Annotated Probabilistic Temporal Logic: Approximate Fixpoint Implementation

Paulo Shakarian, Gerardo I. Simari, V.S. Subrahmanian

Annotated Probabilistic Temporal (APT) logic programs support building applications where we wish to reason about statements of the form "Formula G becomes true with a probability in the range $[L, U]$ within (or in exactly) Δt time units after formula F became true." In this paper, we present a sound, but incomplete fixpoint operator that can be used to check consistency and entailment in APT logic programs. We present the first implementation of APT-logic programs and evaluate both its compute time and convergence on a suite of 23 ground APT-logic programs that were automatically learned from two real-world data sets. In both cases, the APT-logic programs contained up to 1,000 ground rules. In one data set, entailment problems were solved on average in under 0.1 seconds per ground rule, while in the other, it took up to 1.3 seconds per ground rule. Consistency was also checked in a reasonable amount of time. When discussing entailment of APT-logic formulas, convergence of the fixpoint operator refers to $(U - L)$ being below a certain threshold. We show that on virtually all of the 23 automatically generated APT-logic programs, convergence was quick — often in just 2-3 iterations of the fixpoint operator. Thus, our implementation is a practical first step towards checking consistency and entailment in temporal probabilistic logics without independence or Markovian assumptions.

Categories and Subject Descriptors: I.2.4 [**Knowledge Representation Formalisms and Methods**]: Temporal Logic; I.2.3 [**Deduction and Theorem Proving**]: Probabilistic Reasoning

General Terms: Algorithms, Languages

Additional Key Words and Phrases: Probabilistic and Temporal Reasoning, Threads, Frequency Functions, Imprecise Probabilities

1. INTRODUCTION

There are numerous applications where we need to make statements of the form "Formula G becomes true with $50 - 60\%$ probability 5 time units after formula F became true." Statements of this kind arise in a wide variety of application domains. For instance:

(1) When reasoning about **stock markets**, we might automatically learn rules (using machine learning algorithms) such as "There is a $70 - 80\%$ probability of a drop in the Dow Jones index within 3 days of the release of a report showing an increase in unemployment." An example of an APT-program for reasoning about stocks is shown in Figure 1.

(2) When we wish to reason about **environmental phenomena**, we might wish to express statements such as "There is a $20 - 30\%$ probability that over 1,000 birds will die within 3 days of an oil spill."

(3) Likewise, when reasoning about **medical treatments**, we might automatically learn rules from historical data saying things such as "There is a $0 - 5\%$ probability that a patient will experience vomiting within 5 days of beginning a course of drug D."

While logic programmers wrote such rules manually in the past, today we can automatically learn them from historical data, leading to a resurgence of work in probabilistic logic and probabilistic logic programs. The above rules mix time and probabilities. Temporal probabilistic logic programs (or tp-LPs) [Dekhtyar et al. 1999] provide a framework within which we can express rules of the form "If some condition is true, then some atom is true at some time (or time interval) with some probability distribution over the points in the time interval." [Dekhtyar et al. 1999] provided a syntax and semantics for tp-LPs and provided important complexity results. Heterogeneous temporal probabilistic agents (HTP-agents) [Dix et al. 2006] allow us to make statements about the permissions/obligations and forbidden actions that an agent may take at future times if some condition is true now or in the future. Though temporal probabilistic logics have been studied extensively for many years, there is less work on temporal probabilistic logic programs and, until this paper, there has not been a single prototype implementation and/or experimental result.

This paper builds upon recent work on Annotated Probabilistic Temporal (APT) logic programs [Shakarian et al. 2011] which defines a syntax and semantics for APT LPs, together with algorithms to check consistency and entailment of formula by an APT LP $\mathcal{K}$. Although sound and complete, the algorithms of Shakarian et al. [2011], are not practical for general problems. This paper takes a more practical approach. We develop a fixpoint operator for APT-logic that we prove to be sound. We can use this operator to correctly identify many inconsistent APT-programs – although we cannot guarantee a program to be consistent by this means. Additionally, this operator can infer probability ranges for queries, but we cannot guarantee that they are the tightest possible bounds. Most importantly, finding the fixpoint of this operator is efficient to compute. We also show that some of the techniques can also be adopted in a sound algorithm for non-ground APT-programs, where we only require a partial grounding.

We also implement both algorithms and perform experiments on two data sets — the well known Minorities at Risk Organization Behavior (MAROB) data set [Asal et al. 2008] that tracks behaviors of numerous terror groups, and another real-world data counter-insurgency data from the Institute for the Study of War [ISW 2008] (ISW). We used the algorithm APT-EXTRACT in [Shakarian et al. 2011] to automatically learn 23 APT-logic programs — *no bias exists in these APT-logic programs as no one manually wrote them.* We then conducted experiments using those APT-logic programs and entailment problems were solved on an average in under 0.1 seconds per ground rule, while in the other, it took up to 1.3 seconds per ground rule. Consistency was also checked in a reasonable amount of time. *To the best of our knowledge, ours is the first implementation of a system for reasoning simultaneously about logic, time, and probabilities without making independence or Markovian assumptions.* [Shakarian et al. 2011] has demonstrated that Markov Decision Processes (MDPs) can be expressed as APT-logic programs, but not vice-versa, and similar results were also derived for logics like PCTL [Aziz et al. 1995; Hansson and Jonsson 1994b] that do make independence assumptions.

The paper is organized as follows. Section 2 recalls the definition of APT LPs from [Shakarian et al. 2011] and add syntax for integrity constraints (ICs) as well as

probabilistic time formulas (ptf's) – a generalization of the "annotated formulas" from [Dekhtyar et al. 1999; Shakarian et al. 2011]. The material after Section 2 is new. Section 3 shows that consistency and entailment in APT-logic are NP-complete and coNP-complete, respectively, improving on a previous hardness result [Shakarian et al. 2011]. Section 4 describes our approximate fixpoint algorithm which is based on a sound (but not complete) fixpoint operator. The operator works by syntactically manipulating the rules in the APT-program to iteratively tighten the probability bounds of the formula whose entailment is being checked. We adapt the techniques for a consistency-checking and entailment algorithms for non-ground APT-programs in Section 5 (note that these algorithms do not require a full grounding of a program). In Section 6 we present our implementation of the fixpoint approach to solving consistency and entailment problems for ground programs. Finally, in Section 7, we provide an overview of related work.

(1) $\mathsf{sec\_rumor} \wedge \mathsf{rum\_incr(10\%)} \overset{efr}{\rightsquigarrow} \mathsf{stock\_decr(10\%)} : [2, 0.65, 0.97]$
An SEC rumor and a rumor of an earnings increase leads to a stock price decrease of 10% in 2 days with probability $[0.65, 0.97]$.

(2) $\mathsf{sec\_rumor} \wedge \mathsf{rum\_incr(10\%)} \overset{efr}{\rightsquigarrow} \mathsf{stock\_decr(10\%)} \wedge \mathsf{cfo\_resigns} : [1, 0.5, 0.95]$
An SEC rumor and a rumor of an earnings increase of 10% leads to the CFO resigning in exactly 1 days with a probability $[0.5, 0.95]$.

(3) $\mathsf{OCC(cfo\_resigns)} : [0, 1]$
The CFO resigns between 0 and 1 times (*i.e.*, $[\mathsf{lo}, \mathsf{up}] = [0, 1]$).

(4) $\mathsf{BLK(sec\_rumor)} :< 4$
An SEC rumor cannot be reported more than 3 days in a row (*i.e.*, $\mathsf{blk} = 4$).

(5) $(\neg\mathsf{sec\_rumor} \wedge \neg\mathsf{rum\_incr(10\%)} \wedge \neg\mathsf{stock\_decr(10\%)} \wedge \neg\mathsf{cfo\_resigns}) : 1 \wedge$
$(\mathsf{sec\_rumor} \wedge \mathsf{rum\_incr(10\%)} \wedge \neg\mathsf{stock\_decr(10\%)} \wedge \neg\mathsf{cfo\_resigns}) : 2 \wedge$
$(\mathsf{sec\_rumor} \wedge \neg\mathsf{rum\_incr(10\%)} \wedge \mathsf{stock\_decr(10\%)} \wedge \neg\mathsf{cfo\_resigns}) : 3 \wedge$
$(\mathsf{sec\_rumor} \wedge \mathsf{rum\_incr(10\%)} \wedge \neg\mathsf{stock\_decr(10\%)} \wedge \mathsf{cfo\_resigns}) : 4 : [1, 1]$
Based on events that have already occurred, we can state things such as "at day 1 there was no SEC rumor, there is no rumor of a stock increase, the stock price did not decrease, and the CFO did not resign."

Fig. 1. $\mathcal{K}_{stock}$, an example APT-Logic Program about stocks.

Before continuing, we note that applications such as those above use automated rule learning (e.g. using the **APT-Extract** algorithm of [Shakarian et al. 2011]) to automatically learn relationships and correlations between atoms. In particular, the existence of specific such relationships make independence and Markovian assumptions invalid for these types of applications.

2. TECHNICAL BACKGROUND

This section recapitulates the syntax and semantics of APT LPs from [Shakarian et al. 2011] – with the exception of integrity constraints and probabilistic time formulas, this section does not contain new material.

(1) detainment_distr(2) $\wedge$ detainment_relig(1) $\overset{efr}{\rightsquigarrow}$ attack_relig(1):[2, 0.0906, 0.1906]
A detainment in district 2 and detainment in an area where religion 1 dominates is followed by an attack in an area where religion 1 dominates within 2 days with a probability $[0.0906, 0.1906]$.

(2) attack_neigh(28) $\wedge$ attack_relig(1) $\overset{efr}{\rightsquigarrow}$ cache_relig(1):[7, 0.6833, 0.7833]
An attack in neighborhood 28 and an attack in an area where religion 1 dominates is followed by a cache being found in an area where religion 1 dominates within 7 days with a probability $[0.6833, 0.7833]$.

(3) cache_distr(2) $\overset{efr}{\rightsquigarrow}$ detainment_relig(2):[10, 0.6559, 0.7558]
Cache being found in district 2 is followed by a a detainment in an area where religion 2 dominates within 10 days with a probability $[0.6559, 0.7558]$.

(4) detainment_distr(2) $\overset{efr}{\rightsquigarrow}$ attack_distr(7):[10, 0.1346, 0.2346]
A detainment in district 2 is followed by a an attack in district 7 within 10 days with a probability $[0.1346, 0.2346]$.

(5) attack_neigh(28) $\overset{efr}{\rightsquigarrow}$ detainment_distr(2):[9, 0.5410, 0.6500]
An attack in neighborhood 28 is followed by a detainment in district 2 within 9 days with a probability $[0.5410, 0.6500]$.

(6) cache_distr(5) $\overset{efr}{\rightsquigarrow}$ strike_relig(1):[8, 0.2833, 0.3833]
A cache found in district 5 is followed by a precision strike conducted in an area dominated by religion 1 within 8 days with a probability $[0.2833, 0.3833]$.

Fig. 2. $\mathcal{K}_{ISW}$ an APT-Logic Program extracted from counterinsurgency data.

2.1 Syntax

We assume the existence of a logical language $\mathcal{L}$, with a finite set $\mathcal{L}_{cons}$ of constant symbols, a finite set $\mathcal{L}_{pred}$ of predicate symbols, and an infinite set $\mathcal{L}_{var}$ of variable symbols. Each predicate symbol $p \in \mathcal{L}_{pred}$ has an *arity* (denoted *arity(p)*). We also assume the existence of a finite set $\mathcal{F}$ whose members are called *frequency function* symbols [Shakarian et al. 2011] – these are new to APT-logic and, to our knowledge, have not been studied before. A (ground) *term* is any member of $\mathcal{L}_{cons} \cup \mathcal{L}_{var}$ (resp. $\mathcal{L}_{cons}$); if $t_1, \ldots, t_n$ are (ground) terms, and $p \in \mathcal{L}_{pred}$ has arity n, then $p(t_1, \ldots, t_n)$ is a (resp. ground) atom. We use $B_{\mathcal{L}}$ to denote the Herbrand base (set of all ground atoms) of $\mathcal{L}$. It is easy to see that $B_{\mathcal{L}}$ is finite. *Formulas* are defined as follows: A (ground) atom is a (ground) formula. If F_1 and F_2 are (ground) formulas, then $F_1 \wedge F_2$, $F_1 \vee F_2$, and $\neg F_1$ are (ground) formulas.

We assume that all applications are interested in reasoning about an arbitrarily large, but fixed size window of time, and that $\tau = \{1, \ldots, t_{max}\}$ denotes the entire set of time points we are interested in. t_{max} can be as large as an application user wants, and the user may choose his granularity of time according to his needs.

Definition 2.1 *Time Formula.* A **time formula** is defined as follows:

- If F is a (ground) formula and $t \in [1, t_{max}]$ then $F : t$ is an (ground) **elementary** time formula.
- If ϕ, ρ are (ground) time formulas, then $\neg\phi$, $\phi \wedge \rho$, and $\phi \vee \rho$ are (resp. ground) time formulas.

(1) orgst1(1)∧orgst11(2)∧domorgviolence(2) $\overset{efr}{\rightsquigarrow}$ armattack(1):[2, 0.95, 1]
Whenever education and propaganda are used as a minor strategy, coalition building is used as a major strategy, and the group is using domestic violence regularly by targeting security personnel (but not government non-security personnel or civilians), the group carries out armed attacks within two time periods with probability at least 0.95.

(2) orgst1(1)∧orgst11(2)∧domorgviolence(2) $\overset{efr}{\rightsquigarrow}$ dsecgov(1):[3, 0.95, 1]
This rule has the same antecedent as the previous one, but the consequent stands for the group targeting people working for the government in security, or in non-state armed militias.

(3) violrhetrans(0)∧orgst5(0)∧drug(0) $\overset{efr}{\rightsquigarrow}$ armattack(1):[2, 0.58, 0.68]
Whenever the group does not justify targeting transnational entities in public statements, uses non-coercive methods to collect local support (as a minor strategy), and does not engage in drug production/trafficking, armed attacks are carried out within two time periods with probability between 0.58 and 0.68.

(4) orgst1(1)∧orgst11(2)∧orgst8(2) $\overset{efr}{\rightsquigarrow}$ dsecgov(1):[3, 0.9500, 1]
Whenever education and propaganda are used as a minor strategy, coalition building is used as a major strategy, and insurgencies are used as a major strategy, the group targets people working for the government in security, or in non-state armed militias, within 3 time periods with probability at least 0.95.

Fig. 3. $\mathcal{K}_{MAROB}$ an APT-Logic Program extracted from Minorities at Risk Organizational Behavior data.

EXAMPLE 2.1. *Consider the ground atoms in the APT-program from Figure 1. The expression* $(\neg sec\_rumor \wedge \neg rum\_incr(10\%) \wedge \neg stock\_decr(10\%) \wedge \neg cfo\_resigns) : 1$ *is an elementary time formula.*

Throughout, we will use Greek letters ϕ, ρ for time formulas and capital letters F, G for regular formulas. We now extend a time formula to include a probability annotation.

Definition 2.2. If ϕ is a (ground) time formula and $[\ell, u] \subseteq [0, 1]$, then $\phi : [\ell, u]$ is a (resp. ground) **probabilistic time formula**, or **ptf** for short.

Note that when considering ptf's of the form $F : t : [\ell, u]$*, we will sometimes abuse notation and write* $F : [t, \ell, u]$.

EXAMPLE 2.2. *Item 5 in the APT-program from Figure 1 is a ptf.*

Intuitively, $\phi : [\ell, u]$ says time formula ϕ is true with a probability in $[\ell, u]$.[1]

A word on probability intervals: The reader may notice immediately that APT-logic does not use point probabilities but instead probability intervals. There are two purposes for this. First, if extracting ptf's or rules from historical data (as we do in our experiments), it is often the case that probability is determined within a certain range (i.e., "the probability of event X is $45\% + / - 5\%$"). The second

[1] **Assumption:** Throughout the paper we assume, for both ptf's and APT-rules, that the numbers ℓ, u can be represented as rationals a/b where a and b are relatively prime integers and the length of the binary representations of a and b is fixed.

purpose for using intervals rather than point probabilities is that when reasoning without independence assumptions, it is often only possible to obtain a bound on the probability rather than a single value.

Definition 2.3 *Integrity constraint.* Suppose $A_i \in B_{\mathcal{L}}$ and $[\mathsf{lo}_i, \mathsf{up}_i] \subseteq [0, t_{max}]$. Then $\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i]$ is called an *occurrence IC*. If $\mathsf{blk}_i \in [2, t_{max} + 1]$ is an integer, then $\mathsf{BLK}(A_i) :< \mathsf{blk}_i$ is called a *block-size IC*. If A_i is ground then the occurrence (resp. block-size) IC is ground – otherwise it is non-ground.

An occurrence IC $\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i]$ says that A must be true at least lo_i times and at most up_i times. Likewise, the block-size IC says that A cannot be consecutively true for blk_i or more time points. Figure 1 also contains an example occurrence IC and an example block-size IC.

Example 2.3 Integrity Constraints. *Consider the ground atoms in the APT-program from Figure 1 and $t_{max} = 6$. Suppose historical data indicates that for a sequence of 6 days, there is never more than 1 day where the CFO resigns. Hence, we should add the constraint $\mathsf{OCC}(\mathsf{cfo\_resigns}) : [0, 1]$ to the program. There are other types of integrity constraints that could be useful in this domain. For example, a drastic stock price decrease may never occur more than a few times a quarter.*

*To see why block-size constraints are natural, consider the ground atom sec_rumor. Suppose there is never more than 3 days historically where an SEC rumor is reported. This would make the constraint $\mathsf{BLK}(\mathsf{sec\_rumor}) :< 4$ appropriate. Other examples of such constraints in this domain would be reports of profits, which only occur once per quarter (*i.e.*, we would have $\mathsf{blk} = 2$ for such a case).*

We have automatically extracted APT-programs from the ISW and MAROB data sets mentioned earlier. In the case of the ISW data set, occurrence and block-size constraints are needed because militant groups have constrained resources, *i.e.*, a limited amount of personnel and munitions to carry out an attack. Hence, an occurrence integrity constraint can limit the amount of attacks we believe they are capable of in a given time period. Likewise, such groups often limit the amount of consecutive attacks, as police and military often respond with heightened security. Block-size constraints allow us to easily encode this into our formalism.

Definition 2.4 *APT Rules and Programs.* (i) Suppose F, G are (ground) formulas, Δt is a time interval, $[\ell, u]$ is a probability interval, and $\mathsf{fr} \in \mathcal{F}$ is a frequency function symbol. Then $F \overset{\mathsf{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u]$ is an (ground) *APT rule*.
(ii) An (ground) *APT logic program* is a finite set of (ground) APT rules, ptf's, and integrity constraints.
(iii) Given a non-ground APT-logic program $\mathcal{K}^{(ng)}$, the set of ground instances of all rules, ptf's, and IC's in $\mathcal{K}^{(ng)}$ is called the grounding of $\mathcal{K}^{(ng)}$.

Note: Unless specified otherwise, throughout this paper, APT-logic programs, rules, IC's, and ptf's are ground.

Example 2.4. *Figure 1 shows a small APT LP dealing with the stock market, together with an intuitive explanation of each rule.*

(1) $\mathsf{at\_station}(\mathsf{T}, \mathsf{S}_1) \wedge \mathsf{adjEast}(\mathsf{S}_1, \mathsf{S}_2) \overset{efr}{\rightsquigarrow} \mathsf{at\_station}(\mathsf{T}, \mathsf{S}_2) : [4, 0.85, 1]$
If train T is at station S_1 and the station adjacent to it to the East is S_2, T will be at station S_2 within 4 time units with a probability bounded by $[0.85, 1]$.

(2) $\mathsf{at\_station}(\mathsf{T}, \mathsf{S}_1) \wedge \mathsf{adjWest}(\mathsf{S}_1, \mathsf{S}_2) \overset{efr}{\rightsquigarrow} \mathsf{at\_station}(\mathsf{T}, \mathsf{S}_2) : [2, 0.6, 0.7]$
If train T is at station S_1 and the station adjacent to it to the West is S_2, T will be at station S_2 within 2 time units with a probability in the interval $[0.6, 7]$.

(3) $\bigwedge_{t=1}^{t_{max}} \mathsf{adjEast(stnA, stnB)} : t : [1, 1]$, $\bigwedge_{t=1}^{t_{max}} \mathsf{adjEast(stnB, stnC)} : t : [1, 1]$, $\bigwedge_{t=1}^{t_{max}} \mathsf{adjWest(stnB, stnA)} : t : [1, 1]$, $\bigwedge_{t=1}^{t_{max}} \mathsf{adjWest(stnC, stnB)} : t : [1, 1]$
Probabilistic time formulas specifying that Station B is (always) adjacent to the East of A, and C is adjacent to the East of B.

(4) $\mathsf{at\_station(train1, stnA)} : 1 : [0.5, 0.5]$
For a given sequence of events, train 1 will be at station A at time period 1 with a probability of 0.50.

(5) $\mathsf{at\_station(train2, stnA)} : 2 : [0.48, 0.52]$
For a given sequence of events, train 2 will be at station A at time period 2 with a probability bounded by $[0.48, 0.52]$.

Fig. 4. $\mathcal{K}_{train}$, an APT-Logic Program modeling rail transit. Items 1-2 are non-ground APT-Rules, the formulas in 3 are probabilistic temporal formulas, and items 4-5 are annotated formulas. The English translation of each rule is also provided.

We would like to point out that for an APT-rule $F \overset{\mathsf{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u]$, there is no dependence implied or expressed between worlds satisfying formulas F and G. This follows directly from our semantics (described in the next section). A similar implication known as the "leads-to" operator is described in the probabilistic computational tree logic (PCTL) of Hansson and Jonsson [1994a]. However, unlike an APT-rule, the semantics of this operator is completely different as the satisfying structure is a Markov Process in that case, which make independence assumptions between non-consecutive transitions. In Shakarian et al. [2011], we provided a detailed comparison between APT-logic and PCTL.

2.2 Semantics

We now recapitulate the semantics of APT LPs from [Shakarian et al. 2011].

Definition 2.5 *World.* A *world* is any set of ground atoms. ■

The power set of $B_{\mathcal{L}}$ (denoted $2^{B_{\mathcal{L}}}$) is the set of all possible worlds. Intuitively, a world describes a possible state of the (real) world phenomenon being modeled by an APT-logic program. As worlds are just ordinary Herbrand interpretations [Lloyd 1987], we use $w \models F$ to denote the standard definition of satisfaction of a ground formula F by world w as expressed in [Lloyd 1987]. We say world w **satisfies** non-ground formula F iff w satisfies all ground instances of F. A *thread* is a standard temporal interpretation [Emerson and Halpern 1984; Lamport 1980].

Definition 2.6 *Thread.* A *thread* is a mapping $Th : \{1, \ldots, t_{max}\} \rightarrow 2^{B_{\mathcal{L}}}$. ■

$Th(i)$ says that according to thread Th, the world at time i will be $Th(i)$. We use $\mathcal{T}$ to denote the set of all possible threads, and $Th_{\emptyset}$ to denote the "null" thread

which assigns $\emptyset$ to all time points. A thread represents a possible way the domain being modeled will evolve over all time points.

Definition 2.7. (i) Given thread Th and ground time formula ϕ, we say Th **satisfies** ϕ (written $Th \models \phi$) iff:

- $\phi \equiv F : t$: $Th \models \phi$ iff $Th(t) \models F$
- $\phi \equiv \neg\rho$: $Th \models \phi$ iff $Th \not\models \rho$
- $\phi \equiv \rho \wedge \rho'$: $Th \models \phi$ iff $Th \models \rho$ and $Th \models \rho'$
- $\phi \equiv \rho \vee \rho'$: $Th \models \phi$ iff $Th \models \rho$ or $Th \models \rho'$

(ii) Given thread Th and ground occurrence IC $\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i]$, we say Th **satisfies** $\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i]$ iff $|\{i \mid Th(i) \models A_i\}| \in [\mathsf{lo}_i, \mathsf{up}_i]$.
(iii) Given thread Th and block-size IC $\mathsf{BLK}(A_i) :< \mathsf{blk}_i$, we say Th **satisfies** $\mathsf{BLK}(A_i) :< \mathsf{blk}_i$ iff there does not exist an interval $[i, i + \mathsf{blk}_i - 1]$ such that for all $j \in [i, i + \mathsf{blk}_i - 1]$, $Th(j) \models A_i$.
(iv) Th satisfies a non-ground formula or IC iff it satisfies all its ground instances.

Given a set $\mathcal{T}$ of threads and a set IC of integrity constraints, we use $\mathcal{T}(IC)$ to refer to the set $\{Th \in \mathcal{T} \,|\, Th \models IC\}$.

We use the symbol '$\models$' to denote entailment between two time formulas.

Definition 2.8. Given time formulas ϕ, ρ, we say: $\phi \models \rho$ iff $\forall Th \in \mathcal{T}$ s.t. $Th \models \phi$, it is the case that $Th \models \rho$.

If we view time formulas as sets of threads, we can think of $\phi \models \rho$, as equivalent to $\phi \subseteq \rho$. A *temporal probabilistic (tp) interpretation* gives us a probability distribution over all possible threads.

Definition 2.9 *Temporal-Probabilistic Interpretation.* A temporal-probabilistic (tp) interpretation I is a probability distribution over the set of all possible threads, *i.e.*, $\sum_{th \in \mathcal{T}} I(th) = 1$. ■

Thus, a tp-interpretation I assigns a probability to each thread. This reflects the probability that the world will in fact evolve over time in accordance with what the thread says. We now define what it means for a tp-interpretation to satisfy a ptf or integrity constraint.

Definition 2.10. (i) Given interpretation I and ptf $\phi : [\ell, u]$, we say I **satisfies** $\phi : [\ell, u]$ (written $I \models \phi : [\ell, u]$) iff:

$$\ell \leq \sum_{\substack{Th \in \mathcal{T} \\ Th \models \phi}} I(Th) \leq u$$

(ii) Given interpretation I and occurrence IC $\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i]$, we say I **satisfies** $\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i]$ (written $I \models \mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i]$) iff $\forall Th \in \mathcal{T}$ s.t. $Th \not\models \mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i]$, it is the case that $I(Th) = 0$.
(iii) Given interpretation I and block-size IC $\mathsf{BLK}(A_i) :< \mathsf{blk}_i$, we say I **satisfies** $\mathsf{BLK}(A_i) :< \mathsf{blk}_i$ (written $I \models \mathsf{BLK}(A_i) :< \mathsf{blk}_i$) iff $\forall Th \in \mathcal{T}$ s.t. $Th \not\models \mathsf{BLK}(A_i) :< \mathsf{blk}_i$, it is the case that $I(Th) = 0$.
(iv) Interpretation I **satisfies** a non-ground formula or IC iff it satisfies all ground instances of it.

With the above definition, we now define a special type of ptf that can be used to specify a set of threads that start with the same worlds – the intuition is based on the idea of a *prefix* in [Cleaveland et al. 2005].

Definition 2.11. For $n \leq t_{max}$, let $F_1, \ldots, F_i, \ldots, F_n$ be formulas s.t. each F_i is satisfied by exactly one world. Then, the following ptf:

$$F_1 : 1 \wedge \cdots \wedge F_i : i \wedge \ldots \wedge F_n : n : [1, 1]$$

is called a **prefix**.

EXAMPLE 2.5. *Item 5 in the APT-program from Figure 1 is a prefix.*

Intuitively, including a prefix in an APT-program forces the first n worlds of every thread assigned a non-zero probability to satisfy certain formulas. Further, we can use a prefix to force the first n worlds of every thread with a non-zero probability to be the same. For example, if we want the i'th world of thread Th to be set to world w, we would simply use the following formula as F_i in the prefix: $\left(\bigwedge_{a \in w} a\right) \wedge \left(\bigwedge_{a \notin w} \neg a\right)$.

As shown in Shakarian et al. [2011], one of the ways APT-logic separates itself from past work is the introduction of the *frequency function*. The basic idea behind a frequency function is to represent temporal relationships *within* a thread. For instance, we are interested in the frequency with which G will be true Δt units after F is true. When we study this w.r.t. a specific thread Th, we need to identify when F was true in thread Th, and whether G really was true Δt units after that. Consider Figure 5, there are many ways to determine the frequency at which a world satisfying F is followed by world satisfying G within a certain number of time-steps:

- The probability (within the thread of Figure 5) that G follows F in *exactly* two units of time is 0.67 *if we ignore the occurrence of F at time 8*. If on the other hand, we count the occurrence of F at time 8 (even though no times beyond that are possible), then the probability that G follows F in *exactly* two units of time is 0.5.
- The probability that G follows F in *at most* 2 units of time is 1.0 if we ignore the occurrence of F at time 8; otherwise it is 0.75 (this would be an example of the frequency function *efr*, described later).

Each of these intuitions leads to different ways to measure the frequency (within a thread) with which G follows F.[2] Hence, rather than adhering to a single definition of what it means for F to be followed by G, we define this relationship in terms of axioms. Therefore, the use of frequency functions allow us to parameterize this relationship.

Definition 2.12 *Frequency Function.* Let Th be a thread, F and G be **ground** formulas, and $\Delta t > 0$ be an integer. A *frequency function* fr maps quadruples of the form $(Th, F, G, \Delta t)$ to $[0, 1]$ such that the following axioms hold:

[2]**Note:** Throughout this paper, we assume that frequency functions can be computed in polynomial time (*i.e.*, $O(|B_{\mathcal{L}}| \cdot t_{max})$). We also assume that frequency functions return rational numbers and that the length of the binary representations of a and b is fixed.

Fig. 5. Example thread, Th with worlds $Th(1), \ldots, Th(8)$. This figure shows each world that satisfies formula F or formula G.

(FF1) If G is a tautology, then $\mathsf{fr}(Th, F, G, \Delta t) = 1$.
(FF2) If F is a tautology and G is a contradiction, then $\mathsf{fr}(Th, F, G, \Delta t) = 0$.
(FF3) If F is a contradiction, $\mathsf{fr}(Th, F, G, \Delta t) = 1$.
(FF4) If G is not a tautology, and either F or $\neg G$ is not a tautology, and F is not a contradiction, then there exist threads Th_1, $Th_2 \in \mathcal{T}$ such that $\mathsf{fr}(Th_1, F, G, \Delta t) = 0$ and $\mathsf{fr}(Th_2, F, G, \Delta t) = 1$. ■

Axiom FF1 says that if G is a tautology, then $\mathsf{fr}(Th, F, G, \Delta t)$ must behave like material implication and assign 1 to the result. Likewise, if F is a tautology and G is a contradiction, then FF2 says that $\mathsf{fr}(Th, F, G, \Delta t)$ must behave like implication and have a value of 0 ($A \rightarrow B$ is false when A is a tautology and B is a contradiction). Axiom FF3 requires $\mathsf{fr}(Th, F, G, \Delta t)$ to be 1 when F is a contradiction, also mirroring implication. Axiom FF4 ensures that in all cases not covered above, the frequency function will be non-trivial by allowing at least one thread that perfectly satisfies (probability 1) and perfectly contradicts (probability 0) the conditional. Note that any function not satisfying Axiom FF4 can be made to do so as long as it returns distinct values: simply map the lowest value returned to 0 and the highest value returned to 1.

Shakarian et al. [2011] provides several different frequency functions corresponding to different intuitions that satisfy the above axioms. For the sake of simplicity and due to space constraints, in this paper we use the *existential* frequency function from [Shakarian et al. 2011]. The key intuition behind this frequency function is to represent how often a world satisfying formula F is followed by a world satisfying formula G. We also provide special treatment of the boundary condition in this definition, that is where F is satisfied by worlds after time $t_{max} - \Delta t$. If F is satisfied by a world after this time, we do not include it in the denominator unless it is followed by G. The reason for this is that the thread ends at t_{max}, hence we have no knowledge of events after this point. As a result, when a world satisfies F after time $t_{max} - \Delta t$ without being followed by a world in the thread satisfying G, we have no knowledge as to whether an event represented by G occurred within Δt time units later.

Of course, there are other definitions of *efr* that are possible. For instance, perhaps there are some applications where it makes sense to not consider this boundary case. This is why frequency functions are defined axiomatically. The majority of the results in this paper do not hinge on *efr*; when they do, we make a note of it.

Definition 2.13 *Existential Frequency Function.* Let Th be a thread, F and G be formulas, and $\Delta t \geq 0$ be an integer. An *Existential Frequency Function*, denoted $efr(Th, F, G, \Delta t)$, is defined as follows:[3]

[3]Where $efn(Th, F, G, \Delta t, t_1, t_2) = |\{t : (t_1 \leq t \leq t_2)$ *and* $Th(t) \models F$ and there exists $t' \in [t+1, \min(t_2, t+\Delta t)]$ such that $Th(t') \models G\}|$.

$$efr(Th, F, G, \Delta t) = \frac{efn(Th, F, G, \Delta t, 0, t_{max})}{|\{t : (t \leq t_{max} - \Delta t) \wedge Th(t) \models F\}| + efn(Th, F, G, \Delta t, t_{max} - \Delta t, t_{max})}$$

If the denominator is zero (if there is no $t \in [0, t_{max} - \Delta t]$ such that $Th(t) \models F$ and $efn(Th, F, G, \Delta t, t_{max} - \Delta t, t_{max}) = 0$) then we define *efr* to be 1.

In [Shakarian et al. 2011], we show that *efr* satisfies Axioms FF1-FF4. We are now ready to define satisfaction of an Annotated Probabilistic Temporal (APT) rule.

Definition 2.14 *Satisfaction of APT rules.* Let $r = F \overset{\mathsf{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u]$ be an APT rule and I be a tp-interpretation.
(i) If r is a **ground rule**, interpretation I satisfies r (denoted $I \models r$) iff

$$\ell \leq \sum_{Th \in \mathcal{T}} I(Th) \cdot \mathsf{fr}(Th, F, G, \Delta t) \leq u.$$

(ii) Interpretation I satisfies a non-ground rule r iff I satisfies all ground instances of r.

Interpretation I satisfies an APT-program iff it satisfies all rules, ptf's, and IC's in that program. Given an APT-program $\mathcal{K}$, we will often refer to the set of integrity constraints in $\mathcal{K}$ as simply IC.

Intuitively, the APT rule $F \overset{\mathsf{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u]$ evaluates the probability that F leads to G in Δt time units as follows: for each thread, it finds the probability of the thread according to I and then multiplies it by the frequency (in terms of fraction of times) with which F is followed by G in Δt time units according to frequency function fr. This product is like an expected value in statistics where a value (frequency) is multiplied by a probability (of the thread). It then sums up these products across all threads.

To our knowledge, APT-rules represent the first time an attempt to reason about frequencies *within* threads in the temporal-probabilistic logic and LP literature.

Definition 2.15 *Entailment/consistency.* An APT-LP $\mathcal{K}$ is consistent iff there is at least one interpretation that satisfies $\mathcal{K}$. $\mathcal{K}$ *entails* ptf $\phi : [\ell, u]$ iff every interpretation that satisfies $\mathcal{K}$ also satisfies $\phi : [\ell, u]$.

The material from here onwards in the paper is new and is different from that covered in [Shakarian et al. 2011].

3. COMPLEXITY

[Shakarian et al. 2011] showed that consistency and entailment in APT-logic is NP-hard (consistency) and coNP-hard (entailment). In this section, we prove that consistency is in NP and entailment is in coNP. The result is somewhat surprising, because the exact algorithms presented in Shakarian et al. [2011] relied on the solution to linear programs with an exponential number of variables. For example, consider the following linear program.

Definition 3.1 *CONS.* Given an APT-logic program, $\mathcal{K}$, where $IC \subset \mathcal{K}$ is the set of integrity constraints in $\mathcal{K}$, we can create the linear constraints CONS($\mathcal{K}$) as

follows:
For each $Th_j \in \mathcal{T}(IC)$, variable v_j denotes the probability of thread Th_j.

$$
\begin{array}{lll}
\textit{(1)}\ \sum_{j=1}^{|\mathcal{T}(IC)|} v_j = 1 & & \\
\textit{(2)}\ \forall F_i \overset{\mathsf{fr}_i}{\rightsquigarrow} G_i : [\Delta t_i, \ell_i, u_i] \in \mathcal{K} & \textit{(a)}\ \ell_i \leq \sum_{j=1}^{|\mathcal{T}(IC)|} \mathsf{fr}_i(Th_j, F_i, G_i, \Delta t_i) \cdot v_j \\
 & \textit{(b)}\ u_i \geq \sum_{j=1}^{|\mathcal{T}(IC)|} \mathsf{fr}_i(Th_j, F_i, G_i, \Delta t_i) \cdot v_j \\
\textit{(3)}\ \forall \phi_i : [\ell_i, u_i] \in \mathcal{K} & \textit{(a)}\ \ell_i \leq \sum_{Th_j \in \mathcal{T}(IC)\ Th_j \models \phi_i} v_j \\
 & \textit{(b)}\ u_i \geq \sum_{Th_j \in \mathcal{T}(IC)\ Th_j \models \phi_i} v_j
\end{array}
$$

[Shakarian et al. 2011] proved the following:

PROPOSITION 3.2. *$\mathcal{K}$ is consistent iff there is a solution to $\mathsf{CONS}(\mathcal{K})$. Given ptf $\phi : [\ell, u]$, let L be the minimization and U be the maximization of $\sum_{Th_j \in \mathcal{T}(IC)\ Th_j \models \phi} v_j$ subject to $\mathsf{CONS}(\mathcal{K})$. Then $\phi : [\ell, u]$ is entailed by $\mathcal{K}$ iff $[L, U] \subseteq [\ell, u]$.*

However, it turns out that we can be guaranteed a solution to the linear program where only a polynomial number of the variables are set to a value other than 0. Consider the following theorem from Chvtal [1983] and later used in Fagin et al. [1990] to show that deciding the validity of a formula in the logic of [Fagin et al. 1990] is NP-Complete.

THEOREM 3.3 [CHVTAL 1983; FAGIN ET AL. 1990]. *If a system of m linear equalities and/or inequalities has a nonnegative solution, then it has a nonnegative solution with at most m positive variables.*

We can leverage the previous two results to guarantee the existence of an interpretation that assigns a zero probability to all but a polynomial number of threads, thus giving us a "small model" theorem.

THEOREM 3.4. *Deciding if APT-program $\mathcal{K}$ is consistent is NP-complete if $|\mathcal{K}|$ is a polynomial in terms of $|B_{\mathcal{L}}|$.*

THEOREM 3.5. *Deciding if APT-rule r is entailed by APT-program $\mathcal{K}$ is coNP-complete if $|\mathcal{K}|$ is a polynomial in terms of $|B_{\mathcal{L}}|$.*

One may wonder if APT-programs can be made more tractable if we assume a single probability distribution over threads, that is a single tp-interpretation. Unfortunately, even if we assume a uniform probability distribution, this special case is still not tractable.

THEOREM 3.6. *Given APT-program $\mathcal{K}$, interpretation I, and ptf ϕ, determining the maximum ℓ and minimum u such that $\phi : [\ell, u]$ is entailed by $\mathcal{K}$ **and** is satisfied by I is #P-hard. Furthermore, for constant $\epsilon > 0$, approximating either the maximum ℓ and/or minimum u within $2^{|B_{\mathcal{L}}|^{1-\epsilon}}$ is NP-Hard.*

The above theorem is proved using an interpretation that assigns a uniform probability across all threads. The negative approximation result follows from a result of Roth [1996].

In Shakarian et al. [2011], we examined hardness results and provided various sets of linear constraints for solving APT-logic consistency and entailment problems exactly. Methods to approximate the entailment problem within a certain

factor were not examined in that paper. Although it remains an open question if the APT-entailment problem (without the single-interpretation requirement) can be approximated in this manner, the above result is not encouraging.[4] Further, Definition 3.1 illustrates several challenges relating the intractability of this problem. (i) First, we need to compute $\mathcal{T}(IC)$, which is a challenge because $\mathcal{T}$ contains $2^{t_{max} \cdot |B_\mathcal{L}|}$ possible threads and each must be examined to see if it satisfies IC; (ii) Second, the constraints in items (1-2) may contain up to $O\left(2^{t_{max} \cdot |B_\mathcal{L}|}\right)$ variables (this bound can be tightened), so even though linear programming is polynomial [Karmarkar 1984], the input is exponential in the size of t_{max} and $B_\mathcal{L}$. In practice, even if we consider $t_{max} = 10$ and $B_\mathcal{L}$ to consist of just 100 ground atoms, we are looking at the possibility of examining $2^{1,000}$ threads to find $\mathcal{T}(IC)$ and writing constraints containing exponentially large numbers of variables. In practice, we will not be able to even write these constraints. With these intractability results in mind, we proceed to develop heuristics in the next two sections.

4. A SOUND BUT INCOMPLETE FIXPOINT-COMPUTATION ALGORITHM: THE GROUND CASE

This section presents a heuristic algorithm based on a fixpoint operator Γ which maps APT-programs to APT-programs and iteratively tightens the probability bounds on rules and ptf's in the program. To find probability bounds on some time formula ϕ, we simply add the ptf $\phi : [0, 1]$ to the program, iteratively apply Γ until a fixed point is reached, and then examine the bounds on the ptf formed with ϕ in the resulting program. Our approach is sound – so, if the interval $[\ell, u]$ is assigned to ϕ, then $\mathcal{K}$ entails $\phi : [\ell, u]$ (provided, of course, that $\mathcal{K}$ is consistent). However, there may exist some $[\ell', u'] \subset [\ell, u]$ such that $\phi : [\ell', u']$ is also entailed.

Our algorithm requires that $\mathcal{K}$ contain at least one APT-rule of the form $F : [\ell, u]$. This is not really a restriction in most applications where a prefix would exist (cf. Definition 2.11, Page 9). The rest of the section is organized as follows. Section 4.1 describes how to find bounds on a frequency function given *ptf*s. Section 4.2 describes how to use frequency bounds to syntactically manipulate rules and ptf's in APT-programs – which in turn allow us to tighten the probability bounds. Section 4.3 performs various syntactic manipulations in the Γ operator and shows that the operator has a least fixed point. Finally, Section 4.4 demonstrates how Γ can also be used to check the consistency of an APT logic program. Again, such a consistency check is sound but not complete – Γ correctly identifies inconsistent programs but does not guarantee consistency.

4.1 Bounding Frequency Function Values

In this paper, we only use the *efr* frequency function. However, our techniques can be easily adapted to other frequency functions such as *pfr* from Shakarian et al. [2011]. Our first definition is a function, *EFR*, which returns tight bounds on *efr* given F, G, and Δt.

Definition 4.1. Suppose F, G are formulas, Δt is a time point, and ϕ is a time

[4] As an aside, as the construction in the proof of Theorem 3.6 does not depend on multiple time-points, this result holds for the probabilistic logic of [Nilsson 1986] as well.

formula. We define $EFR(F, G, \Delta t, \phi) = [\alpha_{tight}, \beta_{tight}]$ where

$$\alpha_{tight} = \mathbf{inf}\{efr(Th, F, G, \Delta t) \mid Th \in \mathcal{T} \wedge Th \models \phi\}.$$
$$\beta_{tight} = \mathbf{sup}\{efr(Th, F, G, \Delta t) \mid Th \in \mathcal{T} \wedge Th \models \phi\}.$$

The intuition in the above definition is that α_{tight} is the least value of efr (w.r.t. formulas F, G and time interval Δt) for all threads satisfying ϕ. Likewise, β_{tight} is the greatest value of efr (w.r.t. formulas F, G and time interval Δt) for all threads satisfying ϕ. We can easily approximate $[\alpha_{tight}, \beta_{tight}]$ when we make certain assumptions on ϕ. Consider the following special case of a ptf:

Definition 4.2. Suppose $ETF \equiv \{F_1 : t_1, \ldots, F_n : t_n\}$ is a set of elementary time formulas, where $n \leq t_{max}$ and for any two such formulas, $F_i : t_i, F_j : t_j \in ETF$, $t_i \neq t_j$. Then $F_1 : t_1 \wedge \ldots \wedge F_n : t_n$ is a **time conjunction**.

Example 4.1. *Item 5 in the APT-program from Figure 1 is a time-conjunction. We shall refer to this time-conjunction as ϕ_{stock} in later examples.*

We notice right away that a prefix (Definition 2.11, Page 9) is simply a special case of time conjunction annotated with probability $[1, 1]$. One useful property of time conjunctions that we leverage in our operator is the following.

Observation 4.1. *If $F_1 : t_1 \wedge \ldots \wedge F_n : t_n \wedge F_{n+1} : t'_1 \wedge \ldots \wedge F_{n+m} : t'_m$ and $G_1 : t_1 \wedge \ldots \wedge G_n : t_n \wedge G_{n+1} : t''_1 \wedge \ldots \wedge G_{n+k} : t''_k$ are time conjunctions, then*

$$(F_1 \wedge G_1) : t_1 \wedge \ldots \wedge (F_n \wedge G_n) : t_n \wedge F_{n+1} : t'_1 \wedge \ldots \wedge F_{n+m} : t'_m \wedge G_{n+1} : t''_1 \wedge \ldots \wedge G_{n+k} : t''_k$$

is also a time conjunction.

We leverage the above property in the following way: if we know a bound for $EFR(F, G, \Delta t, \phi)$ and $EFR(F, G, \Delta t, \phi \wedge \rho)$, we may be able to use this information to find probability bounds on ρ. We will describe this further when we discuss syntactic manipulation. Next, with a **time conjunction** in mind, we will show how to find a tight bound on *EFR*. In this case, we introduce the following notation and obtain a bound on *EFR* in Proposition 4.4.

Definition 4.3. For formulas F, G, time Δt, and time conjunction ϕ, we define the following:

- $cnt(\phi, F, G, \Delta t) = |\{t \in [1, t_{max} - \Delta t] | \exists t' \in (t, t + \Delta t] \ s.t. \ (\phi \models F : t \wedge G : t')\}|$
- $end(\phi, F, G, \Delta t) = |\{t \in (t_{max} - \Delta t, t_{max}) | \exists t' \in (t, t_{max}] \ s.t. \ (\phi \models F : t \wedge G : t')\}|$
- $denom(\phi, F, \Delta t) = |\{t \in [1, t_{max} - \Delta t] | \exists Th \ s.t. \ (Th \models \phi) \wedge (Th \models F : t)\}|$
- $poss(\phi, F, G, \Delta t) = |\{t \in [1, t_{max} - \Delta t] | \exists t' \in (t, t + \Delta t] \ s.t. \ \exists Th \ s.t. \ (Th \models \phi) \wedge (Th \models F : t \wedge G : t')\}|$
- $endposs(\phi, F, G, \Delta t) = |\{t \in (t_{max} - \Delta t, t_{max}) | \exists t' \in (t, t_{max}] \ s.t. \ \exists Th \ s.t. \ (Th \models \phi) \wedge (Th \models F : t \wedge G : t')\}|$

The intuitions behind the components of Definition 4.3 are as follows. For a given $F, G, \Delta t$, *cnt* is simply the number of times in the first $t_{max} - \Delta t$ timesteps (of all threads satisfying some ptf ϕ) where a world satisfying F is followed by a world satisfying G within Δt time units. Likewise, *end* performs this count for the last Δt time units. Similarly, *poss* and *endposs* perform similar calculations, but rather

than considering all threads that satisfy ϕ, there must only exist a thread satisfying ϕ where a world satisfying F is followed by a world satisfying G in Δt time units. The definition of *denom* captures the total number of times F is satisfied in the first $t_{max} - \Delta t$ worlds (for all threads satisfying ϕ). Due to the boundary condition of *efr* (refer to Section 2 for details), we use *end* and *endposs* to perform this count in the last $t_{max} - \Delta t$ worlds of the threads. Hence, in the below proposition, we are able to show that $EFR(F, G, \Delta t, \phi)$ is a subset of two fractions created from the components we defined.

PROPOSITION 4.4. *For formulas F, G, time Δt, and time conjunction ϕ, $EFR(F, G, \Delta t, \phi) \subseteq$*

$$\left[\frac{cnt(\phi, F, G, \Delta t) + end(\phi, F, G, \Delta t)}{denom(\phi, F, \Delta t) + end(\phi, F, G, \Delta t)}, \frac{poss(\phi, F, G, \Delta t) + endposs(\phi, F, G, \Delta t)}{denom(\phi, F, \Delta t) + endposs(\phi, F, G, \Delta t)}\right]$$

EXAMPLE 4.2. *Consider the APT-program from Figure 1 that includes time conjunction ϕ_{stock} (see Example 4.1). Consider the pre and post conditions of rules 1-2; we shall refer to them as follows (in this and later examples):*

$$\begin{aligned} F_1 &\equiv \quad \textit{sec\_rumor} \wedge \textit{rum\_incr(10\%)} \\ G_1 &\equiv \qquad \textit{stock\_decr(10\%)} \\ F_2 &\equiv \quad \textit{sec\_rumor} \wedge \textit{rum\_incr(10\%)} \\ G_2 &\equiv \textit{stock\_decr(10\%)} \wedge \textit{cfo\_resigns} \end{aligned}$$

Using Definition 4.3, we can determine that:

$$EFR(\phi_{stock}, F_1, G_1, 2) \subseteq [0.5, 1.0]$$

and

$$EFR(\phi_{stock}, F_2, G_2, 1) \subseteq [0.0, 0.667]$$

4.2 Theorems for Syntactic Manipulation

In the last section, we bounded the values that *efr* can have for a thread given some time formula ϕ. This section leverages that information to obtain tighter bounds on ptf's and APT-rules. First, we introduce a simple result that allows for syntactic manipulation of ptf's without these bounds.

LEMMA 4.5. *Let $\rho : [\ell', u']$ be a ptf and I be an interpretation; then:*

(1) If $I \models \phi : [\ell, u]$, then $I \models \phi \wedge \rho : [\max(0, \ell + \ell' - 1), \min(u, u')]$
(2) If $I \models \phi : [\ell, u]$, then $I \models \phi \vee \rho : [\max(\ell, \ell'), \min(1, u + u')]$
(3) If $I \models \phi : [\ell, u]$ and $\phi \models \rho$ then $I \models \rho : [\ell, 1]$
(4) If $I \models \phi : [\ell, u]$ and $\rho \models \phi$ then $I \models \rho : [0, u]$
(5) If $I \models \phi : [\ell, u]$ then $I \models \neg\phi : [1 - u, 1 - \ell]$

EXAMPLE 4.3. *Suppose program $\mathcal{K}_{stock}$ entails ptf sec_rumor : 6 : [0.3, 0.6]. Then, it also entails $\neg$sec_rumor : 6 : [0.4, 0.7].*

We notice right away that syntactic manipulation sometimes identifies inconsistent APT-programs. For example, if $\phi : [0.7, 0.6]$ is entailed via Lemma 4.5, then we know that $\mathcal{K}$ is not consistent. We explore this issue further in Section 4.4.

Next, we use the bounds on *EFR* to syntactically manipulate APT-rules, yielding rules with tighter probability bounds – perhaps uncovering an inconsistent program. Theorem 4.6 tightens the bound when the APT-program includes a ptf that happens with probability 1. Its corollary tightens the lower bound given any *ptf*.

THEOREM 4.6. *Suppose I is an interpretation and ϕ is a time formula such that $I \models \phi : [1,1]$ and $EFR(F, G, \Delta t, \phi) \subseteq [\alpha, \beta]$. Then $I \models F \overset{efr}{\rightsquigarrow} G : [\Delta t, \alpha, \beta]$.*

COROLLARY 4.7. *Suppose I is an interpretation and ϕ is a time formula such that $I \models \phi : [\ell, u]$ and $EFR(F, G, \Delta t, \phi) \subseteq [\alpha, \beta]$. Then $I \models F \overset{efr}{\rightsquigarrow} G : [\Delta t, \alpha \cdot \ell, 1]$.*

The above theorem and corollary are proved by showing that the lower probability bound of an APT-rule has to be at least as much as the lower bound on the associated *EFR* for all threads.

EXAMPLE 4.4. *Consider the scenario from Example 4.2. By the result of that example and Corollary 4.7, we know that $\mathcal{K}_{stock}$ must entail:*

$$\mathit{sec\_rumor} \wedge \mathit{rum\_incr(10\%)} \overset{efr}{\rightsquigarrow} \mathit{stock\_decr(10\%)} : [2, 0.5, 1.0] \text{ and}$$

$$\mathit{sec\_rumor} \wedge \mathit{rum\_incr(10\%)} \overset{efr}{\rightsquigarrow} \mathit{stock\_decr(10\%)} \wedge \mathit{cfo\_resigns} : [1, 0.0, 0.667]$$

Note that we can now find a tighter bound on rule 2, obtaining a probability bound of $[0.5, 0.667]$, that is substantially tighter than $[0.5, 1]$ from the original rule using just one syntactic manipulation.

We can use APT-rules, *EFR*, and Theorem 4.5 to further tighten the bounds on ptf's with the following theorem.

THEOREM 4.8. *Suppose F, G are formulas, ϕ, ρ are time formulas, I is an interpretation, and $[\alpha_1, \beta_1], [\alpha_2, \beta_2]$ are intervals such that $EFR(F, G, \Delta t, \phi) \subseteq [\alpha_1, \beta_1]$ and $EFR(F, G, \Delta t, \phi \wedge \rho) \subseteq [\alpha_2, \beta_2]$, $I \models \phi : [1,1]$ (see note[5]) and $I \models F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u]$. Then:*

(1) If $\beta_2 < \beta_1$, then $I \models \rho : \left[0, \min\left(\frac{\ell - \beta_1}{\beta_2 - \beta_1}, 1\right)\right]$

(2) If $\alpha_2 > \alpha_1$, then $I \models \rho : \left[0, \min\left(\frac{u - \alpha_1}{\alpha_2 - \alpha_1}, 1\right)\right]$

From the above theorem, we can easily obtain the following corollary that can be used with just one time formula (*i.e.*, only ρ). Simply consider the case where ϕ is $\mathsf{TRUE} : t_{max}$ and $[\alpha_1, \beta_1] = [0, 1]$.

COROLLARY 4.9. *Suppose F, G are formulas, ρ is a time formula, I is an interpretation, and $[\alpha, \beta]$ is an interval such that $EFR(F, G, \Delta t, \rho) \subseteq [\alpha, \beta]$ and $I \models F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u]$. Then:*

(1) If $\beta < 1$ then $I \models \rho : \left[0, \min\left(\frac{\ell - 1}{\beta - 1}, 1\right)\right]$

(2) If $\alpha > 0$ then $I \models \rho : \left[0, \min\left(\frac{u}{\alpha}, 1\right)\right]$

[5]Note that Theorem 4.6 requires $\ell \leq \beta_1$ and $\alpha_1 \leq u$

EXAMPLE 4.5. *Following from Example 4.4, consider the time-formula* $\mathsf{stock\_decr(10\%)}:5$. *Using Definition 4.3, we find that* $EFR(\phi_{stock} \wedge \mathsf{stock\_decr(10\%)}:5, F_1, G_1, 2) \subseteq [1,1]$. *Previously, we saw that* $EFR(\phi_{stock}, F_1, G_1, 2) \subseteq [0.5, 1]$. *As the lower bound on frequency increases (by conjuncting the new time formula), that is* $1 > 0.5$, *we apply part 2 of Theorem 4.8 (and/or Corollary 4.9) to obtain an upper probability bound on* $\mathsf{stock\_decr(10\%)}:5$. *Hence, this formula is no more probable than* 0.94.

Finally, we show that we can also use integrity constraints to aid in syntactic manipulation. For certain ptf's with probability 1, a given IC may cause another ptf (or multiple ptf's) to be entailed with a probability of 0, which can also contribute to bounding *EFR*.

PROPOSITION 4.10. *For atom* A_i *and program* $\mathcal{K}$ *where* $\mathsf{BLK}(A_i) :< \mathsf{blk}_i \in \mathcal{K}$, *if there exists a ptf* $\phi : [1,1] \in \mathcal{K}$ *such that* $\phi \models A_i : t - \mathsf{blk}_i + 1 \wedge A_i : t - \mathsf{blk}_i + 2 \wedge \ldots \wedge A_i : t-1$, *then* $\mathcal{K}$ *entails* $A_i : t : [0,0]$.

PROPOSITION 4.11. *For atom* A_i *and program* $\mathcal{K}$ *where* $\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i] \in \mathcal{K}$, *if there exists a ptf* $\phi : [1,1] \in \mathcal{K}$ *such that there are numbers* $t_1, \ldots, t_{\mathsf{up}_i} \in \{1, \ldots, t_{max}\}$ *where* $\phi \models A_i : t_1 \wedge \ldots \wedge A_i : t_{\mathsf{up}_i}$ *then for any* $t \notin \{t_1, \ldots, t_{\mathsf{up}_i}\}$ $\mathcal{K}$ *entails* $A_i : t : [0,0]$.

EXAMPLE 4.6. *Consider* $\mathcal{K}_{stock}$ *from the previous examples. As this program includes* $\mathsf{OCC}(\mathsf{cfo\_resigns}) : [0,1]$ *and entails* $\mathsf{cfo\_resigns} : 4 : [1,1]$ *(by the included prefix), we can conclude that* $\mathsf{cfo\_resigns} : 5 : [0,0]$ *and* $\mathsf{cfo\_resigns} : 6 : [0,0]$ *are entailed by this program.*

4.3 The Fixpoint-Based Heuristic

We are now ready to use the results of the last section to create the Γ operator. First, we present some preliminary definitions to tighten probability bounds for ptf's and rules. Note that the correctness of these bounds follows directly from the results of the previous section. First we show how, given an APT-program, we can tighten the lower and upper bound of a ptf.

Definition 4.12. Suppose $\mathcal{K}$ is an APT-program and ϕ is a time formula. We define:

$$\mathbf{l\_bnd}(\phi, \mathcal{K}) = \mathbf{sup}\left(\{0\} \cup \{\ell \mid \rho : [\ell, u] \in \mathcal{K} \wedge (\rho \models \phi)\}\right).$$

$\mathbf{u\_bnd}(\phi, \mathcal{K})$ is the **inf** of the set:

$$\begin{array}{lll}
\{ & 1 & \} \cup \\
\{ & u & \mid \rho : [\ell, u] \in \mathcal{K} \wedge (\phi \models \rho)\} \cup \\
\{ & \min\left(\frac{\ell - \beta_1}{\beta_2 - \beta_1}, 1\right) & \mid (F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u], \rho : [1,1] \in \mathcal{K} \cup \{\mathsf{true} : t_{max} : [1,1]\}) \wedge \\
 & & (EFR(F, G, \Delta t, \rho) \subseteq [\alpha_1, \beta_1]) \wedge \\
 & & (EFR(F, G, \Delta t, \rho \wedge \phi) \subseteq [\alpha_2, \beta_2]) \wedge (\beta_2 < \beta_1)\} \cup \\
\{ & \min\left(\frac{u - \alpha_1}{\alpha_2 - \alpha_1}, 1\right) & \mid (F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u], \rho : [1,1] \in \mathcal{K} \cup \{\mathsf{true} : t_{max} : [1,1]\}) \wedge \\
 & & (EFR(F, G, \Delta t, \rho) \subseteq [\alpha_1, \beta_1]) \wedge \\
 & & (EFR(F, G, \Delta t, \rho \wedge \phi) \subseteq [\alpha_2, \beta_2]_{\wedge}(\alpha_2 > \alpha_1)\}
\end{array}$$

This bound on a time formula is derived from its relationship with other time formulas (by Lemma 4.5) or it relationship with rules (by Theorem 4.8 and/or Corollary 4.9). Below we show an example.

EXAMPLE 4.7. *Following from Example 4.5, consider, once again, the time-formula* stock_decr(10%) : 5. *For program* $\mathcal{K}_{stock}$, *we know that* **l_bnd**(stock_decr(10%) : $5, \mathcal{K}_{stock}) = 0.0$. *This is due to the simple fact that there is no lower probability bound assigned to the time formula* stock_decr(10%) : 5 *by* $\mathcal{K}_{stock}$ *that is greater than* 0.0. *Examining the upper bound, we consider the* **inf** *of set* $\{1, 0.94\}$ *as* 1 *is the trivial upper bound, there are no other upper probability bounds for* stock_decr(10%) : 5 *seen directly in* $\mathcal{K}_{stock}$ *and we have already used Example 4.5 to derive the upper bound of* 0.94 *based on syntactic manipulation of rules in* $\mathcal{K}_{stock}$ *(which reflects the last two parts of the* **u_bnd** *definition). Hence,* **u_bnd**(stock_decr(10%) : $5, \mathcal{K}_{stock}) =$ 0.94.

Note that for ptf's we do not include any manipulation that relies on the bounds of the negated time formula in the above definitions. We handle this type of manipulation in the definition of the operator. The following are versions of **l_bnd**, **u_bnd** for rules.

Definition 4.13. Suppose $\mathcal{K}$ is an APT-program, F, G are formulas, and $\Delta t > 0$ is an integer.

- The quantity **l_bnd**$(F, G, \Delta t, \mathcal{K})$ is the **sup** of the following set:

$$
\begin{array}{lll}
\{ & 0 & \} \cup \\
\{ & \ell & \mid F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u] \in \mathcal{K} \} \cup \\
\{ & \alpha \cdot \ell & \mid (\phi : [\ell, u], \rho : [1,1] \in \mathcal{K} \cup \{\mathsf{true} : t_{max} : [1,1]\}) \wedge \\
 & & (EFR(F, G, \Delta t, \rho \wedge \phi) \subseteq [\alpha, \beta]) \} \cup \\
\{ & \alpha \cdot (1-u) & \mid (\phi : [\ell, u], \rho : [1,1] \in \mathcal{K} \cup \{\mathsf{true} : t_{max} : [1,1]\}) \wedge \\
 & & (EFR(F, G, \Delta t, \rho \wedge \neg\phi) \subseteq [\alpha, \beta]) \}
\end{array}
$$

- The quantity **u_bnd**$(F, G, \Delta t, \mathcal{K})$ is the **inf** of the following set:

$$
\begin{array}{lll}
\{ & 1 & \} \cup \\
\{ & u & \mid F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u] \in \mathcal{K} \} \cup \\
\{ & \beta & \mid (\rho : [1,1] \in \mathcal{K}) \wedge (EFR(F, G, \Delta t, \rho) \subseteq [\alpha, \beta]) \}
\end{array}
$$

Hence, the new probability bound assigned to a rule is based on how the bounds on the frequency function are tightened given the ptf's present in the program. Given a ptf, we use a bound on *EFR*, which allows us to leverage Theorem 4.6 and Corollary 4.7 to obtain a tighter bound on the rule. Tighter bounds on rules are useful for two reasons: (1) subsequent applications of the fixpoint operator will in turn use these new bounds to tighten bounds on ptf's and (2) they can be used to identify inconsistent program (as we discuss in Section 4.4).

We now define set *formula*($\mathcal{K}$) which intuitively means "all time formulas that appear in $\mathcal{K}$". These are the formulas upon which Definition 4.12 will act, and also through syntactic manipulation, affect other ptf's in $\mathcal{K}$. As stated earlier, we can find bounds for any time formula ρ by adding $\rho : [0, 1]$ to the initial APT program.

Definition 4.14. Given program $\mathcal{K}$ consisting of ptf's and constrained rules, *formula*($\mathcal{K}$) is the following set:

$$\begin{aligned}
&\{ \quad \phi \quad \mid \phi : [\ell, u] \in \mathcal{K} \} \cup \\
&\{ \quad F : t \ \mid (t \in [1, t_{max}]) \wedge (F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u] \in \mathcal{K}) \} \cup \\
&\{ \quad G : t \ \mid (t \in [1, t_{max}]) \wedge (F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u] \in \mathcal{K}) \}
\end{aligned}$$

We now have all the pieces we need to define our operator Γ.

Definition 4.15. Given program $\mathcal{K}$, $\Gamma(\mathcal{K})$ is defined as the following set:

$$\begin{aligned}
\{ \quad & F \overset{efr}{\rightsquigarrow} G : [\Delta t, \textbf{l_bnd}(F, G, \Delta t, \mathcal{K}), \\
& \qquad \textbf{u_bnd}(F, G, \Delta t, \mathcal{K})] && \mid F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u] \in \mathcal{K} \} \cup \\
\{ \quad & \phi : [\textbf{l_bnd}(\phi, \mathcal{K}), \textbf{u_bnd}(\phi, \mathcal{K})] \cap \\
& \quad [1 - \textbf{u_bnd}(\neg\phi, \mathcal{K}), 1 - \textbf{l_bnd}(\neg\phi, \mathcal{K})] && \mid \phi \in \mathit{formula}(\mathcal{K}) \} \cup \\
\{ \quad & A_i : t : [0, 0] && \mid (\mathsf{BLK}(A_i) :< \mathsf{blk}_i \in \mathcal{K}) \wedge (\phi : [1, 1] \in \mathcal{K}) \wedge \\
& && \quad (\phi \models A_i : t - \mathsf{blk}_i + 1 \wedge \ldots \wedge A_i : t - 1)\} \cup \\
\{ \quad & A_i : t : [0, 0] && \mid (\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i] \in \mathcal{K}) \wedge (\phi : [1, 1] \in \mathcal{K}) \wedge \\
& && \quad (\exists t_1, \ldots, t_{\mathsf{up}_i} \in \{1, \ldots, t_{max}\}) \wedge \\
& && \quad (\phi \models A_i : t_1 \wedge \ldots \wedge A_i : t_{\mathsf{up}_i}) \wedge \\
& && \quad (t \notin \{t_1, \ldots, t_{\mathsf{up}_i}\})\} \cup \\
\{ \quad & \mathsf{BLK}(A_i) :< \mathsf{blk}_i && \mid \mathsf{BLK}(A_i) :< \mathsf{blk}_i \in \mathcal{K}\} \cup \\
\{ \quad & \mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i] && \mid \mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i] \in \mathcal{K}\}
\end{aligned}$$

Intuitively, Γ tightens the probability bounds on rules by leveraging probabilistic time formulas using the results we proved in Theorem 4.6 and Corollary 4.7. It tightens the probability bounds on time formulas based on other time formulas, rules, and integrity constraints. This uses the results proved in Lemma 4.5, Theorem 4.8 (and/or Corollary 4.9), and Propositions 4.10-4.11 respectively.

Example 4.8. *Consider the program $\mathcal{K}_{stock}$ from the previous examples. By Definition 4.14, we know that a ptf time-formula* stock_decr(10%) : 5 *will be included in* $\Gamma(\mathcal{K}_{stock})$. *We saw in Example 4.7 that* $\textbf{l\_bnd}(\textit{stock\_decr(10\%)} : 5, \mathcal{K}_{stock}) = 0.0$ *and* $\textbf{u\_bnd}(\textit{stock\_decr(10\%)} : 5, \mathcal{K}_{stock}) = 0.94$. *In the same manner, we can compute that* $\textbf{l\_bnd}(\neg\textit{stock\_decr(10\%)} : 5, \mathcal{K}_{stock}) = 0.0$ *and* $\textbf{u\_bnd}(\neg\textit{stock\_decr(10\%)} : 5, \mathcal{K}_{stock}) = 0.667$ *(this follows from the fact that* $EFR(\phi_{stock} \wedge \neg\textit{stock\_decr(10\%)} : 5, F_1, G_1, 2) \subseteq [0.5, 0.667]$*). Hence, we know that the ptf* stock_decr(10%) : 5 : $[0.333, 0.94]$ *is included in* $\Gamma(\mathcal{K}_{stock})$.

Note that Γ returns an APT-program that is satisfied by the exact same set of interpretations as the original program; this follows directly from the results in the

previous section.

PROPOSITION 4.16. *Suppose I is an interpretation and $\mathcal{K}$ is an APT-program. Then: $I \models \mathcal{K}$ iff $I \models \Gamma(\mathcal{K})$.*

We can also make the following statement about the complexity of the operator.

PROPOSITION 4.17. *One iteration of Γ can be performed in time complexity $O(|\mathcal{K}|^2 \cdot CHK)$ where CHK is the bound on the time it takes to check (for arbitrary time formulas ϕ, ρ) if $\phi \models \rho$ is true.*

One source of complexity is comparing ptf's with other ptf's. If a ptf is formed with an elementary time formula, then it only needs to be compared to other ptf's that share the same time point – this could reduce complexity. As is usual in logic programming, Γ can be iteratively applied as follows.

Definition 4.18. We define multiple applications of Γ as follows.

- $\Gamma(\mathcal{K}) \uparrow 0 = \mathcal{K}$
- $\Gamma(\mathcal{K}) \uparrow (i+1) = \Gamma(\Gamma(\mathcal{K}) \uparrow i)$

Now, we will show that Γ has a least fixed point. First, we define a partial ordering of APT-programs.

Definition 4.19 *Preorder over APT-Programs.* Given $\mathcal{K}_1, \mathcal{K}_2$, we say $\mathcal{K}_1 \sqsubseteq^{pre} \mathcal{K}_2$ if and only if:

- $\forall \phi : [\ell, u] \in \mathcal{K}_1$, $\exists \phi : [\ell', u'] \in \mathcal{K}_2$ s.t. $[\ell', u'] \subseteq [\ell, u]$
- $\forall F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u] \in \mathcal{K}_1$, $\exists F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell', u'] \in \mathcal{K}_2$ s.t. $[\ell', u'] \subseteq [\ell, u]$
- If $\mathsf{BLK}(A_i) :\prec \mathsf{blk}_i \in \mathcal{K}_1$, then $\mathsf{BLK}(A_i) :\prec \mathsf{blk}_i \in \mathcal{K}_2$
- If $\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i] \in \mathcal{K}_1$, then $\mathsf{OCC}(A_i) : [\mathsf{lo}_i, \mathsf{up}_i] \in \mathcal{K}_2$

The intuition behind the above definition is that program $\mathcal{K}_1$ is "below" $\mathcal{K}_2$ if it has less rules or ptf's – or rules/ptf's with tighter probability bounds. Note that if $\mathcal{K}_2$ is above $\mathcal{K}_1$, then $\mathcal{K}_1$ has at least as many satisfying interpretations, and possibly more, than $\mathcal{K}_2$. Let $PROG_{B_\mathcal{L}, t_{max}}$ be the set of all APT-programs given Herbrand base $B_\mathcal{L}$ and time t_{max}. It is easy to see that $\langle PROG_{B_\mathcal{L}, t_{max}}, \sqsubseteq^{pre} \rangle$ is a reflexive and transitive, and therefore a preorder. In the following, we will say that $\mathcal{K}_1 \sim \mathcal{K}_2$, read "$\mathcal{K}_1$ is equivalent to $\mathcal{K}_2$" if and only if $\mathcal{K}_1 \sqsubseteq^{pre} \mathcal{K}_2$ and $\mathcal{K}_2 \sqsubseteq^{pre} \mathcal{K}_1$. The "$\sim$" relation is clearly an equivalence relation; we will use $[\mathcal{K}]$ to denote the equivalence class corresponding to $\mathcal{K}$ w.r.t. this relation.

Definition 4.20 *Partial Ordering of APT-Programs.* Given two equivalence classes $[\mathcal{K}_1], [\mathcal{K}_2]$ w.r.t. relation $\sim$, we say $[\mathcal{K}_1] \sqsubseteq [\mathcal{K}_2]$ if and only if $\mathcal{K}_1 \sqsubseteq^{pre} \mathcal{K}_2$.

The "$\sqsubseteq$" relation is clearly reflexive, antisymmetric, and transitive, and therefore a partial order over sets of APT-programs. Note that when we use the symbol $\sqsubseteq$, we will often write $\mathcal{K}_1 \sqsubseteq \mathcal{K}_2$ as shorthand for $[\mathcal{K}_1] \sqsubseteq [\mathcal{K}_2]$. We will also overload the symbol $PROG_{B_\mathcal{L}, t_{max}}$ to mean "all equivalence classes of APT-programs" (for a given t_{max} and $B_\mathcal{L}$) where appropriate. Therefore, we can now define a complete lattice, where the top element is a set containing all inconsistent programs, and the bottom element is set containing the empty program.

LEMMA 4.21. *Given* $\bot = \{\emptyset\}$ *and* $\top = \{\mathcal{K} \mid \mathcal{K}$ *is inconsistent*$\}$*, then the partial order* $\langle PROG_{B_{\mathcal{L}},t_{max}}, \sqsubseteq\rangle$ *defines a complete lattice.*

What remains to be shown is that Γ is monotonic; if this holds, we can state it has a least fixed point.

LEMMA 4.22. $\mathcal{K} \sqsubseteq \Gamma(\mathcal{K})$.

LEMMA 4.23. Γ *is monotonic.*

By the Tarski-Knaster theorem, Γ has a least fixed point.

THEOREM 4.24. Γ *has a least fixed point.*

4.4 Using Γ for Consistency Checking

As noted earlier, the Γ operator can be used to find "loose" entailment bounds by simply adding an entailment time formula (ϕ) with probability bounds $[0, 1]$ to the logic program, and then examining the tightened bounds after one or more applications of the operator. In this section, we examine how to use Γ for consistency checking. First, we have a straightforward lemma on consistency.

LEMMA 4.25. *Let* $\mathcal{K}$ *be an APT-logic program that entails rule* $F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u]$ *or* $\phi : [\ell, u]$ *such that one of the following is true:*

- $\ell > u$
- $\ell < 0$ *or* $\ell > 1$
- $u < 0$ *or* $u > 1$.

Under this circumstance, $\mathcal{K}$ *is* ***inconsistent****,* i.e.*, there is no interpretation* I *such that* $I \models \mathcal{K}$.

The following result follows immediately.

COROLLARY 4.26. *Let* $\mathcal{K}$ *be an APT-logic program where there exists natural number* i *such that* $\Gamma(\mathcal{K}) \uparrow i$ *entails rule* $F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u]$ *or* $\phi : [\ell, u]$ *such that one of the following is true:*

- $\ell > u$
- $\ell < 0$ *or* $\ell > 1$
- $u < 0$ *or* $u > 1$.

Under this circumstance, $\mathcal{K}$ *is* ***inconsistent****.*

We note that the Γ adds time formulas whose probability bounds is determined by an intersection operation. We observe that an empty intersection of the probability bounds is equivalent to the case where $\ell > u$, which allows us to apply the above corollary to correctly state that the program is not consistent. We illustrate this in the below example.

EXAMPLE 4.9. *Consider* $\mathcal{K}_{stock}$ *from the previous examples. By the definition of* Γ*, the ptf* stock_decr(10%) $\wedge$ cfo_resigns $: 5 : [0.499, 1]$ *is in* $\Gamma(\mathcal{K}_{stock})$*. By Example 4.6, we know that* cfo_resigns $: 5 : [0, 0]$ *is also in* $\Gamma(\mathcal{K}_{stock})$*. However, another application of* Γ *entails* cfo_resigns $: 5 : [0.499, 0]$ *(equivalently,* cfo_resigns $: 5 : \emptyset$*). As* $0.499 > 0$*, we know that* $\mathcal{K}_{stock}$ *is not consistent.*

In addition to checking consistency with the Γ operator, we can check for inconsistencies based on the block and occurrence ICs via the following result.

PROPOSITION 4.27. *If there does not exist at least one thread that satisfies all integrity constraints in an APT-logic program, then that program is inconsistent.*

The **Thread Existence Problem (ThEX)** problem is that of checking existence of a thread that satisfies all block and integrity constraints. Here we show that **ThEX** can be solved in constant time – this can allow us to quickly identify certain inconsistencies in an APT-program. First, we define a partial thread.

Definition 4.28. A *partial thread PTh* is a thread such that for all $1 \leq i \leq t_{max}$, $PTh(i)$ is a singleton set.

For any ground atom A_i with a single associated block-size and occurrence constraint[6] if more than $\left\lceil \frac{(\mathsf{blk}_i - 1) \cdot t_{max}}{\mathsf{blk}_i} \right\rceil$ worlds must satisfy A_i in each partial thread, then all partial threads will have a block of size blk_i. This allows us to derive the following results.

PROPOSITION 4.29. *If* $\mathsf{lo}_i > \left\lceil \frac{(\mathsf{blk}_i - 1) \cdot t_{max}}{\mathsf{blk}_i} \right\rceil$ *then there does not exist a partial thread for ground atom* A_i *such that the single block-size and occurrence IC associated with* A_i *hold.*

PROPOSITION 4.30. *For ground atom* A_i *(with associated ICs), if* $\mathsf{up}_i > \left\lceil \frac{(\mathsf{blk}_i - 1) \cdot t_{max}}{\mathsf{blk}_i} \right\rceil$ *we know that the number of worlds satisfying* A_i *cannot be in the range* $\left[\left\lceil \frac{(\mathsf{blk}_i - 1) \cdot t_{max}}{\mathsf{blk}_i} \right\rceil, up_i \right]$.

The reason for this is simple: it would force the partial thread to have a sequence of blk_i consecutive worlds satisfying A_i. We also notice that these checks can be performed based solely on the values of $\mathsf{lo}_i, \mathsf{up}_i, \mathsf{blk}_i$, and t_{max}. Hence, we have the following proposition.

PROPOSITION 4.31. ***ThEX*** *can be solved in constant time.*

In the next section we extend these results for non-ground APT-programs.

5. CONSISTENCY AND ENTAILMENT ALGORITHMS FOR NON-GROUND PROGRAMS

The fixpoint procedure described via the Γ operator works in the ground case. In this section, we study how we may avoid grounding. We start (Section 5.1) with a sampling based approach for consistency checking of non-ground programs. Section 5.2 defines a non-ground fixpoint operator for entailment queries. This operator avoids grounding the entire program, but guaranteed to provide entailment bounds for a query that are as tight as our ground operator. We remind the reader that both our consistency-checking algorithm and our fixpoint operator presented in this section are sound, but not complete.

[6]There is no loss of generality looking at just one block-size IC per ground atom as multiple such ICs can be coalesced into one by taking the minimum; likewise, there is no loss of generality in considering just one occurrence per ground atom as they can be merged into one by intersecting the $[\mathsf{lo}, \mathsf{up}]$ intervals for that atom.

5.1 Consistency Checking for Non-Ground Programs

In this section, we present a sound algorithm for consistency checking of non-ground programs. We avoid complete grounding of the rules, while still maintaining soundness of the algorithm through random sampling of ground instances of rules. The larger the sample, the more potential inconsistencies can be found.

For a non-ground time formula, ϕ_{ng}, we shall use the notation $gnd(\phi_{ng})$ to refer to the ground formula $\bigwedge\{\phi \mid \textit{is a ground instance of } \phi_{ng}\}$. We are now ready to describe a non-ground analog to the bounds EFR described in the previous section.

Definition 5.1. For non-ground formulas F_{ng}, G_{ng}, time Δt, and non-ground time formula ϕ_{ng}, we define

(1)

$$EFR\_SET(F_{ng}, G_{ng}, \Delta t, \phi_{ng}) = \{EFR(F, G, \Delta t, gnd(\phi_{ng}))| \\ F, G \textit{ are ground instances of } F_{ng}, G_{ng}\}$$

(2)

$$EFR\_IN(F_{ng}, G_{ng}, \Delta t, \phi_{ng}) = (\alpha_{in}, \beta_{in})$$

Where $\exists[\alpha_{in}, \beta'], [\alpha', \beta_{in}] \in EFR\_SET(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$, and $\nexists[\alpha^*, \beta''], [\alpha'', \beta^*] \in EFR\_SET(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$ s.t. $\alpha^* > \alpha_{in}$ and $\beta^* < \beta_{in}$

(3)

$$EFR\_OUT(F_{ng}, G_{ng}, \Delta t, \phi_{ng}) = [\alpha_{out}, \beta_{out}]$$

Where $\exists[\alpha_{out}, \beta'], [\alpha', \beta_{out}] \in EFR\_SET(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$, and $\nexists[\alpha^*, \beta''], [\alpha'', \beta^*] \in EFR\_SET(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$ s.t. $[\alpha^*, \beta^*] \supset [\alpha_{out}, \beta_{out}]$

The intuition behind Definition 5.1 is as follows. *EFR_SET* is the set of all frequency bounds for the different ground instances of F_{ng}, G_{ng}. *EFR_IN* is a pair consisting of the greatest lower bound of *efr* (α_{in}) and the least upper bound of *efr* (β_{in}) of all the elements of *EFR_SET*. $(\alpha_{in}, \beta_{in})$ is a tuple, not a bound. It is possible for $\alpha_{in} > \beta_{in}$. *EFR_OUT* represents the tight bound of efr for any ground instance of F_{ng}, G_{ng}. We now prove these bounds to be tight.

LEMMA 5.2. *Suppose F_{ng}, G_{ng} are non-ground formulas, time $\Delta t > 0$ is an integer, and ϕ_{ng} is a non-ground time formula. Let $(\alpha_{in}, \beta_{in}) = EFR\_IN(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$ and $[\alpha_{out}, \beta_{out}] = EFR\_OUT(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$. If $Th \models \phi_{ng}$, then:*

(1) for all ground instances F, G of F_{ng}, G_{ng} we have $efr(F, G, \Delta t, Th) \in [\alpha_{out}, \beta_{out}]$
(2) there exist ground instances F, G of F_{ng}, G_{ng}, and we have $efr(F, G, \Delta t, Th) \geq \alpha_{in}$
(3) there exist ground instances F, G of F_{ng}, G_{ng}, and we have $efr(F, G, \Delta t, Th) \leq \beta_{in}$

Note that if we were to use the techniques of Section 4 for entailment, we would most likely need to find tight bounds on the elements in the tuple returned by $EFR\_OUT(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$ (specifically a tight lower bound on EFR – as we can be sure that for all ground instances F, G of F_{ng}, G_{ng} that $EFR(F, G, \Delta t, gnd(\phi_{ng}))$ will fall within these bounds). However, there are a few difficulties with this. First,

we conjecture that to find a good bound on *EFR_OUT*, we would most likely have to examine all combinations of ground instances of F_{ng}, G_{ng} – which is most likely equivalent to grounding out the logic program and using Γ. Second, even if we could efficiently find tight bounds on *EFR_OUT*, they would most likely be trivial – i.e., $[0, 1]$.

Conversely, consider the tuple $(\alpha_{in}, \beta_{in}) = \mathit{EFR\_IN}(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$. We know that for all ground instances F, G of F_{ng}, G_{ng} such that for $[\alpha, \beta] = \mathit{EFR}(F, G, \Delta t, \mathit{gnd}(\phi_{ng}))$, we have $\alpha_{in} \geq \alpha$ and $\beta_{in} \leq \beta'$. We also know that finding a lower bound on α_{in} and an upper bound on β_{in} can be done by simply considering any subset of combinations of ground instances of F_{ng} and G_{ng}.

EXAMPLE 5.1. *Consider $\mathcal{K}_{train}$ from Figure 4 with $t_{max} = 4$. Suppose we add the following ptf (called ϕ) to the program.*

$$\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnA}) : 1 \wedge \ \mathsf{adjEast}(\mathit{stnA}, \mathit{stnB}) : 1 \wedge$$
$$\neg(\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnA}) : 2) \wedge \ \mathsf{at\_station}(\mathsf{train1}, \mathsf{stnA}) : 3 \ : [1, 1]$$

Clearly, as $\mathit{EFR}(\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnA}) \wedge \mathsf{adjEast}(\mathit{stnA}, \mathit{stnB}), \mathsf{at\_station}(\mathsf{train1}, \mathsf{stnA}), 2, \phi) = [1, 1]$*, we know for* $(\alpha_{in}, \beta_{in}) = \mathit{EFR\_IN}(\mathsf{at\_station}(\mathsf{T}, \mathsf{S}_1) \wedge \mathsf{adjWest}(\mathsf{S}_1, \mathsf{S}_2), \mathsf{at\_station}(\mathsf{T}, \mathsf{S}_2), 2, \phi)$*, that* $\alpha_{in} = 1$ *and* $\beta_{in} \leq 1$.

Algorithm 1 Finds bounds on *EFR_IN*

FIND-EFR-IN($\mathbf{F}_{sam}, \mathbf{G}_{sam}$ *subsets of ground instances of non-ground formulas* $F_{ng}, G_{ng}, \Delta t$ *natural number* , ϕ_{ng} *non-ground time formula*), returns natural numbers $\alpha^-_{in}, \beta^+_{in}$

(1) Compute $\mathit{gnd}(\phi_{ng})$
(2) Set $\alpha^-_{in} = 0$ and $\beta^+_{in} = 1$
(3) For each $F \in \mathbf{F}_{sam}$
  (a) For each $G \in \mathbf{G}_{sam}$
    i. Let $(\alpha, \beta) = \mathit{EFR}(F, G, \Delta t, \mathit{gnd}(\phi_{ng}))$
    ii. $\alpha^-_{in} = \max(\alpha, \alpha^-_{in})$
    iii. $\beta^+_{in} = \min(\beta, \beta^+_{in})$

Algorithm 1 leverages this technique – if ϕ_{ng} is already ground, algorithm FIND-EFR-IN runs in time quadratic in the size of the sample of ground instances of F_{ng}, G_{ng}. Clearly, this simple algorithm is guaranteed to return a lower bound on α_{in} and an upper bound on β_{in}.

This information can be leveraged in order to perform consistency checks similar to those described in Section 4.4 without resorting to fully grounding out F_{ng}, G_{ng} and considering all combinations of those ground instances. The intuition is simple – if there is just one ground instance of a non-ground rule where $\ell > u$, then the program is inconsistent. The theorem and corollary below mirror Theorem 4.6 and Corollary 4.7 (Page 16) that we described in Section 4.2 for the ground case.

THEOREM 5.3. *Let $\mathcal{K}^{(ng)}$ be a non-ground APT-program that contains the following:*

$$\textbf{\textit{Non-ground rule:}}\ F_{ng} \overset{efr}{\rightsquigarrow} G_{ng} : [\Delta t, \ell, u]$$
$$\textbf{\textit{Non-ground ptf:}}\quad \phi_{ng} : [1, 1]$$

and $(\alpha_{in}, \beta_{in}) = EFR\_IN(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$. If we are given $\alpha^-_{in} \leq \alpha_{in}$ and $\beta^+_{in} \geq \beta_{in}$, then, $\mathcal{K}^{(ng)}$ is not consistent if either $\alpha^-_{in} > u$ or $\beta^+_{in} < \ell$.

COROLLARY 5.4. *Let $\mathcal{K}^{(ng)}$ be a non-ground APT-program that contains the following:*

$$\textbf{\textit{Non-ground rule:}}\ F_{ng} \overset{efr}{\rightsquigarrow} G_{ng} : [\Delta t, \ell, u]$$
$$\textbf{\textit{Non-ground ptf:}}\quad \phi_{ng} : [\ell', u']$$

and $(\alpha_{in}, \beta_{in}) = EFR\_IN(F_{ng}, G_{ng}, \Delta t, \phi_{ng})$. If we are given $\alpha^-_{in} \leq \alpha_{in}$ and $\beta^+_{in} \geq \beta_{in}$, then, $\mathcal{K}^{(ng)}$ is not consistent if $\alpha^-_{in} \cdot \ell' > u$.

Algorithm 2 is a sound (but not complete) method to quickly check for inconsistency in the non-ground case.

Algorithm 2 Checks for inconsistencies in a non-ground program

NG-INCONSIST-CHK($\mathcal{K}^{(ng)}$ *non-ground program*)
returns list of rules that cause inconsistencies

(1) Let L be a list of rules initialized to $\emptyset$
(2) For each ptf $\phi_{ng} : [\ell', u'] \in \mathcal{K}^{(ng)}$ where $u' = 1$, do the following.
  (a) For each rule $F_{ng} \overset{efr}{\rightsquigarrow} G_{ng} : [\Delta t, \ell, u] \in \mathcal{K}^{(ng)}$, do the following.
    i. Generate sample sets $\mathbf{F}_{sam}, \mathbf{G}_{sam}$ of ground instances of F_{ng}, G_{ng}.
    ii. Let $(\alpha^-_{in}, \beta^+_{in}) = \text{FIND-EFR-IN}(\mathbf{F}_{sam}, \mathbf{G}_{sam}, \Delta t, \phi_{ng})$
    iii. If $\alpha^-_{in} \cdot \ell' > u$, then add $F_{ng} \overset{efr}{\rightsquigarrow} G_{ng} : [\Delta t, \ell, u] \in \mathcal{K}^{(ng)}$ to L
    iv. Elseif $\ell' = 1$ and $\beta^+_{in} < \ell$, then add $F_{ng} \overset{efr}{\rightsquigarrow} G_{ng} : [\Delta t, \ell, u] \in \mathcal{K}^{(ng)}$ to L
(3) Return list L

PROPOSITION 5.5. *If the list returned by NG-INCONSIST-CHK contains any elements, then $\mathcal{K}^{(ng)}$ is not consistent.*

Note that the algorithm performs only a quadratic number of comparisons.

PROPOSITION 5.6. *NG-INCONSIST-CHK performs $O(|\mathcal{K}^{(ng)}|^2)$ comparisons.*[7]

[7]**Note:** each comparison requires generating samples of ground instances of two formulas in a rule and running FIND-EFR-IN.

5.2 Entailment for the Non-Ground Case

In this section, we introduce a non-ground operator, $\Lambda_{\mathcal{K}^{(ng)}}$, that maps ground programs to ground programs. Using the same lattice of APT-programs we used in Section 4.3, we show that $\Lambda_{\mathcal{K}^{(ng)}}$ also has a least fixed point. Our intuition is as follows. Suppose we want to find the tightest entailment bounds on some ptf ϕ; if we compute $lfp(\Lambda_{\mathcal{K}^{(ng)}}(\phi : [0,1]))$, the result will be an APT-program (let us call this program $\mathcal{K}_\phi$) s.t. $lfp(\Gamma(\mathcal{K}_\phi))$ will provide the same entailment bounds on ϕ as if we had computed the least fixed point of Γ on the grounding of $\mathcal{K}^{(ng)}$. However, in most cases, $\mathcal{K}_\phi$ will be much smaller than the grounding of $\mathcal{K}^{(ng)}$.

Definition 5.7. For non-ground program $\mathcal{K}^{(ng)}$ and ground program $\mathcal{K}$ (note that $formula(\mathcal{K})$ is a set of ground formulas, as defined in Definition 4.14), $\Lambda_{\mathcal{K}^{(ng)}}$ maps ground programs to ground programs and is defined as follows. $\Lambda_{\mathcal{K}^{(ng)}}(\mathcal{K}) =$

$$
\begin{array}{rl}
\mathcal{K} & \cup \\
\{F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u] | & F \overset{efr}{\rightsquigarrow} G : [\Delta t, \ell, u] \textit{ is a ground instance of a rule in } \mathcal{K}^{(ng)} \textit{ s.t.} \\
& \exists \phi \in \textit{formula}(\mathcal{K}) \textit{ where } \phi \textit{ is ground and} \\
& \exists t \in [1, t_{max}] \textit{ s.t. } \phi \models F : t \textit{ or } \phi \models G : t \\
& \textit{or } \phi \models \neg F : t \textit{ or } \phi \models \neg G : t\} \cup \\
\{\rho : [\ell, u] | & \rho : [\ell, u] \textit{ is a ground instance of a ptf in } \mathcal{K}^{(ng)} \textit{ s.t.} \\
& \exists \phi \in \textit{formula}(\mathcal{K}) \textit{ where } \phi \textit{ is ground and } \phi \models \rho \\
& \textit{or } \phi \models \neg\rho\} \cup \\
\{\mathsf{BLK}(A) :< \mathsf{blk} | & \mathsf{BLK}(A) :< \mathsf{blk} \textit{ is a ground instance of a constraint in } \mathcal{K}^{(ng)} \textit{s.t.} \\
& \exists \phi \in \textit{formula}(\mathcal{K}) \textit{ where } \phi \textit{ is ground and} \\
& \exists t \in [1, t_{max}] \textit{ s.t. } \phi \models A : t \textit{ or } \phi \models \neg A : t\} \cup \\
\{\mathsf{OCC}(A) : [\mathsf{lo}, \mathsf{up}] | & \mathsf{OCC}(A) : [\mathsf{lo}, \mathsf{up}] \textit{ is a ground instance of a constraint in } \mathcal{K}^{(ng)} \textit{s.t.} \\
& \exists \phi \in \textit{formula}(\mathcal{K}) \textit{ where } \phi \textit{ is ground and } \} \\
& \exists t \in [1, t_{max}] \textit{ s.t. } \phi \models A : t \textit{ or } \phi \models \neg A : t\}
\end{array}
$$

We will now present an example for this operator.

EXAMPLE 5.2. *Recall $\mathcal{K}_{train}$ from Figure 4 with $t_{max} = 4$. The following rules comprise the set $\Lambda_{\mathcal{K}_{train}}(\{\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) : 4\})$:*

$$
\begin{array}{rl}
& \mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) : 4 \\
\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnA}) \wedge \mathsf{adjEast}(\mathsf{stnA}, \mathsf{stnB}) & \overset{efr}{\rightsquigarrow} \mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) : [4, 0.85, 1.0] \\
\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) \wedge \mathsf{adjEast}(\mathsf{stnB}, \mathsf{stnB}) & \overset{efr}{\rightsquigarrow} \mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) : [4, 0.85, 1.0] \\
\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnC}) \wedge \mathsf{adjEast}(\mathsf{stnC}, \mathsf{stnB}) & \overset{efr}{\rightsquigarrow} \mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) : [2, 0.85, 1.0] \\
\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnA}) \wedge \mathsf{adjWest}(\mathsf{stnA}, \mathsf{stnB}) & \overset{efr}{\rightsquigarrow} \mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) : [2, 0.6, 0.7] \\
\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) \wedge \mathsf{adjWest}(\mathsf{stnB}, \mathsf{stnB}) & \overset{efr}{\rightsquigarrow} \mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) : [2, 0.6, 0.7] \\
\mathsf{at\_station}(\mathsf{train1}, \mathsf{stnC}) \wedge \mathsf{adjWest}(\mathsf{stnC}, \mathsf{stnB}) & \overset{efr}{\rightsquigarrow} \mathsf{at\_station}(\mathsf{train1}, \mathsf{stnB}) : [2, 0.6, 0.7]
\end{array}
$$

We use the same partial ordering and lattice from Section 4.3, and show the monotonicity of $\Lambda_{\mathcal{K}^{(ng)}}$ as follows.

LEMMA 5.8. *$\mathcal{K} \sqsubseteq \Lambda_{\mathcal{K}^{(ng)}}(\mathcal{K})$ w.r.t. $\langle PROG_{B_{\mathcal{L}},t_{max}}, \sqsubseteq\rangle$*

LEMMA 5.9. *$\Lambda_{\mathcal{K}^{(ng)}}$ is monotonic.*

Now, we show that $\Lambda_{\mathcal{K}^{(ng)}}$ has a least fixed point.

Definition 5.10. We define multiple applications of Λ as follows.

- $\Lambda_{\mathcal{K}^{(ng)}}(\mathcal{K}) \uparrow 0 = \mathcal{K}$
- $\Lambda_{\mathcal{K}^{(ng)}}(\mathcal{K}) \uparrow (i+1) = \Lambda_{\mathcal{K}^{(ng)}}(\Lambda_{\mathcal{K}^{(ng)}}(\mathcal{K}) \uparrow i)$

THEOREM 5.11. *$\Lambda_{\mathcal{K}^{(ng)}}$ has a least fixed point.*

The next two results demonstrate the soundness of Λ. Given non-ground program $\mathcal{K}^{(ng)}$, let *ground*($\mathcal{K}^{(ng)}$) be the grounding of this program. The lemma below follows directly from the definition of the operator. It states that the least fixed point of the operator is a subset of the grounding of $\mathcal{K}^{(ng)}$.

LEMMA 5.12. *Given non-ground program $\mathcal{K}^{(ng)}$, and ground program $\mathcal{K}$, lfp($\Lambda_{\mathcal{K}^{(ng)}}(\mathcal{K})$) $\subseteq$ ground($\mathcal{K}^{(ng)}$) $\cup\, \mathcal{K}$.*

Additionally, the following result states that, for a given entailment query, we obtain the same result whether we use $\Lambda_{\mathcal{K}^{(ng)}}$ or simply ground out $\mathcal{K}^{(ng)}$.

THEOREM 5.13. *Given non-ground program $\mathcal{K}^{(ng)}$*

$$\phi : [\ell, u] \in lfp(\Gamma(lfp(\Lambda_{\mathcal{K}^{(ng)}}(\{\phi : [0,1]\}))))$$

iff

$$\phi : [\ell, u] \in lfp(\Gamma(ground(\mathcal{K}^{(ng)}) \cup \{\phi : [0,1]\}))$$

In the next section, we will show the results of our experimental analysis of the algorithms provided in the previous sections for the non-ground case.

6. EXPERIMENTAL RESULTS

This section reports on experiments carried out in the ground case with our fixpoint algorithm. We demonstrate the Γ operator on 23 different ground APT-programs automatically extracted from two different data sets using a slight improvement of the APT-EXTRACT algorithm from [Shakarian et al. 2011]. We were able to compute fixpoints of APT-programs consisting of over 1,000 ground rules in about 20 minutes (see the left-hand side of Figure 6). Note that this is the time to compute the fixpoint, not to perform a deduction (i.e., via the Λ operator), which can be done for specific entailment queries, and would be faster.

This section is organized as follows. Section 6.1 describes our experimental setup, data set, and how we extracted rules, integrity constraints, and ptf's while Section 6.2 examines the runtime of the fixpoint operator.

6.1 Experimental Setup

All experiments were run on multiple multi-core Intel Xeon E5345 processors at 2.33GHz, 8GB of memory, running the Scientific Linux distribution of the GNU/Linux OS, kernel version 2.6.9-55.0.2.ELsmp.[8] Our implementation consists of approximately 4,000 lines of Java code (JDK 1.6.0).

Iraq Special Groups (ISW) This data-set contains daily counterinsurgency events from Baghdad in 2007-2008. The event data was provided by the Institute for the Study of War (ISW) and augmented with neighborhood data from the International Medical Corps. The historical data was represented with 187 ground atoms over 567 days – which is the time granularity we used. Using the APT-Extract algorithm (presented in Shakarian et al. [2011]), we extracted 3,563 ground rules using the *efr* frequency function.

We considered 13 logic programs from this dataset; each smaller program is a subset of any of the larger ones, so we have $\mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq \ldots \subseteq \mathcal{K}_{12} \subseteq \mathcal{K}_{13}$. In each program, we included a prefix consisting of 50 worlds (for more on prefixes, refer to Definition 2.11 on Page 9). The same prefix was used for each ISW program. We set $t_{max} = 60$ for all ISW programs. Additionally, for all ground atoms appearing in a given program, we added the appropriate block and occurrence integrity constraints. Later we will present our extraction algorithms for these constraints.

Minorities at Risk Organizational Behavior (MAROB) This data set contains yearly attributes for a variety of political and violent groups over a period of 25 years [Wilkenfeld et al. 2007]. Overall, we have extracted over 21.4 million APT-rules from this data set. These rules were also extracted using APT-EXTRACT with the *efr* frequency function.

We considered 10 APT-logic programs from this dataset, each corresponding to a different group. As each of these logic programs is associated with actions for a specific group, all 10 of the MAROB programs are pairwise disjoint. In each MAROB program, we included a unique prefix of 10 worlds specific to the group in the program. We set $t_{max} = 13$ for each MAROB program. Block-size and occurrence constraints were also included in each program. Table I provides some information on these APT-programs.

While integrity constraints (as with rules) could come from an expert, we decided to extract our ICs from the data. We have included the straightforward algorithms OC-EXTRACT and BLOCK-EXTRACT to show how we extracted occurrence and block-size IC's (respectively) for each of the 187 atoms in the data set.

PROPOSITION 6.1. *OC-EXTRACT runs in time* $O((n - t_{max}) \cdot t_{max})$.

PROPOSITION 6.2. *There are no historical threads such that atom* a_i *is satisfied by less than* lo_i *or more than* up_i *worlds when* $\mathsf{lo}_i, \mathsf{up}_i$ *are produced by OC-EXTRACT.*

PROPOSITION 6.3. *BLOCK-EXTRACT runs in time* $O(n)$.

PROPOSITION 6.4. *Given* blk_i *as returned by BLOCK-EXTRACT, there is no sequence of* blk_i *or more consecutive historical worlds that satisfy atom* a_i.

[8] We note that this implementation makes use of only one processor and one core for a single run, though different runs were distributed across the cluster.

Algorithm 3 Extracts occurrence constraints

OC-EXTRACT(a_i *ground atom* $, W_1, \ldots, W_n$ *historical worlds*, t_{max} *maximum time*), returns natural numbers $\mathsf{lo}_i, \mathsf{up}_i$

(1) Set $\mathsf{up}_i = 0$ and $\mathsf{lo}_i = t_{max}$
(2) For $i = 1$, $i \leq n - t_{max} + 1$, loop
  (a) Set $cur = 0$
  (b) For $j = i$, $j < i + t_{max}$ loop
    i. If $W_j \models a_i$, then $cur = cur + 1$
  (c) If $cur < \mathsf{lo}_i$ then set $\mathsf{lo}_i = cur$
  (d) If $cur > \mathsf{up}_i$ then set $\mathsf{up}_i = cur$
(3) Return $\mathsf{lo}_i, \mathsf{up}_i$

Algorithm 4 Extracts block-size constraints

BLOCK-EXTRACT(a_i *ground atom* $, W_1, \ldots, W_n$ *historical worlds*), returns natural number blk_i

(1) Set $cur = 0$
(2) Set $best = 0$
(3) For $i = 1$, $i \leq n$, loop
  (a) If $W_i \models a_i$
    i. $cur = cur + 1$
  (b) Else
    i. If $cur > best$ then set $best = cur$
    ii. Set $cur = 0$
(4) If $cur > best$ set $best = cur$
(5) Set $\mathsf{blk}_i = best + 1$
(6) Return blk_i

6.2 Run Time Evaluation

To evaluate performance, for each logic program, we clocked 10 trials until Γ reached a fixpoint. In all our trials, a fixpoint was reached after only two or three applications (see Table I). We also note that the experimental relationship between run time and the number of rules was linear – we conducted a statistical R^2-test for this and came up with an R^2 value of 0.97 for ISW programs and 0.77 for MAROB programs (refer to Figure 6). We must point out that the disjoint relationship among MAROB programs may account for why the run time relationship is not as linear as that for the ISW programs. This graceful degradation in performance is most likely due to the fact that the number of rules/ptfs that can tighten the bound of a given rule or ptf is much smaller than the set of entire rules, which makes the running time of the inner loop very small. Hence, for practical purposes, the $O(|\mathcal{K}|^2)$ is a loose bound; this worst case is likely to occur only in very rare circumstances.

We checked entailment by looking at the probability bounds of formulas in

Program	Gr. Rules	Post. Gr. Atoms	Range of Δt	t_{max}	Time Points	Γ App.
$\mathcal{K}_1$	92	76	[2,10]	60	567	2
$\mathcal{K}_2$	102	76	[2,10]	60	567	3
$\mathcal{K}_3$	126	76	[2,10]	60	567	3
$\mathcal{K}_4$	144	76	[2,10]	60	567	2
$\mathcal{K}_5$	169	76	[2,10]	60	567	2
$\mathcal{K}_6$	214	76	[2,10]	60	567	3
$\mathcal{K}_7$	241	76	[2,10]	60	567	3
$\mathcal{K}_8$	278	76	[2,10]	60	567	3
$\mathcal{K}_9$	360	79	[2,10]	60	567	3
$\mathcal{K}_{10}$	503	80	[2,10]	60	567	3
$\mathcal{K}_{11}$	644	80	[2,10]	60	567	3
$\mathcal{K}_{12}$	816	80	[2,10]	60	567	3
$\mathcal{K}_{13}$	1081	84	[2,10]	60	567	3
$\mathcal{K}_H$	586	189	[2,3]	13	23	3
$\mathcal{K}_J$	679	192	[3,3]	13	25	2
$\mathcal{K}_A$	661	162	[2,3]	13	25	2
$\mathcal{K}_B$	163	175	[3,3]	13	24	2
$\mathcal{K}_D$	539	176	[3,3]	13	25	2
$\mathcal{K}_{FT}$	482	188	[2,3]	13	22	3
$\mathcal{K}_{FR}$	310	177	[3,3]	13	25	2
$\mathcal{K}_{HA}$	458	168	[3,3]	13	13	2
$\mathcal{K}_{HI}$	330	182	[2,3]	13	25	2
$\mathcal{K}_K$	94	181	[1,3]	13	25	3

Table I. APT-logic programs used in the run time evaluations. Programs $\mathcal{K}_1 - \mathcal{K}_{13}$ (top) are based on the ISW data-set, while the rest (bottom) are based on MAROB.

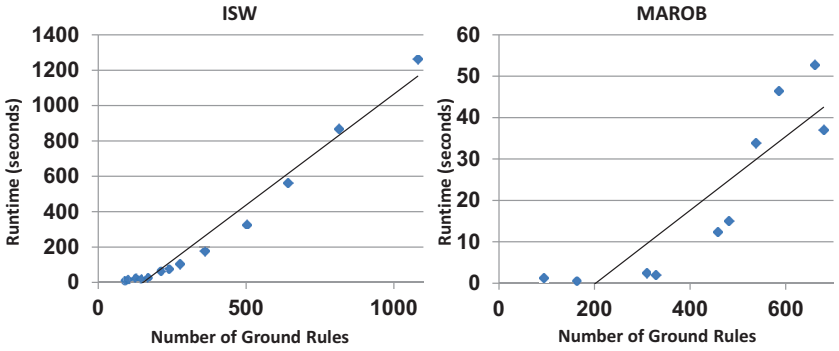

Fig. 6. Number of ground rules vs. run time (Left: ISW, Right: MAROB). Note these run-times include the full computation of the fixed point of the Γ operator.

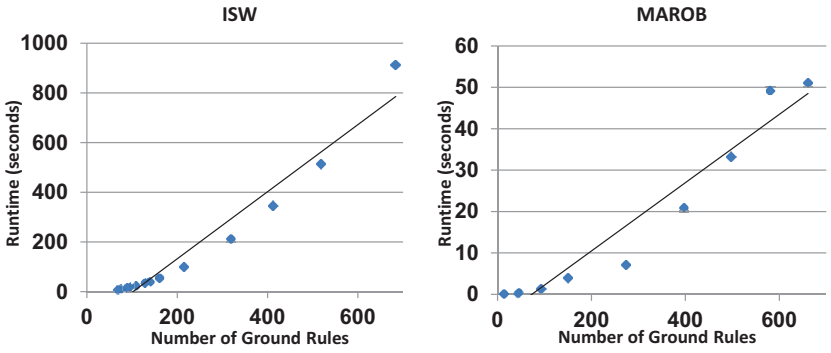

Fig. 7. Number of ground rules vs. run time for entailment checking (Left: ISW, Right: MAROB).

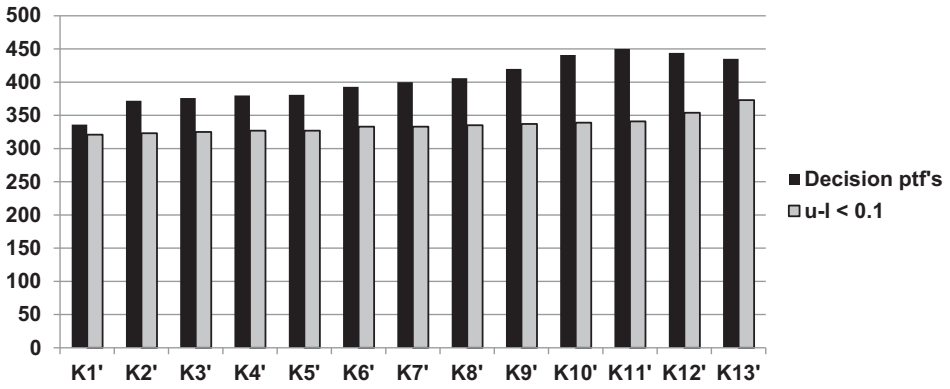

Fig. 8. Attributes of ptf's entailed by the different logic programs (ISW dataset)

formula($\mathcal{K}$) (see Definition 4.14), which is obtained by finding the fixpoint for the Γ operator on a consistent APT-program. After our initial runs of Γ on the 23 logic programs, we found that 21 of them were inconsistent. As inconsistencies are found in a constructive way (refer to Section 4.4 on Page 21), we could eliminate rules that caused inconsistencies (we designate the "consistent" subset of a program with a tick mark, i.e., $\mathcal{K}_2'$ is $\mathcal{K}_2$ with inconsistency-causing rules removed). Using these "consistent" APT-programs, we first looked to revalue the performance of the Γ operator for entailment. Unsurprisingly, as with the run time evaluation we performed for consistency checking, we found that the run time was related linearly to the number of ground rules considered. We obtained R^2 values of 0.95 for ISW programs and 0.94 for MAROB programs. See Figure 7 for details; run times are based on the average of 10 trials for each logic program.

As a consequence of Definition 4.14 (Page 19), the logic program returned by multiple applications of Γ includes several ptf's not in the original program. These ptf's were either based on formulas seen in the rules, or atoms seen in the rules where

an integrity constraint forces the associated atomic ptf to be assigned probability 0. Many of these ptf's have probability bounds tighter than $[0, 1]$ – some extremely tight. We note, as shown in Figure 8, that all of our ISW logic program produce over 300 ptfs where the difference between ℓ and u is less than 0.1 (the number steadily increases with larger ISW programs[9]). We also looked at "decision ptf's"; these are ptf's where either $\ell \leq 0.5$ or $u \leq 0.5$ – the intuition is that the probability mass is either above or below 0.5, allowing a user to make a decision. The Γ operator also was successful in producing many ptf's of this type, producing well over 400 in over half of the logic programs we considered from the ISW dataset.

7. RELATED WORK

APT-Logic distinguishes itself from other temporal logics in the following ways: (i) It supports reasoning about probability of events over time, (ii) Future worlds can depend on more than just the current world (i.e., it does not assume the Markov property). (iii) It provides probability bounds instead of a point probability. (iv) No independence assumptions are made.

Though probabilistic extensions to different logics [Nilsson 1986; Fagin et al. 1990; Giugno and Lukasiewicz 2002] and probabilistic logic programs [Ng and Subrahmanian 1992; Kern-Isberner and Lukasiewicz 2004; Dekhtyar and Dekhtyar 2005] have been extensively studied, together with approaches to logic programming with uncertainty [Kifer and Lozinskii 1992], as well as approaches to learn such programs [De Raedt and Kersting 2003], logic programming with time has not been extensively studied. [Dekhtyar et al. 1999] was the first such effort and provided a declarative semantics for temporal probabilistic LPs. The problem of finding entailment bounds on a formula was not studied there. Moreover, that logic did not include frequency functions. No implementation was proposed and thus no experimental results were studied. [Dix et al. 2006] provided a logic programming language within which agents that reason about time and uncertainty could be encoded. Such agents could make statements about when they were obliged/permitted to take certain actions and when they were forbidden from doing so. A fixpoint semantics for such agents was provided, but there was no notion corresponding to a frequency function, no way of reducing the complexity of the problem, and moreover, no analog of our entailment problem was studied. Our work builds on that in [Shakarian et al. 2011], where APT-programs were proposed and consistency/entailment of ground atoms is studied. However, it did not provide practical algorithms for entailment, or had implementation/experiments.

[Mateus et al. 2001] introduce an extension to the Situation Calculus for handling actions with uncertain effects. The semantics of their logical language is given in terms of a "Randomly Reactive Automaton", which allows for probabilistic effects of actions but has no notion of the current time apart from that implied by the sequence of actions. They examine next move determination where the results of a move are dependent on the move chosen as well as on draws from single or from multiple distributions.

[9]It is important to point out that all numbers of ptf's with tight bounds are associated with a world outside the range of the prefix.

Santos and Young [Santos and Young 1999] propose the Probabilistic Temporal Network model (PTNs), which allows to represent temporal (and atemporal) information in combination with probabilistic semantics. PTNs are suitable for representing causality constrained by time, conditions for the occurrence of events (and at what time they occur), and periodic and recurrent processes. This model is based on Bayesian networks (for the probabilistic aspect) and on work by Allen [Allen and Ferguson 1994] on temporal interval algebra for the temporal aspect. Even though this work's goals overlap to some extent with those of our own, the fundamental difference lies in the initial assumptions made. In order to build a PTN, one must have available information regarding dependencies, prior probabilities for all random variables, temporal causal relationships between random variables in temporal aggregates, etc. The focus of our work is to reason about events making no independence assumptions, and only based on limited information relating small subsets of events. The PTN framework is, however, very useful for scenarios in which the required information is available, as is the case in probabilistic reasoning with traditional Bayesian Networks. The key aspect that separates APT-logic from PTN's, is the fact that **APT-logic makes no assumptions about independence.** For example, consider item 1 of Theorem 4.5, one of the key building blocks of our fixpoint heuristic. In this case, if $I \models \phi : [p, p]$ and $\rho : [p', p']$, then $I \models \phi \wedge \rho : [\max(0, p + p' - 1), \min(p, p')]$. If we had assumed independence, then $I \models \phi \vee \rho : [p^2, p^2]$ – clearly a different answer and not appropriate for domains where we do not wish to make assumptions about dependence/independence (i.e., the counter-insurgency data that we used for our experiments). This also is our motivation for the use of probability intervals – rather than point probabilities.

7.1 Work in Verification and PRISM

Logics merging time and probabilities have been studied quite a bit in the area of verification. [Vardi 1985] was one of the pioneers in this, followed by many including probabilistic CTL [Hansson and Jonsson 1994b], and others [Cleaveland et al. 2005]. Building on this work, Kwiatkowska et al. developed a tool known as PRISM [Kwiatkowska et al. 2000; 2009] to perform this type of model checking. PRISM has the following characteristics:

(1) The user specifies a **model** – a discrete-time Markov chain (DTMC), continuous-time Markov chain (CTMC) or Markov decision processes (MDP)
(2) The user also specifies a **property** – which is normally a CTL formula
(3) PRISM returns a **value** (normally a probability or expected value) associated with the property

One can view our implementation in the same light – taking an APT-program as a model, time formula as a property, and returning entailment bounds as the value. However, PRISM operates under some very different assumptions than APT-logic which are appropriate for some applications but not for all.

(1) The **model** specified by the user in PRISM is a stochastic process that assumes the Markov property - that is the probability of being in the next state only depends on the current state and action. Conversely, an APT-program **does not assume the Markov property**. Further, we demonstrated translations

from stochastic processes to APT-programs in Shakarian et al. [2011]. Also, in that paper, we showed how it is easy to construct a very simple APT-program where there is no analogous MDP (using a natural construction).

(2) Based on the **model** specified by the user, PRISM also makes an independence assumption. Suppose we are in initial state S_1 and consider the following sequence of states, actions, and probabilities in an MDP: $S_1 \xrightarrow{a}_{p_1} S_2 \xrightarrow{b}_{p_2} S_3$ which states that "state 1 transitions to state 2 on action a with probability p_1 and state 2 transitions to state on action b with probability p_2." PRISM would calculate the probability of such a sequence – $p_1 \cdot p_2$ – hence it has assumed independence between the two transitions. Likewise, consider the formulas $F(S_1), F(S_2), F(S_3)$ – formulas satisfied exactly by states S_1, S_2, S_3. Using the natural translation described in Shakarian et al. [2011], we can create an analogous APT-program as follows:
- $(F(S_1) \wedge a \wedge \neg b) : 1 \wedge F(S_2) : 2 : [p_1, p_1]$
- $(F(S_2) \wedge b \wedge \neg a) : 2 \wedge F(S_3) : 3 : [p_2, p_2]$

By item 1 of Theorem 4.5, the following ptf is tightly entailed:
$(F(S_1) \wedge a \wedge \neg b) : 1 \wedge (F(S_2) \wedge b \wedge \neg a) : 2 \wedge F(S_3) : 3 : [\max(0, p_1 + p_2 - 1), \min(p_1, p_2)]$
With APT-logic, we allow for uncertainty - **all** we can say about the sequence is it has a probability in $[\max(0, p_1 + p_2 - 1), \min(p_1, p_2)]$ – which is clearly different from $p_1 \cdot p_2$.

(3) The **property** specified by the user in PRISM is based on PCTL [Aziz et al. 1995; Hansson and Jonsson 1994b]. Although there are constructs in PCTL that appear similar to the syntax of APT-logic, as our semantics differ substantially, the statements have different meanings. Even if an MDP is encoded in an APT-program, a "leads-to" PCTL operator (which has a strikingly similar intuition to an APT-rule) has a very different meaning. We explore the specifics of these differences in Shakarian et al. [2011].

Basically, PRISM is best suited for situations where the underlying model can be represented as a stochastic process. Popular applications have included software verification and certain biology problems that can be easily represented as stochastic processes. APT-logic is best suited for situations where there are no independence or Markov assumptions made about the model – which is often the case when we are working with extracted rules. We have shown APT-logic to be viable for studying the actions of militia groups in a counter-insurgency environment. Other applications where APT-logic is well suited include policy analysis and stock price movement.

8. CONCLUSIONS

Logical reasoning with time and probabilities is essential in any application where the occurrence of certain conditions at time t may cause or imply that other phenomena may occur δ units in the future. There are numerous such applications including ones relating to how stock markets will move in the future based on current or past conditions, medicine where the condition of a patient in the future depends on various things true now, behavior modeling where the behavior of an

individual or group in the future may depend on his current/past situation. In addition, most applications where we reason about the future are fraught with uncertainty. Annotated Probabilistic Temporal Logic (APT-logic for short) was first introduced in Shakarian et al. [2011] as a paradigm for reasoning about sentences of the form "If formula F is true at time t, then formula G will be true at time Δt with a probability in the range $[L, U]$." More importantly, APT-logic programs were introduced in a manner that did not require independence or Markovian assumptions, many of which are inapplicable for several applications.

To date, no implementation of probabilistic temporal logic exists that does not make use of Markovian or independence assumptions. To our knowledge, this is the first paper that attempts any implementation of such logics. However, due to the high complexity of such reasoning (which may also explain why implementations may not exist), practical temporal probabilistic reasoning systems may not always be complete.

In this paper, we developed, implemented, and evaluated a fixpoint-based heuristic for consistency and entailment problems in APT-logic programs. This paper makes the following contributions:

(1) We show NP-completeness of the APT-logic consistency problem, and coNP-completeness of the APT-logic entailment problem, extending hardness results in Shakarian et al. [2011].

(2) We developed a **fixpoint based** heuristic from the following observations:
   - The presence of ptf's with the probability of 1 in an APT-program allows us to tightly bound values for frequency functions.
   - The bound on frequency functions, in turn, allows us to tighten the bounds of elements in an APT-program
   - The above two characteristics can be employed in an operator that maps APT-programs to APT-programs and has a least fixed point

(3) We developed consistency and entailment algorithms for the non-ground case.

(4) We implemented our fixpoint heuristic and applied it to 23 *real world* APT-logic programs derived *automatically* from two different real world data sets. This suite of test programs was not written by us. Our experiments show that our fixpoint based heuristical can calculate fixpoints in time roughly linear w.r.t. the number of ground rules

(5) We also show that using our implementation, we can solve the "tight entailment problem" where the goal is to find the tightest interval $[\ell, u]$ such that $F : [t, \ell, u]$ is entailed by an APT-logic program for a given time t and formula F.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: `http://www.acm.org/pubs/citations/journals/tocl/2010-V-N/p1-URLend`.

Acknowledgements. Paulo Shakarian is funded under the US Army ACS/West Point Instructor (EECS) program.
Some of the authors of this paper were funded in part by AFOSR grant FA95500610405, ARO grant W911NF0910206 and ONR grant N000140910685.

REFERENCES

Allen, J. F. and Ferguson, G. 1994. Actions and events in interval temporal logic. *J. of Logic and Computation 4*, 531–579.

Asal, V., Carter, J., and Wilkenfeld, J. 2008. Ethnopolitical violence and terrorism in the middle east. In *Peace and Conflict 2008*, J. Hewitt, J. Wilkenfeld, and T. Gurr, Eds. Paradigm.

Aziz, A., Singhal, V., Balarin, F., Brayton, R. K., and Sangiovanni-vincentelli, A. L. 1995. It usually works: The temporal logic of stochastic systems. Springer, 155–165.

Chvtal, V. 1983. *Linear Programming*. W.H.Freeman, New York.

Cleaveland, R., Iyer, P., and Narasimha, M. 2005. Probabilistic temporal logics via the modal mu-calculus. *Theoretical Computer Science 342,* 2-3, 316–350.

De Raedt, L. and Kersting, K. 2003. Probabilistic logic learning. *SIGKDD Explor. Newsl. 5,* 1, 31–48.

Dekhtyar, A. and Dekhtyar, M. I. 2005. Revisiting the semantics of interval probabilistic logic programs. In *Proc. Intl. Conf. on Logic Programming and Non-Monotonic Reasoning (LPNMR)*. 330–342.

Dekhtyar, A., Dekhtyar, M. I., and Subrahmanian, V. S. 1999. Temporal probabilistic logic programs. In *ICLP 1999*. The MIT Press, Cambridge, MA, USA, 109–123.

Dix, J., Kraus, S., and Subrahmanian, V. S. 2006. Heterogeneous temporal probabilistic agents. *ACM TOCL 7,* 1, 151–198.

Emerson, E. A. and Halpern, J. Y. 1984. "sometimes" and "not never" revisited: on branching versus linear time. Tech. rep., Austin, TX, USA.

Fagin, R., Halpern, J. Y., and Megiddo, N. 1990. A logic for reasoning about probabilities. *Information and Computation 87*, 78–128.

Giugno, R. and Lukasiewicz, T. 2002. P-shoq(d): A probabilistic extension of shoq(d) for probabilistic ontologies in the semantic web. In *JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence*. Springer-Verlag, London, UK, 86–97.

Hansson, H. and Jonsson, B. 1994a. A logic for reasoning about time and probability. *Formal Aspects of Computing 6*, 512–535.

Hansson, H. and Jonsson, B. 1994b. A logic for reasoning about time and reliability. *Formal Aspects of Computing 6*, 102–111.

ISW. 2008. Map of Special Groups Activity in Iraq, Institute for the Study of War.

Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica 4,* 4, 373–395.

Kern-Isberner, G. and Lukasiewicz, T. 2004. Combining probabilistic logic programming with the power of maximum entropy. *Artif. Intell. 157,* 1-2, 139–202.

Kifer, M. and Lozinskii, E. L. 1992. A logic for reasoning with inconsistency. *Journal of Automated Reasoning 9,* 2, 179–215.

Kwiatkowska, M., Norman, G., and Parker, D. 2000. Verifying randomized distributed algorithms with PRISM. In *Proc. Workshop on Advances in Verification (Wave'2000)*.

Kwiatkowska, M., Norman, G., and Parker, D. 2009. Prism: probabilistic model checking for performance and reliability analysis. *SIGMETRICS Perform. Eval. Rev. 36,* 4, 40–45.

Lamport, L. 1980. "sometime" is sometimes "not never": on the temporal logic of programs. In *POPL 1980*. ACM, New York, NY, USA, 174–185.

Lloyd, J. W. 1987. *Foundations of Logic Programming, Second Edition*. Springer-Verlag.

Mateus, P., Pacheco, A., Pinto, J., Sernadas, A., and Sernadas, C. 2001. Probabilistic situation calculus. *AMAI 32*, 393–431(39).

Ng, R. T. and Subrahmanian, V. S. 1992. Probabilistic logic programming. *Information and Computation 101,* 2, 150–201.

Nilsson, N. 1986. Probabilistic logic. *Artificial Intelligence 28*, 71–87.

Roth, D. 1996. On the hardness of approximate reasoning. *Artificial Intelligence 82*, 273–302.

Santos, E. and Young, J. D. 1999. Probabilistic temporal networks: A unified framework for reasoning with time and uncertainty. *Inter. Journal of Approximate Reasoning 20*.

Shakarian, P., Parker, A., Simari, G., and Subramanian, V. 2011. Annotated probabilstic temporal logic. *ACM Transactions on Computational Logic 12,* 2.

Vardi, M. Y. 1985. Automatic verification of probabilistic concurrent finite state programs. *Symp. on Foundations of Comp. Sci. 0*, 327–338.

Wilkenfeld, J., Asal, V., Johnson, C., Pate, A., and Michael, M. 2007. The use of violence by ethnopolitical organizations in the middle east. Tech. rep., National Consortium for the Study of Terrorism and Responses to Terrorism. February.

Received June 2010; accepted March 2011