds(`$A \rightarrow B$`, k)` $\wedge$ `begin(k)=`t_1 $\wedge$ `end(k)=`t_2. When using functions, the time point or interval associated with `f(K)` might be constrained to occur before or after the time entity associated with `K`.

Example 2 : The constraint

```
time(K)-time(prev(K))=8
```

implies that the function `prev` goes backwards in time. In contrast, `time(next(K))-time(K)=8` goes forward in time.

Semantics

Ground token terms are constants, but their interpretation on the temporal domain is not fixed. In other words, **ground token terms have a dual role:** they are constants for the logic program and variables for constraint propagation.

Example 3 : To continue with example 1, the atoms `p(a,`k_0`)` and `interval(next(`k_0`))`{`meets`}`interval(`k_0`)` are facts even though the term `interval(next(`k_0`))` is an interval variable whose end points are not fixed (e.g. we can assign `begin(`k_0`)=3.01`). In contrast, the atoms `p(a,K)`, `interval(next(K))`{`meets`}`interval(`k_0`)` are not facts because the *token term* '`K`' is not ground.

Together with restrictions 2,4, this dual role of tokens is sufficient to allow the use of constraint atoms in the heads of rules.

Example 4 : Consider the following two-rule program which violates restrictions 2,4:

```
medicine(K).
K∈[0,120] :- medicine(K).
```

The semantics of this program is unclear. To perform bottom-up evaluation we need to fix K, namely to assign K a value. This is a problem because for K∉[0,120] we need to add K∈[0,120] as a constraint fact even though this constraint is falsified[1]. Using Token-Datalog, we rewrite:

```
medicine(K).
time(K)∈[0,120] :- medicine(K).
```

In the modified formulation, time(K) is a constraint variable (representing a time point). Now, we can fix K without fixing time(K) and without introducing inconsistencies. Thus, bottom-up evaluation is well defined.

An interpretation $I =< F, \mathcal{C} >$ consists of set of non-constraint facts F and constraint facts $\mathcal{C}$. A ground non-constraint atom A evaluates to *true* iff $A \in F$. A constraint atom C, whose token terms are ground, evaluates to *true* iff it is entailed by $\mathcal{C}$, namely iff $\mathcal{C} \models C$. A rule r of the form H :- $B_1, \ldots, B_k$ evaluates to *true* in I, denoted $I \models r$, iff r satisfies, for every substitution θ,

$$if \quad \forall i \;\; I \models B_i\theta \quad then \quad I \models H\theta \; .$$

An interpretation I is a model of a program P iff

1. every rule r in P is satisfied by I, namely $\forall r \in P$, $I \models r$, and
2. I does not contain an incompatibility set.

The definition of incompatibility sets is not straight forward and relies on restrictions 2,3,4. For brevity, we present an informal example.

Example 5 : Let F_1=p(k_1), F_2=¬p(k_2) and let $\mathcal{C}$ ⊨interval(k_1){overlaps} interval(k_2). Substituting k_1,k_2 with k results in F_1'=p(k), F_2'=¬p(k). Since the conjunction $F_1' \wedge F_2'$ is inconsistent, $\{F_1, F_2\}$ is an incompatibility set.

Theorem 1 : *[12] Every program Ψ has a unique minimal model $M_\Psi =< F_\Psi, \mathcal{C}_\Psi >$.*

A direct consequence of the semantics is the following method for answering queries:

Theorem 2 : *For every* k $\in$ *fusion($\mathcal{K}$), a predicate $p(x_1, \ldots, x_m$,k) evaluates to* true *if exists a token* k'$\in \mathcal{K}$ *that satisfies* begin(k')< begin(k) < end(k)< end(k') *and $p(x_1, \ldots, x_m$,k') evaluates to* true *(this implements the homogeneity property with open intervals).*

[1]To address this problem, CLP-like languages use constraint facts of the form a:-c where a is a non-constraint atom and c is a constraint atom.

The restrictions presented in the preliminaries section affect these two results as follows: Lifting restriction 1 renders bottom-up evaluation unsound, thus the intended fixed point semantics is lost. Lifting restriction 2 falsifies Theorem 1 since it results in loosing the uniqueness of the intersection of all models. Lifting restriction 3 falsifies the Theorem 2 since it invalidates the notion of *incompatibility sets* (i.e. the intended semantics is lost).

Constraint Unification

A necessary inference step in the introductory example is computing the constraint relationship between time(k_0), time(k_1), time(next(k_0)) and time(next^{-1}(k_1)). In general, we are interested in the relationship between the time entities associated with *pairs* of tokens. The approach we propose here is to isolate and formalize the temporal relations as a Temporal Constraint Satisfaction Problem (TCSP) and apply constraint processing techniques to answer entailment queries. The challenge is to isolate the set of relevant temporal constraints at every resolution step, and show how to use constraint processing techniques to determine which constraint atoms are entailed by the program Ψ. We start with the following definition:

Definition 1 : Let $\mathcal{C}_\Psi$ be the set of constraint atoms in M_Ψ (the unique minimal model of Ψ) and let $\mathcal{C}_U$ be the subset of the unit clauses of a program Ψ (i.e. rules with empty bodies) consisting of constraint atoms.

$\mathcal{C}_\Psi$ is the desired isolated TCSP and $\mathcal{C}_U$ is an approximation to it. Consequently, constraint processing techniques aimed at deciding whether a constraint atom A is entailed by $\mathcal{C}_U$ are sound for deciding whether $\Psi \models A$.

Example 6 : Consider the introductory example where $\mathcal{C}_U$={time(k_0)=0, time(next(k_0))-time(k_0)=8 } and consider the query :- time(k_0)∈[0,120]. Deciding whether the atom A=(time(k_0)∈[0,120]) is entailed by Ψ can be done using constraint processing techniques as follows. Examine the conjunction $\mathcal{C}_U \wedge \neg A$ which subsumes the conjunction time(k_0)=0 ∧ ¬(time(k_0)∈[0,120]). Because this conjunction is inconsistent, we infer that $\Psi \models A$. This inconsistency is always detected by computing the minimal TCSP. It is sometimes detected by polynomial algorithms, such as path-consistency, aimed at approximating the minimal TCSP [1, 10, 9, 8, 11].

The TCSP $\mathcal{C}$ entails a constraint fact A, denoted $\mathcal{C} \models A$, iff A is satisfied by all solutions of $\mathcal{C}$.

Lemma 1 : *For every* constraint fact[2] *A, $\Psi \models A$ iff $\mathcal{C}_\Psi \models A$. Also, if $\mathcal{C}_U \models A$ then $\mathcal{C}_\Psi \models A$.*

In other words, all the temporal constraints that are entailed can be inferred by temporal constraints in M_Ψ.

A necessary step for deciding the entailment $\mathcal{C}_U \models A$ is to perform constraint unification that verifies

[2]A constraint atom whose token terms are ground.

whether the constraint terms in A appear in C_U. The particular technique we use to introduce tokens (restrictions 2,3,4) induces a specific structure of constraint atoms.

Definition 2 : A constraint atom is a binary predicate of the form

$$C(g_1(\mathtt{K}_1), g_2(\mathtt{K}_2))$$

where C is the temporal relation (i.e. constraint), $g_1(\mathtt{K}_1)$ and $g_2(\mathtt{K}_2)$ are constraint terms (i.e. constraint variables), g_1, g_2 are functions mapping token terms to points or interval variables, and $\mathtt{K}_1, \mathtt{K}_2$ are token terms.

Example 7 : The atom `interval(next(K))` `{overlaps}interval(K)` evaluates to *true* whenever the intervals associated with `K` and `next(K)` overlap. For this atom, $\mathtt{K}_1$=`next(K)`, $\mathtt{K}_2$=`K`, g_1=g_2=`interval` and $C = C(X, Y)$ is the relation `X{ overlaps} Y`. The atom `begin(next(K))-end(K)=8` evaluates to *true* whenever the beginning of the interval associated with the token term `next(K)` is 8 time units after the end of the interval associated with the token term `K`. For this atom, $\mathtt{K}_1$ = `next(K)`, $\mathtt{K}_2$=`K`, g_1=`begin`, g_2=`end` and $C = C(X, Y)$ is the relation $X - Y \in [8, 8]$.

Next, algorithm UnifyConstraints (Figure 2) is illustrated and analyzed.

Example 8 : To illustrate the execution of algorithm UnifyConstraints, let C_U include the constraint C = `time(`$\mathtt{k}_1$`)-time(`$\mathtt{k}_0$`)=8` and let the query A be A = `time(next(K))-time(K)`$\in$`[4,10]`. Since $g_1 = g_3$ and $g_2 = g_4$ we try to unify `next(K)`, `K` with $\mathtt{k}_1, \mathtt{k}_0$ respectively. The unification succeeds with the substitution θ = `(next(K)/`$\mathtt{k}_1$`)(K/`$\mathtt{k}_0$`)` which has the desired property that $C \models A\theta$ implying that $C_U \models A\theta$ and that $\Psi \models A\theta$.

Lemma 2 : *Applying algorithm UnifyConstraints on a TCSP defined by C_U and a constraint atom A,*

1. *succeeds iff $\exists\theta \;\; C_U \models A\theta \;\vee\; C_U \models \neg A\theta$, and*
2. *computes θ such that $C_U \models A\theta$ or $C_U \models \neg A\theta$.*
3. *step 1 is intractable (we need an approximation) but steps 2-9 terminate in $O(nm)$ steps where n is the number of constraints in C_U and m is the maximum length of the terms unified.*

Algorithm UnifyConstraints (Figure 2) deviates from the standard unification algorithms in that it unifies two atoms even if they differ in the constraint (i.e. predicate) they specify. Not unifying different constraints on the same pair of variables restricts inference to facts explicitly listed in the database only, thus not utilizing the ability of constraint propagation to infer implicit constraints.

Example 9 : In the introductory example, without constraint unification we could not infer that `time(`$\mathtt{k}_0$`)=time(next`$^{-1}$`(`$\mathtt{k}_1$`))`. Moreover, constraint unification may unify disjunctions. Consider the substitution θ = `(next(K)/`$\mathtt{k}_1$`)(K/`$\mathtt{k}_0$`)`, which unifies

Algorithm UnifyConstraints
Input: A constraint atom A and a *consistent* TCSP C.
Output: A unifier θ such that $C \models A\theta$ or $C \models \neg A\theta$ (if θ exists).

1. Compute C_{min}, the minimal TCSP of C.
2. **For** every constraint C in C_{min} **do**
3. Define $C = C(g_1(K_1), g_2(K_2))$ and $A = A(g_3(K_3), g_4(K_4))$.
4. **If** $g_1 = g_3$ and $g_2 = g_4$ and
   $\mathtt{K}_1, \mathtt{K}_2$ are unifiable with $\mathtt{K}_3, \mathtt{K}_4$ respectively **then**
5. Assign θ = `(`$\mathtt{K}_3$`/`$\mathtt{K}_1$`)(`$\mathtt{K}_4$`/`$\mathtt{K}_2$`)`.
6. **If** $C \models A\theta$ or $C \models \neg A\theta$ **then** return θ.
7. **End-if**
8. **End-for**
9. Exit with failure.

Figure 2: Unifying constraints.

the constraint `interval(`$\mathtt{k}_1$`){meets,starts, overlaps}interval(`$\mathtt{k}_2$`)` with the constraint `interval(next(K)) {meets,overlaps}interval(K)`. The predicate calculus representation of the first constraint is equivalent to the disjunction

```
Meets(interval(k1),interval(k2)) ∨
Starts(interval(k1),interval(k2)) ∨
Overlaps(interval(k1),interval(k2))
```

while the second constraint is equivalent to the disjunction

```
Meets(interval(next(K)),interval(K)) ∨
Overlaps(interval(next(K)),interval(K))
```

The restrictions presented in the preliminaries section affect the applicability of this algorithm as follows: Lifting restriction 1 renders bottom-up evaluation unsound and thus the intended semantics is lost. Lifting restriction 2,4 disables constraint unification as it invalidates the central assumption on the form of constraints atoms presented in definition 2. Lifting restriction 3 does not affect the above result.

Token-Fusion

Token-fusion is the operation of unifying two ground token terms. To illustrate the problem addressed by token-fusion, consider the use of resolution for detecting inconsistency of the conjunction of the facts

$$\mathtt{P}(\mathtt{x}_1,\ldots,\mathtt{x}_n,\mathtt{k}_1) \wedge \neg\mathtt{P}(\mathtt{x}_1,\ldots,\mathtt{x}_n,\mathtt{k}_2) \wedge \mathtt{time}(\mathtt{k}_1)=\mathtt{time}(\mathtt{k}_2).$$

In other words, we would like to identify *incompatibility sets*[3] using resolution.

Inconsistencies such as the above cannot be detected using traditional unification techniques because that would require substituting $(\mathtt{k}_1/\mathtt{k}_2)$, which cannot be done when both $\mathtt{k}_1$ and $\mathtt{k}_2$ are ground. Consequently, we are seeking a formal condition defining when two token terms are fusible, namely when it is possible to introduce substitutions such as θ=$(\mathtt{k}_1/\mathtt{k}_2)$.

[3]defined in the preliminaries section as part of the semantics of Token-Datalog

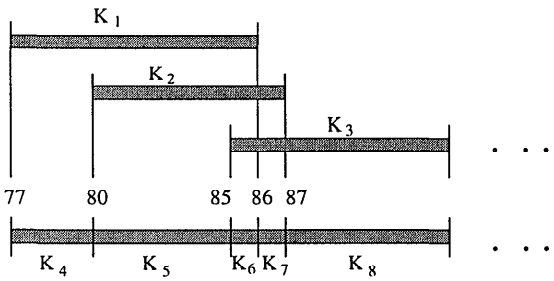

Figure 3: Token fusion.

Example 10 : Consider the statement "John got married in 1984" and "John bought a house in 1984". Using tokens, a possible formalization of this example is as follows:

```
GetMarried(John, k1).   time(k1)=1984.
BuyHouse(John, k2).     time(k2)=1984.
```

In many cases we may need to infer that "John got married and bought a house in 1984". To answer the query

```
:- GetMarried(X, K), BuyHouse(X, K)
```

we need to substitute k_2 with k_1 and obtain the facts: `GetMarried(John, k1)`, `BuyHouse(John, k1)` and `time(k1)=1984`. With this substitution, the query above can be answered directly.

Definition 3 : Two tokens k_1 and k_2 are *fusible* iff C_Ψ entails that the time entities associated with k_1 and k_2 overlap, namely there are three possible cases: 1) k_1, k_2 are point tokens and $C_\Psi \models$ `time(k1)=time(k2)`, 2) k_1 is a point token while k_2 is an interval token and $C_\Psi \models$ `begin(k2)<time(k1)<end(k2)`, and 3) both k_1, k_2 are interval tokens and $C_\Psi \models$ `begin(k1)<end(k2)`, `begin(k2)<end(k1)`.

Next, we compute the fusion of a set of tokens whose end points are linearly ordered.

Example 11 : Consider the statement "Mary lived in Arrowhead from 1977 to 1986, when she moved to San Francisco. John lived in Arrowhead from 1980 to 1987, when he joined Mary in San Francisco. They are married since 1985." A possible formalization of this example is as follows:

```
Live(Marry, Arrowhead, k1).
Live(John, Arrowhead, k2).
Married(Mary, John, k3).

        begin(k1)=1977.   end(k1)=1986.
        begin(k2)=1980.   end(k2)=1987.
        begin(k3)=1985.   end(k3)>1987.
```

Here, the fusion of $\{k_1, k_2, k_3\}$ is $\{k_4, k_5, k_6, k_7, k_8\}$ (illustrated in figure 3) where:

```
begin(k4)=1977,          end(k4)=begin(k5)=1980,
end(k5)=begin(k6)=1985,  end(k6)=begin(k7)=1986,
end(k7)=begin(k8)=1987,  end(k8)>1987.
```

Algorithm FuseTokens
Input: Two interval token terms K_1, K_2 and a *consistent* TCSP $\mathcal{C}$.
Output: If they are fusible, output $fusion(\{K_1, K_2\})$.

1. **if** $\mathcal{C} \models begin(K_1) < end(K_2)$ **and**
   $\mathcal{C} \models end(K_1) > begin(K_2)$ **then**
2. **if** $\mathcal{C} \models begin(K_1) \leq begin(K_2)$ **or**
   $\mathcal{C} \models begin(K_1) \geq begin(K_2)$ **then**
3. **if** $\mathcal{C} \models end(K_1) \leq end(K_2)$ **or**
   $\mathcal{C} \models end(K_1) \geq end(K_2)$ **then**
4. **output** $fusion(\{K_1, K_2\})$ (see Definition 4).

Figure 4: Fusing a pair of interval token terms.

Definition 4 : Let $\mathcal{K}$ be a set of tokens whose end points are *linearly ordered*, and let $E = \{e_1, \ldots, e_n\}$ be the ordered set of end points of the time entities associated with these tokens (obtained by `time`, `begin`, `end`). The fusion of the set $\mathcal{K}$, denoted $fusion(\mathcal{K})$, is the set $k_1, \ldots, k_{n-1}$ where `end(k_i)=begin(k_{i+1})=e_{i+1}` and `begin(k1)=e1`, `end(k_{n-1})=e_n`.

To compute the fusion of a pair of token terms we use algorithm FuseTokens described in Figure 4. Step 1 tests whether the two interval token terms are fusible, namely whether the constraints C_Ψ entail that their intervals overlap. Steps 2,3 tests whether the end points of the token terms are linearly ordered. Step 4 computes the fusion according to the definition above.

In addition to computing $fusion(\mathcal{K})$, we need to determine which predicates evaluate to *true* at each of these tokens.

Example 12 : To continue example 11, we would like to state that from 1986 to 1987, Mary was in San Francisco while John was in Arrowhead, but they were married. Thus we obtain `Live(Marry, SanFrancisco, k7)`, `Live(John, Arrowhead, k7)`, `Married(Marry, John, k7)`.

Lemma 3 : *Let Ψ be a program whose C_U[4] induces a linear order on the end points of* all *the token terms in Ψ, and let $\mathcal{K}$ be the set of token terms occurring in Ψ.*

1. *The truth value of the predicates in Ψ may change only at the end points of the token terms in $fusion(\mathcal{K})$ (see Example 11).*
2. *Computing $fusion(\mathcal{K})$ and the set of predicates that evaluate to true for each token in $fusion(\mathcal{K})$ requires $O(n(n+m))$ where n is the number of tokens in $\mathcal{K}$ and m is the total number of predicates.*

The token fusion algorithm in Figure 4 utilizes this result.

The restrictions presented in the preliminaries section affect the applicability of this algorithm as follows: Lifting restriction 1 renders bottom-up evaluation unsound and thus the intended semantics is lost. Lifting restriction 2,4 disables constraint unification and thus does not allow to identify fusible token terms. Lifting restriction 3 invalidates the notion of *incompatibility sets* and thus the intended semantics is lost.

[4] All unit clauses containing constraint atoms.

Completeness

In this section we show that the combination of the two algorithms presented above is sufficient for making correct inferences with Token-Datalog. However, we make no claims regarding additional modification to the unification algorithms that may be required for temporal reasoning in its most general form. For clarity of presentation, we consider three classes of languages that have increasing levels of expressiveness and complexity:

- $\mathcal{L}_0$ is a subclass of Token-Datalog in which there are no constraint atoms and all the token terms are ground.
- $\mathcal{L}_1$ extends $\mathcal{L}_0$ by allowing constraint atoms (token terms are ground).
- $\mathcal{L}_2$ is the general form of Token-Datalog which extends $\mathcal{L}_1$ by allowing non-ground token terms.

Let R be a generalized resolution algorithm, let R^c be R augmented with constraint unification, let R^k be R augmented with token fusion and let R^{kc} be R augmented with both. R^{kc} modifies line 4 in algorithm *UnifyConstraints* to include token fusion, namely the unification of $\mathtt{K}_1,\mathtt{K}_2$ with $\mathtt{K}_3,\mathtt{K}_4$ when $\mathtt{K}_1,\mathtt{K}_2,\mathtt{K}_3,\mathtt{K}_4$ are all ground token terms.

Lemma 4 : *If algorithm R is complete for $\mathcal{L}_0$ then algorithm R^c is sound and complete for $\mathcal{L}_1$.*

Next, to show that R^{kc} is complete for $\mathcal{L}_2$ we need to show that the homogeneity axiom holds.

Lemma 5 : *If algorithm R is complete for $\mathcal{L}_0$ then R^{kc} is sound and for every program $\Psi \in \mathcal{L}_2$, if R^{kc} terminates without identifying an inconsistency then for every interval token term k and point token k' if* $\Psi \models \mathtt{p(k)} \wedge (\mathtt{begin(k)} < \mathtt{time(k')} < \mathtt{end(k)})$ *then* $\Psi \models \mathtt{p(k')}$.

Recall that without token fusion we might not find a ground token term `k'` that enables us to detect the inconsistency.

Theorem 3 : *Generalized resolution augmented with constraint propagation and token fusion is complete for Token-Datalog.*

This result hangs on all the restrictions, as described separately for each of the two algorithms above.

Conclusion

Top-down evaluation is a commonly used query processing method. For regular (i.e. non-temporal) logic programs, this method correctly decides whether a query is a logical consequence of the input program. In contrast, for temporal logic programs some problems arise when using a token based temporal qualification method.

We identified and demonstrated two major difficulties: (i) the need unify a pair of ground token terms, and (ii) the need to unify constraint atoms. To address these problems we introduced the *token fusion* and *constraint unification* methods. We analyzed the properties of these methods and proposed a modification to the standard top-down evaluation method.

These results sheds some light on the applicability of logic programming in general, and CLP in particular, to temporal reasoning. Introducing token fusion may extend CLP to support a token based temporal qualification method. Introducing constraint unification may enable (i) having constraint atoms in the heads of the rules, (ii) processing constraints that do not appear in the database explicitly by inferring implicit constraints, and (iii) making inferences with constraints even when all temporal variables are not instantiated.

Acknowledgments

We would like to thank Lluís Vila for fruitful discussions, and Rina Dechter for comments on an earlier draft of this paper. We would also like to thank the anonymous reviewers that helped improve this paper.

References

[1] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[2] J. Chomicki. Depth-bounded bottom-up evaluation of logic programs, Journal of Logic Programming, Oct 1995.

[3] J. Chomicki. Finite representation of infinite query answers. *ACM Transaction on Database Systems*, June 1995.

[4] T. Dean and D. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1987, 1987.

[5] R. Dechter. *Encyclopedia of Artificial Intelligence*, chapter Constraint Networks. John Wiley&Sons Inc., 2nd edition edition, 1992.

[6] A. Galton. Reified temporal theories and how to unreify them. In *Proc. IJCAI'91*, pages 1177–1182, 1991.

[7] J. Jaffar and J. Lassez. Constraint logic programming: A survey. *Journal of Logic Programming*, 19,20:503–581, 1994.

[8] H. Kautz and P. Ladkin. Integrating metric and qualitative temporal reasoning. In *Proc. AAAI'91*, pages 241–246, 1991.

[9] I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87(1-2):295–342, 1996.

[10] R.Dechter, I.Meiri, and J.Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[11] E. Schwalb and R. Dechter. Processing temporal constraint networks. *Artificial Intelligence*, (summer 97).

[12] E. Schwalb, L. Vila, and R. Dechter. Deductive databases with temporal constraints. Technical report, UCI, 1997.

[13] Y. Shoham. *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*. The MIT Press, 1988.

[14] P. vanHentenrick. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.

[15] L. Vila and H. Reichgelt. The token reification approach to temporal reasoning. *Artificial Intelligence*, 83(1):59–74, May 1996.

[16] L. Vila and E. Schwalb. A theory of time and temporal incidence based on instants and periods. Technical Report 96-04, ICS Dept. UCI, 1996.

[17] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. AAAI'86*, pages 377–382, 1986.