

Annotated Probabilistic Temporal Logic

Paulo Shakarian, Austin Parker, Gerardo Simari, V.S. Subrahmanian

The semantics of most logics of time and probability is given via a probability distribution over “threads” where a thread is a structure specifying what will be true at different points in time (in the future). When assessing the probabilities of statements such as “Event a will occur within 5 units of time of event b ”, there are many different semantics possible, even when assessing the truth of this statement within a single thread. We introduce the syntax of annotated probabilistic temporal (APT) logic programs and axiomatically introduce the key notion of a frequency function (for the first time) to capture different types of intra-thread reasoning, and then provide a semantics for intra-thread and inter-thread reasoning in APT logic programs parameterized by such frequency functions. We develop a comprehensive set of complexity results for consistency checking and entailment in APT logic programs, together with sound and complete algorithms to check consistency and entailment. The basic algorithms use linear programming, but we then show how to substantially (and correctly) reduce the sizes of these linear programs to yield better computational properties. We describe a real world application we are developing using APT logic programs.

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.” We now give four examples of how such statements might be applied.

Stock Market Prediction. There is ample evidence [Fujiwara et al. 2008] that reports in newspapers and blogs [De Choudhury et al. 2008] have an impact on stock market prices. For instance, major investment banks invest a lot of time, effort and money attempting to learn predictors of future stock prices by analyzing a variety of indicators together with historical data about the values of these indicators. As we will show later in Figure 1, we may wish to write rules such as “The probability that the stock of company C will drop by 10% at time $(T + 2)$ is over 70% if at time T , there is a news report of a rumor of an SEC investigation of the company and (at time T) there is a projected earnings increase of 10%. It is clear that such rules

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2009 ACM 1529-3785/2009/0700-0001 \$5.00

- | |
|--|
| <p>(1) $\text{scandal} \xrightarrow{pfr} \neg\text{scandal} : [1, 0.89, 0.93, 0.8, 1.0]$
 For a given sequence of events, if there is a scandal in the headlines, this will be followed by there not being a scandal in 1 time unit with probability $[0.89, 0.93]$.</p> <p>(2) $\text{sec_rumor} \wedge \text{earn_incr}(10\%) \xrightarrow{pfr} \text{stock_decr}(10\%) : [2, 0.65, 0.97, 0.7, 1.0]$
 For a given sequence of events, if there is a rumor of an SEC investigation and an earnings increase of 10%, then the stock price will decrease by 10% in exactly 2 time units frequency range $[0.7, 1.0]$ and probability $[0.65, 0.97]$.</p> <p>(3) $\text{sec_rumor} \wedge \text{earn_incr}(10\%) \xrightarrow{pfr} \text{stock_decr}(10\%) \wedge \text{cfo_resigns} : [2, 0.68, 0.95, 0.7, 0.8]$
 For a given sequence of events, if there is a rumor of an SEC investigation and an earnings increase of 10%, this will be followed by a stock price decrease of 10% and the CFO resigning in exactly 2 time units with a frequency range $[0.7, 0.8]$ and probability bounds $[0.68, 0.95]$.</p> |
|--|

Fig. 1. \mathcal{K}_{stock} , an APT-Logic Program modeling the behavior to reactions of stock-related news feeds. As all of these rules are constrained, this is a constrained program. The English translation of each rule is also provided.

can be learned from historical data using standard machine learning algorithms. Financial companies have the means to derive large sets of such rules and make predictions based on them.

Reasoning about Terror Groups. Our group has extensively dealt with historical data on over 40 terrorist groups from the Minorities at Risk project [Wilkenfeld et al. 2007] and has published detailed analyses of some of these groups’ behaviors (Hezbollah [Mannes et al. 2008] and Hamas [Mannes et al. 2008]). Our SOMA Terror Organization Portal [Martinez et al. 2008b] has registered users from over 12 US government agencies and contains thousands of (automatically) extracted rules about the behaviors of these groups. For such groups, we might want to say: “Hezbollah targets domestic government security institutions/lives with a probability of 87 to 97% within 3 years (time periods) of years when their major organizational goals were focused on eliminating ethnic discrimination and when representing their interests to government officials was a minor part of their strategy.” Figure 2 provides a list of such rules associated with Hezbollah. Clearly, analysts all over the world engaged in counter-terrorism efforts need to be able to reason with such rules and make appropriate forecasts; in separate work, we have also done extensive work on making such forecasts [Martinez et al. 2008a; 2009].

Reasoning about Trains. All of us want to reason about train schedules and plane schedules. More importantly, railroad companies, airlines, and shipping companies have an even more urgent need to do such reasoning as it directly impacts their planning process. In such settings, a railroad company may learn rules of the form “If train 1 is at station A at time T , then it will be at station B at time $(T+4)$ with over 85% probability.” Once such rules are learned from historical data, various types of reasoning need to be performed in order for the railroad company to make its plans. Figure 3 shows a small toy example of rules associated with trains.

Reasoning about a Power Grid. Utility companies need to reason constantly about power grids. Decisions about which lines and transformers should be repaired next

are based not only on the costs of these repairs, but also when these components are likely to fail, and many other factors. Thus, for example, a power company may derive rules of the form “if the transformer tr and power line ln are functioning at time T , then there is a probability of over 95% that they will continue to be functioning at time $(T + 3)$ ”. Figure 4 shows a small toy example of rules associated with power grids.

- | |
|---|
| <p>(1) $(\text{INTERORGCON} = 1) \xrightarrow{\text{efr}} (\text{ARMATTACK} = 1) : [2, 0.85, 0.95]$
 Armed attacks are carried out within two years of inter-organizational conflicts arising, with probability between 0.85 and 0.95.</p> <p>(2) $(\text{DIASUP} = 0) \wedge (\text{MILITIAFORM} = 2) \xrightarrow{\text{efr}} (\text{KIDNAP} = 1) : [3, 0.68, 0.78]$
 Kidnappings are carried out within three years when no support from diaspora is received, and Hezbollah has a standing military wing, with probability between 0.68 and 0.78.</p> <p>(3) $(\text{ORGST2} = 1) \wedge (\text{ORGDOMGOALS} = 1) \xrightarrow{\text{efr}} (\text{DSECGOV} = 1) : [3, 0.87, 0.97]$
 Domestic government/state lives and security are targets of terrorism within three years if Hezbollah represents interests to officials as a minor strategy, and its major organizational goals are focused on eliminating discrimination, with probability between 0.87 and 0.97.</p> <p>(4) $(\text{ORGST4} = 1) \wedge (\text{INTERORGCON} = 1) \wedge (\text{MILITIAFORM} = 1) \xrightarrow{\text{efr}} (\text{BOMB} = 0) : [1, 0.56, 0.66]$
 Hezbollah does <i>not</i> carry out bombings within the following year if it solicits external support as a minor strategy, there are inter-organizational conflicts, and its military wing is being created, with probability between 0.56 and 0.66.</p> |
|---|

Fig. 2. A sample of the rules extracted by APT-Extract from the Hezbollah dataset. The atoms in the rules are represented as a variable and its value. A plain English explanation of each rule is also provided.

The examples above illustrate the syntax of an APT-logic program; we will give the formal details as we develop the technical material in the paper. While it is possible for designers to write such programs manually, we expect that machine learning programs can be used to automatically learn such programs from historical data using standard machine learning algorithms, as we did previously in our work on ap-programs [Khuller et al. 2007]. Though this is not claimed as a major contribution of this paper, in order to show that it is possible to automatically learn APT-programs, we have developed a simple algorithm called APT-Extract and used it to learn models of certain behaviors exhibited by 18 terror groups.

This paper proceeds as follows. In Section 2 we introduce the syntax and semantics of APT-logic programs, including a quick treatment of our notion of a *frequency function*, a structure unique to APT-logic. In Section 3 we introduce several methods to check consistency of APT-logic programs, along with appropriate complexity analysis. We introduce several algorithms for consistency checking: one that straightforwardly applies the semantics, one that exploits the relationships between formulas in the heads and bodies of APT-rules, and one that works only on specific sorts of APT-rules but often offers substantial speedup when it is possible.

- | | |
|-----|--|
| (1) | $\text{at_station}(\text{train1}, \text{stnA}) \xrightarrow{\text{efr}} \text{at_station}(\text{train1}, \text{stnB}) : [4, 0.85, 1]$
If train 1 is at station A, train 1 will be at station B within 4 time units with a probability bounded by $[0.85, 1.00]$ |
| (2) | $\text{at_station}(\text{train1}, \text{stnB}) \xrightarrow{\text{pfr}} \text{at_station}(\text{train1}, \text{stnC}) : [2, 0.75, 0.9]$
If train 1 is at station B, train 1 will be at station C in exactly 2 time units with a probability bounded by $[0.75, 0.90]$ |
| (3) | $\text{at_station}(\text{train1}, \text{stnA}) \xrightarrow{\text{pfr}} \text{at_station}(\text{train2}, \text{stnB}) : [1, 0.95, 1]$
If train 1 is at station A, train 2 will be at station B in exactly 1 time units with a probability bounded by $[0.95, 1.00]$ |
| (4) | $\text{at_station}(\text{train1}, \text{stnA}) : [1, 0.5, 0.5]$
For a given sequence of events, train 1 will be at station A at time period 1 with a probability of 0.50. |
| (5) | $\text{at_station}(\text{train2}, \text{stnA}) : [2, 0.48, 0.52]$
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. 3. $\mathcal{K}_{\text{train}}$ an APT-Logic Program modeling rail transit. Items 1-3 are APT-Rules while items 4-5 are annotated formulas. The English translation of each rule is also provided.

- | | |
|-----|--|
| (1) | $\text{func}(\text{ln}) \xrightarrow{\text{pfr}} \neg \text{func}(\text{ln}) : [1, 0.05, 0.1]$
If the power line is functional, in exactly 1 time unit it will be non-functional with a probability bounded by $[0.05, 0.10]$ |
| (2) | $\neg \text{func}(\text{ln}) \xrightarrow{\text{efr}} \text{func}(\text{ln}) : [2, 0.99, 1]$
If the power line is not functional, within 2 time units it will functional with a probability bounded by $[0.99, 1.00]$ |
| (3) | $\text{func}(\text{tr}) \wedge \text{func}(\text{ln}) \xrightarrow{\text{pfr}} \neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln})) : [1, 0.025, 0.03]$
If the transformer is functional and the line is functional, then in exactly 1 time unit, at least one of them is not functional with a probability bounded by $[0.025, 0.030]$ |
| (4) | $\neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln})) \xrightarrow{\text{efr}} \text{func}(\text{tr}) \wedge \text{func}(\text{ln}) : [3, 0.95, 1]$
If the transformer and/or the line is not functional, then within 3 time units, they both are functional with a probability bounded by $[0.95, 1.00]$ |
| (5) | $\text{func}(\text{tr}) \wedge \text{func}(\text{ln}) : [1, 0.8, 0.95]$
For a given sequence of events, the transformer and the power line are functional at the first time point with a probability bounded by $[0.80, 0.95]$. |

Fig. 4. $\mathcal{K}_{\text{power}}$ an APT-Logic Program modeling a power grid. Items 1-4 are APTRules, while item 5 is an annotated formula. The English translation of each rule is also provided.

General Complexity Results		
Problem	Complexity	Reference
Consistency of Single Unconstrained Rule	NP-complete	Thm 3.2
Consistency of Single Constrained Rule	NP-complete	Thm 3.4
Consistency of a mixed PCD Program with additional restrictions on lower probability bounds	Guaranteed consistent	Thm 3.7
Entailment of an annotated formula by an program	coNP-hard	Thm 4.2

Table I. Summary of General Complexity Results

Type of Linear Constraints	Number of Constraints	Number of Variables	Cost of Identifying Equivalence Classes
SLC	$2 \mathcal{K} + 1$	$2^{ \mathcal{B}_{\mathcal{L}} t_{max}}$	(equivalence classes not used)
WELC	$2 \mathcal{K} + 1$	$2^{2 \mathcal{K} t_{max}}$	$O(2^{2 \mathcal{K} +B_{\mathcal{L}}})$
FELC using BFECA to identify classes	$2 \mathcal{K} + 1$	$2^{ \mathcal{K} }$	$O(2^{ \mathcal{B}_{\mathcal{L}} t_{max}} \cdot F(t_{max}) \cdot \mathcal{K})$
FELC using WEFE to identify classes	$2 \mathcal{K} + 1$	$2^{ \mathcal{K} }$	$O(2^{2 \mathcal{K} \cdot t_{max}} \cdot t_{max} \cdot \mathcal{K}) + O(2^{2 \mathcal{K} +B_{\mathcal{L}}})$
FELC w. PCD restrictions on \mathcal{K}	$2 \mathcal{K} + 1$	$2^{ \mathcal{K} }$	(equivalence classes guaranteed)

Table II. Comparison of Linear Constraints for Consistency, see Section 3

These techniques can also be applied to the problem of entailment, which is covered in Section 4. In Section 5, we explore some applications of APT-logic programs and finally, we spend a great deal of effort in Section 6 distinguishing this logic from other probabilistic logics. In particular, we examine the relationship between APT-logic programs and Markov Decision Processes (MDPs for short) [Puterman 1994], showing that one can create APT-logic programs “equivalent” to a given MDP and policy, but under natural assumptions, there is no MDP “equivalent” to certain APT-logic programs. We further address the relationship between APT-logic and a well known logic called Probabilistic Computation Tree Logic (PCTL for short) [Hansson and Jonsson 1994a] and provide examples demonstrating that PCTL cannot express various things expressible in APT-logic programs.

The entire set of complexity results for APT-logic programs derived in this paper is summarized in Table I. Consistency of APT-logic programs is determined by solving certain linear programs. In this paper, we develop successively more sophisticated linear programs that try to use different types of “equivalence classes” to collapse multiple variables in the linear program into one variable; Table II summarizes the main results related to linear program size reduction for consistency checking. Table II also provides an analogous summary related to reduction of size of the linear program when considering entailment by APT-logic programs.

2. APT-LOGIC PROGRAMS

In this section, we will first define the syntax of APT-logic programs, and then define the formal semantics.

Algorithm	Intuition	Reference
SLC-ENT	Determining both the minimization and maximization of a constraint wrt SLC	Section 4
ALC-ENT	Determining both the minimization and maximization of a constraint wrt FELC or WELC	Section 4

Table III. Comparison of Linear Constraints for Entailment

2.1 Syntax

We assume the existence of a first order 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 $arity(p)$). A (ground) *term* is any member of $\mathcal{L}_{cons} \cup \mathcal{L}_{var}$ (resp. \mathcal{L}_{cons}); if t_1, \dots, t_n are (ground) terms, and $p \in \mathcal{L}_{pred}$, then $p(t_1, \dots, t_n)$ is a (resp. ground) atom. A *formula* is defined recursively as follows.

Definition 2.1. 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 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.

We assume that all applications reason about an arbitrarily large, but fixed size window of time, and that $\tau = \{1, \dots, t_{max}\}$ 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. For instance, in the stock market and power grid examples, the unit of time used might be days, and t_{max} may be arbitrarily set to (say) 1,095 denoting interest in stock market and power grid movements for about 3 years. In the case of the train example, however, the unit of time might be seconds, and the application developer might set t_{max} to 93,600, reflecting that we are only interested in reasoning about one day at a time, but at a temporal resolution of one second. In the case of the terrorism application, on the other hand, our temporal resolution might be one month, and t_{max} might be 360 reflecting an interest in events over a 30-year time span.

Definition 2.2 Annotated Formula. If F is a formula, $t \in \tau$ is a time point, and $[\ell, u]$ is a probability interval, then $F : [t, \ell, u]$ is an *annotated formula*. ■

Intuitively, $F : [t, \ell, u]$ says F will be true at time t with probability in $[\ell, u]$.¹

EXAMPLE 2.1. *Let us reconsider the program \mathcal{K}_{train} from Figure 3. The annotated formula $at_station(train1, stnB) : [4, 0.85, 1]$ says that the probability that train1 will be at station stnB at time point 4 is between 85 and 100%.*

Throughout this paper, we assume the existence of a finite set \mathcal{F} of symbols called *frequency function symbols*. Each of these symbols will denote a specific “frequency function” to be defined later when we define our formal APT semantics. We are now

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

ready to define the syntax of Annotated Probabilistic Temporal (APT for short) rules and logic programs which will form the main topic of study for this paper.

Definition 2.3 APT Rule. Let F, G be two formulas, Δt be a time interval, ℓ, u be a probability interval, $\text{fr} \in \mathcal{F}$ be a frequency function symbol and $\alpha, \beta \in [0, 1]$.

(1) $F \overset{\text{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u]$ is called an *unconstrained APT rule*.

(2) $F \overset{\text{fr}}{\hookrightarrow} G : [\Delta t, \ell, u, \alpha, \beta]$ is called a *constrained APT rule*.

An *APT logic program* is a finite set of APT rules and annotated formulas. ■

Note that we use the symbol ' $\overset{\text{fr}}{\rightsquigarrow}$ ' for unconstrained APT rules with frequency function symbol fr , while the symbol ' $\overset{\text{fr}}{\hookrightarrow}$ ' is used for constrained rules with frequency function fr . The formal semantics of these rules is quite complex and will be explained shortly. But informally speaking, both types of rules try to check the probability that a formula F is true Δt units before a formula G becomes true.

Figures 1, 2, 3, and 4 respectively show the APT-logic programs associated with our stock market, counter-terrorism, trains, and power grid applications. We now define three types of APT-logic programs.

Definition 2.4 Types of APT-Logic Programs.

- An *unconstrained* APT-Logic Program consists *only* of unconstrained APT-rules.
- A *constrained* APT-Logic Program consists *only* of constrained APT-rules.
- A *mixed* APT-Logic Program consists *both* of constrained and unconstrained APT-rules. ■

From this example, we see that \mathcal{K}_{stock} is a constrained APT-logic program, \mathcal{K}_{trains} , \mathcal{K}_{power} , and \mathcal{K}_{terror} are unconstrained APT-logic programs.²

2.2 Semantics of APT-logic programs

In this section, we will provide a formal declarative semantics for APT-logic programs. As the syntax of these programs is quite complex, we will do this one step at a time. We start with the well known definition of a world.

Definition 2.5. 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 or real world phenomenon being modeled by an APT-logic program. The following are examples of worlds:

EXAMPLE 2.2. Consider the atoms present in the program \mathcal{K}_{train} from Figure 3. A few possible worlds are: $\{\text{at_station}(\text{train1}, \text{stnA}), \text{at_station}(\text{train2}, \text{stnB})\}$, $\{\text{at_station}(\text{train1}, \text{stnB})\}$, and $\{\}$.

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].

²Notably absent from the types of APT-Logic Programs described above are annotated formulas. We will show later in Theorem 2.20 that APT-rules can be used to express annotated formulas and hence there is no loss of expressive power.

$Th(1) = \{\text{at_station}(\text{train1}, \text{stnA})\},$	$Th(2) = \{\},$
$Th(3) = \{\},$	$Th(4) = \{\text{at_station}(\text{train1}, \text{stnB})\},$
$Th(5) = \{\},$	$Th(6) = \{\text{at_station}(\text{train1}, \text{stnC})\},$
$Th(7) = \{\},$	$Th(8) = \{\text{at_station}(\text{train1}, \text{stnB})\},$
$Th(9) = \{\},$	$Th(10) = \{\text{at_station}(\text{train1}, \text{stnA})\}$

Fig. 5. Example thread for the train scenario from Figure 3, where only one train is present.

Definition 2.6 Satisfaction of a formula by a world. Let f be a ground formula and w be a world. We say that w satisfies f (denoted $w \models f$) iff:

- If $f = a$ for some ground atom a , then $a \in w$.
- If $f = \neg f'$ for some ground formula f' then w does not satisfy f' .
- If $f = f_1 \wedge f_2$ for formulas f_1 and f_2 , then w satisfies f_1 and w satisfies f_2 .
- If $f = f_1 \vee f_2$ for formulas f_1 and f_2 , then w satisfies f_1 or w satisfies f_2 . ■

We say a formula f is a tautology if for all $w \in 2^{B\mathcal{L}}$, $w \models f$. We say f is a contradiction if for all $w \in 2^{B\mathcal{L}}$, $w \models \neg f$.

A *thread*, defined below, is nothing but a standard temporal interpretation [Emerson and Halpern 1984; Lamport 1980] in temporal logic.

Definition 2.7 Thread. A thread is a mapping $Th : \{1, \dots, t_{max}\} \rightarrow 2^{B\mathcal{L}}$. ■

$Th(i)$ implicitly says that according to the thread Th , the world at time i will be $Th(i)$. We will use \mathcal{T} to denote the set of all possible threads, and Th_\emptyset to denote the “null” thread, *i.e.*, the thread which assigns \emptyset to all time points.

EXAMPLE 2.3. Consider the train scenario shown in Figure 3 and the worlds described in Example 2.2. Let $\tau = \{0, \dots, 9\}$ represent one-hour time periods in a day from 9:00am to 6:00pm, *i.e.*, 0 represents 9-10am, 1 represents 10-11am, and so forth. Figure 5 shows a sample thread for this setting, where only one train is present. According to this thread, the train is at station A at 9 o'clock; at 10 o'clock the thread has an empty world, since the train is still between stations, reaching station B at 12. The thread shows how the train moves throughout the rest of the day.

A thread represents a possible way the domain being modeled (*e.g.*, where the train is) will evolve over all time points. A *temporal probabilistic (tp) interpretation* gives us a probability distribution over all possible threads.

Definition 2.8 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 about the state of the world at various points in time.

EXAMPLE 2.4. Consider once again the setting of Figure 3. A very simple example of a tp-interpretation is the probability distribution that assigns probability 1

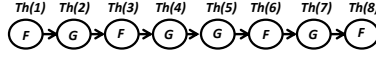


Fig. 6. Example thread, Th with worlds $Th(1), \dots, Th(8)$. This figure shows each world that satisfies formula F or formula G .

to the thread from Figure 5 and 0 to every other possible thread. Another example would be a distribution that assigns probability 0.7 to the thread from Figure 5 and 0.3 to the thread Th' defined as follows: $\langle Th'(1) = \{\text{at_station}(\text{train1}, \text{stnA})\}$, $Th'(2) = \{\}$, $Th'(3) = \{\}$, $Th'(4) = \{\}$, $Th'(5) = \{\text{at_station}(\text{train1}, \text{stnB})\}$, $Th'(6) = \{\text{at_station}(\text{train1}, \text{stnC})\}$, $Th'(7) = \{\}$, $Th'(8) = \{\text{at_station}(\text{train1}, \text{stnB})\}$, $Th'(9) = \{\}$, $Th'(10) = \{\text{at_station}(\text{train1}, \text{stnA})\}\rangle$; this thread specifies that the train's trip from station A to station B takes one time unit longer than specified by the previous thread (Th).

We now define what it means for a tp-interpretation to satisfy an annotated formula.

Definition 2.9 Satisfaction of an Annotated Formula. Let $F : [t, \ell, u]$ be an annotated formula, and I be a tp-interpretation. We say that I satisfies $F : [t, \ell, u]$, written $I \models F : [t, \ell, u]$, iff $\ell \leq \sum_{Th \in \mathcal{T}, Th(t) \models F} I(Th) \leq u$. ■

Thus, to check if I satisfies $F : [t, \ell, u]$, we merely sum up the probabilities assigned to those threads $Th \in \mathcal{T}$ which make F true at time t . If this sum is in $[\ell, u]$ then I satisfies $F : [t, \ell, u]$.

2.3 Frequency Functions

When defining the syntax of APT-logic programs, we defined frequency function symbols. Each frequency function symbol denotes a 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. For instance, consider the thread shown in Figure 6. Here, F is true at times 1, 3, 6, and 8. G is true at times 2, 4, 5, and 7. F and G should be true at the times indicated above.

- The probability (within the thread of Figure 6) that G follows F in *exactly* two units of time is 0.33 if we ignore the occurrence of F at time 8. If, on the other hand, we do count that 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 0.25.
- The probability that G follows F in *at most* 2 units of time is 100% if we ignore the occurrence of F at time 8; otherwise it is 0.75.

Each of these intuitions leads to different ways to measure the frequency (within a thread) with which G follows F . As we will show shortly, many other possibilities exist as well. *To the best of our knowledge, no past work on reasoning with time and uncertainty deals with frequencies within threads; as a consequence, past works are not able to aggregate frequencies across multiple threads in \mathcal{T} or w.r.t.*

tp-interpretations. This capability, we will show, is key for the types of applications described in the Introduction of this paper.

We see above that there are many different ways to define this frequency from a given body of historical data. Rather than make a commitment to one particular way and in order to allow applications and users to select the frequency function that best meets their application needs, we now define *axioms* that any frequency function must satisfy. Later, we will define some specific frequency functions.³

Definition 2.10 Frequency Function. Let Th be a thread, F and G be formulas, and $\Delta t > 0$ be an integer. A *frequency function* fr is one that maps quadruples of the form $(Th, F, G, \Delta t)$ to $[0, 1]$ such that it satisfies the following axioms:

(FF1) If G is a tautology, then $fr(Th, F, G, \Delta t) = 1$.

(FF2) If F is a tautology and G is a contradiction, then $fr(Th, F, G, \Delta t) = 0$.

(FF3) If F is a contradiction, $fr(Th, F, G, \Delta t) = 1$.

(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 $fr(Th_1, F, G, \Delta t) = 0$ and $fr(Th_2, F, G, \Delta t) = 1$. ■

Axiom FF1 says that if G is a tautology, then $fr(Th, F, G, \Delta t)$ 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 $fr(Th, F, G, \Delta t)$ 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 $fr(Th, F, G, \Delta t)$ 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. We now give examples of two frequency functions.

Definition 2.11 Point Frequency Function. Let Th be a thread, F and G be formulas, and $\Delta t \geq 0$ be an integer. A *Point Frequency Function*, denoted $pfr(Th, F, G, \Delta t)$, is defined as:

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

If there is no $t \in [0, t_{max} - \Delta t]$ such that $Th(t) \models F$ then we define pfr to be 1.

The point frequency function expresses a simple concept: it specifies how frequently G follows F in Δt time points. Mathematically, this is done by finding all time points from $[1, t_{max} - \Delta t]$ at which F is true and of all such time points t , then finding those for which G is true at time $t + \Delta t$. The ratio of the latter to the former is the value of pfr . The following lemma says that this is a valid frequency function. Note that the denominator of the point frequency function does not include times

³**Note:** Throughout this paper, we will assume that frequency function for a given thread can be computed in polynomial time (i.e. $O(|B_{\mathcal{L}}| \cdot t_{max})$). Additionally, we shall assume that a frequency function will return number that can be represented as a rational number a/b where a and b are relatively prime and the length of the binary representations of a and b is fixed.

where the thread satisfies F after $t_{max} - \Delta t$ because the “end of time” of our finite time model comes before Δt units elapse after F becomes true.

LEMMA 2.12. *pfr satisfies Axioms FF1-FF4.*

EXAMPLE 2.5 POINT FREQUENCY FUNCTION. *Consider thread Th from Figure 5. Suppose we want to calculate $pfr(Th, \text{at_station}(\text{train1}, \text{stnB}), \text{at_station}(\text{train1}, \text{stnC}), 2)$. In English, this is the ratio of time $\text{at_station}(\text{train1}, \text{stnB})$ is followed by $\text{at_station}(\text{train1}, \text{stnC})$ in two units of time in thread Th . We can see that $\text{at_station}(\text{train1}, \text{stnB})$ is satisfied by two worlds: $Th(4)$ and $Th(8)$. We also notice that $Th(6) \models \text{at_station}(\text{train1}, \text{stnC})$ and $Th(10) \not\models \text{at_station}(\text{train1}, \text{stnC})$. Hence, the pfr is simply 0.5.*

Our second type of frequency function, called an *existential* frequency function, does not force G to occur exactly Δt units of time after F is true. It can occur at or before Δt units of time elapse after F becomes true.

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:⁴

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

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.

Note that in the denominator of efr , after time $t_{max} - \Delta t$, we only count satisfaction of F if it is followed by satisfaction of G within $[t_{max} - \Delta t, t_{max}]$.

LEMMA 2.14. *efr satisfies Axioms FF1-FF4.*

The point frequency function expresses what is desired in situations where there is a precise temporal relationship between events (*i.e.*, if one drops an object from a height of 9.8 meters in a vacuum, it will hit the ground in exactly $\sqrt{2}$ seconds). However, it can be very brittle. Consider mail delivery where one knows a package will arrive in at most 5 business days 95% of the time. The existential frequency function efr allows for the implied condition to fall within some specified period of time rather than after exactly $\sqrt{2}$ seconds as before.

EXAMPLE 2.6 EXISTENTIAL FREQUENCY FUNCTION. *Consider thread Th' from Example 2.4. Suppose we want to calculate*

$$efr(Th', \text{at_station}(\text{train1}, \text{stnB}), \neg \text{at_station}(\text{train1}, \text{stnC}), 2).$$

In English, this is the ratio of times that $\text{at_station}(\text{train1}, \text{stnB})$ is followed by $\neg \text{at_station}(\text{train1}, \text{stnC})$ in two units of time in thread Th' . We can see that formula $\text{at_station}(\text{train1}, \text{stnB})$ is satisfied by two worlds: $Th'(5)$ and $Th'(8)$. Consider world $Th'(6)$, which occurs one time unit after world $Th'(5)$. We can easily see that $Th'(6) \not\models \neg \text{at_station}(\text{train1}, \text{stnC})$. However, $Th'(7)$, two units later, does satisfy $\neg \text{at_station}(\text{train1}, \text{stnC})$. As $Th'(9)$ also satisfies $\neg \text{at_station}(\text{train1}, \text{stnC})$, we have

⁴Where $efn(Th, F, G, \Delta t, t_1, t_2) = |\{t : (t_1 \leq t \leq t_2) \text{ and } Th(t) \models F \text{ and there exists } t' \in [t + 1, \min(t_2, t + \Delta t)] \text{ such that } Th(t') \models G\}|$.

a world within two time units after every world that satisfies `at_station(train1, stnB)`. Hence, the *efr* is 1 in this case.

Properties of *pfr*: Because of the requirement for F_2 to be satisfied after a specific Δt , *pfr* has several properties (all formulas F_1, F_2 below are assumed to be satisfiable).

- (1) $pfr(Th, F_1, F_2 \vee F_3, \Delta t) \geq \max(pfr(Th, F_1, F_2, \Delta t), pfr(Th, F_1, F_3, \Delta t))$ (valid for *efr* as well)
- (2) $pfr(Th, F_1, F_2 \wedge \neg F_3, \Delta t) = pfr(Th, F_1, F_2 \wedge F_3, \Delta t) - pfr(Th, F_1, F_3, \Delta t)$
- (3) $pfr(Th, F_1, F_2, \Delta t) \leq pfr(Th, F_1 \wedge F_3, F_2, \Delta t) \Rightarrow pfr(Th, F_1 \wedge \neg F_3, F_2, \Delta t) \leq pfr(Th, F_1, F_2, \Delta t)$
- (4) $pfr(Th, F_1, F_2 \wedge F_3, \Delta t) \leq \min(pfr(Th, F_1, F_2, \Delta t), pfr(Th, F_1, F_3, \Delta t))$
- (5) If $pfr(Th, F_1, F_2, \Delta t) = a$ and $pfr(Th, F_1, F_3, \Delta t) = b$ then $pfr(Th, F_1, F_2 \wedge F_3, \Delta t) \geq a + b - 1$.

Properties of *efr*: *efr* satisfies all the properties that *pfr* has above. In addition, *efr* has the property that:

$$efr(Th, F_1, F_2, \Delta t) \geq efr(Th, F_1, F_2, \Delta t - 1)$$

The following result provides some links between *pfr* and *efr*.

PROPOSITION 2.15. *Let Th be a thread, F and G be formulas,*

- (1) *Let Δt_1 and Δt_2 be two positive integers. If $\Delta t_1 \leq \Delta t_2$, then:*

$$pfr(Th, F, G, \Delta t_1) \leq efr(Th, F, G, \Delta t_2).$$

- (2) *Let Δt be a temporal interval. The following inequality always holds:*

$$efr(Th, F, G, \Delta t) \leq \sum_{i=1}^{\Delta t} pfr(Th, F, G, i)$$

2.4 Satisfaction of Rules and Programs

We are now ready to define satisfaction of an Annotated Probabilistic Temporal (APT) rule.

Definition 2.16 Satisfaction of APT rules. Let r be an APT rule with frequency function fr and I be a tp-interpretation.

- (1) For $r = F \overset{fr}{\rightsquigarrow} G : [\Delta t, \ell, u]$, we say that I *satisfies* r (denoted $I \models r$) iff

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

- (2) For $r = F \overset{fr}{\leftrightarrow} G : [\Delta t, \ell, u, \alpha, \beta]$, we say that I *satisfies* r (denoted $I \models r$), iff

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

■

Intuitively, the unconstrained APT rule $F \overset{\text{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u]$ 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 that 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 a little bit like an expected value computation in statistics where a value (frequency) is multiplied by a probability (of the thread). It then sums up these products across all threads in much the same way as an expected value computation.

On the other hand, in the case of constrained rules, the probability is computed by first finding all threads such that the frequency of F leading to G in Δt time units is in the $[\alpha, \beta]$ interval, and then summing up the probabilities of all such threads. This probability is the sum of probabilities assigned to threads where the frequency with which F leads to G in Δt time units is in $[\alpha, \beta]$. To satisfy the constrained APT rule $F \overset{\text{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u, \alpha, \beta]$, this probability must be within the probability interval $[\ell, u]$.

EXAMPLE 2.7. *Coming back to the train scenario from Figure 3, the following is an example of an unconstrained rule (r_1) and a constrained rule (r_2):*

$$\begin{aligned} r_1 &: \text{at_station}(\text{train1}, \text{stnC}) \overset{\text{efr}}{\rightsquigarrow} \text{at_station}(\text{train1}, \text{stnB}) : [2, 0.85, 1] \\ r_2 &: \text{at_station}(\text{train1}, \text{stnB}) \overset{\text{efr}}{\rightsquigarrow} \text{at_station}(\text{train1}, \text{stnC}) : [2, 0.9, 1, 0.5, 1] \end{aligned}$$

Consider the second tp -interpretation from Example 2.4, which we will call I . By analyzing the two threads considered possible by I , it is clear that $I \models r_1$, since both threads have the property that after being at station C the train reaches station B within two time units, and thus the probability of this event is 1. A similar analysis leads us to confirm that $I \models r_2$, but we must now verify that the constraints placed by the rule on the threads hold; these constraints require that at least half of the times in which the train is at station B , station C be reached within 2 time units. This is indeed the case, since the train stops twice at station B , once going towards C and once going towards A on its way back. As before, the sum of probabilities of reaching the station within 2 time units is 1. Finally, consider the rule:

$$r_3 : \text{at_station}(\text{train1}, \text{stnA}) \overset{\text{efr}}{\rightsquigarrow} \text{at_station}(\text{train1}, \text{stnC}) : [2, 0.5, 0.6]$$

Clearly, $I \not\models r_3$, since neither of the threads considered possible by the tp -interpretation satisfy the condition that the train reaches station C within two time units of being at station A .

The following proposition says that any tp -interpretation that satisfies certain kinds of constrained or unconstrained APT-logic programs also satisfies a certain APT rule that can be easily constructed from the APT-rules in the original APT-logic program.

PROPOSITION 2.17. *Let I be a temporal interpretation, F and G be formulas, and Δt be a temporal interval.*

$$(1) \text{ If } I \models \bigcup_{i=1}^{\Delta t} \left\{ F \overset{\text{pfr}}{\rightsquigarrow} G : [i, \ell_i, u_i] \right\} \text{ then } I \models F \overset{\text{efr}}{\rightsquigarrow} G : \left[\Delta t, \max(\ell_i), \min \left(\sum_{i=1}^{\Delta t} u_i, 1 \right) \right].$$

- (2) If $I \models F \xrightarrow{fr} G : [\Delta t, \ell_p, u_p, a, b]$ then $\forall a_\ell, b_\ell, a_u, b_u$ such that $a_\ell \leq a \leq a_u$ and $b_\ell \leq b \leq b_u$ we have $I \models F \xrightarrow{fr} G : [\Delta t, \ell_p, 1, a_\ell, b_u]$ and $I \models F \xrightarrow{fr} G : [\Delta t, 0, u_p, a_u, b_\ell]$.

Note that in unconstrained APT-rules, the ℓ, u probability bounds account for the frequency function as well. In the case of constrained APT-rules, the ℓ, u probability bounds *do not* account for the frequency function. We now show that using a special frequency function called a *query frequency function*, we can use constrained and unconstrained rules to express annotated formulas.

Definition 2.18 Query Frequency Function. Let Th be a thread, F and G be formulas, and $\Delta t \geq 0$ be an integer. A *query frequency function*, denoted $qfr(Th, F, G, \Delta t)$ is defined as follows:

- (1) If G is a tautology then $qfr(Th, F, G, \Delta t) = 1$
- (2) If F is a tautology and G is a contradiction, then $qfr(Th, F, G, \Delta t) = 0$
- (3) If F is a contradiction then $qfr(Th, F, G, \Delta t) = 1$
- (4) If $Th(1) \models F$ and $Th(\Delta t) \models G$ then $qfr(Th, F, G, \Delta t) = 1$
- (5) Else, $qfr(Th, F, G, \Delta t) = 0$ ■

The following result shows that qfr is a valid frequency function.

LEMMA 2.19. *qfr satisfies Axioms FF1-FF4.*

qfr allows us to construct constrained and unconstrained rules that are equivalent to arbitrary annotated formulas.

THEOREM 2.20. *Let $q = Q : [t, \ell, u]$ be an annotated formula, and I be an interpretation.*

- (1) For constrained rule $r = \text{TRUE} \xrightarrow{qfr} Q : [t, \ell, u, 1, 1]$, $I \models q$ iff $I \models r$.
- (2) For unconstrained rule $r = \text{TRUE} \xrightarrow{qfr} Q : [t, \ell, u]$, $I \models q$ iff $I \models r$.

The following is an example of how an annotated formula can be expressed as a rule using qfr .

EXAMPLE 2.8. *Consider the train setting from Figure 3. One of the annotated formulas given in this example was `at_station(train1, stnA) : [1, 0.5, 0.5]`. By applying Theorem 2.20, this formula is equivalent to the constrained rule r_1 and the unconstrained rule r_2 :*

$$\begin{aligned} r_1 &: \text{TRUE} \xrightarrow{qfr} \text{at_station}(\text{train1}, \text{stnA}) : [1, 0.5, 0.5, 1, 1] \\ r_2 &: \text{TRUE} \xrightarrow{qfr} \text{at_station}(\text{train1}, \text{stnA}) : [1, 0.5, 0.5] \end{aligned}$$

3. CONSISTENCY

3.1 Complexity of Consistency Checking

We are now ready to study the complexity of the problem of checking consistency of APT-logic programs. We say that an APT-logic program \mathcal{K} is *consistent* iff there is a tp-interpretation I such that $I \models \mathcal{K}$. Before stating complexity results, we give results that hold for any frequency function and any APT-rule. The first result follows from axioms FF1-FF4 on frequency functions.

LEMMA 3.1. Consider the APT-Program $\{r = F \overset{fr}{\rightsquigarrow} G : [\Delta t, \ell, u]\}$.

- (1) If G is a tautology, then $\{r\}$ is consistent iff $u = 1$.
- (2) If F is a tautology and G is a contradiction, then $\{r\}$ is consistent iff $\ell = 0$.
- (3) If F is a contradiction, then $\{r\}$ is consistent iff $u = 1$.
- (4) If F is not a contradiction, G is not a tautology, and either F is not a tautology or G is not a contradiction then $\{r\}$ is consistent.

Using this lemma, we can show that for any unconstrained APT-rule, the problem of determining if an APT-logic program consisting of just that APT-rule is consistent using any frequency function is NP-complete.

THEOREM 3.2. Deciding the consistency of an APT-logic program containing a single unconstrained APT-rule is NP-complete in the size of $B_{\mathcal{L}}$.

The proof of hardness above is by reduction from the SAT problem, while membership in NP relies on manipulating Lemma 3.1.

In deciding the consistency of a single constrained rule, we take a slightly different approach. The intuition is that if the lower probability bound is not zero, we *must* have a thread whose frequency function value falls within $[\alpha, \beta]$. Otherwise, there is no thread available that would ensure a non-zero probability mass as per the definition of satisfaction. The idea of classifying threads in this manner for constrained rules comes into play later when we present consistency-checking algorithms in Section 3.4.

LEMMA 3.3. Let $\mathcal{K} = \{r = F \overset{fr}{\rightsquigarrow} G : [\Delta t, \ell, u, \alpha, \beta]\}$ be a constrained APT-logic program consisting of a single rule. \mathcal{K} is consistent iff at least one of the following conditions hold.

- $u = 1$ **and** there exists a thread Th_{in} such that $\alpha \leq fr(Th_{in}, F, G, \Delta t) \leq \beta$.
- $\ell = 0$ **and** there exists a thread Th_{out} such that either $\alpha > fr(Th_{out}, F, G, \Delta t)$ or $\beta < fr(Th_{out}, F, G, \Delta t)$.
- There exists a thread Th_{in} such that $\alpha \leq fr(Th_{in}, F, G, \Delta t) \leq \beta$ and a thread Th_{out} such that either $\alpha > fr(Th_{out}, F, G, \Delta t)$ or $\beta < fr(Th_{out}, F, G, \Delta t)$.

Lemma 3.3, used in conjunction with the frequency function axioms, allow us to prove that deciding the consistency of a single constrained rule is also NP-complete.

THEOREM 3.4. Deciding the consistency of an APT-logic program containing a single constrained APT-rule is NP-complete in the size of $B_{\mathcal{L}}$.

The NP-hardness of consistency checking for APT programs (whether constrained, unconstrained, or mixed) with more than one rule follows trivially from Theorems 3.2 and 3.4. We suspect that the problem of consistency checking for a general APT program is not in NP.

However, if we assume that certain conditions hold, we can show that consistency for an APT-logic program containing multiple APT-rules can be guaranteed. These restrictions are termed Pre-Condition Disjoint, or PCD; intuitively, they refer to an APT-Program such that there exists a unique world that satisfies exactly one of the rule pre-conditions (the F formulas). Hence, we say that the pre-conditions are

“disjoint” from each other. Perhaps such conditions could be specified by a tool used to learn the rules from the data-set.

Definition 3.5 Pre-Condition Disjoint (PCD) APT-Logic Program. Let \mathcal{K} be an APT-Logic Program such that $\mathcal{K} = \{r_1, \dots, r_n\}$, where $r_i = F_i \xrightarrow{\text{fr}} G_i : [\Delta t_i, \ell_i, u_i]$ or $r_i = F_i \xrightarrow{\text{cfr}} G_i : [\Delta t_i, \ell_i, u_i, \alpha_i, \beta_i]$. \mathcal{K} is Pre-Condition Disjoint (PCD) if the following conditions hold true.

- (1) $\forall i$, if r_i is constrained, then $\beta_i = 1$.
- (2) $\forall i$, $\Delta t_i \geq 1$.
- (3) $\forall i$ there exists a world w_i such that $w_i \models F_i$ and $\forall j$ where $j \neq i$, $w_i \not\models F_j$.
- (4) $\forall i$, fr_i is equal to either pfr , or efr .
- (5) $t_{\max} \geq |\mathcal{K}| \cdot \max(\Delta t_i)$ (where t_{\max} is the length of each thread).
- (6) \exists world w_\emptyset such that $\forall i$ $w_\emptyset \not\models F_i$ and $w_\emptyset \not\models G_i$.
- (7) $\forall r_i \in \mathcal{K}$, $u_i = 1$. ■

While somewhat limiting, this restriction still allows APT-Logic Programs that are useful. Consider the following example.

EXAMPLE 3.1. Consider the set of rules shown in Figure 3. These rules do not constitute a PCD program for various reasons. For instance, the upper bound on the probability of the second rule is not 1. Likewise, condition 3 is not satisfied since the first and third rule have the same antecedent. However, the following set of rules satisfies all of the conditions for being a PCD program:

$$\begin{aligned} & \text{at_stn}(\text{trn1}, \text{stnA}) \wedge \neg \text{at_stn}(\text{trn1}, \text{stnB}) \wedge \neg \text{at_stn}(\text{trn1}, \text{stnC}) \xrightarrow{\text{efr}} \\ & \quad \text{at_stn}(\text{trn1}, \text{stnB}) : [4, 0.85, 1] \\ & \text{at_stn}(\text{trn1}, \text{stnB}) \wedge \neg \text{at_stn}(\text{trn1}, \text{stnA}) \wedge \neg \text{at_stn}(\text{trn1}, \text{stnC}) \xrightarrow{\text{pfr}} \\ & \quad \text{at_stn}(\text{trn1}, \text{stnC}) : [2, 0.75, 1] \\ & \text{at_stn}(\text{trn1}, \text{stnC}) \wedge \neg \text{at_stn}(\text{trn1}, \text{stnA}) \wedge \neg \text{at_stn}(\text{trn1}, \text{stnB}) \xrightarrow{\text{efr}} \\ & \quad \text{at_stn}(\text{trn1}, \text{stnB}) : [3, 0.9, 1] \end{aligned}$$

Conditions 1, 2, 4, and 7 are trivially satisfied, and t_{\max} can be easily chosen to satisfy condition 5. Condition 3 can be seen to hold by noting that no two antecedents of rules can be satisfied at once. Finally, condition 6 holds since the empty world does not satisfy any of the formulas involved in the rules.

The useful feature in a PCD program is that (based on the axioms) we are guaranteed threads with certain frequency function values for each rule. Consider Lemma 3.6 below, where for any subset of a given APT-program, we are guaranteed the existence of a thread whose frequency is 1 according to the rules in the subset and is 0 according to the other rules.

LEMMA 3.6. Consider APT-Program $\mathcal{K} = \{r_1, \dots, r_i, \dots, r_n\}$ where $r_i = F_i \xrightarrow{\text{fr}_i} G_i[\Delta t_i, \ell_i, u_i, \alpha_i, \beta_i]$ or $r_i = F_i \xrightarrow{\text{cfr}_i} G_i : [\Delta t_i, \ell_i, u_i]$, depending on whether r_i is a constrained or unconstrained rule. If \mathcal{K} is PCD, then for any disjoint partition of rules, $\mathcal{K}_1, \mathcal{K}_2$, there exists a thread Th such that for all rules $r_i \in \mathcal{K}_1$, $\text{fr}_i(Th, F_i, G_i, \Delta t_i) = 1$ and for all rules $r_i \in \mathcal{K}_2$, $\text{fr}_i(Th, F_i, G_i, \Delta t_i) = 0$.

The PCD conditions add a “one-tailed” requirement (the first requirement of Definition 3.5) to the constrained rules so that β is always one. This allows us to be guaranteed the existence of threads in the $[\alpha, \beta]$ bounds. As it turns out, if the lower bounds on the probabilities are less than a certain amount, we can create an interpretation to guarantee the consistency of the PCD program.

THEOREM 3.7. *For a mixed PCD APT-Program $\mathcal{K} = \{r_1, \dots, r_i, \dots, r_n\}$, if for all r_i , $\ell_i \leq \frac{|\mathcal{K}| - 1}{|\mathcal{K}|}$ then \mathcal{K} is consistent.*

In the appendix, we show how PCD assumptions can be leveraged for a significant reduction in complexity for constrained APT-programs.

3.2 Linear Constraints for Consistency Checking

A *straightforward* algorithm to find a satisfying interpretation given an APT-logic program \mathcal{K} is a brute-force approach that considers each thread. Given k atoms and t_{max} timepoints, there are 2^k possible worlds at each timepoint, and $2^{k \cdot t_{max}}$ possible threads. For ease of notation, we shall refer to the number of threads as n . Hence, note that a function that is linear in the number of threads is exponential in the number of atoms.

Let $\mathcal{T} = \{Th_1, \dots, Th_i, \dots, Th_n\}$ be the set of threads. In our linear program, we will use the variables $V = \{v_1, \dots, v_j, \dots, v_n\}$. Each v_i represents the (as yet unknown) probability of thread Th_i . We will design the linear program so that solutions of the linear program are in a one to one correspondence with interpretations that satisfy the APT-logic program. Thus, if θ is a solution of the linear program, we want to be sure that the tp-interpretation I_θ such that $I_\theta(Th_i) = \theta(v_i)$ is an interpretation that satisfies \mathcal{K} .

Hence, given an APT-logic program \mathcal{K} , we will construct a set of “straightforward” linear constraints $SLC(\mathcal{K})$ over variables $V = \{v_1, \dots, v_j, \dots, v_n\}$, such that the interpretation I_θ associated as above with any solution θ satisfies \mathcal{K} . The set of constraints are as follows:

Definition 3.8 Straightforward Linear Constraints (SLC). Let \mathcal{K} be an APT-logic program; the set of *straightforward linear constraints* contains exactly the following:

- (1) $\sum_{j=1}^n v_j = 1$
- (2) For each unconstrained rule $F_i \xrightarrow{fr_i} G_i : [\Delta t_i, \ell_i, u_i] \in \mathcal{K}$
 - (a) $\ell_i \leq \sum_{j=1}^n fr_i(Th_j, F_i, G_i, \Delta t_i) \cdot v_j$
 - (b) $u_i \geq \sum_{j=1}^n fr_i(Th_j, F_i, G_i, \Delta t_i) \cdot v_j$
- (3) For each constrained rule $F_i \xrightarrow{fr_i} G_i : [\Delta t_i, \ell_i, u_i, \alpha_i, \beta_i] \in \mathcal{K}$
 - (a) $\ell_i \leq \sum_{Th_j \in \mathcal{T}} \alpha_i \leq fr_i(Th_j, F_i, G_i, \Delta t_i) \leq \beta_i v_j$
 - (b) $u_i \geq \sum_{Th_j \in \mathcal{T}} \alpha_i \leq fr_i(Th_j, F_i, G_i, \Delta t_i) \leq \beta_i v_j$

We refer to this set as $SLC(\mathcal{K})$. ■

The first constraint above says that the threads are exhaustive. The second constraint is derived from the formula for satisfaction of an unconstrained rule, while the third constraint is derived from the formula for satisfaction of a constrained rule.

Algorithm 1 Compute consistency of \mathcal{K} using SLC.

SLC-CONSISTENT(*APT-Program* \mathcal{K})

- (1) Construct SLC(\mathcal{K}).
 - (2) Attempt to solve SLC(\mathcal{K}).
 - (3) If solvable, return *consistent*, otherwise, *inconsistent*.
-

Note that the coefficient of v_j in constraints (2) and (3) above are both constants (after the calculations are performed), so these constraints are all linear.

EXAMPLE 3.2. Recall the program \mathcal{K}_{power} from Figure 4. In this simple example, we supposed the power plant delivers power to a transformer (named *tr*), which is in turn connected via a power line (named *ln*) to a home. Hence, the atoms $\text{func}(\text{tr})$ and $\text{func}(\text{ln})$ denote that the various components are functioning, and the home receives power only if both *tr* and *ln* are *func*. Therefore, we have four possible worlds: $w_0 = \{\text{func}(\text{tr}), \text{func}(\text{ln})\}$, $w_1 = \{\text{func}(\text{tr})\}$, $w_2 = \{\text{func}(\text{ln})\}$, and $w_3 = \emptyset$. If we set the time limit to 4 days, then there are $4^4 = 256$ possible threads (each world may occur at each time point). We name these threads Th_0, \dots, Th_{255} so that the world at time point t of thread Th_i is $((i/4^t) \bmod 4)$ (i.e. Th_{25} is $\langle w_1, w_2, w_1, w_0 \rangle$) and associate the variable v_i with $I(Th_i)$. We now show the constraints in SLC(\mathcal{K}_{power}):

- (1) $\sum_{i=0}^{i<256} v_i = 1$
- (2) $0.025 \leq \sum_{i=0}^{i<256} \text{pfr}(Th_i, \text{func}(\text{tr}) \wedge \text{func}(\text{ln}), \neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln})), 1) \cdot v_i \leq 0.03$
- (3) $0.95 \leq \sum_{i=0}^{i<256} \text{efr}(Th_i, \neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln})), \text{func}(\text{tr}) \wedge \text{func}(\text{ln}), 3) \cdot v_i \leq 1$
- (4) $0.05 \leq \sum_{i=0}^{i<256} \text{pfr}(Th_i, \text{func}(\text{ln}), \neg\text{func}(\text{ln}), 1) \cdot v_i \leq 0.1$
- (5) $0.99 \leq \sum_{i=0}^{i<256} \text{efr}(Th_i, \neg\text{func}(\text{ln}), \text{func}(\text{ln}), 2) \cdot v_i \leq 1$

Given a solution θ of these constraints, we can see immediately that I_θ satisfies \mathcal{K} .

We provide the following proposition about correctness of the above procedure for mixed programs.

PROPOSITION 3.9. For mixed APT-Logic Program \mathcal{K} , \mathcal{K} is consistent iff SLC(\mathcal{K}) has a solution.

The size of the linear program for SLC follows immediately from the definition. As each rule requires two linear constraints, and one linear constraint is required to ensure the variables sum to 1, we have $2|\mathcal{K}| + 1$ constraints. The number of variables is equal to the number of threads.

Remark 3.10. SLC contains $2|\mathcal{K}| + 1$ constraints and $2^{|B_{\mathcal{L}}| \cdot t_{max}}$ variables.

Using SLC we can create Algorithm 1, which is guaranteed to give a correct answer to the question of consistency for any APT-Logic Program. However, the linear program's size is exponential in terms of $|B_{\mathcal{L}}| \cdot t_{max}$, making it a very expensive operation in many situations. There are several obvious ways to reduce this cost. One such way would be to consider the set of atoms to be *only* the atoms present in the rules. An obvious method to reduce the other factor in the exponent, t_{max} , would be to adjust the granularity of time used. For example, convert all time to

hours instead of minutes. However, this would only provide a correct result in terms of the new granularity. This is an issue we intend to explore in future research.

It turns out that for arbitrary sets of rules and annotated formulas, one need not use one variable for each of the $2^{|B_{\mathcal{L}}| \cdot t_{max}}$ threads. Some threads are equivalent, and may in fact be considered together. We provide two such methods that consider equivalent threads. One that reduces the number of worlds based on *world equivalence* and one that reduces the number of threads based on *frequency equivalence*.

3.3 World Equivalence

World equivalence uses the following intuition: when two worlds satisfy exactly the same formulas from the APT-program, they are identical from the APT-program's point of view. By partitioning the set of worlds into classes of identical worlds, and working with the classes instead of the individual worlds, we can create smaller linear programs by associating just one variable with each equivalence class (rather than one variable with each world as is the case of SLC).

Consider the rule $F \xrightarrow{\text{fr}} G : [\Delta t, \ell, u, \alpha, \beta]$. The four world-based equivalence classes resulting from this rule would be the sets of worlds that satisfy $F \wedge G$, $F \wedge \neg G$, $\neg F \wedge G$, and $\neg F \wedge \neg G$. We apply this concept to APT-Logic Programs and divide the set of worlds accordingly. We can treat these resulting equivalence classes as worlds and create world-based thread equivalence classes, and use them instead of threads. This reduces the number of linear constraints for an algorithm similar to SLC. One must note, however, that the equivalence classes must be computed first, which we will show to be NP-complete.

As world equivalence for APT-Logic is based on the formulas found in APT-Rules and annotated formulas, we will formalize the set of formulas associated with a program. We introduce the notation $formula(\mathcal{K})$ to denote the set of all formulas present in an APT-logic program:

$$formula(\mathcal{K}) = \{F, G \mid F \xrightarrow{\text{fr}} G : [\Delta t, \ell, u, \alpha, \beta] \in \mathcal{K}\} \cup \{F, G \mid F \xrightarrow{\text{tr}} G : [\Delta t, \ell, u] \in \mathcal{K}\}$$

EXAMPLE 3.3. Recall the program \mathcal{K}_{power} from Figure 4. The set $formula(\mathcal{K}_{power})$ is then

$$\{\text{func}(\text{ln}), \neg \text{func}(\text{ln}), \text{func}(\text{tr}) \wedge \text{func}(\text{ln}), \neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln}))\},$$

since these are the only formula appearing in \mathcal{K}_{power} .

The cardinality of $formula(\mathcal{K})$ for a given APT-Logic Program is bounded by $2|\mathcal{K}|$ since APT-Rules have two formulas, F and G . We notice that for each world w in $2^{B_{\mathcal{L}}}$ there is a subset of $formula(\mathcal{K})$ that w satisfies and a disjoint subset of $formula(\mathcal{K})$ that w does not satisfy. Hence, with respect to a given set of formulas, certain worlds are indistinguishable: that is, they satisfy exactly the same formulas from the set. We call such worlds \mathcal{K} -equivalent.

Definition 3.11 World Equivalence. For APT-logic program \mathcal{K} , a world w is \mathcal{K} -equivalent to a world w' (denoted $w \equiv_{\mathcal{K}} w'$) iff for all $F \in formula(\mathcal{K})$, $w \models F$ iff $w' \models F$. ■

EXAMPLE 3.4. Continuing with \mathcal{K}_{power} from Figure 4, recall the 4 worlds: $w_0 = \{\text{func}(\text{tr}), \text{func}(\text{ln})\}$, $w_1 = \{\text{func}(\text{tr})\}$, $w_2 = \{\text{func}(\text{ln})\}$, and $w_3 = \emptyset$ and the formula from \mathcal{K}_{power} :

$$\text{formula}(\mathcal{K}_{power}) = \{\text{func}(\text{ln}), \neg\text{func}(\text{ln}), \text{func}(\text{tr}) \wedge \text{func}(\text{ln}), \neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln}))\}.$$

Here w_1 is \mathcal{K}_{power} -equivalent to w_3 , since both w_1 and w_3 do not satisfy the first formula, do satisfy the second formula, do not satisfy the third formula, and do satisfy the fourth formula. However, w_1 is not \mathcal{K}_{power} -equivalent to w_2 since w_1 satisfies $\neg\text{func}(\text{ln})$ (the second formula), while w_2 does not.

The relation $\equiv_{\mathcal{K}}$ can be extended to threads in the obvious way.

Definition 3.12 *Thread Equivalence.* For APT-logic program \mathcal{K} , a thread Th_1 is \mathcal{K} -equivalent to a thread Th_2 (denoted $Th_1 \equiv_{\mathcal{K}} Th_2$) iff for all time points t , the world $Th_1(t)$ is \mathcal{K} -equivalent to world $Th_2(t)$. ■

EXAMPLE 3.5. In Example 3.4, we saw that w_1 is \mathcal{K}_{power} -equivalent to w_3 . Assuming four time points, then the thread $Th = \langle w_3, w_1, w_1, w_0 \rangle$ will be equivalent to $Th' = \langle w_1, w_3, w_3, w_0 \rangle$, since at every time point t , $Th(t)$ is a world that is \mathcal{K} -equivalent to world $Th'(t)$.

The relation $\equiv_{\mathcal{K}}$ is an equivalence relation (*i.e.*, it is transitive, reflexive, and symmetric) both for threads and for worlds; therefore, it can be used to construct a partitioning of threads into equivalence classes. Let $\mathcal{T}[\equiv_{\mathcal{K}}] = \{P_1, \dots, P_m\}$ be that partitioning. All threads in each P_i are \mathcal{K} -equivalent. The following result states that these partitions have the useful property that all threads in any partition P_i have the same value for *pfr*, *efr*, or *qfr* for formulas in $\text{formula}(\mathcal{K})$:

LEMMA 3.13. For APT-logic program \mathcal{K} , partitioning P_1, \dots, P_m of \mathcal{T} induced by $\equiv_{\mathcal{K}}$, for all threads $Th, Th' \in P_i$, all $F, G \in \text{formula}(\mathcal{K})$, and all Δt ;

- (1) $pfr(Th, F, G, \Delta t) = pfr(Th', F, G, \Delta t)$
- (2) $efr(Th, F, G, \Delta t) = efr(Th', F, G, \Delta t)$
- (3) $qfr(Th, F, G, \Delta t) = qfr(Th', F, G, \Delta t)$

Lemma 3.13 tells us that each partition P_i has a unique value for *pfr*, *efr*, and *qfr* (for each F , G , and Δt). We introduce the notation $pfr(P_i, F, G, \Delta t)$, $efr(P_i, F, G, \Delta t)$, and $qfr(P_i, F, G, \Delta t)$ to denote these values. For technical reasons, we associate a *label* with each thread Th such that all threads in the same partition P_i have the same label. To define the label, we first order the set $\text{formula}(\mathcal{K}) = \{F_1, \dots, F_n\}$. Then, for a thread Th , we assign $\text{label}(Th)$ to be a length $t_{max} \cdot n$ bitstring where bit $t' \cdot i$ ($1 \leq t' \leq t_{max}$ and $1 \leq i \leq n$) is 1 if $Th(t') \models F_i$ and 0 if $Th(t') \not\models F_i$.

Clearly, all Th, Th' in the same partition P_i have the same label. Also, all partitions P_i have a unique label equivalent to the labels of the contained threads and denoted $\text{label}(P_i)$. There are at most as many partitions as there are length $t_{max} \cdot n$ bitstrings, and determining if there is a partition associated with a given bitstring b can be done by checking if there is thread whose label is b .

EXAMPLE 3.6. Using \mathcal{K}_{power} from Figure 4, we number $\text{formula}(\mathcal{K}_{power})$ as follows:

$$\{F_1 = \text{func}(\text{ln}), F_2 = \neg\text{func}(\text{ln}), F_3 = \text{func}(\text{tr}) \wedge \text{func}(\text{ln}), F_4 = \neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln}))\}.$$

Here, the label for $Th = \langle w_3, w_1, w_1, w_0 \rangle$ (worlds w_i defined in Example 3.4) is

$$\underbrace{0101}_{w_3} \underbrace{0101}_{w_1} \underbrace{0101}_{w_1} \underbrace{1010}_{w_0}.$$

To see this, consider the first four digits 0101 for world w_3 . World w_3 does not satisfy F_1 , hence the first 0. It does, however, satisfy F_2 and F_4 causing the second and fourth digits to be 1.

The thread $Th' = \langle w_1, w_3, w_1, w_0 \rangle$ has the same label: 01010101011010; any two threads which are \mathcal{K}_{power} -equivalent will have the same labels.

We immediately notice that the number of thread partitions is potentially smaller than the number of threads. While there are $2^{B_{\mathcal{L}} \cdot t_{max}}$ threads, there are only $2^{|formula(\mathcal{K})| \cdot t_{max}} \leq 2^{2|\mathcal{K}| \cdot t_{max}}$ partitions. Therefore, using these partitions, rather than threads, is preferable in designing linear constraints. We can use Lemma 3.13 to construct smaller sets of linear constraints than SLC. For these constraints, we introduce the variable \hat{v}_{lbl} , where lbl is a length $t_{max} \cdot |formula(\mathcal{K})|$ bitstring ($lbl \in \{0, 1\}^{|formula(\mathcal{K})| \cdot t_{max}}$) representing the probability mass assigned to the set of threads in the partition labeled lbl ($\hat{v}_{lbl} = \sum_{Th \in P_i, label(P_i)=lbl} I(Th)$). We can now define the first alternative set of linear constraints.

Definition 3.14 World Equivalence Linear Constraints (WELC). Let \mathcal{K} be an APT-logic program that uses only the frequency functions pfr and efr ; the set of *World Equivalence Linear Constraints*, $WELC(\mathcal{K})$, contains exactly the following:

- (1) $\sum_i \hat{v}_i = 1$.
- (2) For $F \xrightarrow{fr} G : [\Delta t, \ell, u, \alpha, \beta]$
 - (a) $\sum_{lbl \in \{l | \alpha \leq fr(P_i, F, G, \Delta t) \leq \beta \wedge l = label(P_i)\}} \hat{v}_{lbl} \geq \ell$
 - (b) $\sum_{lbl \in \{l | \alpha \leq fr(P_i, F, G, \Delta t) \leq \beta \wedge l = label(P_i)\}} \hat{v}_{lbl} \leq u$
- (3) For $F \xrightarrow{tr} G : [\Delta t, \ell, u]$
 - (a) $\sum_{P_i} fr(P_i, F, G, \Delta t) \hat{v}_{label(P_i)} \geq \ell$
 - (b) $\sum_{P_i} fr(P_i, F, G, \Delta t) \hat{v}_{label(P_i)} \leq u$
- (4) For all $lbl \in \{0, 1\}^{|formula(\mathcal{K})| \cdot t_{max}}$ for which there is no P_i such that $lbl = label(P_i)$, $\hat{v}_{lbl} = 0$. ■

EXAMPLE 3.7. $WELC(\mathcal{K}_{power})$ (based on program \mathcal{K}_{power} from Figure 4) is constructed using variables \hat{v}_{lbl} for each of the $2^{4 \cdot 4} = 65,536$ possible labels. Due to constraint 4, at most 256 of these variables will be non-zero, since there are 256 worlds to populate these 65,536 possible equivalence classes. We will therefore be able to eliminate all but at most 256 of the variables from the representation altogether, since they will be known to be zero in every possible solution. As such, we only need to use the variables not eliminated via constraint 4 when constructing $WELC(\mathcal{K}_{power})$, and we will do so in this example. The only labels that will have associated threads are those that are combinations of the labels for the worlds w_0 , w_1 , w_2 , and w_3 (defined in Example 3.2). With $formula(\mathcal{K}_{power})$ being:

$\{F_1 = \text{func}(\text{ln}), F_2 = \neg \text{func}(\text{ln}), F_3 = \text{func}(\text{tr}) \wedge \text{func}(\text{ln}), F_4 = \neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln}))\}$, these labels are $lbl(w_0) = 1010$, $lbl(w_1) = 0101$, $lbl(w_2) = 1001$ and $lbl(w_3) = 0101$. So, for any label lbl , each four digit sequence must be 1010, 0101, or 1001. Otherwise

there cannot possibly be a thread Th such that $\text{label}(Th) = \text{lbl}$. In fact, since there are only 3 labels for the worlds (w_1 and w_2 , being $\mathcal{K}_{\text{power}}$ -equivalent, share a label), we know that when there are four time points, there are only $3^4 = 81$ variables that can be non-zero in our linear program (one label at each time point). So, leaving out the zeroing constraints and supposing each sum \sum_{lbl} sums over those 81 variables not known to be zero via the zeroing constraints, the set of linear constraints is: $\text{WELC}(\mathcal{K}_{\text{power}}) =$

- (1) $\sum_{\text{lbl}} \hat{v}_{\text{lbl}} = 1$
- (2) $0.025 \leq \sum_{\text{lbl}} \text{pfr}(Th_i, \text{func}(\text{tr}) \wedge \text{func}(\text{ln}), \neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln})), 1) \cdot \hat{v}_{\text{lbl}} \leq 0.03$
- (3) $0.95 \leq \sum_{\text{lbl}} \text{efr}(Th_i, \neg(\text{func}(\text{tr}) \wedge \text{func}(\text{ln})), \text{func}(\text{tr}) \wedge \text{func}(\text{ln}), 3) \cdot \hat{v}_{\text{lbl}} \leq 1$
- (4) $0.05 \leq \sum_{\text{lbl}} \text{pfr}(Th_i, \text{func}(\text{ln}), \neg\text{func}(\text{ln}), 1) \cdot \hat{v}_{\text{lbl}} \leq 0.1$
- (5) $0.99 \leq \sum_{\text{lbl}} \text{efr}(Th_i, \neg\text{func}(\text{ln}), \text{func}(\text{ln}), 2) \cdot \hat{v}_{\text{lbl}} \leq 1$

Note that this set of linear constraints is substantially smaller than $\text{SLC}(\mathcal{K}_{\text{power}})$, which used 256 variables where $\text{WELC}(\mathcal{K}_{\text{power}})$ uses only 81 variables and exactly the same number of constraints (after removal of trivial zeroing constraints).

PROPOSITION 3.15. *For any APT-program \mathcal{K} , $\text{WELC}(\mathcal{K})$ is solvable iff \mathcal{K} is consistent.*

This approach can provide a substantial speedup. As we noted earlier, the number of partitions is bounded by $2^{2^{|\mathcal{K}|} \cdot t_{\text{max}}}$ which will often be much smaller than the number of threads, $2^{|\mathcal{B}\mathcal{L}| \cdot t_{\text{max}}}$. Further, the number of partitions is bound by the number of threads, regardless of the size of \mathcal{K} .

PROPOSITION 3.16. *WELC requires $2^{|\mathcal{K}|} + 1$ constraints and at most $2^{2^{|\mathcal{K}|} t_{\text{max}}}$ variables.*

Algorithm 2 Compute consistency of \mathcal{K} using WELC.

WELC-CONSISTENT(APT-Program \mathcal{K})

- (1) Construct $\text{WELC}(\mathcal{K})$.
 - (2) Attempt to solve $\text{WELC}(\mathcal{K})$.
 - (3) If solvable, return *consistent*, otherwise, *inconsistent*.
-

This suggests Algorithm 2 for checking consistency of \mathcal{K} . The complexity of Algorithm 2 comes from both creating and solving WELC. Proposition 3.16 gives the number of constraints required of a linear program to implement WELC-CONSISTENT. Building WELC is also difficult: we have constraint 4, which requires the inclusion of the constraint $\hat{v}_{\text{lbl}} = 0$ if there is no non-empty partition in $\mathcal{T}[\equiv_{\mathcal{K}}]$ with label lbl . Unfortunately, this is an NP-complete operation.

THEOREM 3.17. *For APT-Logic Program, \mathcal{K} , and label lbl , determining if there is non-empty $P_i \in \mathcal{T}[\equiv_{\mathcal{K}}]$ such that $\text{label}(P_i) = \text{lbl}$ is NP-complete.*

To properly construct WELC, we must solve SAT for every subset of $\text{formula}(\mathcal{K})$. As $\text{formula}(\mathcal{K}) \leq 2^{|\mathcal{K}|}$, this amounts to $O(2^{2^{|\mathcal{K}|}})$ calls to a SAT solver. Assuming $O(2^{|\mathcal{B}\mathcal{L}|})$ operations per SAT solution procedure, this operation will take time

Thread	$pfr(Th, \text{scandal}, \neg\text{scandal}, 1)$
$\langle \text{scandal}, \text{scandal}, \text{scandal} \rangle$	0
$\langle \text{scandal}, \text{scandal}, \neg\text{scandal} \rangle$	1/2
$\langle \text{scandal}, \neg\text{scandal}, \text{scandal} \rangle$	1
$\langle \text{scandal}, \neg\text{scandal}, \neg\text{scandal} \rangle$	1
$\langle \neg\text{scandal}, \text{scandal}, \text{scandal} \rangle$	0
$\langle \neg\text{scandal}, \text{scandal}, \neg\text{scandal} \rangle$	1
$\langle \neg\text{scandal}, \neg\text{scandal}, \text{scandal} \rangle$	1
$\langle \neg\text{scandal}, \neg\text{scandal}, \neg\text{scandal} \rangle$	1

Fig. 7. For a set of atoms consisting of `scandal`, and t_{max} of 3 time points, the above chart shows the pfr for all possible threads based on a program consisting only of rule `scandal \xrightarrow{pfr} \neg scandal` : [1, 0.89, 0.93, 0.8, 1.0] from Figure 1. Figure 8 groups these threads in frequency equivalence classes based on pfr .

$O(2^{2|\mathcal{K}|+|B_{\mathcal{L}}|})$. However, as for most linear program implementations, the running time for WELC-CONSISTENT will be exponential in terms of $7|\mathcal{K}|t_{max}$ [Karmarkar 1984], the generation of world equivalence classes will be dominated by WELC itself. Therefore, in most cases, Algorithm 2 will have a better big-O run time than solving the set of straightforward linear constraints.

3.4 Frequency Equivalence

For constrained rules it is possible to develop a different set of linear constraints. Rather than considering equivalent worlds, we develop a partition of the set of threads based on the value of the frequency function with respect to each rule in the program. We will then create a new set of linear constraints based on this equivalence, as with WELC, in order to improve performance.

Therefore, the partitions will depend on the thread's relationship to the probability interval $[\alpha, \beta]$, which we shall refer to as the *frequency bounds* for a given rule. Due to the requirement of considering the frequency bounds, this type of thread equivalence will be referred to as *frequency equivalence* and apply only to constrained rules, though there are manipulations one can apply to include annotated formulas; we first define an equivalence relation over threads.

Definition 3.18 Frequency Equivalence. For threads Th_1 and Th_2 , and constrained rule $r = F \xrightarrow{fr} G : [\Delta t, \ell, u, \alpha, \beta]$, we say Th_1 is r -frequency-equivalent to Th_2 (denoted $Th_1 \sim^r Th_2$) iff $(\alpha \leq fr(Th_1, F, G, \Delta t) \leq \beta \Leftrightarrow \alpha \leq fr(Th_2, F, G, \Delta t) \leq \beta)$. For APT-Logic Program \mathcal{K} containing only constrained conditionals, we say Th_1 is \mathcal{K} -frequency-equivalent to Th_2 (denoted $Th_1 \sim^{\mathcal{K}} Th_2$) iff for all rules $r \in \mathcal{K}$, $Th_1 \sim^r Th_2$. ■

EXAMPLE 3.8. Consider rule `scandal \xrightarrow{pfr} \neg scandal` : [1, 0.89, 0.93, 0.8, 1.0] from Figure 1, where we used APT-Rules to represent the behavior of stock price based on news reports. Let \mathcal{K}_{fr-ex} be an APT-program containing exactly this rule. We will consider the set of atoms to consist only of `scandal` and t_{max} to be 3. In Figure 7 we compute the pfr based on this single rule for all possible threads. In Figure 8 we can then group these threads into two equivalence classes, those whose pfr is within [0.8, 1] and those whose frequency is outside this range.

$$\mathcal{K}_{fr-ex} = \{\text{scandal} \xrightarrow{pfr} \neg\text{scandal} : [1, 0.89, 0.93, 0.8, 1.0]\}$$

$$\mathcal{T}[\sim^{\mathcal{K}_{fr-ex}}] = \left\{ \begin{array}{l} E_1 = \left\{ \begin{array}{l} \langle \text{scandal}, \neg\text{scandal}, \text{scandal} \rangle, \\ \langle \text{scandal}, \neg\text{scandal}, \neg\text{scandal} \rangle, \\ \langle \neg\text{scandal}, \text{scandal}, \neg\text{scandal} \rangle, \\ \langle \neg\text{scandal}, \neg\text{scandal}, \text{scandal} \rangle, \\ \langle \neg\text{scandal}, \neg\text{scandal}, \neg\text{scandal} \rangle \end{array} \right\}, \\ E_2 = \left\{ \begin{array}{l} \langle \text{scandal}, \text{scandal}, \text{scandal} \rangle, \\ \langle \text{scandal}, \text{scandal}, \neg\text{scandal} \rangle, \\ \langle \neg\text{scandal}, \text{scandal}, \text{scandal} \rangle \end{array} \right\} \end{array} \right\}$$

Fig. 8. For a program consisting only of rule $\text{scandal} \xrightarrow{pfr} \neg\text{scandal} : [1, 0.89, 0.93, 0.8, 1.0]$ from Figure 1, we have frequency equivalence classes E_1 and E_2 based on the pfr for all possible threads seen in Figure 7.

For instance, threads $\langle \text{scandal}, \text{scandal}, \text{scandal} \rangle$ and $\langle \text{scandal}, \text{scandal}, \neg\text{scandal} \rangle$ both have a pfr less than 0.8. Therefore, we have that $\langle \text{scandal}, \text{scandal}, \text{scandal} \rangle \sim^{\mathcal{K}_{fr-ex}} \langle \neg\text{scandal}, \text{scandal}, \text{scandal} \rangle$.

The relation $\sim^{\mathcal{K}}$ satisfies several common properties of relations.

PROPOSITION 3.19. *For any constrained APT-logic program \mathcal{K} , $\sim^{\mathcal{K}}$ is reflexive, symmetric, and transitive.*

Therefore $\sim^{\mathcal{K}}$ is an equivalence relation, and we can partition \mathcal{T} (the set of all possible threads) into equivalence classes according to a given $\sim^{\mathcal{K}}$. We let $\mathcal{T}[\sim^{\mathcal{K}}]$ be this partitioning, where each set $E \in \mathcal{T}[\sim^{\mathcal{K}}]$ contains only \mathcal{K} -frequency-equivalent threads. We then assign each set E a binary string $str(E)$ of length m (the number of constrained formulas in \mathcal{K}) where digit i is 1 if for all $Th \in E$, $\alpha_i \leq fr(Th, F_i, G_i, \Delta t_i) \leq \beta_i$, and 0 otherwise.

EXAMPLE 3.9. *In Figure 8 we see a partitioning of the threads $\mathcal{T}[\sim^{\mathcal{K}_{fr-ex}}]$ with two partitions: E_1 and E_2 . The associated binary strings are: $str(E_1) = 1$ and $str(E_2) = 0$. Notice that we only have two frequency equivalence classes of threads, which is only 25% of the 8 threads we had originally.*

In the following linear program, we introduce variables \bar{v}_b for each binary string b of length $|\mathcal{K}|$.

Definition 3.20 Frequency-Equivalence Linear Constraints. For constrained APT-Logic Program \mathcal{K} , the set of Frequency-Equivalence Linear Constraints $FELC(\mathcal{K})$ contains only the following:

- (1) $\sum_{E \in \mathcal{T}[\sim^{\mathcal{K}}]} \bar{v}_{str(E)} = 1$ (where $str(E)$ is the binary number that labels frequency equivalence class E)
- (2) For all length $|\mathcal{K}|$ binary strings b if there is no $E \in \mathcal{T}[\sim^{\mathcal{K}}]$ such that $str(E) = b$ then $\bar{v}_b = 0$
- (3) For all $F_i \xrightarrow{fr} G_i : [\Delta t_i, \ell_i, u_i, \alpha_i, \beta_i] \in \mathcal{K}$, $\ell_i \leq \sum_{s \in [0,1]^m, s_i=1} \bar{v}_s \leq u_i$ ■

THEOREM 3.21. *For constrained APT-Logic Program \mathcal{K} , \mathcal{K} is consistent iff there is a solution to FELC(\mathcal{K}).*

As FELC provides a correct result for consistency, we can use it to develop the consistency-checking algorithm FELC-CONSISTENT shown below.

Algorithm 3 Compute consistency of \mathcal{K} using FELC.

FELC-CONSISTENT(*APT-Program* \mathcal{K})

- (1) Construct FELC(\mathcal{K}).
 - (2) Attempt to solve FELC(\mathcal{K}).
 - (3) If solvable, return *consistent*, otherwise, *inconsistent*.
-

If the frequency equivalence classes of threads for a given program are known, FELC also offers an improvement in complexity over SLC.

PROPOSITION 3.22. *FELC requires $2|\mathcal{K}| + 1$ constraints and $2^{|\mathcal{K}|}$ variables.*

EXAMPLE 3.10. *Consider the APT-Program \mathcal{K}_{stock} from Figure 1. Let $B_{\mathcal{L}}$ be the set of atoms seen in that program (hence $|B_{\mathcal{L}}| = 5$). We consider a t_{max} of 4. From Proposition 3.10, we know that using SLC to determine the consistency of \mathcal{K}_{stock} would require 7 constraints and $2^{20} = 1,048,576$ variables. We show below a set of linear constraints based on FELC below that requires 7 constraints and only $2^3 = 8$ variables. For the program \mathcal{K}_{stock} , we have the following linear constraints:*

- For rule *scandal* $\xrightarrow{pfr} \neg \text{scandal} : [1, 0.89, 0.93, 0.8, 1.0]$
 $0.89 \leq \bar{v}_{001} + \bar{v}_{011} + \bar{v}_{101} + \bar{v}_{111} \leq 0.93$
- For rule *sec_rumor* \wedge *earn_incr(10%)* \xrightarrow{pfr} *stock_decr(10%)* : $[2, 0.65, 0.97, 0.7, 1.0]$
 $0.65 \leq \bar{v}_{010} + \bar{v}_{011} + \bar{v}_{110} + \bar{v}_{111} \leq 0.97$
- For rule *sec_rumor* \wedge *earn_incr(10%)* \xrightarrow{pfr} *stock_decr(10%)* \wedge *cfo_resigns* : $[2, 0.68, 0.95, 0.7, 0.8]$
 $0.68 \leq \bar{v}_{100} + \bar{v}_{101} + \bar{v}_{110} + \bar{v}_{111} \leq 0.95$
- $\bar{v}_{000} + \bar{v}_{001} + \bar{v}_{010} + \bar{v}_{011} + \bar{v}_{100} + \bar{v}_{101} + \bar{v}_{110} + \bar{v}_{111} = 1$

The running time of consistency checking via FELC is *independent* of the number of atoms or time points or number of worlds. Thus, even though it runs in time exponential in $|\mathcal{K}|$, it will in many cases run faster than SLC, which runs in time linear in $|\mathcal{K}|$ and exponential in the number of worlds or the number of time points. Further, since the size of \mathcal{K} , the number of worlds, and the number of time points are all known in advance, one can tell which approach will be faster dynamically, and dispatch the smaller, faster linear program.

However, as with WELC, significant computation cost is required to construct the linear constraints, specifically in identifying the frequency equivalence classes that are empty. We refer to the obvious, exhaustive, and exact method for identifying empty frequency equivalence classes as the Brute Force Frequency Equivalence Class Algorithm or BFECA.

As BFECA exhaustively considers all threads, we have the following trivial proposition concerning correctness.

Algorithm 4 Find Frequency Equivalence Classes of Constrained Program \mathcal{K}
 BFECA(*APT-Program* \mathcal{K})

- (1) Generate all possible threads.
 - (2) For each thread, Th , for all i , compute $fr_i(Th, F_i, G_i, \Delta t_i)$.
 - (3) Determine for each thread, Th , for each rule, r_i , if the associated frequency function, fr_i for Th falls within the range $[\alpha_i, \beta_i]$.
 - (4) Based on the result of step 3, determine which frequency equivalence class Th belongs to.
 - (5) After all threads are generated, return EMPTY if there are no threads found for a given frequency equivalence class is empty and OK otherwise.
-

PROPOSITION 3.23. *For each frequency equivalence class C , if C is empty BFECA returns EMPTY; otherwise, if C contains at least one thread, BFECA returns OK.*

For each thread, BFECA calculates the frequency function with regard to each rule. Hence, for each of the $2^{|B_{\mathcal{L}}|t_{max}}$ threads, it calculates $|\mathcal{K}|$ frequency functions. This leads us to the complexity result below.

PROPOSITION 3.24. *The complexity of BFECA is:*

$$O\left(2^{|B_{\mathcal{L}}|t_{max}} \cdot F(t_{max}) \cdot |\mathcal{K}|\right)$$

where $F(t_{max})$ is defined as follows. Suppose $time_i$ is the time required to compute $fr_i(Th, F_i, G_i, \Delta t_i)$. Then $F(t_{max})$ equals $\max_i(time_i)$.

Note that if $F(t_{max})$ is linear, then the complexity of finding the frequency equivalence classes and then performing FELC is still better than SLC. The dominating term in the complexity of FELC has an exponent of $|B_{\mathcal{L}}| \cdot t_{max}$ when BFECA is used. SLC, on the other hand, will have an exponent of $3.5 \cdot |B_{\mathcal{L}}| \cdot t_{max}$ for most linear program solvers [Karmarkar 1984]. The following example shows how BFECA works.

EXAMPLE 3.11. *Consider the FELC constraints set up for \mathcal{K}_{stock} in Example 3.10. Look at rules $sec_rumor \wedge earn_incr(10\%) \xrightarrow{pfr} stock_decr(10\%) : [2, 0.65, 0.97, 0.7, 1.0]$ and $sec_rumor \wedge earn_incr(10\%) \xrightarrow{pfr} stock_decr(10\%) \wedge cfo_resigns : [2, 0.68, 0.95, 0.7, 0.8]$. For a given thread, Th , consider the pfr's associated with those rules. Let $p_1 = pfr(Th, sec_rumor \wedge earn_incr(10\%), stock_decr(10\%), 2)$ and $p_2 = pfr(Th, sec_rumor \wedge earn_incr(10\%), stock_decr(10\%) \wedge cfo_resigns, 2)$. We note that p_2 **must** be less than or equal to p_1 as the G formula for both rules differs only by one conjuncted atom. Therefore, there is no possible Th such that $p_2 > p_1$. Hence, variables \bar{v}_{100} and \bar{v}_{101} from the FELC constraints in Example 3.10 **must** be set to zero.*

To find such variables, BFECA calculates the frequency function for all possible threads. However, with SLC-CONSISTENT, the dominating term in this example requires 2^{70} operations, where BFECA requires only 2^{20} operations. Note that the complexity of BFECA often will dominate the complexity of FELC-CONSISTENT.

As suggested earlier, FELC can be used on programs that consist of both constrained rules and annotated formulas. We can include annotated formulas in our constrained program by writing rules that are essentially equivalent to annotated formulas, as described earlier through use of the Query Frequency Function in Definition 2.18.

Note that if the PCD conditions are met (Page 16), we can often be guaranteed that all FELC equivalence classes will be non-empty, making the BFECA algorithm unnecessary. See the Appendix for a complete discussion of this special case.

3.5 Combining World and Frequency Equivalence

We have introduced two improved methods for computing consistency: FELC-CONSISTENT/BFECA and WELC-CONSISTENT. We now introduce a hybrid approach that uses the world-equivalence classes of WELC to ease the computation necessary to compute the frequency-equivalence classes needed in FELC. World-equivalence can be used to determine if a frequency equivalence class is empty or not. The intuition is simple: we follow the approach of BFECA, generating the set of threads and finding the frequency function for each one. However, rather than generating the set of threads, we generate the set of world-based thread partitions and find their frequency functions. As shown in the discussion of WELC, the number of world-based thread partitions can be considerably less than the number of threads. Hence, we present world equivalence for finding frequency equivalence, or WEFE.

Algorithm 5 World Equivalence for finding Frequency Equivalence Classes of Constrained Program \mathcal{K}

WEFE(*APT-Program* \mathcal{K})

- (1) Find the world equivalence classes based on $formula(\mathcal{K})$.
 - (2) Generate all world-equivalence based thread partitions for \mathcal{K} .
 - (3) For each world-equivalence thread partition, P , for all i , compute $fr_i(P, F_i, G_i, \Delta t_i)$.
 - (4) For each rule, r_i let IN_i be the set of thread partitions such that $\alpha_i \leq fr_i(P, F_i, G_i, \Delta t_i) \leq \beta_i$. For each rule, let OUT_i be all partitions not in IN_i .
 - (5) For string $s \in [0, 1]^{|\mathcal{K}|}$ let the set $PCLASS_s$ be defined as $\{\bigcap_{s_i=1} IN_i\} \cap \{\bigcap_{s_i=0} OUT_i\}$.
 - (6) For each class cl_s return EMPTY if $PCLASS_s \equiv \emptyset$ and OK otherwise.
-

As WEFE exhaustively considers all world equivalence based thread partitions, and each thread belongs to exactly one partition, WEFE provides a correct answer.

PROPOSITION 3.25. *If a given frequency equivalence class is empty, WEFE returns EMPTY. If there is a thread in a given frequency equivalence class, WEFE returns OK.*

The computational complexity of this algorithm is dependent upon the number of thread-partitions resulting from world-equivalence. As stated before, this is

$2^{2^{|\mathcal{K}|} \cdot t_{max}}$. Further, the cost of calculating the frequency function for each thread is only $O(t_{max})$ as checking the satisfiability of the F and G formulas in a rule by a world equivalence class is a trivial operation, since the satisfaction is pre-determined when the world-equivalence classes are generated.

PROPOSITION 3.26. *The complexity of WEFE is*

$$O\left(2^{2^{|\mathcal{K}|} \cdot t_{max}} \cdot t_{max} \cdot |\mathcal{K}|\right)$$

when the set of world-equivalence classes for \mathcal{K} is known.

WEFE/FELC-CONSISTENT is generally preferable for checking the consistency of constrained programs: because it considers threads on a world-equivalence basis rather than individually, it should generally have a shorter run time than BFECA even taking into account the costs of constructing world-equivalence classes. We illustrate this in the following example:

EXAMPLE 3.12. *Suppose we want to build FELC constraints for \mathcal{K}_{stock} as we did in Example 3.10 where $t_{max} = 4$. We note that $formula(\mathcal{K}_{stock})$ consists of the following:*

- (1) scandal
- (2) \neg scandal
- (3) sec_rumor \wedge earn_incr(10%)
- (4) stock_decr(10%)
- (5) stock_decr(10%) \wedge cfo_resigns

Although the number of world equivalence classes, based on $formula(\mathcal{K}_{stock})$ would be 2^5 , which is also the number of worlds due to there only being 5 atoms referenced in the program, we note that many of the world equivalence classes are empty. For example, we know that there can be no world that satisfies both of the first two formulas, which immediately reduces our number of world equivalence classes by a factor of two. Further, there can be no world that does not satisfy `stock_decr(10%)` but satisfies `stock_decr(10%) \wedge cfo_resigns`. Hence, the number of world equivalence classes is 12 in this case, a significant reduction from the 32 worlds originally considered.

Therefore, WEFE only considers $12^4 = 20,736$ world-equivalent threads, as opposed to BFECA, which considers $32^4 = 1,048,576$ threads. Note that if the world equivalence classes are known, this cost of WEFE may still dominate FELC-CONSISTENT. This is a vast improvement over the 2^{70} operations required by SLC-CONSISTENT.

4. ENTAILMENT BY APT-LOGIC PROGRAMS

Now that we have dealt with consistency, we can explore the issue of entailment, which is defined in the usual way.

Definition 4.1 Entailment. Let \mathcal{K} be an APT-logic program, r be a rule, and af be an annotated formula. We say that \mathcal{K} entails af iff for all models I of \mathcal{K} , $I \models af$, and that \mathcal{K} entails r iff for all models I of \mathcal{K} , $I \models r$. ■

EXAMPLE 4.1 ENTAILMENT. Recall that in Example 3.8 we presented the following APT-Program:

$$\mathcal{K}_{fr-ex} = \{\text{scandal} \xrightarrow{pfr} \neg \text{scandal} : [1, 0.89, 0.93, 0.8, 1.0]\}$$

Suppose we form the following rule as a hypothesis.

$$r_{hyp} = \text{scandal} \xrightarrow{pfr} \neg \text{scandal} : [1, 0.88, 0.94, 0.8, 1.0]$$

Does \mathcal{K}_{fr-ex} entail r_{hyp} ? A quick examination of the only rule in the program and the hypothesis tells us that except for the probability bounds, they are the same. Notice that the rule in \mathcal{K}_{fr-ex} has probability bounds $[0.89, 0.93]$ and the probability bounds of r_{hyp} are a superset, $[0.88, 0.94]$. Therefore, we know that any interpretation in which the sums of the probabilities of threads with a frequency ratio between $[0.8, 1.0]$ sum to a quantity in $[0.89, 0.93]$, are also in $[0.88, 0.94]$. So, by the definitions of satisfaction and entailment, we can say that \mathcal{K}_{fr-ex} entails r_{hyp} .

The following result shows that checking entailment of an annotated formula by an APT-logic program is coNP-hard.

THEOREM 4.2. Given an APT-logic program \mathcal{K} and an annotated formula, af , deciding if \mathcal{K} entails af is coNP-hard in $|B_{\mathcal{L}}|$ (the number of atoms).

4.1 Linear Constraints for Entailment

We shall now provide algorithms for computing entailment based on the linear constraints SLC, WELC, and FELC. In all cases, the method is straightforward: we determine the minimal and maximal probability for the annotated formula in interpretations satisfying the original knowledgebase by minimizing and maximizing the appropriate sum subject to some set of linear constraints. Due to the fact that any annotated formula can be viewed as a constrained rule, we will not describe the entailment of annotated formulas in this section.

Algorithm 6 Entailment of Rule r by Program \mathcal{K} with SLC

SLC-ENT(APT-Program \mathcal{K})

- (1) If r is unconstrained, ($r = F \xrightarrow{fr} G : [\Delta t, \ell, u]$), create rule $r' = F \xrightarrow{fr} G : [\Delta t, \ell', u']$ where ℓ', u' are variables.
 - (2) If r is constrained, ($r = F \xrightarrow{fr} G : [\Delta t, \ell, u, \alpha, \beta]$) create rule $r' = F \xrightarrow{fr} G : [\Delta t, \ell', u', \alpha, \beta]$ where ℓ', u' are variables.
 - (3) Create set of linear constraints $SLC(\mathcal{K} \cup \{r'\})$.
 - (4) Let $\bar{\ell}'$ be the minimization of ℓ' subject to $SLC(\mathcal{K} \cup \{r'\})$.
 - (5) Let \bar{u}' be the maximization of u' subject to $SLC(\mathcal{K} \cup \{r'\})$.
 - (6) If $[\bar{\ell}', \bar{u}'] \subseteq [\ell, u]$ return ENTAILS otherwise return NOT ENTAILS.
-

We can show Algorithm 6 to be correct and to take time exponential in $|B_{\ell}|$ (as expected due to Theorem 4.2).

PROPOSITION 4.3 CHECKING ENTAILMENT USING SLC. *For unconstrained rule $r = F \xrightarrow{fr} G : [\Delta t, \ell, u]$ or constrained rule $r = F \xrightarrow{fr} G : [\Delta t, \ell, u, \alpha, \beta]$ and program \mathcal{K} , SLC-ENT returns ENTAILS iff \mathcal{K} entails r and returns NOT ENTAILS iff \mathcal{K} does not entail r*

PROPOSITION 4.4. *SLC-ENT requires solving at most two linear programs. Each linear program has $2|\mathcal{K}| + 1$ constraints and $2^{|\mathcal{B}_{\mathcal{L}}| \cdot t_{max}}$ variables.*

We now give an example of how Algorithm 6 will run in practice.

EXAMPLE 4.2. *Consider APT-Program \mathcal{K}_{stock} introduced in Figure 1 with $t_{max} = 4$. Suppose we want to see if \mathcal{K}_{stock} entails the annotated formula query = earn_decr(10%) : [3, 0.50, 0.80].*

First, we re-write the query as a rule using qfr. Hence, $query_{rule} = \text{TRUE} \xrightarrow{qfr} \text{earn_decr}(10\%) : [3, 0.50, 0.80, 1, 1]$. From this rule, we create $query'_{rule} = \text{TRUE} \xrightarrow{qfr} \text{earn_decr}(10\%) : [3, \ell', u', 1, 1]$.

We now consider all possible threads given $\mathcal{K}_{stock} \cup \{query'_{rule}\}$ and $t_{max} = 4$. As there are 6 atoms in the union of the program and query, we have $2^{2^4} = 16,777,216$ possible threads ($|\mathcal{T}| = 2^{2^4}$). Hence, we set up the following linear constraints:

- For rule $scandal \xrightarrow{pfr} \neg scandal : [1, 0.89, 0.93, 0.8, 1.0]$
 $0.89 \leq \sum_{Th_j \in \mathcal{T}}_{0.8 \leq pfr(Th_j, scandal, \neg scandal, 1) \leq 1.0} v_j$
 $0.93 \geq \sum_{Th_j \in \mathcal{T}}_{0.8 \leq pfr(Th_j, scandal, \neg scandal, 1) \leq 1.0} v_j$
- For rule $sec_rumor \wedge \text{earn_incr}(10\%) \xrightarrow{pfr} \text{stock_decr}(10\%) : [2, 0.65, 0.97, 0.7, 1.0]$
 $0.65 \leq \sum_{Th_j \in \mathcal{T}}_{0.7 \leq pfr(Th_j, sec_rumor \wedge \text{earn_incr}(10\%), \text{stock_decr}(10\%), 2) \leq 1.0} v_j$
 $0.97 \geq \sum_{Th_j \in \mathcal{T}}_{0.7 \leq pfr(Th_j, sec_rumor \wedge \text{earn_incr}(10\%), \text{stock_decr}(10\%), 2) \leq 1.0} v_j$
- For rule $sec_rumor \wedge \text{earn_incr}(10\%) \xrightarrow{pfr} \text{stock_decr}(10\%) \wedge \text{cfo_resigns} : [2, 0.68, 0.95, 0.7, 0.8]$
 $0.68 \leq \sum_{Th_j \in \mathcal{T}}_{0.7 \leq pfr(Th_j, sec_rumor \wedge \text{earn_incr}(10\%), \text{stock_decr}(10\%) \wedge \text{cfo_resigns}, 2) \leq 0.8} v_j$
 $0.95 \geq \sum_{Th_j \in \mathcal{T}}_{0.7 \leq pfr(Th_j, sec_rumor \wedge \text{earn_incr}(10\%), \text{stock_decr}(10\%) \wedge \text{cfo_resigns}, 2) \leq 0.8} v_j$
- For rule $query'_{rule} = \text{TRUE} \xrightarrow{qfr} \text{earn_decr}(10\%) : [3, \ell', u', 1, 1]$
 $\ell' \leq \sum_{Th_j \in \mathcal{T}}_{1 \leq qfr(Th_j, \text{TRUE}, \text{earn_decr}(10\%), 3) \leq 1.0} v_j$
 $u' \geq \sum_{Th_j \in \mathcal{T}}_{1 \leq qfr(Th_j, \text{TRUE}, \text{earn_decr}(10\%), 3) \leq 1.0} v_j$
- $\sum_{j=0}^{2^{2^4}} v_j = 1.$

*As it turns out, the minimization of ℓ' is 0 and the maximization of u' is 1. Since $[0, 1] \not\subseteq [0.5, 0.8]$, we can say that \mathcal{K}_{stock} does **not** entail query.*

SLC-ENT uses the SLC set of linear constraints. However, one could easily substitute WELC or FELC for SLC in SLC-ENT. We present an algorithm for alternate linear constraints, ALC-ENT, that mirrors SLC-ENT and leverages these other constraints in the appendix.

There is a further improvement that can be made in practice: if we solve the linear program once, and find that the minimization of ℓ' is less than ℓ , we have

determined that the rule is not entailed by the program, and solving the linear program again is not necessary to decide entailment.

5. APPLICATIONS OF APT LOGIC

Algorithm 7 The APT-Extract Algorithm.

APT-Extract(T , $ActCond$, $MaxBody$, Δ , $SuppLB$, σ , STAT-Test)

- (1) $Rules := \emptyset$;
 - (2) for each combination (*environment variable, value*) choose $1, \dots, MaxBody$ {
 - (3) let $Body$ be the current combination; $supportBody := 0$; $supportBoth := 0$;
 - (4) for $t = 1$ to $maxTime(T)$ {
 - (5) $bodyHappened := false$;
 - (6) if $Body$ is true at time t then
 - (7) $bodyHappened := true$; $actHappened := false$;
 - (8) for $d = 1$ to Δ {
 - (9) if $ActCond$ is true at time $t + d$ then $actHappened := true$;
 - (10) break for;
 - (11) }
 - (12) if $bodyHappened$ then $supportBody := supportBody + 1$;
 - (13) if $bodyHappened$ and $actHappened$ then $supportBoth := supportBoth + 1$;
 - (14) }
 - (15) if $supportBody \neq 0$ then $confidence := supportBoth / supportBody$;
 - (16) else $confidence := 0$;
 - (17) if $(supportBoth > suppLB) \wedge STAT_TEST(Body, ActCond)$ then
 - (18) add $Body \stackrel{pfr}{\rightsquigarrow} ActCond : [\Delta, confidence - \sigma, confidence + \sigma]$ to $Rules$;
 - (19) }
 - (20) return $Rules$;
-

APT-logic programs have many possible applications; in this section we will briefly describe an effort to learn conditions under which various terror groups took various actions, in the form of APT-programs. We assume that the data is given in the form of a table that contains two kinds of attributes: *action* and *environment*, and that each tuple represents the values of each of these attributes for a certain time point. A good example of this kind of data is the “Minorities at Risk Organizational Behavior” (MAROB) data set [Wilkenfeld et al. 2007]. This data set has identified around 150 parameters to monitor for about 300 groups around the world that are either involved in terrorism or are at risk of becoming full-fledged terrorist organizations. The 150 attributes describe aspects of these groups, such as whether or not the group engaged in violent attacks, if financial or military support was received from foreign governments, and the type of leadership the group has. It was a simple task to divide the attributes into actions that could be taken by the group (*i.e.*, bombings, kidnappings, armed attacks, etc.) and environmental conditions (*i.e.*, the type of leadership, the kind and amount of foreign support, whether the group has a military wing, etc.). Values for these 150 parameters

are available for up to 24 years per group, though it is less for some groups (*e.g.*, groups that have been around for a shorter duration). For each group, MAROB provides a table whose columns correspond to the 150 parameters and the rows correspond to the years. There are many social science data sets that use such data. These include the KEDS data set from the University of Kansas that tracks country stability data (rather than terror group data) [Schrodt and Gerner 1998] and the Political Instability Task Force (PITF) data [Goldstone et al. 2005].

The APT-Extract algorithm provides a basic approach to extracting APT-rules⁵. The inputs are: a table of historic data, a condition on an action variable (variable name and value), a maximum size for the body, a value for Δ , a lower bound for the *support* of the rule, and a real number $\sigma \in [0, 1]$ that will determine the width of the probability annotations for the extracted rules, and an arbitrary statistical test (*e.g.*, a t-test or something based on p-values in statistics) selected by the user that measures the correlation between the values of the body of a possible rule and the head. We use the standard measurements of support and confidence from the literature on association rules: given table T , the *support* of a condition C in T is the number of tuples for which C is true; given conditions C_1 and C_2 , the confidence in the fact that C_1 is accompanied by C_2 is the ratio of the support of $C_1 \wedge C_2$ to the support of C_1 . As an example of the kind of rules that can be extracted by this algorithm, some of the rules extracted from the data for Hezbollah are given in Figure 2.

6. RELATED WORK

In addition to the authors' past work on probabilistic logic programming [Ng and Subrahmanian 1992; 1991], probabilistic logic programs were studied in [Kiessling et al. 1992], [Kifer and Subrahmanian 1992], and [Lakshmanan and Sadri 1994a; 1994b; Lakshmanan and Shiri 1997], who showed how to introduce various probabilistic dependencies into probabilistic LPs. [Lukasiewicz 1999; Lukasiewicz et al. 1999] made major contributions to bottom up computations of probabilistic LPs.

[Lehmann and Shelah 1982] and [Hart and Sharir 1986] were among the first to provide a logic to integrate time and probability. [Kanazawa 1991] also studied the integration of time and probability in order to facilitate efficient planning. He was primarily interested in how the probability of facts and events change over time. [Haddawy 1991] developed a logic for reasoning about actions, probability and time using an interval time model. [D. Dubois and Prade 1991] developed methods to extend possibilistic logic to handle temporal information. This logic associates, with each formula of possibilistic logic, a set of time points describing when the formula has a possibilistic truth value. [Halpern and Tuttle 1992] studied the semantics of reasoning about distributed systems where uncertainty is present using a logic where a process has knowledge about the probability of events for decision making by the process. [Fagin et al. 1990; Fagin and Halpern 1994] developed logics of time and belief to model the behavior of distributed systems, while [Thomas 1995] developed a framework that integrates beliefs, time, commitment, desires, and multiple agents.

⁵Note that this algorithm is not a novel one, and simply performs calculations to capture interesting relationships present in the data in order to build rules. More complex algorithms for rule extraction are outside the scope of this paper.

[Baral et al. 2002] developed a language to reason about actions in a probabilistic setting; their models use static and dynamic causal laws together with background (unknown) variables whose values are determined by factors not in the model. Building on top of past work by [Dekhtyar et al. 1999], [Dix et al. 2006] introduce heterogeneous temporal probabilistic agents to model agent behavior and develop a model theory and fixpoint semantics focusing on agents built using legacy code.

Though there has been extensive work on temporal reasoning, the key difference between APT logic programs and past works in verification [Lamport 1980; Emerson and Halpern 1984; Vardi 1985; R. Cleaveland and Narasimha 2005; Glabbeek et al. 1995; Larsen and Skou 1991] is the use of frequency functions in our work to define the frequency with which a given formula G holds (some given time) after a given formula F holds. We show that such a definition can be given in many different ways and, rather than committing to one such definition, we provide axioms that any frequency function should satisfy. A result of our introduction of the frequency function is that the probability an event occurs at time t is dependent on the events that occur in interval $[1, t]$ and interval $[t, t_{max}]$.

APT-Logic distinguishes itself from other temporal logics in the following ways:

- (1) It provides for reasoning about probability of events within a sequence of events and probabilistic comparison between sequences of events.
- (2) Future worlds can depend on more than just the current world.
- (3) It provides bounds on probabilities rather than just a point probability.
- (4) It does not make any independence assumptions.

6.1 Markov Decision Processes

Many temporal logics, whether probabilistic or not, make use of some sort of state transition system as an underlying structure. A state-transition system is said to conform to the *Markov Property* if each transition probability only depends on the current state [Russell and Norvig 2003]. We demonstrate that while APT-Logic Programs maintain much of the expressiveness of most state-transition systems, they also have the ability of expressing non-Markovian sequences of events. Specifically, the semantic structures used in APT-Logic (worlds, threads, interpretations) can be represented by state transition systems when the following restrictions are applied:

- (1) As APT-Logic only deals with finite temporal sequences, only the first t_{max} states generated by an MDP will be considered.
- (2) By definition, each world represents a unique set of atoms. Therefore, a corresponding state transition system must have the restriction that each state is uniquely labeled; *i.e.*, each state in the MDP represents exactly one world.
- (3) Each transition in the MDP takes one unit of time.

Our notation for an MDP most resembles the *reactive probabilistic labeled transition system* (RPLTS) [R. Cleaveland and Narasimha 2005; Glabbeek et al. 1995; Larsen and Skou 1991]. Below, we will formally define an MDP with respect to a set of actions Act , and a set of atomic propositions, $B_{\mathcal{L}}$. When comparing MDPs to APT-Programs, we will assume that the APT-Program uses the same set of ground

atoms, and that each state in an MDP has a unique atomic label. In this manner, we can equate MDP states with worlds in tp-interpretations. Hence, an MDP is defined as follows:

Definition 6.1 MDP. A Markov Decision Process (MDP) consists of a 4-tuple $L = (S, \delta, P, lbl, s_1)$ where:

- S is a finite set of states
- $\delta \subseteq S \times Act \times S$ is the transition relation
- $P : \delta \rightarrow [0, 1]$ is the transition probability distribution, which satisfies:
 - $\forall s \in S, \forall a \in Act \sum_{s':(s,a,s') \in \delta} P(s, a, s') \in [0, 1]$
 - $\forall s \in S, \forall a \in Act (\exists s'(s, a, s') \in \delta) \Rightarrow \sum_{s':(s,a,s') \in \delta} P(s, a, s') = 1$
- $lbl : S \rightarrow 2^{Bc}$ is the labeling of each state that specifies the set of propositions that are true in a state. Each state has a unique set of propositions.
- $s_1 \in S$ is the initial state. ■

When an MDP is employed with policy π , it means that in state s_i , action $\pi(s_i)$ is taken. An MDP that uses only a single policy is often referred to as a *Stochastic Process*, or *Markov Process*. With the definition of an MDP and notion of a policy, we can now state what it means for a tp-interpretation to satisfy an MDP.

Definition 6.2. Let L be an MDP, π be a policy, I be a tp-interpretation, and t_{max} be the maximum value of time. We say that I **satisfies** the pair (L, π) iff: for all sequences of $n = t_{max}$ states, $seq \equiv s_1 \rightarrow \dots \rightarrow s_i \rightarrow \dots \rightarrow s_n$, there exists a thread Th such that:

- For every s_i in seq , $a \in lbl(s_i)$ iff $a \in Th(i)$
- $\prod_{i=1}^{n-1} P(s_i, \pi(s_i), s_{i+1}) = I(Th)$ ■

Further, we say that an interpretation I satisfies an MDP L and set of policies POL iff there exists a policy $\pi \in POL$ such that $I \models (L, \pi)$.

We can extend the notion of *entailment* described earlier to MDP's and describe entailment relationships between MDP's and APT-Programs. Based on this idea, we now can define a notion of *equivalence* between an MDP and an APT-Program as follows.

Definition 6.3 Equivalence/Entailment. An MDP L and set of policies POL is **equivalent** to APT-Program \mathcal{K} when tp-interpretation $I \models (L, POL)$ iff $I \models \mathcal{K}$. (L, POL) is said to *entail* \mathcal{K} if for all tp-interpretations I , if $I \models (L, POL)$ then $I \models \mathcal{K}$. Finally, \mathcal{K} is said to *entail* (L, POL) if for all tp-interpretations I , if $I \models \mathcal{K}$ then $I \models (L, POL)$. ■

With this notion, given an MDP and policy, we can now create an APT-Logic Program such that the set of satisfying interpretations for the MDP and policy is the same as the set of satisfying interpretations for the APT-Logic Program. We use these notions of entailment and equivalence to specify the semantic relationship between APT-Logic: if for any APT-Program there is an equivalent MDP and a set of policies, then we will consider APT-Logic to be no more expressive than MDPs. Soon we will see this is not the case, and that APT-Logic is in fact more expressive than MDPs.

First however, we provide the following formula notation. F is a mapping of states to formulas such that $F(s) \equiv (\bigwedge_{a \in lbl(s)} a) \wedge (\bigwedge_{b \notin lbl(s)} \neg b)$. Second, we provide the following probability measurement of a t -length sequence starting with state s_1 and ending with state s_t . We use the notation $s \rightarrow^t s'$ to denote the set of sequences of t transitions from s to s' .

Definition 6.4 Sequence Probability Measure. Let L be an MDP, π be a policy, s_1, s_t be states, and t be a positive integer. The *sequence probability measure*, SPM is defined as follows:

$$SPM_{L,\pi}(s_t, t) = \sum_{s_1 \rightarrow^{t-1} s_t} \left(\prod_{i=1}^{t-1} P(s_i, \pi(s_i), s_{i+1}) \right)$$

■

So, the SPM totals the probabilities of all sequences from the initial state to s_t in $t - 1$ transitions.

Next, we will present Algorithm 8 that, given an MDP and set of policies (L, POL) , creates an APT-Program \mathcal{K} such that (L, POL) entails \mathcal{K} . This construction is guaranteed to be correct by the following theorem.

Algorithm 8 Generate APT-Program that is entailed by a given MDP and set of policies.

MAKE-APT(*MDP* L , *PolicySet* POL)

- (1) Create annotated formula $F(s_1) : [1, 1, 1]$.
 - (2) For each state s , and each time point t , there are $|POL|$ SPM's, one for each policy. Let $min(SPM_{L,\pi}(s, t))$ be the minimum such SPM.
 - (3) For each state s , and each time point t , let $max(SPM_{L,\pi}(s, t))$ be the maximum SPM.
 - (4) For each time point $t \in [1, t_{max}]$, and each state s_i , create the following annotated formula: $F(s_i) : [t, min(SPM_{L,\pi}(s_i, t)), max(SPM_{L,\pi}(s_i, t))]$.
-

THEOREM 6.5. *If an interpretation I satisfies MDP L with set of policies L , then it satisfies APT-Program \mathcal{K} generated from MAKE-APT.*

Clearly, if we restrict the MDP to a single policy, then we can create an APT-Program using MAKE-APT that is equivalent to the MDP and single policy.

COROLLARY 6.6. *An interpretation I satisfies MDP L with policy π , iff it satisfies APT-Program \mathcal{K} generated from MAKE-APT.*

It is interesting to note, however, that although we can create an APT-Logic Program that is entailed by a given MDP and set of policies, we cannot always create an APT-Logic Program that *entails* an MDP and a set of policies. The intuition is that, in certain circumstances we are guaranteed that an APT-Logic Program has an infinite number of satisfying interpretations. If an MDP and set of policies are created such that these circumstances hold, then creating an APT-Program that

entails the given MDP and set of policies is impossible. Hence, we first make the claim of the special circumstance that guarantees an infinite number of satisfying interpretations. The claim is that for APT-Program \mathcal{K} , if there exists satisfying tp-interpretations for \mathcal{K} , I_1 , I_2 , such that for threads Th_1 , Th_2 , $I_1(Th_1) = 1$ and $I_2(Th_2) = 1$, then there is an infinite number of satisfying interpretations for \mathcal{K} . We describe why this is true in the following paragraph.

Let $c \in (0, 1)$ and $b \in (c, 1)$. Let I_3 represent an infinite number of interpretations such that $I_3(Th_1) = b$ and $I_3(Th_2) = (1 - b)$. \mathcal{K} is then satisfied by an infinite number of interpretations if all possible I_3 interpretations satisfy \mathcal{K} . Suppose by way of contradiction that some I_3 does not satisfy \mathcal{K} . We have two cases:

Case 1: There exists an unconstrained rule, r such that $I_3 \not\models r$.

Let $r = F \xrightarrow{\text{fr}} G : [\Delta t, \ell, u]$. Let $a_1 = \text{fr}(Th_1, F, G, \Delta t)$ and $a_2 = \text{fr}(Th_2, F, G, \Delta t)$. Let $a_1 \leq a_2$. By the definition of satisfaction, we know that $[a_1, a_2] \subseteq [\ell, u]$. By the definition of satisfaction, we know that $\sum_{Th \in \mathcal{T}} I_3(Th) \text{fr}(Th, F, G, \Delta t) < \ell$ or $\sum_{Th \in \mathcal{T}} I_3(Th) \text{fr}(Th, F, G, \Delta t) > u$ as $I_3 \not\models r$. Therefore, $b \cdot a_1 + (1 - b) \cdot a_2 < \ell$ or $b \cdot a_1 + (1 - b) \cdot a_2 > u$. However, clearly, $b \cdot a_1 + (1 - b) \cdot a_2 \subseteq (a_1, a_2)$ which implies $b \cdot a_1 + (1 - b) \cdot a_2 \subseteq [\ell, u]$. Hence, we have a contradiction.

Case 2: There exists a constrained rule, r such that $I_3 \not\models r$.

Let $r_i = F \xrightarrow{\text{fr}} G : [\Delta t, \ell, u, \alpha, \beta]$. We have three cases:

Case 2.1: $Th_1, Th_2 \in \text{ATS}_i$

Then, $\ell \leq 1 \leq u$ and the probabilities of both threads summed together must fall in this probability bounds. As $I_3(Th_1) + I_3(Th_2) = 1$, I_3 then must satisfy r_i , so we have a contradiction.

Case 2.2: Either $Th_1 \in \text{ATS}_i$ or $Th_2 \in \text{ATS}_i$

If $\ell \neq 1$, then there exists $c \in (0, 1)$ such that there is an infinite number of interpretations as per the definition of I_3 such that $I_3 \models r_i$. If $\ell = 1$, then either I_1 or I_2 does not satisfy r_i . Hence, we have a contradiction.

Case 2.3: $Th_1, Th_2 \notin \text{ATS}_i$

In this case, any interpretation that assigns probabilities only to Th_1 and Th_2 satisfies r_i . Therefore, I_3 must satisfy r_i .

Now we consider a very simple MDP with only two policies. We see that this MDP causes the above mentioned circumstances to occur. Hence, we cannot construct an APT-Program that entails the MDP and set of policies.

Let L be an MDP, the set of atoms, $B_{\mathcal{L}}$, be $\{a\}$, $S = \{s_1, s_2\}$ be such that $\text{lbl}(s_1) \equiv \{a\}$ and $\text{lbl}(s_2) \equiv \emptyset$, $\text{Act} = \{x, y\}$, $P(s_1, x, s_1) = 1$ and $P(s_1, x, s_2) = 0$, $P(s_1, y, s_1) = 0$, and $P(s_1, y, s_2) = 1$. We define the set of policies, $\text{POL} = \{\pi_1, \pi_2\}$ such that $\pi_1(s_1) = x$ and $\pi_2(s_1) = y$. Let $t_{\max} = 2$. We claim that it is impossible to construct an APT-Program that entails (L, POL) .

So, we can see why there does not exist an APT-Program that entails the MDP described above. Assume by way of contradiction that we can create an APT-Logic Program \mathcal{K} such that all interpretations that satisfy (L, π_1) or (L, π_2) satisfy \mathcal{K} . As each MDP-policy tuple is satisfied by exactly one interpretation, we have the following threads and interpretations based on the set of worlds $W = \{w_1, w_2\}$ where $w_1 \equiv \text{lbl}(s_1)$ and $w_2 \equiv \text{lbl}(s_2)$.

- Thread $Th_1 \equiv \langle w_1, w_2 \rangle$. Let I_1 be an interpretation such that $I_1(Th_1) = 1$ and sets the probability of all other threads to zero.
- Thread $Th_2 \equiv \langle w_1, w_1 \rangle$. Let I_2 be an interpretation such that $I_2(Th_2) = 1$ and sets the probability of all other threads to zero.

Hence, APT-Logic Program \mathcal{K} must be satisfied by exactly I_1 and I_2 . However, by the claim above, any program satisfied by these two interpretations is also satisfied by an infinite number of interpretations, so we have a contradiction.

So, based on the earlier definition of *equivalence*, while we can construct an equivalent APT-Program for an MDP and a single policy, we cannot do so for an MDP and set of policies. However, is the opposite true? It is: it would be trivial to construct an MDP that entails an APT-Program, since the null MDP can accomplish this. This highlights a difference between MDPs and APT-Logic Programs: we cannot have rules that say *this relationship holds with probability p_1 or probability p_2* . However, we can express ranges of probabilities.

While we cannot create an APT-Program that entails a given MDP and set of policies, APT-Programs can be satisfied by tp-interpretations that cannot satisfy *any* MDP. In other words, there are APT-programs and tp-interpretations that satisfy those APT-programs where there is no MDP that is satisfied by that tp-interpretation. Consider the set of ground atoms $B_{\mathcal{L}} = \{a\}$ and $t_{max} = 4$ and the below APT-Logic Program, \mathcal{K} :

- $a : [1, 1, 1]$
- $a \xrightarrow{pfr} \neg a : [1, 0.5, 0.5]$ (or $a \xrightarrow{pfr} \neg a : [1, 0.5, 0.5, 1, 1]$)

We included an alternate second rule to illustrate that this type of expressiveness result is true about both constrained and unconstrained programs. Consider worlds $w_1 \equiv \{a\}$ and $w_2 \equiv \emptyset$. Let I be an interpretation that assigns probabilities to the threads below:

- $Th_1 \equiv \langle w_1, w_2, w_1, w_2, \rangle$, $I(Th_1) = 0.5$
- $Th_2 \equiv \langle w_1, w_1, w_1, w_1, \rangle$, $I(Th_2) = 0.5$

It is trivial to show that $I \models \mathcal{K}$. We claim that it is impossible to build an MDP L with set of policies POL such that tp-interpretation $I \models (L, POL)$.

Let $S = \{s_1, s_2\}$ such that $lbl(s_1) \equiv w_1$ and $lbl(s_2) \equiv w_2$. Suppose by way of contradiction that $I \models (L, POL)$. Therefore, there exists a policy, $\pi \in POL$ such that I satisfies (L, π) . Hence, the following must be true:

- $P(s_1, \pi(s_1), s_2) \cdot P(s_2, \pi(s_2), s_1) \cdot (s_1, \pi(s_1), s_2) = I_1(th_1) = 0.5$
- $P(s_1, \pi(s_1), s_1) \cdot P(s_1, \pi(s_1), s_1) \cdot (s_1, \pi(s_1), s_1) = I_1(th_2) = 0.5$

Refer to the left side of Figure 9 for a graphical representation of what follows. Let $P(s_1, \pi(s_1), s_2) = p$. Then, by the definition of an MDP, $P(s_1, \pi(s_1), s_1) = 1 - p$. By the above equalities, $1 - p > 0$. Let $P(s_2, \pi(s_2), s_1) = r$. Therefore, $p^2 \cdot r = 0.5$. Now consider the sequence $seq \equiv s_1 \rightarrow s_2 \rightarrow s_1 \rightarrow s_1$. The probability of this sequence must be set to zero, by the definition of I . Then, $P(seq) = p \cdot r \cdot (1 - p) = 0$. However, we know that $p \cdot r$ cannot be zero and we know that $1 - p > 0$. Hence, we have a contradiction.

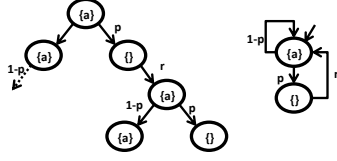


Fig. 9. Left: Unrolled MDP in an attempt to create an MDP that satisfies interpretation I in the text. Notice how the sequence $\{\{a\}, \{\}, \{a\}, \{a\}\}$ must be assigned a non-zero probability. Right: A standard representation of the MDP on the left. Notice that the MDP must allow for non-zero probability of threads that are given a zero probability in interpretation I .

The above discussion illustrates the differences between MDPs and APT-Logic. One could argue that the use of policies is overly restrictive for an MDP, *i.e.*, that perhaps the action should be decided based on time, or a combination of time and the current state. However, we can easily modify the above claim based on time or actions based on time and current state and obtain the same result. We suspect that it is not possible to have an MDP that replicates an APT-Logic Program without breaking the Markov Property, or causing a massive increase in the number of states, which also would change the assumption about the relationship between worlds and states.

6.2 Comparison with Probabilistic Computation Tree Logic (PCTL)

In this section, we show that APT-Logic rules differ significantly in meaning from similar structures presented in PCTL [Aziz et al. 1995; Hansson and Jonsson 1994b], a well-known probabilistic temporal logic.

A derived operator in LTL with an intuition similar to that of our APT-Rules was introduced by Susan Owicki and Leslie Lamport in [Owicki and Lamport 1982]. The operator, known as *leads-to* and an equivalent LTL formula are shown below (p and q are state formulas).

$$(p \leadsto q) \equiv G(p \Rightarrow F(q))$$

This formula intuitively says that if p is true in a state, then q must be true in the same (or future) state. As Owicki and Lamport’s operator is based on LTL, it does not describe the correlation between p and q with probabilities or with reference to a specific time interval; q merely must happen sometime after (or with) p . A probabilistic version of CTL, known as PCTL [Aziz et al. 1995; Hansson and Jonsson 1994b] introduces another operator based on a similar intuition; the authors refer to this operator as “leads-to” as well. This derived operator, and the equivalent PCTL formula, are shown below (f_1 and f_2 are state formulas).

$$f_1 \leadsto_{\geq p}^{\leq t} f_2 \equiv \left[G \left[(f_1 \Rightarrow F_{\geq p}^{\leq t} f_2) \right] \right]_{>1}$$

Intuitively, this operator reads as “ f_2 follows f_1 within t periods of time with a probability of p or greater”. As PCTL formulas are satisfied by a Markov Process (an MDP with a single policy), satisfaction is determined by the transition probabilities. So, to determine if a Markov Process satisfies the above leads-to formula, we must compute the minimum probability of all sequences that start in a state

satisfying f_1 and satisfy f_2 in t units of time or less. Note that this is determined by the transition probabilities of the Markov Process; hence, whether a Markov Process satisfies the lead-to operator depends on the interval between f_1 and f_2 , but *not* on the total length of the sequence of states. So, if we limit the number of states being considered, using an operator such as $G_{\geq 1}^{\leq t_{max}}$ which PCTL provides to limit consideration to only the first t_{max} states, the Markov Process will satisfy the formula *regardless* of the value of t_{max} . Note that $G_{\geq 1}^{\leq t_{max}}$ placed at the head of the PCTL has no effect on the satisfaction of the formula as there is already a G path-quantifier included at the beginning of the leads-to operator.

As previously described, the frequency function is often highly sensitive to t_{max} . Our two primary examples of frequency functions, pfr and efr , are based on ratios of numbers of worlds in a given thread. For example, if we create a thread Th on a single atom a , we can see that for thread $\langle \{a\}, \{a\}, \{\} \rangle$, the value of $pfr(Th, a, \neg a, 1)$ is much greater than if Th were $\langle \{a\}, \{a\}, \{\}, \{a\}, \{a\}, \{a\}, \{a\} \rangle$. The fact that the length of the thread has an effect on the frequency function further illustrates how APT-Logic allows for reasoning beyond the restrictions of the Markov Property. The limited thread length forces us to consider worlds before and after a time-point we wish to reason about. If our probabilities were fixed, based on transition probabilities, they would not, and we would conform to the Markov Property.

Even though there are syntactic similarities, in the Appendix we provide a short example illustrating semantic differences between APT-rules and PCTL.

7. CONCLUSION

Statements of the form “Formula G is/was/will be true with a probability in the range $[\ell, u]$ in/within Δt units of time after formula F became true” are common. In this paper, we have provided examples from four domains (stock markets, counter-terrorism, reasoning about trains, and power grids), but many more examples exist. They could be used, for instance, to describe when the health or environmental effects of industrial pollution may arise after a polluting event occurred, to the time taken for a medication to produce (with some probability) some effects. In the same way, they can be used in domains as widely divergent as industrial control systems to effects of educational investment on improved grades or graduation rates.

In this paper, we have provided the concept of Annotated Probabilistic Temporal (APT) logic programs within which such statements can be expressed. APT-logic programs consist of two kinds of rules: *unconstrained* and *constrained* rules with an expected value style semantics and a more ordinary semantics. Both types of rules are parameterized by the novel concept of a *frequency function*. Frequency functions capture the probability that G follows F in exactly (or within) T time units *within a thread* (temporal interpretation). We show that this notion of “follows” can intuitively mean many different things, each leading to a different meaning. We propose an *axiomatic definition* of frequency functions which is rich enough to capture these differing intuitions and then provide a formal semantics for APT-logic programs.

We then study the problems of consistency and entailment for APT-logic programs. We show that the consistency problem is computationally intractable and is naturally solved via linear programming. We develop three successively more

sophisticated linear programs for consistency checking and show that they lead to smaller linear programs (though not always). We also develop a suite of complexity results characterizing the entailment problem and provide algorithms to solve the entailment problem.

A natural question that arises in any probabilistic logic framework is “Where do the probabilities come from?” In order to answer this question, we develop the (straightforward) APT-Extract algorithm that shows how APT-logic programs can be derived from certain types of databases. We have applied APT-Extract to extract APT-rules about 18 terror groups.

Last, but not least, we have developed a detailed comparison between our APT-framework and two well known frameworks: Markov decision processes [Puterman 1994] and probabilistic computation tree logic [Hansson and Jonsson 1994a]. We show the former can be captured within APT-logic program framework (but not vice versa). The latter has a more complex relationship with APT-logic programs, but cannot express intra-thread properties of the type expressed via APT-logic programs.

There is much work that remains to be done in order to reduce APT-logic programs to practice. We are implementing APT-logic programs within our SOMA Terror Organization Portal [Martinez et al. 2008b] which has registered users from 12 US government organizations. Initially, users will be able to browse the APT-logic programs associated with a given terror group’s behavior. Consistency checks are expensive, but need to be performed only once. In contrast, entailment checks are needed when answering queries against such programs. Scaling the entailment checks is a major priority that needs to be achieved. We plan to leverage some of the scaling methods developed for SOMA programs (which are similar to APT-logic programs, but with no temporal component) [Khuller et al. 2007].

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/2009-V-N/p1-URLend>.

Acknowledgements. Some of the authors of this paper were funded in part by AFOSR grant FA95500610405, ARO grant W911NF0910206 and ONR grant N000140910685.

REFERENCES

- 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.
- BARAL, C., TRAN, N., AND TUAN, L. 2002. Reasoning about actions in a probabilistic setting. In *Proc. AAAI 2002*. 507–512.
- D. DUBOIS, J. L. AND PRADE, H. 1991. Timed possibilistic logic. *Fundamenta Informaticae XV*, 211–234.
- DE CHOUDHURY, M., SUNDARAM, H., JOHN, A., AND SELIGMANN, D. D. 2008. Can blog communication dynamics be correlated with stock market activity? In *HT '08: Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*. ACM, New York, NY, USA, 55–60.
- 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. AND HALPERN, J. Y. 1994. Reasoning about knowledge and probability. *Journal of the ACM* 41, 340–367.
- FAGIN, R., HALPERN, J. Y., AND MEGIDDO, N. 1990. A logic for reasoning about probabilities. *Information and Computation* 87, 78–128.
- FUJIWARA, I., HIROSE, Y., AND SHINTANI, M. 2008. *Can News be a Major Source of Fluctuation: A Bayesian DGSE Approach*. Vol. Discussion Paper Nr. 2008-E-16. Institute for Monetary and Economic Studies, Bank of Japan.
- GLABBEEK, R. J. V., SMOLKA, S. A., AND STEFFEN, B. 1995. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation* 121, 130–141.
- GOLDSTONE, J. A., BATES, R., GURR, T. R., LUSTIK, M., MARSHALL, M. G., ULFELDER, J., AND WOODWARD, M. 2005. A global forecasting model of political instability. In *Proc. Annual Meeting of the American Political Science Association*.
- HADDAWY, P. 1991. Representing plans under uncertainty: A logic of time, chance and action. *PhD Thesis, Univ. of Illinois*.
- HALPERN, J. AND TUTTLE, M. 1992. Knowledge, probability, and adversaries. In *IBM Thomas J. Watson Research Center Tech Report*.
- 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.
- HART, S. AND SHARIR, M. 1986. Probabilistic propositional temporal logic. *Information and Control* 70, 97–155.
- KANAZAWA, K. 1991. A logic and time nets for probabilistic inference. In *Proc. AAAI 1991*.
- KARMARKAR, N. 1984. A new polynomial-time algorithm for linear programming. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM, New York, NY, USA, 302–311.
- KHULLER, S., MARTINEZ, M. V., NAU, D., SIMARI, G. I., SLIVA, A., AND SUBRAHMANIAN, V. S. 2007. Action probabilistic logic programs. *Annals of Mathematics and Artificial Intelligence* 51, 2–4, 295–331.
- KIESSLING, W., THONE, H., AND GUNTZER, U. 1992. Database support for problematic knowledge. In *Proceedings of EDBT 1992, Springer LNCS Volume 580*. 421–436.
- KIFER, M. AND SUBRAHMANIAN, V. 1992. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming* 12.
- LAKSHMANAN, L. V. AND SHIRI, N. 1997. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*.
- LAKSHMANAN, V. AND SADRI, F. 1994a. Modeling uncertainty in deductive databases. In *Proceedings of DEXA 1994*. Springer LNCS Vol. 856, 724–733.
- LAKSHMANAN, V. AND SADRI, F. 1994b. Probabilistic deductive databases. In *Proceedings of the Intl. Logic Programming Symposium (ILPS)*. MIT Press.
- LAMPORT, L. 1980. “sometime” is sometimes “not never”: on the temporal logic of programs. In *POPL 1980*. ACM, New York, NY, USA, 174–185.
- LARSEN, K. G. AND SKOU, A. 1991. Bisimulation through probabilistic testing. *Inf. Comput.* 94, 1, 1–28.
- LEHMANN, D. AND SHELAH, S. 1982. Reasoning about time and chance. *Information and Control* 53, 165–198.
- LLOYD, J. W. 1987. *Foundations of Logic Programming, Second Edition*. Springer-Verlag.
- LUKASIEWICZ, T. 1999. Many-valued disjunctive logic programs with probabilistic semantics. In *In Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning, volume 1730 of LNAI*. Springer, 277–289.

- LUKASIEWICZ, T., LUKASIEWICZ, T., KERN-ISBERNER, G., AND KERN-ISBERNER, G. 1999. Probabilistic logic programming under maximum entropy. In *In Proc. ECSQARU-99, LNCS 1638*. Springer, 279–292.
- MANNES, A., MICHAELL, M., PATE, A., SLIVA, A., SUBRAHMANIAN, V., AND WILKENFELD, J. April 1-2, 2008. Stochastic opponent modelling agents: A case study with hezbollah. In *Proc. 2008 First Intl. Workshop on Social Computing, Behavioral Modeling and Prediction*. Springer Verlag.
- MANNES, A., SLIVA, A., SUBRAHMANIAN, V., AND WILKENFELD, J. Sep. 2008. Stochastic opponent modeling agents: A case study with hamas. In *Proc. 2008 Intl. Conf. on Computational Cultural Dynamics*. AAAI Press, 49–54.
- MARTINEZ, V., SIMARI, G., SLIVA, A., AND SUBRAHMANIAN, V. 2008a. Convex: Similarity-based algorithms for forecasting group behavior. *IEEE Intelligent Systems* 23, 4, 51–57.
- MARTINEZ, V., SIMARI, G., SLIVA, A., AND SUBRAHMANIAN, V. 2009. Cape: Automatically predicting changes in terror group behavior. *to appear in Mathematical Methods in Counterterrorism (ed. N. Memon)*.
- MARTINEZ, V., SIMARI, G., SLIVA, A., AND SUBRAHMANIAN, V. S. 2008b. The soma terror organization portal (stop): Social network and analytic tools for the real-time analysis of terror groups. In *Proceedings of the First International Workshop on Social Computing, Behavioral Modeling and Prediction*, H. Liu and J. Salerno, Eds.
- NG, R. T. AND SUBRAHMANIAN, V. S. 1991. A semantical framework for supporting subjective and conditional probabilities in deductive databases. In *Proceedings of ICLP '91*, K. Furukawa, Ed. The MIT Press, 565–580.
- NG, R. T. AND SUBRAHMANIAN, V. S. 1992. Probabilistic logic programming. *Information and Computation* 101, 2, 150–201.
- OWICKI, S. AND LAMPORT, L. 1982. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst.* 4, 3, 455–495.
- PUTERMAN, M. L. 1994. *Markov Decision Processes*. Wiley.
- R. CLEAVELAND, S. I. AND NARASIMHA, M. 2005. Probabilistic temporal logics via the modal mu-calculus. *Theoretical Computer Science* 342, 2-3, 316–350.
- RUSSELL, S. AND NORVIG, P. 2003. *Artificial Intelligence: A Modern Approach*, 2nd edition ed. Prentice-Hall, Englewood Cliffs, NJ.
- SCHRODT, P. AND GERNER, D. 1998. Cluster analysis as an early warning technique for the middle east. *Preventive Measures: Building Risk Assessment and Crisis Early Warning Systems (eds. John L. Davies and Ted Robert Gurr)*.
- THOMAS, S. R. 1995. The placa agent programming language. In *ECAI-94: Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents*. Springer-Verlag New York, Inc., New York, NY, USA, 355–370.
- 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 August 2009; revised March 2010; accepted April 2010