Temporal Probabilistic Logic Programs

Alex Dekhtyar
University of Maryland
dekhtyar@cs.umd.edu

Michael I. Dekhtyar
Tver State University
Michael.Dekhtyar@tversu.ru

V.S. Subrahmanian
University of Maryland
vs@cs.umd.edu

May 13, 1999

Abstract

There are many applications where the precise time at which an event will occur (or has occurred) is uncertain. Temporal probabilistic logic programs (TPLPs) allow a programmer to express knowledge about such events. In this paper, we develop a model theory, fixpoint theory, and proof theory for TPLPs, and show that the fixpoint theory may be used to enumerate consequences of a TPLP in a sound and complete manner. Likewise the proof theory provides a sound and complete inference system. Last, but not least, we provide complexity results for TPLPs, showing in particular, that reasonable classes of TPLPs have polynomial data complexity.

1 Introduction

There are a vast number of applications where uncertainty and time are indelibly intertwined. For example, the US Postal Service (USPS) as well as most commercial shippers have detailed statistics on how long shipments take to reach their destinations. Likewise, we are working on a Viennese historical land deed application where the precise time at which certain properties passed from one owner to another is also highly uncertain. Historical radio carbon dating methods are yet another source of uncertainty, providing approximate information about when a piece was created.

Logical reasoning in situations involving temporal uncertainty is definitely important. For example, an individual querying the USPS express mail tracking system may want to know when he can expect his package to be delivered today — he may then choose to stay home during the period when the probability of delivery seems very high, and leave a note authorizing the delivery official to leave the package by the door at other times.

In this paper, we propose the concept of a *Temporal Probabilistic Logic Program* (or TPLP for short). We define the syntax of TPLPs and provide a formal model theoretic and fixpoint semantics that are shown to coincide. We then develop polynomial time bottom-up, sound and complete fixpoint computation algorithms. Then we present a "brute force" sound and complete proof procedure. Though this procedure is inefficient, it is transparent and easy to understand. Subsequently, we develop a more sophisticated proof procedure that manipulates succinct representations of temporal-probabilistic information, and is still sound and complete — this proof procedure is much more efficient than the previous one.

2 Temporal Probabilistic Programs: Syntax

Let L be a language generated by finitely many constant and predicate symbols. We assume that L has no ordinary function symbols, but it may contain *annotation function* symbols for a fixed family of functions.The interpretation of these function symbols is given below, at this moment we only mention that annotation function symbols are split into two disjoint sets of *probabilistic annotated functions* and *temporal annotated functions.* If p is an n-ary predicate symbol and $a_1, \ldots, a_n$ are either constants of variables, then $p(a_1, \ldots, a_n)$ is called a *simple event atom.* When all the a_i's are constants, $p(a_1, \ldots, a_n)$ is said to be *ground.* Each ground event atom represents one unique instance of an instantaneous event of the "real world". We use B_L to denote the set of ground simple event atoms.

A calendar τ is any initial segment of the set of natural numbers: $\tau = \{1, \ldots, t_{\max}\}$ for some $t_{\max}$. We will denote such calendar $\tau = [1, t_{\max}]$.

Definition 1 *Let $A_1, \ldots A_k$ all be simple event atoms. Then $A_1 \wedge \ldots \wedge A_k$ and $A_1 \vee \ldots \vee A_k$ are called* compound event atoms. Simple event atoms *and compound event atoms are both* event atoms.

Definition 2 A *probabilistic annotation function* f_p of arity n is a total function $f_p : [0, 1]^n \longrightarrow [0, 1]$.
A *temporal annotation function* f_t of arity n is a total function $f_t : \tau^n \longrightarrow \tau$

We assume that associated with each *annotation function* is a body of terminating s oftware code implementing that function.

We also assume that all variable symbols from L are partitioned into three classes. We will call one class *object variables* and this class will contain the regular first order logic variable symbols. The second and third classes of variable symbols, *probabilistic annotation variables* and *temporal annotation variables* will contain variable symbols that range over the interval $[0, 1]$ and over calendar τ respectively. These variables can appear only inside *annotation items*, which are defined below:

Definition 3 An *annotation item* δ based on a set of constants $\hat{C}$, set of variables $\hat{V}$ and set of *annotation functions* $\hat{\mathcal{F}}$ is one of the following:

- a constant $\alpha \in \hat{C}$;
- a variable $v \in \hat{V}$;
- an expression of the form $f(\delta_1, \ldots, \delta_k)$ where $\delta_1, \ldots \delta_k$ are all annotation items based on $\langle \hat{C}, \hat{V}, \hat{\mathcal{F}} \rangle$ and $f \in \hat{\mathcal{F}}$ is a k-ary function symbol.

In this paper we will consider two types of annotation items: *probabilistic annotation items* and *temporal annotation items*. Both are defined below.

Definition 4 A *probabilistic annotation item* δ is an annotation item based on the set of constants from the $[0, 1]$ interval, and on the sets of probabilistic annotation variables and probabilistic annotation functions from L.

A *temporal annotation item* δ is an annotation atom based on the set of all constants from τ, and on the sets of temporal annotation variables and temporal annotation functions from L.

Definition 5 A temporal constraint $c = c(y, y_1, \ldots, y_k)$ with *independent variables* $y_1, \ldots y_k$ and *dependent variable* y is one of the following:

- Let λ be a temporal annotation items with $y_1, \ldots y_k$ being its only variable symbols. Then y *op* λ where *op* $\in \{=, <, >, \leq, \geq, \neq\}$ is a temporal constraint.
- Let λ_1 and λ_2 be temporal annotation items that contain only variables $y_1, \ldots y_k$. Then $y : \lambda_1 \sim \lambda_2$ is a temporal constraint.
- Let c_1 and c_2 be temporal constraints with the same dependent variable. Then $c_1 \wedge c_2$, $c_1 \vee c_2$ and $\neg c_1$ are temporal constraints.

A temporal constraint c is called *ground* if it contains no independent variables.

We will slightly abuse notation and sometimes instead of constraints $y = t$ and $y : t \sim t$ we will simply write t.

Definition 6 *Let $c = c(y)$ be a ground temporal constraint. The* solution set *of c, denoted $sol(c)$ is defined as follows:*

1. *c is atomic. $sol(c)$ is determined by the following table:*

Case	$sol(C)$
$y \le t$	$\{x \in \tau \vert x \le t\}$
$y < t$	$\{x \in \tau \vert x < t\}$
$y = t$	$\{t\}$
$y \neq t$	$\{x \in \tau \vert x \neq t\}$
$y > t$	$\{x \in \tau \vert x > t\}$
$t \ge t$	$\{x \in \tau \vert x \ge t\}$
$y : t \sim t'$	$\{x \in \tau \vert x \ge t \wedge x \le t'\}$

2. *If $c = c_1 \wedge c_2$, $c_1 \vee c_2$ or $\neg c_1$ then $sol(c)$ is defined as $sol(c_1) \cap sol(c_2)$, $sol(c_1) \cup sol(c_2)$ and $\tau - sol(c_1)$ respectively.*

Definition 7 *Let $c = c(y, y_1, \ldots y_k)$ be a (non-ground) temporal constraint. We define $sol(c(y, y_1, \ldots, y_k))$ as a function $sol(c) : \tau^k \longrightarrow 2^\tau$ such that $(\forall (a_1, \ldots a_k) \in \tau^k)(sol(c)(a_1, \ldots, a_k) = sol(c(y, a_1, \ldots, a_k))$.*

Definition 8 *A* probabilistic weight function *w associated with a subset T of calendar τ is a function $w : T \longrightarrow [0, 1]$.*

If $c = c(y, y_1, \ldots, y_k)$ is a temporal constraint, a generalized probabilistic weight function *ω_c is defined as a function that takes as arguments (i) a substitution $\bar{a} = (a_1, \ldots, a_k)$ of values for $y_1, \ldots y_k$ and (ii) a timepoint $t \in \tau$ and returns a number between 0 and 1. We only require that if $\omega_c(\bar{a}, t) \neq 0$, then $t \in sol(c(t, a_1, \ldots, a_k))$.*

The intuition underlying the above definition is as follows. Consider a constraint $c = c(y, y_1, \ldots, y_k)$ and let $\bar{a}$ be a vector of k time points. and let $\theta = [y_1, \ldots, y_k]/\bar{a}$. Then $c\theta$ determines a set of time points, viz. those that make c true by making an assignment to the dependent variable y. A probabilistic weight function assigns a probability to each time point in this set.

Example 1 *For instance, consider the temporal constraint $c = c(y, y_1, y_2)$ of the form $2 \le y \wedge y \le y_1 \wedge y_1 \le y_2 \wedge y_2 \le 4$. When we set $y_1 = y_2 = 4$, i.e. $\bar{a} = (4, 4)$, this constraint determines the set of time points $\{2, 3, 4\}$. A probabilistic weight function $omega_c(\bar{a}, t)$ may associate the respective values $1, 1, 0.5$ with these time points.*

We will use probabilistic weight functions as follows. Now suppose we consider a formula A, and suppose we want to say that A is true with probability $\omega_c(\bar{a}, t)$ at any time t which is a solution of c. Then this means that A is true with 100% probability at times 2 and 3 and 50% probability at time 4.

Notice that if temporal constraint c is *ground* then any generalized probabilistic weight function ω_c is reduced to a simple probabilistic weight function defined on $sol(C)$. In this case, as no independent variables are present in c, we will write $\omega_c(y)$ and not distinguish between it and the simple weight function.

Let $c(y, y_1, \ldots, y_k)$ be a temporal constraint such that $(\forall (a_1, \ldots, a_k) \in \tau^k)(|sol(c(y, a_1, \ldots, a_k))| = 1$. We will denote by $\sharp$ the generalized probabilistic weight function ω_c, such that $\omega_c(t, a_1, \ldots a_l) = 1$ iff $t = sol(c)(a_1, \ldots, a_k)$. This defines $\sharp$ to be a *universal identity* weight function.

Also, sometimes we will specify the weight function in a form of a set of values: $\{v_1, \ldots, v_m\}$. We can do this when we know that the $|sol(c)| = k$. For example, if $c(y) = y : 3 \sim 5$, a weight function ω_c can be represented as $\{0.5, 1, 0\}$. This will mean $\omega_c(3) = 0.5, \omega_c(4) = 1, \omega_c(5) = 0$.

Definition 9 *A* temporal probabilistic annotation *is a quadruple $\langle c, l, u, \omega_c\rangle$ where c is a temporal constraint, l and u are probabilistic annotation items and ω is a generalized probabilistic weight function.*

Definition 10 *Let $F = A_1 * \ldots * A_k$ be an event atom ($* \in \{\vee, \wedge\}$) and $\mu = \langle c, l, u, \omega_c\rangle$ be a tp-annotation. Then $F : \mu$ is a* tp-annotated basic formula *or just an* annotated basic formula.

Intuitively when F is ground, $F : \langle c, l, u, \omega_c \rangle$ represents the fact that the events described by F happened at a point in $sol(c)$ with a probability in the interval $[l, u]$ and that the probability that these events occured at a particular timepoint $t \in sol(c)$ is given by the weight function ω_c.

Definition 11 *Let $A : \mu, F_1 : \mu_1, \ldots F_m : \mu_m$ be tp-annotated basic formulas, $A \in B_L$. Then $A : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_m : \mu_m$ is called a* temporal probabilistic clause *or a* tp-clause.

Definition 12 *A temporal probabilistic program (tp-program) is a finite set of tp-clauses. If P is a tp-program, we let $ground(P)$ denote the set of all ground instances of rules of P.*

Example 2 Let $\tau = [0, \ldots 30]$. Consider the following ground tp-program:

```
arrived(Item,Place): [y : 3 ~ 5, 0.5, 0.8, {0.5, 0.3, 0.2}] <---
        sent(Item,Place): [y = 1, 0.9, 1, ♯].
arrived(Item,Place): [y : 6 ~ 8, 0.2, 0.4, {0.75, 0, 0.25}] <---
        sent(Item,Place): [y = 1, 0.9, 1, ♯].
arrived(Item,paris): [y : 3 ~ 4, 0.5, 0.9, {0.6, 0.4}] <---
        sent(Item,paris): [y = 1, 0.95, 1, ♯] ∧
        express_mail(Item): [y = 1, 1, 1, ♯].
sent(shoes,rome): [y = 1, 1, 1, ♯] <--- .
sent(letter,paris): [y = 1, 1, 1, ♯] <--- .
express_mail(letter,paris): [y = 1, 1, 1, ♯] <--- .
```

This program represents a small part of a deductive database of some company that deals with projected arrivals of the packages shipped by the company. First two rules of the program provide the information on the probability distribution of the arrival time of an arbitrary package sent to any place. The third rule gives some extra information about the arrival time of packages sent to Paris via express-mail. Three facts about shipments complete this simple program.

3 Temporal Probabilistic Programs: Model Theory and Fixpoint Semantics

3.1 Model Theory

In this section we introduce the model theory of tp-programs.

Definition 13 (Thread) *A* thread *is a function $th : B_L \to 2^\tau$.*

Intuitively, a thread th contains information about the times when *each* event occurs. It specifies one possible way events could occur in time. If for some ground event atom A, $th(A) = \emptyset$, we interpret it as meaning that the event associated with A does not occur in thread th at all. The notion of TP-world below says that we may not know which of several possible threads actually describes the occurrence of events over time.

Definition 14 *A TP-world M is a pair $M = \langle TH, p \rangle$ where $TH = \{th\}$ is a set of threads and $p : TH \to [0, 1]$ is a probability distribution function such that $\sum_{th \in TH} p(th) \leq 1$.*

Here, TH represents a set of possible ways events could occur, while p specifies the probability of each thread. Given a TP-world M, we may specify the probability of a formula via the following definition.

Definition 15 *Suppose $M = \langle TH, p \rangle$ is a TP-world and $A \in B_L$. The probability $p_M(A, t)$ that the event denoted by A occurred at time t according to the TP-model M is defined as follows:*

$$p_M(A, t) = \sum_{th \in TH, th(A) \ni t} p(th).$$

If $F = A_1 \wedge \ldots \wedge A_k$ and $G = A_1 \vee \ldots \vee A_k$ are ground compound event atoms*, then $p_M(F,t)$ and $p_M(G,t)$ are definable as:*

$$p_M(F,t) = \sum_{th \in TH, th(A_1) \ni t, \ldots, th(A_k) \ni t} p(th).$$

$$p_M(G,t) = \sum_{th \in TH, (\exists i \in \{1, \ldots k\})(th(A_i) \ni t)} p(th).$$

The above definition specifies the probability of a formula being true at a given point in time. We can extend this to define the probability that a formula is true at some (possibly more than one) time point in a solution of a ground temporal constraint c.

Definition 16 *If $M = \langle TH, p\rangle$ is a TP-world, $F = A_1 \wedge \ldots \wedge A_k$ and $G = A_1 \vee \ldots \vee A_k$ are event atoms, and $c = c(y)$ is a ground temporal constraint. Then $p_M(F,c)$ is defined as:*

$$p_M(F,c) = \sum_{th \in TH, th(A_1) \cap \ldots \cap th(A_k) \cap sol(c) \neq \emptyset} p(th).$$

$$p_M(G,c) = \sum_{th \in TH, (\exists i \in \{1, \ldots k\})(th(A_i) \cap sol(c) \neq \emptyset)} p(th).$$

We are now in position to define satisfaction for *ground formulas.*

Definition 17 (Satisfaction) *Let $M = \langle TH, p\rangle$ be a TP-world.*

- $M \models F : [c, l, u, \omega_c]$, *where* $c = c(y)$ **iff**
  - $p_M(F,c) \in [l, u]$
  - $(\forall t \in sol(c)) p_M(F,t) \in [\omega_c(t) \cdot l, \omega_c(t) \cdot u]$.
- $M \models F_1 : \mu_1 \wedge \ldots \wedge F_k$ **iff** $\forall i \in \{1, \ldots k\} M \models F_i : \mu_i$.
- $M \models F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_k$ **iff** *either* $M \models F : \mu$ *or* $M \not\models F_1 : \mu_1 \wedge \ldots \wedge F_k$.

A TP-world M is called a model *of a tp-program P ($M \models P$)* **iff** $(\forall r \in P)(M \models r)$*. A tp-program P is* consistent *if it has a model.*

As usual we say that $F : \mu$ is a *consequence* of P **iff** for every model M of P, it is the case that $M \models F : \mu$.

Example 3 *Picking up from where we left off in Example* **??**, *let us define a simple TP-world which would satisfy a few rules from that example. Consider the following TP-world M which consists of two tp-threads th_1 and th_2.*

The threads are defined as follows:

th_1(sent(letter,paris)) $= \{1, 3, 7\}$
$th_1(textsfexpress\_mail(letter, paris)) = \{1, 7\}$
th_1(arrived(letter, paris)) $= \{3, 6, 8\}$

th_1(sent(letter,paris)) $= \{1\}$
th_1(express_mail(letter,paris)) $= \{1\}$
th_1(arrived(letter, paris)) $= \{4\}$

The probability distribution p is: $p(th_1) = 0.6$ and $p(th_2) = 0.4$.

The first thread states that three letters had been sent to Paris on dates 1, 3 and 7, two of them (on dates 3 and 7) had been sent via express mail, and that the letters arrived to Paris on dates 3, 6 and 8 respectively. The second thread has information that only one letter to Paris had been sent (on date 1), it was sent via express-mail and arrived to Paris on date 4.

We can now find that the probability that the letter arrived to Paris on date 3 is 0.6 according to M and the probability that it arrived on date 4 is 0.4.

3.2 Fixpoint Semantics

We now provide a fixpoint procedure to compute the *semantics* of tp-programs. The fixpoint procedure maps certain structures called tp-interpretations to tp-interpretations — in order to define tp-interpretations, we need to define some intermediate structures called tp-piles and tp-sets below.

Definition 18 *Let F be a ground event atom, $t \in \tau$ be a timepoint and $[l,u] \subseteq [0,1]$. Then the quadruple (F,t,l,u) is called a* tp-tuple.

A collection (multiset) of tp-tuples is called a TP-pile. *If R is a TP-pile we will use $R[F,t]$ to denote the set $\{(F,t,\_,\_) \in R\}$. If for each pair F,t the size of $R[F,t]$ is at most 1, then we call R a* TP-set.

Intuitively, each tp-tuple contains information about the probability of an event associated with F (which can be compound) at timepoint t. A TP-pile is an arbitrary collection of such information. In a TP-pile there may be two or more tp-tuples that have information about the probability of some event F at some time t.

A TP-set R is complete iff $(\forall F)(\forall t)(R[F,t] \neq \emptyset)$. On non-complete TP-sets we define a *completion* operation *compl* as follows: $compl(R) = R \cup \{(F,t,0,1) | R[F,t] = \emptyset\}$. Clearly for each TP-set R there is a uniquely defined completion of it. **Therefore without loss of generality from now on we will consider only *complete TP-sets*.** On such sets we define the satisfaction as follows:

Definition 19 *Let R be a* TP-set *and $F : [c,l,u,\omega_c]$ be an event atom where $c = c(y, y_1, \ldots, y_k)$ is a temporal constraint.*

- *R satisfies $F : [c,l,u,\omega_c]$ ($R \models F : [c,l,u,\omega_c]$)* **iff** *for all $\bar{a} = (a_1, \ldots, a_n) \in \tau^k$ such that $sol(c)(\bar{a}) \neq \emptyset$ and for all $t \in sol(c)(\bar{a})$ there exists such interval $[l_t, u_t]$ that*
  - $(F,t,l_t,u_t) \in R$
  - $[l_t,u_t] \subseteq [\omega_c(\bar{a},t) \cdot l, \omega_c(\bar{a},t) \cdot u]$
- $R \models F_1 : [c_1,l_1,u_1,\omega_{c_1}] \wedge \ldots \wedge F_n : [c_n,l_n,u_n,\omega_{c_n}]$ **iff** $(\forall i \in \{1, \ldots n\}) R \models F_i : [c_i,l_i,u_i,\omega_{c_i}]$
- $R \models F : [c,l,u,\omega] \longleftarrow F_1 : [c_1,l_1,u_1,\omega_1] \wedge \ldots \wedge F_n : [c_n,l_n,u_n,\omega_n]$ **iff** $R \models F : [c,l,u,\omega_c]$ *or* $F \not\models F_1 : [c_1,l_1,u_1,\omega_{c_1}] \wedge \ldots \wedge F_n : [c_n,l_n,u_n,\omega_{c_n}]$
- *R satisfies a tp-program P ($R \models P$)* **iff** *R satisfies every clause in P.*

A TP-interpretation is a TP-set that satisfies certain simple axioms. It will turn out that these axioms are exactly "right" from the point of view of making our fixpoint procedure compute the notion of logical consequence associated with tp-programs.

Definition 20 *A TP-set R is called a TP-interpretation* **iff** *the following conditions hold:*

- Conjunctive ignorance.
  $(\forall F = A_1 \wedge \ldots \wedge A_k, G = B_1 \wedge \ldots \wedge B_m)(\forall t \in \tau)((F,t,l_1,u_1) \in R \wedge (G,t,l_2,u_2) \in R) \Rightarrow (R \models (F \wedge G) : [t, \max(0, l_1 + l_2 - 1), min(u_1,u_2), \sharp])$
- Disjunctive ignorance.
  $(\forall F = A_1 \vee \ldots \vee A_k, G = B_1 \vee \ldots \vee B_m)(\forall t \in \tau)((F,t,l_1,u_1) \in R \wedge (G,t,l_2,u_2) \in R) \Rightarrow (R \models (F \vee G) : [t, max(l_1,l_2), min(1, u_1 + u_2), \sharp])$

The intervals $[\max(0, l_1 + l_2 - 1), min(u_1,u_2)]$ and $[max(l_1,l_2), min(1, u_1 + u_2)]$ represent the intervals in which the probability of respectively conjunction and disjunction of two events with probability intervals $[l_1,u_1]$ and $[l_2,u_2]$ will lie, if *nothing is known about the relationship between the events.*

In order to define a fixpoint operator, we first define an intermediate operator T_P^* which applied to a TP-interpretation produces a TP-pile. The operator T_P corrects the result of T_P^* to make it a $TP{-}interpretation$. In the definition of T_P^* we will employ the $\oplus$ notation defined below to "split" a compound event atom.

Definition 21 *Let* $F = F_1 * \ldots * F_n$, $G = G_1 * \ldots * G_k$ $(k > 0)$, $H = H_1 * \ldots * H_m$ $(m > 0)$ *where* $* \in \{\wedge, \vee\}$. *We write* $G \oplus H = F$ **iff:**
(a) $\{G_1, \ldots, G_k\} \cup \{H_1, \ldots, H_m\} = \{F_1, \ldots, F_n\}$ *and*
(b) $\{G_1, \ldots, G_k\} \cap \{H_1, \ldots, H_m\} = \emptyset$.

This definition may now be used to define the operator T_P^* which takes TP-interpretations as input and produces TP-piles as output.

Definition 22 *Let* P *be a tp-program.*

- *Let* $F = A$ *be an atom of* B_L. *Let* $t \in \tau$
  $T_P^*(R)(A, t) = \{(A, t, l, u) | (\exists r \in ground(P))$
  $(r = A : [c, l', u', \omega_c] \longleftarrow Body; R \models Body;$
  $t \in sol(c)$ and $[l, u] = [\omega_c(t) \cdot l', \omega_c(t) \cdot u'])\}$
- *Let* $F = A_1 \wedge \ldots \wedge A_k$.
  $T_P^*(R)(F, t) = \{(F, t, l, u) | (\exists H, G)$
  $(F \equiv H \oplus G; (H, t, l_1, u_1), (G, t, l_2, u_2) \in R$
  and $[l, u] = [\max(0, l_1 + l_2 - 1), \min(u_1, u_2)])\}$
- *Let* $F = A_1 \vee \ldots \vee A_k$.
  $T_P^*(R)(F, t) = \{(F, t, l, u) | (\exists H, G)$
  $(F \equiv H \oplus G; (H, t, l_1, u_1), (G, t, l_2, u_2) \in R$
  and $[l, u] = [\max(l_1, l_2), \min(1, u_1 + u_2)])\}$

We may now extend the definition of T_P^* to map tp-interpretations to tp-interpretations.

Definition 23 *Let* P *be a tp-program. We define operator* T_P *as follows:* $T_P(R)(F, t) = (F, t, l, u)$, *where* $[l, u] = \cap\{[l', u'] | (F, t, l', u') \in T_P^*(R)(F, t)\}$.

In order to describe the fixpoint procedure based on the operators described above, we need to introduce an ordering on TP-interpretations, and prove that T_P operator is monotonic w.r.t. this order.

Definition 24 *Let* R *and* S *be two TP-sets. We say that* $R \leq S$ **iff** *for every* F *and* $t \in \tau$ *if* $(F, t, l, u) \in R$ *and* $(F, t, l', u') \in S$ *then* $[l, u] \supseteq [l', u']$.

The following statement shows that the set $\mathcal{R}_{B_L}$ of all TP-interpretations for the formulas constructed out of atoms of B_L forms a complete lattice.

Lemma 1 $\langle \mathcal{R}_{B_L}, \leq \rangle$ *is a complete lattice.*

We can now show that our T_P operator is monotonic and proceed with the fixpoint procedure.

Theorem 1 *Let* $R \leq S$ *and let* P *be a TP-program. Then* $T_P(R) \leq T_P(S)$.

The following definition specifies how we might iteratively apply the T_P operator.

Definition 25 *Let* P *be a tp-program.*

- $T_P^0 = \bot = compl(\emptyset)$
- $T_P^{i+1} = T_P(T_P^i)$
- $T_P^\lambda = \cap_{i \leq \lambda}(T_P^i)$

The following theorem is important. It shows that T_P has a least fixpoint, $lfp(T_P)$, and that this least fixpoint precisely captures the model-theoretic notion of logical consequence associated with tp-programs. Thus, iterative application of the T_P operator yields all ground event atoms that are logical consequences of P.

Theorem 2 *1.* $R \models P$ **iff** $T_P(R) \leq R$

2. $lfp(T_P) \models P$

3. $(\forall F)(P \models F : [c, l, u, \omega_c]$, *where* $c = c(y, y_1, \ldots y_k)$ **iff** $(\forall \bar{a} \in \tau^k)(sol(c)(\bar{a}) \neq \emptyset) \Rightarrow (\forall t \in sol(c)(a))(\exists l_t, u_t)((F, t, l_t, u_t) \in lfp(T_P) \wedge [l_t, u_t] \subseteq [\omega_c(\bar{a}, t) \cdot l, \omega_c(\bar{a}, t) \cdot u])))$.

4 Fixpoint Computation and Entailment Problem (Ground Case)

In this section, we develop algorithms and associated complexity results for computation of the fixpoint of a ground tp-program, checking its consistency, and query answering. To simplify time bounds we assume that such elementary problems as checking "$t \in sol(c)$" and computing $\omega_c(\bar{a}, t)$ can be done in constant time[1] , and enumeration of $sol(c)$ can be done in time linear w.r.t. $|sol(c)|$.

For any TP-set (TP-interpretation) R, let $atoms(R) = \{(A, t, l, u) \in R, A \textit{ is a simple event atom}\}$. Let P be a ground tp-program over the set of ground simple event atoms $\{A_1, ..., A_N\}$ and the calendar $\tau = [0, t_{\max}]$. Let us denote $lfp(T_P)$ by R_P. The following lemma shows that R_P can be easily defined by $atoms(R_P)$..

Lemma 2 *For all* $t \in \tau$ *and all ground compound event atoms* F

- *if* $F = A_1 \wedge \ldots \wedge A_k$ *and for each* $1 \leq i \leq k \quad (A_i, t, l_i, u_i) \in atoms(R_P)$ *then* $(F, t, l, u) \in R_P$, *for* $l = l_1 + \ldots + l_k + 1 - k$ *and* $u = \min\{u_1, \ldots, u_k\}$;
- *if* $F = A_1 \vee \ldots \vee A_k$. *and for each* $1 \leq i \leq k \quad (A_i, t, l_i, u_i) \in atoms(R_P)$ *then* $(F, t, l, u) \in R_P$, *for* $l = \max\{l_1, \ldots, l_k\}$ *and* $u = \min\{1, u_1 + \ldots + u_k\}$.

This lemma allows to propose an efficient algorithm to construct $atoms(R_P)$ for the ground case. This algorithm will also check the consistency of tp-programs. Let P be a ground tp-program over the set $B_L = \{A_1, ..., A_N\}$ and let $\{F_1, ..., F_s\}$ be the set of all compound event atoms included in clauses of P.

Algorithm LFP-atoms.
Input: a ground tp-program P.
Output: $atoms(R_P)$.

BEGIN (algorithm)
1. $OLD := \emptyset$; $NEW := \emptyset$;
2. FOR $i = 1$ TO N DO
FOR EACH $alpha \in \tau$ DO
$NEW := NEW \cup \{(A_i, t, 0, 1)\}$;
3. FOR $j = 1$ TO s DO
FOR EACH $t \in \tau$ DO
$NEW := NEW \cup \{(F_j, t, 0, 1)\}$;
4. WHILE $OLD \neq NEW$ DO
BEGIN
$OLD := NEW$;

[1] We do this to make our complexity results independent of the issues such as the implementation of the ω_c functions. We can then factor the complexity of computing $\omega(\bar{a}, t)$ back into our complexity estimate if needed. It should be noted that for ground programs on which we are concentrating in this section the assumption that $\omega_c(t)$ can be computed in constant time is not all that unreasonable.

5. FOR EACH $r = A : [c, l, u, \omega_c] \leftarrow$ *Body* $\in P$ such that *OLD* $\models$ *Body* DO
6. FOR EACH $t \in sol(c)$ DO
BEGIN
LET $(A, t, l', u') \in NEW$;
$l_1 := \max(l', \omega_c(t) * l)$;
$u_1 := \min(u', \omega_c(t) * u)$;
IF $u_1 < l_1$ THEN RETURN $\emptyset$;
$NEW := (NEW \setminus \{(A, t, l, u)\}) \cup \{(A, t, l_1, u_1)\}$;
$P := P \setminus \{r\}$
END;
7. FOR $j = 1$ TO s DO
8. FOR EACH $t \in \tau$ DO
BEGIN
LET $F_j = B_1^j * \ldots * B_k^j$ AND FOR EACH $1 \le p \le k$ $(B_p^j, t, l_p^j, u_p^j) \in NEW$
IF $* = \wedge$ THEN BEGIN
$l := l_1^j + \ldots + l_k^j + 1 - k$; $u := \min\{u_1^j, \ldots, u_k^j\}$ END
ELSE BEGIN $\{* = \vee\}$
$l := \max\{l_1^j, \ldots, l_k^j\}$; $u := \min\{1, u_1^j + \ldots + u_k^j\}$ END;
$NEW := (NEW \setminus \{(F_j, t, \_, \_)\}) \cup \{(F_j, t, l, u)\}$
END;
END;
9. RETURN $atoms(NEW)$
END.

The following theorem shows that algorithm **LFP-atoms** is a correct way of checking P for inconsistency and (if P is consistent) of computing the least fixpoint of T_P for simple event atoms. It also establishes the polynomial time complexity of the algorithm. The proof of this result uses lemma 2.

Theorem 3 *Let P be any ground tp-program and let P contain m clauses. Let $|P|$ be the size of P (under some standard encoding). Then* **Algorithm LFP-atoms** *returns $\emptyset$ iff P is inconsistent. If P is consistent then it computes $atoms(R_P)$ in time $O(m \cdot |P| \cdot t_{max})$.*

Now we consider the entailment problem: given a tp-program P and a ground query G, check whether $P \models G$. Our queries will be conjunctions of annotated basic formulas of the form $F : [c, l, u, \omega_c]$ where F is ground. One way to answer such queries is to add F to the list of compound events $\{F_1, ..., F_s\}$ of P run start algorithm **LFP-atoms**, and after the algorithm finishes, check the values (F, t, l, u) for all $t \in sol(c)$. A better way is to use algorithm **LFP-atoms** one time as a preprocessor to obtain $atoms(R_P)$, and then to answer to queries using the following simple algorithm.

Algorithm Simple query.
Input: set $atoms(R_P)$ and a simple query $F : [c, l, u, \omega_c]$.
Output: "YES" if $P \models F : [c, l, u, \omega_c]$, otherwise "NO".

BEGIN (algorithm)
LET $F = B_1 * \ldots * B_k$
FOR EACH $t \in sol(c)$ DO
BEGIN LET FOR EACH $1 \le p \le k$ $(B_p, t, l_p, u_p) \in atoms(R_P)$;
IF $* = \wedge$ THEN BEGIN
$L := l_1 + \ldots + l_k + 1 - k$; $U := \min\{u_1, \ldots, u_k\}$ END
ELSE BEGIN $\{* = \vee\}$
$L := \max\{l_1, \ldots, l_k\}$; $U := \min\{1, u_1 + \ldots + u_k\}$ END;
IF $(L < l * \omega_c(t))$ OR $(M > u * \omega_c(t))$
THEN RETURN "NO"
END
RETURN "YES"
END.

The following theorem shows that after polynomial time preprocessing the entailment problem can be solved in linear time.

Theorem 4 *(1) Algorithm* **Simple query** *gives a correct answer to any simple query of the form* $F : [c, l, u, \omega_c]$ *in time* $O(|F| + |sol(c)|)$.

(2) There exists an algorithm which given the set $atoms(R_P)$ *answers to any query* G *of the form* $F_1 : [c_1, l_1, u_1, \omega_1] \wedge ... \wedge F_n[c_n, l_n, u_n, \omega_n]$ *in time linear of* $(|G| + \sum_{i=1}^{n} |sol(c_i)|\ )$.

5 Proof Procedure(s)

In this section, we present two basic proof procedures in this section – the first is inefficient, but easy to understand, while the second is more efficient but more complex to present.

5.1 Unification

If $*$ is either $\wedge$ or $\vee$, then we say that two event atoms $A_1 * \ldots * A_k$ and $B_1 * \ldots * B_m$ are unifiable iff there is a substitution θ such that $\{A_1\theta, \ldots, A_k\theta\} = \{B_1\theta, \ldots, B_m\theta\}$. This notion of unification was introduced in [23] and it was proved that though most general unifiers are not necessarily unique, a corresponding notion of a *maximally* general unifier (max-gu) exists. For space reasons, we do not go into details of this here.

5.2 A Basic (Inefficient) Proof Procedure

In this section, we describe how we can expand a tp-program P into a program called its *closure*. This closure may then be used to define a resolution based proof procedure.

Definition 26 (Explosions) *Let* $F : [c, l, u, \omega_c]$ *be a ground formula. The* explosion *of* $F : [c, l, u, \omega_c]$, *denoted* $\mathcal{E}(F : [c, l, u, \omega_c])$ *is defined as* $\{F : [t, \omega_c(t) \cdot l, \omega_c(t) \cdot u, \sharp] | t \in sol(c)\}$. *The* exploded basic formula *for* $F : [c, l, u, \omega_c]$, *denoted* $\mathcal{E}_{\mathcal{F}}(F : [c, l, u, \omega_c])$, *is* $\bigwedge_{t \in sol(c)} F : [t, \omega_c(t) \cdot l, \omega_c(t) \cdot u, \sharp]$.

Let $r = F : [c, l, u, \omega_c] \longleftarrow Body$ *be a tp-clause. Then the* explosion *of* r, *denoted* $\mathcal{E}(r)$ *is defined as* $\{F : [t, \omega_c(c, t) \cdot l, \omega_c(c, t) \cdot u, \sharp] \longleftarrow Body | t \in sol(c)\}$.

If P *is a tp-program, then the* explosion *of* P, *denoted* $\mathcal{E}(P)$ *is defined as* $\{\mathcal{E}(r) | r \in P\}$

The explosion operation explicitly enumerates the probabilities of an event at all time points associated with a constraint. (As this set of time points can potentially be very large, we use the term explosion to describe this operation). Using this, we can now define the timepoint-based closure of program P.

Definition 27 (Timepoint-based Closure) *Let* P *be a tp-program.*

- $REDUN(P) = \mathcal{E}(P) \cup \{F : [t, 0, 1, \sharp] \longleftarrow | F \in B_L; t \in \tau\}$
- *We define the closure of* P *(denoted* $TCL(P)$*) iteratively:*

1. $TCL^0(P) = REDUN(P)$
2. *To construct* $TCL^{i+1}(P)$ *given* $TCL^i(P)$ *we have to apply the following rule to* all possible pairs of clauses *from* $TCL^i(P)$ *:*
   - Clarification rule:
     If clauses $F : [t, l, u, \sharp] \longleftarrow Body$ *and* $F' : [t, l', u', \sharp] \longleftarrow Body'$ *are in* $TCL^i(P)$ *and* F *and* F' *are unifiable via max-gu* Θ, *add the clause*
     $)F : [t, \max(l, l'), \min(u, u'), \sharp] \longleftarrow Body \wedge Body')\Theta$ *to* $TCL^{i+1}(P)$.
   - $\wedge$-composition rule:
     If clauses $F : [t, l, u, \sharp] \longleftarrow Body$ *and* $F' : [t, l', u', \sharp] \longleftarrow Body'$ *are in* $TCL^i(P)$, $F = A_1 \wedge \ldots \wedge A_k$ *and* $G = B_1 \wedge \ldots \wedge B_m$ $(k, m \geq 1)$, *add the clause*
     $(F \wedge G) : [t, \max(0, l + l' - 1), \min(u, u'), \sharp] \longleftarrow Body \wedge Body'$ *to* $TCL^{i+1}(P)$.

- $\vee$-composition rule:
If clauses $F : [t, l, u, \sharp] \longleftarrow Body$ *and* $F' : [t, l', u', \sharp] \longleftarrow Body'$ *are in* $TCL^i(P)$, $F = A_1 \vee \ldots \vee A_k$ *and* $G = B_1 \vee \ldots \vee B_m$ $(k, m \geq 1)$*, add the clause*
$(F \vee G) : [t, \max(l, l'), \min(u + u', 1), \sharp] \longleftarrow Body \wedge Body'$ *to* $TCL^{i+1}(P)$.

3. $TCL(P) = \cup_{i \geq 0} TCL^i(P)$.

The following result says that TCL(P) is sound w.r.t. P.

Lemma 3 *For every clause* $r \in TCL(P)$, $P \models r$.

Notice that the syntax of TCL(P) is somewhat different from the syntax of P, since we allow the rules in TCL(P) to have non-atomic heads. However this extension is supported by our definitions of satisfaction on TP-models (def. 17) and TP-sets (def. 19). We are now ready to proceed with the resolution procedure.

Definition 28 *A* tp-query *is an expression of the form* $\exists(H_1 : [c_1, l_1, u_1] \wedge \ldots \wedge H_n : [c_n, l_n, u_n])$.
The explosion *of a tp-query* Q*, denoted* $\mathcal{E_F}(Q)$ *is an expression of the form:* $\bigwedge_{i=1}^{n} \mathcal{E_F}(H_i : [c, l_i, u_i, \omega_c])$

In other words, the explosion of a tp-query is a conjunction of the explosions of all basic formulas in the query.

Definition 29 *Let* Q *be a query and let* $\mathcal{E_F}(Q) = F_1 : [t_1, l_1, u_1, \sharp] \wedge \ldots F_n : [t_n, l_n, u_n, \sharp]$. *Let* $r \equiv G : [t, l, u, \sharp] \longleftarrow G_1 : \mu_1 \wedge \ldots G_k : \mu_k \in TCL(P)$ *and* G *be unifiable with* F_i *via max-gu* Θ. *Then*

$$\exists((F_1 : [t_1, l_1, u_1, \sharp] \wedge \ldots \wedge F_{i-1} : [t_{i-1}, l_{i-1}, u_{i-1}, \sharp] \wedge \mathcal{E_F}(G_1 : \mu_k) \wedge \ldots \wedge \mathcal{E_F}(G_k : \mu_k) \wedge$$
$$F_{i+1} : [t_{i+1}, l_{i+1}, u_{i+1}, \sharp] \wedge \ldots \wedge F_n : [t_n, l_n, u_n, \sharp])\Theta)$$

is a tp-resolvent *of* r *and* Q **iff** $[l, u] \subseteq [l_i, u_i]$.

Definition 30 Let $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ be an initial query, and P a tp-program. A tp-deduction of Q from P is a sequence $< Q_1, r_1, \Theta_1 > \ldots < Q_s, r_s, \Theta_s > \ldots$ where, $Q_1 = \mathcal{E_F}(Q)$, for all $i \geq 1$, r_i is a renamed version of a clause in TCL(P), and Q_{i+1} is a *tp-resolvent* of Q_i and r_i via max-gu Θ_i.
A tp-refutation of Q from P is a *finite tp-deduction* $< Q_1, r_1, \Theta_1 > \ldots < Q_s, r_s, \Theta_s >$ where, the tp-resolvent of Q_s and r_s via Θ_s is the empty query. $\Theta_1 \ldots \Theta_r$ is called the *computed answer substitution.*

The following two theorems state that our first proof procedure is sound and complete.

Theorem 5 *[Soundness of tp-refutation]*
Let P *be a tp-program, and* Q *be an initial query. If there exists a tp-refutation of* $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ *from* P *with the answer substitution* Θ *then* $P \models \forall((F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)\Theta)$.

Theorem 6 *[Completeness of tp-refutation]*
Let P *be a consistent tp-program and* Q *be a query. Then, if* $P \models Q$ *then there exists a tp-refutation of* Q' *from* P.

5.3 A More Efficient Temporal Constraint-based Proof Procedure

It is clear that the above proof procedure is highly inefficient because the *explosion* operation is applied to the entire program and entire query, regardless of any specific aspects of either that may allow for optimization. A better procedure would allow us to keep as much information as possible in a "compressed" state, and would allow resolution of such "compressed" formulas. We start by introducing two extensions of the tp-program syntax. We will use expressions in one format of the extended syntax to store the new closure of a tp-program. The other extended syntax format will be used during the resolution procedure to keep track of the parts of the query which had already been "established" and the parts which need to be checked.

First we need to give one definition.

Definition 31 *Let $c = c(y, y_1, \ldots, y_k)$ and $d = d(y, y_1, \ldots y_k)$ be two temporal constraints. We say that $sol(d) \subset sol(c)$* ***iff*** *for all $(a_1, \ldots a_k) \in \tau^k$ $sol(d)(a_1, \ldots, a_k) \subseteq sol(c)(a_1, \ldots, a_k)$.*

Definition 32 *Let $c = c(y, y_1, \ldots, y_k)$ and $d = d(y, y_1, \ldots y_k)$ be two temporal constraints such that $sol(d) \subseteq sol(c)$. Then we say that an expression of the form $F : [c, d, l, u, \omega_c]$ is in* extended query syntax.
For an annotated basic formula $F : [c, l, u, \omega_c]$ let $Ext_{\mathcal{Q}}(F : [c, l, u, \omega_c]) = F : [c, c, l, u, \omega_c]$.
We also define $Ext_{\mathcal{Q}}(F_1 : [c_1, l_1, u_1, \omega_{c_1}] \wedge \ldots \wedge F_n : [c_n, l_n, u_n, \omega_{c_n}]) = \bigwedge_{i=1}^{n} Ext_{\mathcal{Q}}(F_i : [c_i, l_i, u_i, \omega_{c_i}])$.

Let ω_c^l and ω_c^u be generalized weight functions. Then we say that an expression of the form $F : [c, l, u, \omega_c^l, \omega_c^u]$ is in extended program syntax. *For annotated basic formula $F : [c, l, u, \omega_c]$ let $Ext_{\mathcal{P}}(F : [c, l, u, \omega_c]) = F : [c, l, u, \omega_c, \omega_c]$. If $r = F : [c, l, u, \omega_c] \longleftarrow Body$ is a tp-clause, then we define $Ext_{\mathcal{P}}(r) = Ext_{\mathcal{P}}(F : [c, l, u, \omega_c]) \longleftarrow Body$.*

Expressions in *extended query syntax* have an extra temporal constraint in them. Throughout the query resolution process, the temporal constraint c of a query in extended syntax will not change and will be equal to the initial constraint. Constraint d in the meantime will contain information about the timepoints in $sol(c)$ for which we still need to establish probability ranges. Initially $d = c$, as when the query is asked we have no information about any timepoints. In our query resolution procedure we will make sure that whenever a formula is added to the query it is added in the extended query syntax format.

Expressions in *extended program syntax* have two distribution functions. In an annotated formula $F : [c, l, u, \omega_c]$, ω_c describes how both lower and upper bounds of the probability interval $[l, u]$ are distributed among the elements of $sol(c)$. However, as the resolution procedure described below proceeds, we may wind up in a situation when for for a newly created temporal constraint the lower and the upper bounds of the probability intervals computed are distributed differently. The *extended program syntax* format is used for the purpose of keeping track of both distributions. As our resolution process will be described in terms of resolution against a closure of a program P, the clauses of the closure of P will be in the extended program syntax format. *Note that these extended syntaxes are used by a query processing procedure and they will not be visible to users, nor will they change either the semantics of tp-programs.* We can now define the temporal constraint-based closure of program P.

Definition 33 (Temp. Constraint-based Closure) *Let P be a tp-program.*

- $REDUN_{Ext}(P) = \{Ext_{\mathcal{P}}(r) | r \in P\} \cup$
  $\{F : [c^\star, 0, 1, \top, \top] \longleftarrow | F \in B_L;$
  $c^\star = \neg(\bigvee_{F:[c,l,u,\omega_c] \longleftarrow \in P} c)\}$
- *We define the* temporal constraint-based closure *of P (denoted $CCL(P)$) iteratively:*

1. $CCL^0(P) = REDUN_{Ext}(P)$
2. *To construct $CCL^{i+1}(P)$ given $CCL^i(P)$ we have to apply the following rule to* all possible pairs of clauses *from $CCL^i(P)$:*

   - Clarification rule:
   If clauses $F : [c, l, u, \omega_c^l, \omega_c^u] \longleftarrow Body$ and $F' : [c', l', u', \omega_{c'}^{\prime l}, \omega_{c'}^{\prime u}] \longleftarrow Body'$ are in $CCL^i(P)$, F and F' are unifiable via max-gu Θ, and $sol(c \wedge c') \neq \emptyset$ add to $CCL^{i+1}(P)$ the clause $(F : [c^\star, l^\star, u^\star, \omega_{c^\star}^{\star l}, \omega_{c^\star}^{\star u}] \longleftarrow Body \wedge Body')\Theta$, where $c^\star = c \wedge c'$,

$l^\star = \sum_{t \in sol(c \wedge c')} \max(\omega_c^l(t) \cdot l, \omega_{c'}^{\prime l}(t) \cdot l')$	$u^\star = \sum_{t \in sol(c \wedge c')} \min(\omega_c^u(t) \cdot u, \omega_{c'}^{\prime u}(t) \cdot u')$
$\omega_{c^\star}^{\star l}(t) = \frac{\max(\omega_c^l(t) \cdot l, \omega_{c'}^{\prime l}(t) \cdot l')}{l^\star}$; if $l^\star \neq 0$	$\omega_{c^\star}^{\star u}(t) = \frac{\min(\omega_c^u(t) \cdot u, \omega_{c'}^{\prime u}(t) \cdot u')}{u^\star}$; if $u^\star \neq 0$
and $\omega_{c^\star}^{\star u}(t) = 0$ otherwise; $t \in sol(c \wedge c')$	and $\omega_{c^\star}^{\star u}(t) = 0$ otherwise; $t \in sol(c \wedge c')$

   - $\wedge$-composition rule:
   If clauses $F : [c, l, u, \omega_c^l, \omega_c^u] \longleftarrow Body$ and $G : [c', l', u', \omega_c^{\prime l}, \omega_{c'}^{\prime u}] \longleftarrow Body'$ are in $CCL^i(P)$, $F = A_1 \wedge \ldots \wedge A_k$ and $G = B_1 \wedge \ldots \wedge B_m$ and $sol(c \wedge c') \neq \emptyset$, then add to $CCL^{i+1}(P)$ the clause $(F \wedge G) : [c^\star, l^\star, u^\star, \omega_{c^\star}^{\star}, \omega_{c^\star}^{\star u}] \longleftarrow Body \wedge Body'$, where $c^\star = c \wedge c'$ and

$l^\star = \sum_{t\in sol(c\wedge c')} \max(\omega_c^l(t)\cdot l + \omega'^l_{c'}(t)\cdot l' - 1, 0)$	$u^\star = \sum_{t\in sol(c\wedge c')} \min(\omega_c^u(t)\cdot u, \omega'^u_{c'}(t)\cdot u')]$
$\omega_{c^\star}^{\star l}(t) = \frac{\max(\omega_c^l(t)\cdot l+\omega'^l_{c'}(t)\cdot l'-1,0)}{l^\star}$; if $l^\star \neq 0$	$\omega_{c^\star}^{\star u}(t) = \frac{\min(\omega_c^u(t)\cdot u,\omega'^u_{c'}(t)\cdot u')}{u^\star}$; if $u^\star \neq 0$
and $\omega_{c^\star}^{\star l}(t) = 0$ otherwise; $t \in sol(c\wedge c')$	and $\omega_{c^\star}^{\star u}(t)(t) = 0$ otherwise; $t \in sol(c\wedge c')$

- $\vee$-composition rule:
If clauses $F : [c, l, u, \omega_c^l, \omega_c^u] \longleftarrow Body$ *and* $G : [c', l', u', \omega'^l_{c'}, \omega'^u_{c'}] \longleftarrow Body'$ *are in* $CCL^i(P)$, $F = A_1 \vee \ldots \vee A_k$ *and* $G = B_1 \vee \ldots \vee B_m$ *and* $sol(c\wedge c') \neq \emptyset$, *then add to* $CCL^{i+1}(P)$ *the clause* $(F \vee G) : [c^\star, l^\star, u^\star, \omega_l^\star, \omega_u^\star] \longleftarrow Body \wedge Body'$, *where* $c^\star = c\wedge c'$ *and*

$l^\star = \sum_{t\in sol(c\wedge c')} \max(\omega_c^l(t)\cdot l, \omega'^l_{c'}(t)\cdot l')$	$u^\star = \sum_{t\in sol(c\wedge c')} \min(\omega_c^u(t)\cdot u + \omega'^u_{c'}(t)\cdot u', 1)]$
$\omega_{c^\star}^{\star l}(t) = \frac{\max(\omega_c^l(t)\cdot l,\omega'^l_{c'}(t)\cdot l')}{l^\star}$;if $l^\star \neq 0$	$\omega_{c^\star}^{\star u}(t) = \frac{\min(\omega_c^u(t)\cdot u+\omega'^u_{c'}(t)\cdot u',1)}{u^\star}$; if $u^\star \neq 0$
and $\omega_l^\star(t) = 0$ otherwise; $t \in sol(c\wedge c')$	and $\omega_{c^\star}^{\star u}(t) = 0$ otherwise; $t \in sol(c\wedge c')$

3. $CCL(P) = \cup_{i\geq 0} CCL^i(P)$.

The following result says that CCL(P) is sound w.r.t. P.

Lemma 4 *For every clause* $r \in CCL(P)$, $P \models r$.

Our resolution procedure will convert the initial query into the extended query syntax. "Chunks" of each conjunct of the query can be resolved against the appropriate rules of the CCL(P). If a query conjunct is of a form $F : [c, d, l, u, \omega_c]$, CCL$(P)$ contains a clause r with the head $F' : [c', l', u', \omega'_l, \omega'_u]$, F and F' are unifiable and $d\wedge c'$ has a non-empty set of solutions, then if we can prove the body of r then we can also extract some information about probability of F over the $sol(d\wedge c')$. To prove the initial query we will still need to prove $F : [c, d\wedge\neg c', l, u, \omega_c]$. This intuition is formalized in the definitions that follow.

Definition 34 *Let* Q *be a query in extended query format and let* $Q = F_1 : [c_1, d_1, l_1, u_1, \omega_c] \wedge \ldots F_n : [c_n, d_n, l_n, u_n, \omega_n]$. *Let* $r \equiv G : [c, l, u, \omega_l, \omega_u] \longleftarrow Body \in CCL(P)$ *and* G *be unifiable with* F_i *via max-gu* Θ. *Then* $\exists((F_1 : [c_1, d_1, l_1, u_1, \omega_1] \wedge \ldots \wedge F_i : [c_i, d_i\wedge\neg c, l_i, u_i, \omega_i] \wedge Ext_{\mathcal{Q}}(Body) \wedge F_{i+1} : [c_{i+1}, d_{i+1}, l_{i+1}, u_{i+1}, \omega_{i+1}]$ $\wedge \ldots \wedge F_n : [c_n, d_n, l_n, u_n, \omega_n])\Theta)$
is a ctp-resolvent *of* r *and* Q **iff**
$(1) sol(d_i\wedge c) \neq \emptyset$
$(2)(\forall t \in sol(d_i\wedge c))([\omega_i(c_i,t)\cdot l, \omega_i(c_i,t)\cdot u] \supseteq [\omega_c^l(c,t)\cdot l, \omega_c^u(c,t)\cdot u])$

Note that if $sol(d_i) = sol(c)$ *we will have* $sol(d_i\wedge\neg c) = \emptyset$. *In this case,* $F_i : [c_i, d_i\wedge\neg c, l_i, u_i, \omega_i]$ *should not be added to the resolvent, as its proof is finished.*

Definition 35 Let $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ be an initial query, and P a tp-program. A tp-deduction of Q from P is a sequence $< Q_1, r_1, \Theta_1 > \ldots < Q_s, r_s, \Theta_s > \ldots$ where, $Q_1 = \exists(Ext_{\mathcal{Q}}(Q)$, for all $i \geq 1$, r_i is a renamed version of a clause in CCL(P) and Q_{i+1} is a ctp-resolvent of Q_i and r_i via max-gu Θ_i.

n ctp-refutation of Q from P is a *finite ctp-deduction* $< Q_1, r_1, \Theta_1 > \ldots < Q_s, r_s, \Theta_s >$ where, the ctp-resolvent of Q_s and r_s via Θ_s is the empty query. $\Theta_1 \ldots \Theta_r$ is called the *computed answer substitution.*

Theorem 7 *[Soundness of ctp-refutation]*
Let P *be a tp-program, and* Q *be an initial query. If there exists a ctp-refutation of* $Q \equiv \exists(F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)$ *from* P *with the answer substitution* Θ *then* $P \models \forall((F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n)\Theta)$.

Theorem 8 *[Completeness of ctp-refutation]*
Let P *be a consistent tp-program and* Q *be a query. Then, if* $P \models Q$ *then there exists a tp-refutation of* Q' *from* P.

The second procedure is a major improvement on the first. First, while *tp-resolution* mandates the explosion of the entire program and all queries down to the timepoint level, our *ctp-resolution* avoids this. This makes the closure set significantly smaller. Second, during resolution, we may employ knowledge about specific distribution functions and probability intervals to avoid computing the probability intervals for every timepoint. This provides a wide variety of opportunities for the optimization of the implementation of this proof procedure. For example if we are combining two rules where the distributions in the heads are uniform, it is sufficient to compute the probability intervals only for a single timepoint from each temporal constraint, which saves *a lot* of computations. Similar optimizations can be a applied to some other distribution functions such as geometric or binomial.

In [2] the authors showed how closure-based proof procedure similar to the ones described here can be converted into a proof procedure for probabilistic logic programs based on a set of inference rules. A similar conversion for our proof procedures is straightforward: instead of constructing the closure using Clarification, and two types of Composition rules, we can apply these rules (together with explosion of extension operations) to the clauses of P dynamically to produce the next rule against which current query should be resolved. Due to space limitations we omit the formal description of these conversions.

6 Related Work

To date, there has been no work on temporal probabilistic logic programming that we are aware of — hence, we compare our work with work on probabilistic logic programming, and with logics of probability and time.

In addition to the authors' works, probabilistic logic programs were studied by Thone *et al.*[25], and Lakshmanan [17] who showed how different probabilistic dependencies can be encoded into logic programs. Kiessling's group [14, 25] and Lukasiewicz [21] made important contributions to bottom up computations of logic programs. The work reported in this paper may be viewed as an extension of the above works (as well as [23, 24, 2, 4]) to handle temporal-probabilistic information. In addition to the model theory, we have developed both bottom up fixpoint computation algorithms, as well as alternative proof procedures for TPLPs.

Lehmann and Shelah [19] and Hart and Sharir [11] were among the first to integrate time and probability. Kanazawa also studied the integration of probability and time with a view to developing efficient planning algorithms [12]. Their main interest is in how probabilities of facts and events change over time., Haddawy [10] develops a logic for reasoning about actions, probabilities, and time using an interval time model. Our framework is different from theirs in that (i) we allow arbitrary distributions in our syntax, (ii) we provide a fixpoint theory, (iii) we provide and manipulate constraint based representations of time, and (iv) we provide constraint based proof procedures and complexity results for the "Horn clause like" fragment of this logic.

Dubois and his colleagues [5] have studied the integration of uncertainty and time – they extend the well-known possibilistic logic theory to a "timed possibilistic logic." This logic associates, with each formula of possibilistic logic, a set of time points reflecting the times at which the formula has a given possibilistic truth value. However, this framework is not probabilistic.

Last but not least, our use of possible worlds models was inspired by the work of Fagin and Halpern ([7],[9]), in which similar in spirit models had been introduced for probabilistic nontemporal logics.

7 Conclusions

In this paper, we have defined temporal probabilistic (tp-) logic programs that allow us to reason about instantaneous events in a probabilistic environment, We have provided a formal syntax and model theory for tp-programs, developed a fixpoint theory that is equivalent to the model theory and developed sound and complete bottom up computation procedures for entailment. We also developed two sound and complete proof procedures for tp-programs, and showed that the second proof procedure is an improvement on the first. These proof procedures support a resolution based query processing procedure for tp-programs.

Acknowledgments

The work of the second author had been partially supported by Russian Fundamental Studies Foundation (Grants 97-01-00973). The other authors were supported by the Army Research Office under Grants DAAH-

04-95-10174, DAAH-04-96-10297, and DAAH04-96-1-0398, by the Army Research Laboratory under contract number DAAL01-97-K0135, by an NSF Young Investigator award IRI-93-57756, and by a TASC/DARPA grant J09301S98061.

References

[1] M. Baudinet. (1992)*A Simple Proof Of The Completeness Of Temporal Logic Programming*, in Intensional Logics and Programming (eds. L.G. del Cerro, M. Pentonen), pp 51-83, Clarendon Press, 1992.

[2] A. Dekhtyar and V.S. Subrahmanian. (1998) *Hybrid Probabilistic Logic Programs*, accepted to Journal of Logic Programming, Feb. 1999. Early version in Proc. 1997 Intl. Conf. on Logic Programming (ed. L. Naish), MIT Press.

[3] A. Dekhtyar, R.Ross and V.S. Subrahmanian. (1998) *Probabilistic Temporal Databases, I: Algebra*, available as University of Maryland tech report CS-TR-3987

[4] M. Dekhtyar, A. Dekhtyar and V.S. Subrahmanian. (1998) *Hybrid Probabilistic Programs: Algorithms and Complexity*, accepted to Uncertainty in AI'99, extended version available as University of Maryland tech. report CS-TR-3969.

[5] D. Dubois, J. Lang and H. Prade. (1991) *Timed Possibilistic Logic*, Fundamenta Informaticae, XV, pps 211–234.

[6] C. Dyreson and R. Snodgrass. (1998) *Supporting Valid-Time Indeterminacy*, ACM Transactions on Database Systems, Vol. 23, Nr. 1, pps 1—57.

[7] R Fagin, J.Halpern, Uncertainty, belief, and probability, Computational Intelligence 7, 1991, pp. 160-173

[8] R. Fagin,J. Halpern, N. Megiddo, A logic for reasoning about probabilities, Information and Computation 87:1,2, 1990, pp. 78-128

[9] R. Fagin, J. Halpern, Reasoning about knowledge and probability, Journal of the ACM 41:2, 1994, pp. 340-367

[10] P. Haddawy. (1991) *Representing Plans under Uncertainty: A Logic of Time, Chance and Action*, Ph.D. Thesis. Available as University of Illinois Tech. Report UIUCDCS-R-91-1719.

[11] S. Hart and M. Sharir. (1986) *Probabilistic propositional temporal logic*, Information and Control, 70:97–155.

[12] K. Kanazawa. (1991) *A Logic and Time Nets for Probabilistic Inference*, AAAI-91, pps 360–365.

[13] S. Kraus and D. Lehmann. (1988) *Knowledge, Belief and Time*, Theoretical Computer Science 58, pp 155-174.

[14] W. Kiessling, H. Thone and U. Guntzer. (1992) *Database Support for Problematic Knowledge*, Proc. EDBT-92, pps 421–436, Springer LNCS Vol. 580.

[15] V.S. Lakshmanan, N. Leone, R. Ross and V.S. Subrahmanian. ProbView: A Flexible Probabilistic Database System. ACM TRANSACTIONS ON DATABASE SYSTEMS, Vol. 22, Nr. 3, pps 419–469, Sep. 1997.

[16] V.S. Lakshmanan and F. Sadri. (1994) *Modeling Uncertainty in Deductive Databases*, Proc. Int. Conf. on Database Expert Systems and Applications, (DEXA'94), September 7-9, 1994, Athens, Greece, Lecture Notes in Computer Science, Vol. 856, Springer (1994), pp. 724-733.

[17] V.S. Lakshmanan and F. Sadri. (1994) *Probabilistic Deductive Databases*, Proc. Int. Logic Programming Symp., (ILPS'94), November 1994, Ithaca, NY, MIT Press.

[18] V.S. Lakshmanan and N. Shiri. (1997) *A Parametric Approach with Deductive Databases with Uncertainty*, accepted for publication in IEEE Transactions on Knowledge and Data Engineering.

[19] D. Lehmann and S. Shelah. (1982) *Reasoning about Time and Chance*, Information and Control, 53, pps 165–198.

[20] T. Lukasiewicz. (1998) *Probabilistic Logic Programming*, in Procs. 13th biennial European Conference on Artificial Intelligence, pps 388-392, Brighton, UK, August 1998.

[21] T. Lukasiewicz. (1998) *Magic Inference Rules for Probabilistic Deduction under Taxonomic Knowledge*, Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, pps 354-361, Madison, Wisconsin, USA, July 1998.

[22] J.W. Lloyd. (1987) *Foundations of Logic Programming*, Springer.

[23] R. Ng and V.S. Subrahmanian. (1993) Probabilistic Logic Programming, INFORMATION AND COMPUTATION, 101, 2, pps 150–201, 1993.

[24] R. Ng and V.S. Subrahmanian.(1993) A Semantical Framework for Supporting Subjective and Conditional Probabilities in Deductive Databases, JOURNAL OF AUTOMATED REASONING, 10, 2, pps 191–235, 1993.

[25] H. Thone, W. Kiessling and U. Guntzer. (1995) *On Cautious Probabilistic Inference and Default Detachment*, Annals of Operations Research, 55, pps 195–224.