IFI TECHNICAL REPORTS

Institute of Computer Science,
Clausthal University of Technology

IfI-05-03

Clausthal-Zellerfeld 2005

Heterogeneous Temporal Probabilistic Agents*

Jürgen Dix Sarit Kraus V.S. Subrahmanian

Clausthal University of Technology, Institut für Informatik
Julius-Albert-Str. 4, 58678 Clausthal, Germany, dix@tu-clausthal.de
Bar-Ilan Univ. and University of Maryland
Dept. of CS, Ramat-Gan, 52900 Israel, sarit@macs.biu.ac.il
Department of Computer Science, University of Maryland
College Park, MD 20742, vs@cs.umd.edu

Abstract

To date, there has been no work on temporal probabilistic agent reasoning on top of heterogeneous legacy databases and software modules. We will define the concept of a *heterogeneous temporal probabilistic* (HTP) agent. Such agents can be built on top of existing databases, data structures, and software code bases without explicitly accessing the internal code of those systems and can take actions compatible with a policy or operating principles specified by an agent developer. We will develop a formal semantics for such agents through the notion of a feasible temporal probabilistic status interpretation (FTPSI for short). Intuitively, an FTPSI specifies what all an HTP agent is permitted/forbidden/obliged to do at various times t. As changes occur in the environment, the HTP agent must compute a new FTPSI. HTP agents continuously compute FTPSI's in order to determine what they should do and hence, the problem of computing FTPSI's is very important. We give a sound and complete algorithm to compute FTPSI's for a very large class of HTP agents called *strict* HTP agents. In a given state, many FTPSI's may exist. These represent alternative courses of action that the HTP agent can take. We provide a notion of an *optimal* FTPSI that selects an FTPSI optimising an objective function and give a sound and complete algorithm to compute an optimal FTPSI.

1 Introduction

Past works on reasoning about time and uncertainty overwhelmingly assume (i) that the state of the world is represented in some uniform (usually in logic) way selected by the designer of the reasoning mechanism, and (ii) that all data in the state can be

*This work was supported in part by the Army Research Lab under contract DAAL0197K0135, the CTA on Advanced Decision Architectures, by ARO contract DAAD190010484, and by DARPA/RL contract number F306029910552 and by NSF grant IIS0222914.

directly manipulated by the reasoning paradigm. Unfortunately, in today's world, these assumptions are true only in a small class of applications.

First, note that most real world systems must manipulate existing databases and must use existing, tried and tested code. For example, a simple stock market application that requires reasoning about time and uncertainty (when will a given stock reach a given price?) may access a wide variety of legacy sources. These may include databases (e.g. of past stock performance, past Dow Jones data, performance of similar stocks in the past, etc.), stock market models encoded as legacy code that analyzes stock performance (numerous such models exist), and corporate risk assessment programs (of which numerous models exist). In order for an agent to recommend stock portfolios to individual or corporate investors, it needs to leverage all these existing, proven databases and software modules. We are not aware of any existing method for temporal probabilistic reasoning that can explicitly do this.

Second, most real world data systems only provide *limited access* to their data. In other words, many corporate databases only allow third party programs to access their database through programs that they provide. There are at least two good reasons for this. The database may be owned by a third party who only wants their data accessed by code that they have written (and hence trust). In addition, they may not want *all* of their data to be accessible to third parties (e.g. their databases may contain profiles of the clients which they may want to keep hidden). If the state of the world is contained in such data sources, this means that a reasoning program only has access to that state via third party function calls. We are not aware of any existing method to reason about time and uncertainty that can support this.

Third, consider a simple corporate employee database. Such a database almost always has at least ten columns (including fields like `firstname, lastname` (with domain string), social security number (with domain all 9 digit integers), as well as many other fields like `streetnum, streetname, zip, projectid, salary, hire-date`, and many more. Even if we consider just these three fields and assume the length of strings to be at most 10 characters long. we have a total state space of $26^{10} \times 26^{10} \times 10^9$. This is an enormous state space for what almost all database managers would consider a *very tiny* database ! Real databases (e.g. the US military's GCCS and MIDB databases, or Walmart's inventory database) are orders of magnitude larger. We are not aware of any existing method to reason about time and uncertainty that can reason about such large state spaces. Furthermore, almost every commercial database supports an *SQL API* (application program interface) which means authorized users can update the database via appropriate programs containing *SQL* updates statement - this in turn is an infinite set of possible statements. An attempt to treat this via a tree-like search space will encounter a difficult scalability problem because of the infinite branching of such a tree (in addition to the enormous number of states).

In this paper, we propose the concept of heterogeneous temporal probabilistic (HTP) software agents that address the above shortcomings of past paradigms:

1. HTP agents provide a syntax within which agents that perform temporal and

probabilistic reasoning on top of legacy databases and data sources can be programmed with application specific action policies. To date, we have seen no single *uniform* syntax to perform temporal-probabilistic reasoning on top of legacy pieces of code. There are certainly applications (e.g. [Chalupsky et al., 2001]) which use Markov decision processes [Puterman, 1994] to make decisions on top of *specific* information sources, but no principles to do this across *arbitrary databases and/or legacy software applications* have been articulated to our knowledge.

2. HTP agents are defined directly in terms of the application program interfaces (*API*'s) that a third party piece of code and/or database provides. Note that virtually every decent commercial software package has an *API*. This allows HTP agents to support the goal of encoding decision making policies in the presence of limited access to state information. No paradigm we are aware of to date can do this.

3. The *state* of an HTP agent consists of whatever data is stored inside the data structures of the code on top of which the agent is built. This state may or may not reflect information about the state of the real world, but it is the set of all computational objects that the program is aware of. HTP agents consist of a set of rules that are built on top of the underlying code's *API* calls in such a way that they can automatically take actions in response to changes in their state. Such actions can include for example, updates, alerts, modifying requests and processing them, and virtually anything that can be executed by a piece of code.

4. We provide a formal syntax and a formal semantics for this. The semantics is defined in terms of a structure called a *feasible temporal probabilistic (*FTP*) status interpretation (*FTPSI *for short)* . FTPSI's extend the concept of a feasible status set defined in [Eiter et al., 1999]—as feasible status sets themselves extend various well known semantics in logic programming, our semantics extends those semantics as well.

5. An HTP agent continuously executes a decision cycle. The HTP agent ensures that it's state always satisfies various integrity constraints. Changes (i.e. updates) to their state can occur in many ways. For instance, the receipt of a message changes the state of their messaging data structures. Likewise, the tick of a clock can change the state of the current time variable. When such changes occur, the HTP agent must find a set of actions to perform that satisfies various requirements (described later on in the paper). These requirements could include conditions that must hold (now), conditions that may hold with some probability (now or in the future). The execution of the actions must not cause any integrity constraints to be violated by the resulting state. And so on. The agent then executes these actions (or schedules them for future execution). When state changes occur in a future state, the same computations are performed based on the new state.

Thus the agents are engaged in a continuous cycle of *detect state change* $\longrightarrow$ *determine what actions to perform now or in the future* (via computation of an FTPSI) $\longrightarrow$ *execute such actions or schedule them for future execution* $\longrightarrow$ *detect state change* ... . Supporting this decision cycle is a key part of our system. When an update occurs, the agent automatically computes a "feasible temporal probabilistic status interpretation (FTPSI)" that tells it what to do now in response to the given change as well as what actions to take in the future (of course a later update can cause those decisions to be revisited).

6. As there has been no formal syntax and semantics for performing temporal-probabilistic reasoning on top of arbitrary legacy pieces of code, there have been no provably sound and complete algorithms for this purpose. We provide a sound and complete algorithm for computing FTPSI's. Past works that provide completeness results for temporal probabilistic reasoning that we are aware of assume that the data on top of which the reasoning is performed is represented in some uniform fashion chosen by the designer of the reasoning system (usually logic). It is very rare in the corporate/military world to have the opportunity to decide how all the data should be stored. Usually legacy data sources need to be used. Furthermore, legacy applications that provide various kinds of reasoning capabilities need to be leveraged not re-implemented from scratch. Past work generally assumes that all the reasoning involved is performed by them whereas our paradigm allows legacy reasoners (e.g. route planners, sensing programs, etc.) to participate in the reasoning process.

The goal of this paper is therefore quite simple: provide a programming language within which agents that must make decisions involving time and uncertainty can be programmed by programmers do so in the presence of (and leveraging) legacy databases, data structures and software programs. We provide a syntax for such HTP agents and a semantics along with appropriate soundness and completeness results.

Throughout this paper (excluding the related work section), we use the word "reasoning" to describe determining how to take actions now or in the future based on the data the agent currently has access to via zero or more legacy data sources and based on leveraging existing software programs. HTP *agents can be built on top of any such legacy data sources of existing software applications independently of what those software applications compute. All kinds of classical AI frameworks traditionally called reasoning such as image processing, predictions, Markov decision processes, abduction, AI planning, operations research, decision theoretic planning, could be represented by these legacy programs. The agent developer can encode whatever rules he wants his agents to follow by leveraging whichever of these resources he has access to.*

Our group has built two applications (one jointly with *BBN*, *Lockheed Martin*, *US Navy* and several other partners [Mittu and Ross, 2003], the other jointly with *BBN*, *BAE Systems*, *Fantastic Data* and several other partners

[T. Hammel and Rogers, 2003]) to track battlefield information and take actions based on what is seen. For example, in the first application [Mittu and Ross, 2003], spatio-temporal-probabilistic predictions are made about where an enemy submarine will be at future points in time. This is done on the basis of sensor data as well as using third party (legacy) code describing the reliability of the sensors as well as commercial legacy code for performing predictions, as well as terrain databases describing the coastline involved. Where and when to intercept the enemy submarine is a problem involving reasoning about time and probabilities in the presence of the legacy databases and legacy software code bases described above. This problem was solved using the principles described in this paper. In the second application again, we used live ground sensor data to identify enemy vehicles in a road network, predict where they might be going based on properties of the road network, and determining how and where to intercept the enemy based on existing resources available for interception.

Our work builds on top of the *IMPACT* agent platform [Subrahmanian et al., 2000]. In *IMPACT*, agents manage a set of data types/structures (including a message box) through a set of application program interface (*API*) function calls. The state of the agent at a given point in time is a set of objects belonging to these data types. Each agent has a set of integrity constraints that its state must always satisfy. When an agent's state changes (due to external events such as receipt of a message), the agent tries to modify its state so that the integrity constraints are satisfied. To do this, it has a suite of actions, and an *agent program* that specifies the operating principles (what is permitted, what is forbidden, what is obligatory, etc., and under what conditions?). We emphasise that our framework is not static. HTP agents extend the above agents so as to reason about time and uncertainty, and make decisions based on such analyses. HTP agents continuously engage in the following computational cycle: identify changes to state $\rightarrow$ compute FTPSI $\rightarrow$ take actions prescribed by FTPSI $\rightarrow$ identify changes to state $\rightarrow$ ... . Thus, the computation of FTPSI's is a continuous ongoing process *intended to handle updates and changes in agent state in a manner that is consistent with the rules of behaviour the agent developer intended.*

2 Motivating Examples

In this section, we first present a detailed energy trading example to motivate our research. We then present a (less detailed) stock trading example. Both examples will be used throughout the paper to motivate the basic definitions introduced here.

2.1 Energy Market example

Consider an energy market (such as those to trade energy in markets such as the US and the UK). Energy producers in such a market tend to make major decisions based on predicted future demand. In general, uncertainty in energy (and other) markets can be expressed via statements of the form: *The probability that there will be demand for D*

units of item I at price P is p. Demand curves—used extensively in economics—are a graphical representation of a set of such statements.

Usually, short term demand in energy markets is invariant to price fluctuations. Thus, in the context of energy markets, demand related data involves statements of the form "The Boston energy market will need K_1 Megawatts per hour of energy between 4 and 5pm tomorrow with probability p_1, K_2 Megawatts per hour of energy between 4 and 5pm with probability p_2 ..." and so on. Of course, many similar such statements could be made for other time slots. Most regions are served by multiple energy producers. Energy distributors use auction mechanisms to determine which vendors to buy energy from in a given time slot. Things are somewhat more complex than laid out here because of additional constraints. For example, a distributor cannot place an order for a huge amount of energy from one company in the 10-11am time slot, and leave the adjacent time slots (9-10,11-12, etc.) with an order of zero units of energy (as such machines require a "ramp up" time and a "ramp down" time which is often expensive).

Executives of such energy producing companies need to continuously reason in the presence of time and uncertainty. For example, they need to determine how much to ask for a Megawatt of energy in a given time slot in tomorrow's auction market. By pricing energy too high (and getting too greedy), they may get orders for lower quantities. Likewise, by pricing energy too low, they may lose profits. Thus, they need to make decisions based not only on their perception of how much energy will be needed in the future (e.g. tomorrow), but also based on their perception of what their competitors will do. Furthermore, *updates* are important: as new data becomes available, old decisions may need to be discarded and replaced with newer ones. In this paper, we show how problems such as this may be modelled using our temporal probabilistic agent framework. We do not solve this problem, as that is not the purpose of this paper. Rather, the ability to reason *automatically and simultaneously* about time and probabilities in a setting which requires access to heterogeneous data sources and software sources in such a setting is important for agents. We will use a simplified version (presented in Appendix A of [Wolfram, 1998]) of such a market.

> Suppose there are two energy generators. Generator 1 has one unit of capacity with constant marginal cost c. Likewise, generator 2 has one unit of capacity with constant marginal cost c, but it also has m (where $m > 0$) additional units of capacity with zero marginal cost. Demand is assumed to be stochastic and varies between $m + 1$ and $m + 2$ with probability ϕ and $1 - \phi$, respectively ($0 < \phi < 1$). In other words, generator 2's m units with zero marginal cost are always used and depending on demand, either one or both of the generators' units) with marginal costs equal to c are used. We assume that the generators simultaneously submit process p_1 and p_2 for their two units with marginal costs c before the level of demand is realised. The two units are ranked according to the submitted prices and once demand is realised, the marginal unit sets the price for all units used in this period. The generator offer prices are constrained to be below some

price P^{max}.

2.2 Stock Market Example

Consider a very simple agent that tracks stocks and executes trades automatically for clients. A specific client may wish to trade stocks based on some simple rules. For example, if a prediction program predicts that a given stock is going to go up to $50 per share with a high probability (e.g. 80% or more) sometime during the next 10-20 days and the user already owns this stock, then she may want to buy the stock sometime in the next 9 days. However, if the user does not own this stock (and if it is consistent with the user's diversified investment strategy) then the user may want a 90% (or more) probability that the stock will go up to $50 in order to buy it. Note that again, updates can occur naturally—tomorrow, the same prediction program may say that the given stock will only go up to $ 30 per share instead of $ 50. The decision to buy the stock in the next 9 days may need to be revised by the agent.

Throughout this paper, we will use this simple example to motivate the need for taking actions automatically during the presence of temporal uncertainty. As will become clear, our notion of temporal probabilistic agents can be easily used to write far more sophisticated agents with more complex rules than those described above.

3 Temporal Probabilistic Code Call's

In examples such as the energy example above, agents need to be able to make decisions based on the data stored in various legacy data sources. Furthermore, in arriving at these decisions, they may use capabilities provided by various software packages. [Eiter et al., 1999][1] represent a piece of software code as a pair $\mathcal{S} =_{def} (\mathcal{T}_\mathcal{S}, \mathcal{F}_\mathcal{S})$ where $\mathcal{T}_\mathcal{S}$ is a finite set of types and $\mathcal{F}\_\mathcal{S}$ is a finite set of *API* functions whose input and output types are drawn from $\mathcal{T}_\mathcal{S}$. They also introduce the concept of a *code call* which is a simple syntax to access the data and functionality of such legacy data and software sources. In this section, we introduce the concept of a *temporal probabilistic code call (*TPCC*)* using which we may reason about sources that provide temporally uncertain information.

We first start by recapitulating the notion of a code call.

Definition 3.1 ((Code Call (*cc*))) *Suppose $\mathcal{S} =_{def} (\mathcal{T}_\mathcal{S}, \mathcal{F}_\mathcal{S})$ is some software code, $f \in \mathcal{F}_\mathcal{S}$ is a predefined function with n arguments, and $\mathtt{d}_1, \ldots, \mathtt{d}_n$ are objects or variables such that each $\mathtt{d}_i$ respects the type requirements of the i'th argument of f. Then, $\mathcal{S} : f(\mathtt{d}_1, \ldots, \mathtt{d}_n)$ is a* code call. *A code call is* ground *if all the $\mathtt{d}_i$'s are objects.*

[1] Henceforth, we use terminology introduced in a series of previous papers. To make the paper self contained, all necessary definitions are given in an appendix.

A code call atom *is an expression of the form* $\mathbf{in}(\mathtt{a}, \mathcal{S} : f(\mathtt{d_1}, \ldots, \mathtt{d_n}))$ *where* a *is either a variable or a constant and* $\mathcal{S} : f(\mathtt{d_1}, \ldots, \mathtt{d_n})$ *is a code call. Such a code call atom says that* a *is in the set of objects returned by* $\mathcal{S} : f(\mathtt{d_1}, \ldots, \mathtt{d_n})$.

A code call executes an *API* function and returns as output a set of objects of the appropriate output type.

For example, in the energy domain, $\mathtt{energy} : demand(\mathtt{A})$ is a code call that intuitively returns the energy demand in area A. $\mathbf{in}(\mathtt{d}, \mathtt{energy} : demand(\mathtt{A}))$ is an example code call atom.

Code calls are often used to formulate more complicated conditions and to express relationships between several variables.

Definition 3.2 ((Code Call Condition (ccc))) *A* code call condition χ *is defined as follows:*

1. *Every code call atom is a code call condition.*
2. *If* $\mathtt{s}, \mathtt{t}$ *are either variables or objects, then* $\mathtt{s} = \mathtt{t}$ *is a code call condition.*
3. *If* $\mathtt{s}, \mathtt{t}$ *are either integers/real valued objects, or are variables over the integers/reals, then* $\mathtt{s} < \mathtt{t}, \mathtt{s} > \mathtt{t}, \mathtt{s} \geq \mathtt{t}, \mathtt{s} \leq \mathtt{t}$ *are code call conditions.*
4. *If* χ_1, χ_2 *are code call conditions, then* $\chi_1 \,\&\, \chi_2$ *is a code call condition.*

A code call condition satisfying any of the first three criteria above is an atomic *code call condition.*

Instead of giving examples of code call conditions, we first generalise them to *temporal probabilistic code call* and provide examples of those.

Notice that in our energy example, a code call of the form $\mathtt{energy} : demand(\mathtt{A})$ may be uncertain about the answer. For example, it may say that the demand for energy in area A in the 10am to 11am time slot tomorrow is 5 MegaWatts (with probability 20-25%), 6 MegaWatts (with probability 35-50%), and 7 (with probability 20-40%). Similar statements may be made for other time slots as well. In this case, what is happening is that the code call returns a set of random variables, one associated with each time point.

Definition 3.3 ((Coherent set of random variables)) *A* random variable $\mathbf{RV}$ *of type* τ *is a pair* $\mathbf{RV} = (Obj, \wp)$ *where* Obj *is a finite set of elements of type* τ *and* $\wp$ *is a probability distribution over* Obj *that assigns real numbers in the unit interval* $[0, 1]$ *to members of* $\mathbf{RV}$ *such that* $\Sigma_{o \in Obj} \wp(o) \leq 1$.

A set S *of random variables of type* τ *is* coherent *iff whenever we consider two distinct* $(Obj_1, \wp_1), (Obj_2, \wp_2) \in S$, *it is the case that* $Obj_1 \cap Obj_2 = \emptyset$.

Intuitively, when a set $\{(Obj_1, \wp_1), \ldots (Obj_n, \wp_n)\}$ of random variables is returned by some code call, then we interpret this as saying that some object in Obj_1 is in the

answer, some object in Obj_2 is in the answer, and so on. *Note that the answers returned by a code call depend on the agent's state.* A code call may return one set $\mathbf{Ans}(\mathtt{t})$ of random variables at time $\mathtt{t}$. This reflects the answer to the code call based on the state of the agent at time $\mathtt{t}$. If a message arrives within an infinitesimal amount of time after $\mathtt{t}$ (but before $\mathtt{t}+1$), the answer $\mathbf{Ans}(\mathtt{t}+1)$ returned at time $\mathtt{t}+1$ may not be the same as $\mathbf{Ans}(\mathtt{t})$. This is because the receipt of a message causes the state of the agent to change (in particular the contents of its mailbox has changed). In general, if *any* update is made to the state of the agent between time $\mathtt{t}$ and $\mathtt{t}+1$, $\mathbf{Ans}(\mathtt{t}+1)$ may be different from $\mathbf{Ans}(\mathtt{t})$.

The precise probability that an object $o \in Obj_i$ is in the answer is given by $\wp_i(o)$. The condition of coherence requires that no object occur in two sets Obj_i, Obj_j above. The reason for this is that if o occurs in both Obj_i and Obj_j, then to compute the probability that o appears in the answer, we need to combine the probabilities $\wp_i(o)$ and $\wp_j(o)$. How to combine these two probabilities depends upon our knowledge of the dependencies between the objects in the sets Obj_i, Obj_j.

Example 3.4 *Let us return to our stock market example. Consider a code call* $\mathtt{stock}:up(\mathtt{P})$ *asking for an estimate of the stocks that will go over the value P tomorrow. This code call may return two random variables.*

$$\begin{aligned}\mathbf{RV}_1 &= \langle\{IBM, HP\}, \wp_1\rangle.\\ \mathbf{RV}_2 &= \langle\{GM, Ford\}, \wp_2\rangle.\end{aligned}$$

This says that either the value of IBM's stock or HP's stock will go over the P tomorrow and either the value of GM's stock or Ford's stock will go over P tomorrow. If we assume that $\wp_1$ assigns 0.5 to both IBM and HP and $\wp_2$ assigns 0.7 to GM and 0.3 to Ford then, we know for example that the probability of the value of IBM's stock to go over P is 0.5.

However, suppose that this code call return the following two random variables.

$$\begin{aligned}\mathbf{RV}'_1 &= \langle\{IBM, HP\}, \wp_1\rangle.\\ \mathbf{RV}'_2 &= \langle\{IBM, Ford\}, \wp_2\rangle.\end{aligned}$$

If we assume that $\wp_1$ assigns 0.5 to both IBM and HP and $\wp_2$ assigns 0.7 to Ford and 0.3 to IBM then what is the probability that the value of IBM will go over P tomorrow? There is no simple answer to this question, without making various types of independence assumptions (or other assumptions). This is why we have the requirement of coherence. Note, that it is still not simple even in the first example to answer the question of the probability that both the value of HP and the value of GM will go over P.

We are now ready to introduce the concept of a TP code call.

Definition 3.5 ((Temporal probabilistic *cc* (TPCC))) *A* temporal probabilistic code call based on *cc , denoted cc^{TP}, returns a* mapping *from natural numbers to coherent sets of random variables of type τ.*

Suppose we have a code call $\texttt{stock} : over(\texttt{C}, \texttt{P})$ which ordinarily returns as output, a member of $\{\mathbf{true}, \mathbf{false}\}$ indicating whether the company C's stock price is P or more. A TPCC based on this code call would return a mapping as output. This mapping would associate with each time point t, a coherent set of random variables. For example, at time 1 it may return only one random variable, $(\{\mathbf{true}, \mathbf{false}\}, \delta_u)$ saying that at time 1 there is a 50% probability of the stock being greater than or equal to P (i.e., δ_u is the uniform distribution function). At time 2, it may return the same set, but with the distribution $\wp(\mathbf{true}) = 0.8, \wp(\mathbf{false}) = 0.2$ indicating that there is now an 80% probability of C's stock exceeding P.

In the energy domain, a TPCC based on the code call $\texttt{energy} : demand(\texttt{A})$ that originally returned a demand, returns a mapping from time points (periods) to random variables of the form $\{\{m+1, m+2\}, p\}$, where $p(m+1) = \phi$ and $p(m+2) = 1-\phi$. The actual value of ϕ may be different for different time periods.

In the rest of this paper, we often write *cc* to denote both ordinary code calls as well as TPCC's. The intended meaning should be clear from context.

It is well known that even if we know the exact probabilities of two events e_1, e_2, it is not always easy to obtain a precise point probability for the conjunction or disjunction of these events, though it is possible to obtain a "tightest" possible interval for these probabilities (cf. [Boole, 1854], [Fagin et al., 1990]). In general, depending upon exactly what is known about the two events, the probabilities of their conjunction and disjunction can be computed in many different ways. The following definition (due to [Lakshmanan et al., 1997]) proposes the notion of a probabilistic conjunction/disjunction strategy which provides an abstract view of how to obtain probabilities of conjunctive/disjunctive events.

Given two probability intervals $[L_1, U_1]$ and $[L_2, U_2]$, we say that $[L_1, U_1] \leq [L_2, U_2]$ *iff* $L_1 \leq L_2$ and $U_1 \leq U_2$.

Definition 3.6 ((Probabilistic conj/disj strategy)) *Let events e_1, e_2 be associated probabilistic intervals $[L_1, U_1]$ and $[L_2, U_2]$ respectively. Then a* probabilistic conjunction strategy *(*probabilistic disjunction strategy*) is a binary operation $\otimes$ ($\oplus$) which uses this information to compute the probabilistic interval $[L, U]$ for event "$e_1 \wedge e_2$" ("$e_1 \vee e_2$"). When the events involved are clear from context, we use $[L, U] = [L_1, U_1] \otimes [L_2, U_2]$ to denote $(e_1 \wedge e_2, [L, U]) = (e_1, [L_1, U_1]) \otimes (e_2, [L_2, U_2])$ and we use $[L, U] = [L_1, U_1] \oplus [L_2, U_2]$ to denote $(e_1 \vee e_2, [L, U]) = (e_1, [L_1, U_1]) \oplus (e_2, [L_2, U_2])$. Every strategy must conform to the following postulates:*

Generic postulates *($* $* \in \{\otimes, \oplus\}$ *)*	
1. Commutativity	$([L_1, U_1] * [L_2, U_2]) = ([L_2, U_2] * [L_1, U_1])$
2. Associativity	$(([L_1, U_1] * [L_2, U_2]) * [L_3, U_3])$ = $([L_1, U_1] * ([L_2, U_2]) * [L_3, U_3]))$
3. Monotonicity	$([L_1, U_1] * [L_2, U_2]) \leq ([L_1, U_1] * [L_3, U_3])$ *if* $[L_2, U_2] \leq [L_3, U_3]$
Probabilistic Conjunction postulates	
4.a. Bottomline	$([L_1, U_1] \otimes [L_2, U_2])$ $\leq [\min(L_1, L_2), \min(U_1, U_2)]$
5.a. Identity	$([L_1, U_1] \otimes [1, 1]) = [L_1, U_1]$
6.a. Annihilator	$([L_1, U_1] \otimes [0, 0]) = [0, 0]$
7.a. Ignorance	$([L_1, U_1] \otimes [L_2, U_2]) \subseteq$ $[\max(0, L_1 + L_2 - 1), \min(U_1, U_2)]$
Probabilistic Disjunction postulates	
4.b. Bottomline	$([L_1, U_1] \oplus [L_2, U_2]) \geq$ $[\max(L_1, L_2), \max(U_1, U_2)]$
5.b. Identity	$([L_1, U_1] \oplus [0, 0]) = [L_1, U_1]$
6.b. Annihilator	$([L_1, U_1] \oplus [1, 1]) = [1, 1]$
7.b. Ignorance	$([L_1, U_1] \oplus [L_2, U_2]) \subseteq$ $[\max(L_1, L_2), \min(1, U_1 + U_2)]$

A detailed explanation of why these axioms are reasonable axioms for probabilistic reasoning is given in [Lakshmanan et al., 1997]. Another approach towards combination strategies, which concentrates especially on a clean probabilistic semantics is given in [Eiter et al., 2001]. Both approaches generalise the well known concepts of triangular norms and co-norms which are used for similar purposes. Conjunction and disjunction strategies are more general than T-norm axioms because of additional requirements like Bottomline and Monotonicity.

Each agent has a set of actions that the agent is capable of executing. Actions change the state of the agent. An action has five components: (1) a name $\alpha(X_1, ..., X_n)$ where the $\mathtt{X}_i$'s are variables, (2) a *precondition*, (3) an add list, (4) a delete list, all three of which consist of a set of code calls and temporal probabilistic code calls, and (5) an action *code* which is a body of code that executes the action. Each agent has a notion of *concurrency* specifying how to combine a set of actions into a single action, and a set of *action constraints* that define the circumstances under which uncertain actions may be concurrently executed. A formal model of such agents (with no time and no uncertainty) is given in [Eiter et al., 1999]. For simplicity, we assume (in this paper) that an action has no duration (see [Dix et al., 2001] for how to handle actions with durations).

Definition 3.7 ((Status condition, Status set)) *If* $\alpha(\vec{t})$ *is an action, and* $\mathsf{Op} \in \{\mathbf{P}, \mathbf{F}, \mathbf{W}, \mathbf{Do}, \mathbf{O}\}$*, then* $\mathsf{Op}\alpha(\vec{t})$ *is called a* action status atom*. If* $A_1, \ldots, A_n$ *are status atoms, then* $(A_1 \wedge \ldots \wedge A_n)$ *is a* action status condition*. A* status set *is a finite set of ground action status atoms.*

$\mathbf{P}\alpha$ means α is permitted. $\mathbf{F}\alpha$ means α is forbidden. $\mathbf{Do}\,\alpha$ means α is actually done.

$\mathbf{O}\alpha$ means α is obligatory, and $\mathbf{W}\alpha$ means that the obligation to perform α is waived.

4 Syntax of HTP Agent Programs

Our main aim in this section is to define *temporal probabilistic* agents. A first step is to extend ccc's and status conditions (Definition 3.7) to annotated versions of them (Definition 4.5). We start with the notion of a temporal expression te which may be used to denote time points. Time in our model is discrete because computers clocks are discrete and because it makes computation somewhat easier. Logical agent programming frameworks that use continuous time models are presented in, for example, [Reiter, 1998]—however, these frameworks are unable to deal with leveraging legacy code and assume all data is stored in logic.

Definition 4.1 ((Temporal interval (ti): $[\mathsf{te}_1, \mathsf{te}_2]$) *(1) Every integer is a temporal expression. (2) $\mathtt{X}_{now}$ is a temporal expression. (3) If te_1, te_2 are temporal expressions, then so is $(te_1 + te_2)$.*

If te_1, te_2 are temporal expressions, then $[te_1, te_2]$ is a temporal interval*, denoted ti. It denotes the set of all time points between te_1 and te_2.*

For example, 5, $\mathtt{X}_{\mathsf{now}} + 3$ and $\mathtt{X}_{\mathsf{now}} + \mathtt{X}_{\mathsf{now}} + 13$ are all temporal expressions. We will assume that there is an oracle that automatically assigns a value to $\mathtt{X}_{\mathsf{now}}$ (in an implementation, this can be done by looking at the system clock). Hence, a temporal expression can always be evaluated to a value.

It is easy to see that for two temporal intervals, their intersection is also a temporal interval.

Temporal intervals $[\mathsf{te}_1, \mathsf{te}_2]$ can also be seen as constraints for a time variable $\mathtt{t}$: $\mathsf{te}_1 \leq \mathtt{t} \leq \mathsf{te}_2$.

For example, $[\mathtt{X}_{\mathsf{now}} + 2, \mathtt{X}_{\mathsf{now}} + 7]$ is a temporal interval ti. Thus, if $\mathtt{X}_{\mathsf{now}} = 3$, then $[\mathtt{X}_{\mathsf{now}} + 2, \mathtt{X}_{\mathsf{now}} + 7] = \{5, 6, 7, 8, 9, 10\}$.

We assume the existence of a special set of variables called *probabilistic variables*. Each of these variables is denoted $\mathbf{X_i}$ and ranges over *real values* in the unit interval $[0, 1]$. We also assume the existence of some set of function symbols, each with an associated arity—these function symbols are pre-interpreted and map $[0, 1]^a$ to $[0, 1]$ for appropriate arities a.

Definition 4.2 ((Probabilistic interval (pi): $[\ell_1, \ell_2]$)) *(1) Every real number in the unit interval $[0, 1]$ is a probabilistic item. (2) Every probabilistic variable $\mathbf{X_i}$ is a probabilistic item. (3) If $\ell_1, \ldots, \ell_n$ are probabilistic items and f is an n-ary probabilistic function symbol, then $f(\ell_1, \ldots, \ell_n)$ is a probabilistic item.*

If ℓ_1, ℓ_2 are probabilistic items, then $[\ell_1, \ell_2]$ is called a probabilistic interval.

For example, if $\mathbf{X}$ is a probabilistic variable, then $\frac{\mathbf{X}+1}{2}$ is an example of a probabilistic item. $[\mathbf{X}, \frac{\mathbf{X}+1}{2}]$ is a probabilistic interval.

Definition 4.3 ((Probability distribution function (pdf) δ on a ti))
Suppose $ti = [te_1, te_2]$ *is a time interval. A probability distribution function (**pdf**) w.r.t.* ti *is a mapping* δ *from* $\langle \mathtt{X}_{now}, ti \rangle$ *to* $[0, 1]$ *such that for all values* $\mathtt{t}_0$ *of* $\mathtt{X}_{now}$*:* $\Sigma_{\mathtt{t} \in [te_1(\mathtt{t}_0), te_2(\mathtt{t}_0)]} \delta(\mathtt{t}) = 1$ *(here the expressions* $te_i(\mathtt{t}_0)$ *are the values of* te_i *when* $\mathtt{X}_{now}$ *is replaced by the value* $\mathtt{t}_0$*.*

We also write $\langle [te_1, te_2], \delta \rangle$ *to emphasise the fact that* δ *is a **pdf** over* $[te_1, te_2]$*.*

The above ingredients form the basic building blocks of our temporal probabilistic agents. We are now ready to start putting these building blocks together in a step-by-step effort to define HTP agent programs. We start with the notion of a TP-annotation.

Definition 4.4 ((TP-annotation $[\otimes, \langle [\mathtt{t}_1, \mathtt{t}_2], \delta \rangle, [\ell_1, \ell_2]]$)) *A* TP*-annotation is a triple* $[\otimes, \langle [\mathtt{t}_1, \mathtt{t}_2], \delta \rangle, [\ell_1, \ell_2]]$ *where* $\otimes$ *is a probabilistic conjunction strategy,* $\langle [\mathtt{t}_1, \mathtt{t}_2], \delta \rangle$ *is a temporal interval with a **pdf** δ over it, and* $[\ell_1, \ell_2]$ *is a probabilistic interval. A* TP*-annotation is ground if* ℓ_1, ℓ_2 *are ground.*

For example, $[\otimes_{ig}, \langle [\mathtt{X}_{\mathsf{now}} + 2, \mathtt{X}_{\mathsf{now}} + 7], \delta_u \rangle, [\mathbf{X}, \frac{\mathbf{X}+1}{2}]]$ is a TP-annotation where $\otimes_{ig}$ represents the "ignorance" conjunction strategy and δ_u represents the uniform distribution. This annotation is not ground due to the presence of the variable $\mathbf{X}$. However, $[\otimes_{ig}, \langle [\mathtt{X}_{\mathsf{now}} + 2, \mathtt{X}_{\mathsf{now}} + 7], \delta_u \rangle, [0.3, 0.5]]$ is considered to be ground.

We are now ready to define a temporal probabilistic extension of a code call condition. We also define an extension of the notion of a *status condition* to the temporal probabilistic case.

Definition 4.5 ((TP-CCC, TP-ASC)) *If* χ *is a (ground) code call condition (ccc),* $(A_1 \wedge \ldots \wedge A_n)$ *is a (ground) action status condition, and* $[\otimes, \langle [\mathtt{t}_1, \mathtt{t}_2], \delta \rangle, [\ell_1, \ell_2]]$ *is a (ground) annotation, then* $\chi : [\otimes, \langle [\mathtt{t}_1, \mathtt{t}_2], \delta \rangle, [\ell_1, \ell_2]]$ *is a (ground)* TP CODE CALL CONDITION *(*TP-CCC*), and* $(A_1 \wedge \ldots \wedge A_n) : [\otimes, \langle [\mathtt{t}_1, \mathtt{t}_2], \delta \rangle, [\ell_1, \ell_2]]$ *is a (ground)* TP ACTION STATUS CONDITION *(*TP-ASC*).*

The following is an example TP code call condition:

$$\mathbf{in}(\mathtt{c},\, \mathtt{d} : f(\mathtt{a}, \mathtt{b})) : [\otimes_{ig}, \langle [\mathtt{X}_{\mathsf{now}} + 2, \mathtt{X}_{\mathsf{now}} + 7], \delta_u \rangle, [\mathbf{X}, \frac{\mathbf{X}+1}{2}]].$$

This condition says that there is a probability in the probabilistic interval $[\mathbf{X}, \frac{\mathbf{X}+1}{2}]$ that at some time point between $\mathtt{X}_{\mathsf{now}} + 2$ and $\mathtt{X}_{\mathsf{now}} + 7$, the code call $\mathtt{d} : f(\mathtt{a}, \mathtt{b})$ will return c in its output. Furthermore, for any specific time point in this time interval, the specific probability that c will be returned is uniformly distributed. Similarly,

$$\big(\mathbf{Do}\ buy(\mathtt{ibm}) \ \wedge \ \ \mathbf{Do}\ sell(\mathtt{lucent})\big) : [\otimes_{ig}, \langle [\mathtt{X}_{\mathsf{now}} + 2, \mathtt{X}_{\mathsf{now}} + 7], \delta_u \rangle, [\mathbf{X}, \frac{\mathbf{X}+1}{2}]]$$

is an TP action status condition. It says that there is a probability in the interval $[\mathbf{X}, \frac{\mathbf{X}+1}{2}]$ that at some time point in the temporal interval $[\mathtt{X}_{\mathsf{now}} + 2, \mathtt{X}_{\mathsf{now}} + 7]$, the agent

will both buy IBM stock and sell Lucent stock. Furthermore, the probability of both events occurring is computed from their individual probabilities by using the conjunction strategy of ignorance. In the energy domain the TP code call condition

$$\mathbf{in}(\mathtt{m}+1, \mathtt{energy} : demand(\mathtt{bal})) : [\otimes_{ig}, \langle[\mathtt{X_{now}}, \mathtt{X_{now}}+2], \delta_u\rangle, [0.4, 0.7]]$$

says that there is a probability of 40% to 70% that the demand in Baltimore will be $m+1$ units sometime between $\mathtt{X_{now}}$ and $\mathtt{X_{now}}$ +2. Similarly,

$$\mathbf{Do}\ Bid(\mathtt{c}+5, \mathtt{bal}) : [\otimes_{ig}, \langle[\mathtt{X_{now}}, \mathtt{X_{now}}+1], \delta_u\rangle, [0.6, 0.8]]$$

says that there is a probability between 0.6 and 0.8 that today or tomorrow the generator will bid the price $c+5$ for Baltimore.

We are now ready to define temporal probabilistic rules. As in *IMPACT* [Subrahmanian et al., 2000], the head of a rule is a status atom. The body of a rule is a conjunction of TP-ASC's and TP-CCC's.

Definition 4.6 ((TP-rule)) *A (ground)* temporal probabilistic *(*TP*) rule is an expression of the form*

$$SA_0 \leftarrow tpccc_1 \wedge \ldots \wedge tpccc_n \wedge asc_1 \wedge \ldots \wedge asc_n,$$

where SA_0 is a (ground) TP-ASC *containing only one status atom in it, $tpccc_1, \ldots, tpccc_m$ are (ground)* TP-CCC*'s and $asc_1, \ldots, asc_n$ are (ground)* TP-ASC*'s.*

A temporal probabilistic *(*TP*) rule is strict if any annotation that appears in the rule and is associated with a status atom is of the form $[\otimes, \langle \mathsf{ti}, \delta\rangle, [1,1]]$. We also write $[\otimes, \langle \mathsf{ti}, \delta\rangle]$ for such rules.*

TP rules are used to determine what all actions an agent is permitted to do, obliged to do, forbidden from doing, etc. at a given time. Strict rules allow rule bodies to contain uncertainty about what is true in an agent state (and when)- however, the agent's decisions are not allowed to be uncertain. This is the case when the agent apply deterministic polices in which an agent is either going to do something (or is obliged/permitted/forbidden to do something) with certainty. If a designer of an agent would like his agent to act in a non-deterministic way, then he can implement an action of "flipping a coin" and use the result of the action to make a decision on what other actions to take (see an example in Section 5.1). In the rest of the paper we will consider only strict rules and thus will shorten the annotation to $[\otimes, \langle \mathsf{ti}, \delta\rangle]$.

Definition 4.7 ((TP Agent Program ($\mathcal{TPP}$))) *A* temporal probabilistic agent program *(denoted by $\mathcal{TPP}$) is a finite set of* TP *rules.*[2]

[2]Note that we consider only strict rules.

Example 4.8 ((Stock example: $\mathcal{TPP}$)) *The following two rules encode the simple stock example described earlier.*

$$\begin{array}{l}\mathbf{Do}\ buy(\mathtt{X}) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+5], \delta_u\rangle] \leftarrow \\ \quad \mathbf{in}(\mathtt{X}, \mathtt{stock} : myportfolio()) \quad : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}], \delta_u\rangle] \ \wedge \\ \quad \mathbf{in}(\texttt{"}yes\texttt{"}, \mathtt{stock} : over(\mathtt{X}, 50)) \quad : [\otimes_{ig}, \langle[\mathtt{X}_{now}+10, \mathtt{X}_{now}+20], \delta_u\rangle, [0.8, 1]]\end{array}$$

$$\begin{array}{l}\mathbf{Do}\ buy(\mathtt{X}) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+3], \delta_u\rangle] \leftarrow \\ \quad \big(\mathbf{in}(\texttt{"}no\texttt{"}, \mathtt{stock} : myportfolio()) \ \wedge \\ \quad\ \mathbf{in}(\texttt{"}ok\texttt{"}, \mathtt{stock} : diversify(\mathtt{X}))\big) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}], \delta_u\rangle] \ \wedge \\ \quad\ \mathbf{in}(\texttt{"}yes\texttt{"}, \mathtt{stock} : over(\mathtt{X}, 50)) \ : [\otimes_{ig}, \langle[\mathtt{X}_{now}+10, \mathtt{X}_{now}+20], \delta_u\rangle, [0.9, 1]]\end{array}$$

The first rule says that if stock X is in my current portfolio and it is expected (with 80% probability or more) to go over $50 per share sometime between day 10 and 20 from now, then buy this stock in the next 5 days. The second rule says that if the stock is not in my current portfolio and acquiring it is consistent with the goal to maintain a diversified portfolio, then we should buy the stock in the next 3 days as long as there is an over 90% probability of its going up to $50 per share.

Updates. *It is important to note that neither of these rules, by themselves, say what is true in the current agent state. The current state might say that stock X is in fact in my current portfolio and the current prediction says that it will go up to over $50 in 10 to 20 days from now. Thus, the agent may conclude* now *that it should execute a "buy" order on this stock in the next 5 days. But it may not execute the buy order just yet. On the next day, an update occurs to the stock database which may lead to the estimation that the stock will only rise to $40. This will invalidate the inference that the stock must be bought in the next five days (of course, in the event the stock was already bought on the preceding day, this might trigger some other actions not shown above).*

Example 4.9 ((Energy example: $\mathcal{TPP}$)) *The following rule encodes a simple rule for a generator in the energy example.*

$$\begin{array}{l}\mathbf{P}\,Bid(\mathtt{P}^{\mathtt{max}}, \mathtt{bal}) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+5], \delta_u\rangle] \leftarrow \\ \quad \mathbf{in}(\mathtt{m}+2, \mathtt{energy} : demand(\mathtt{bal})) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+5], \delta_u\rangle, [0.9, 1]]\end{array}$$

The rule says that the generator is permitted to bid the highest price between $\mathtt{X}_{now}$ and $\mathtt{X}_{now}$ +5, if there is a probability between 0.9 and 1 that the demand in Baltimore at some time between $\mathtt{X}_{now}$ and $\mathtt{X}_{now}$ +5 will be $m+2$.

$$\mathbf{F}\,Bid(\mathtt{X}, \mathtt{bal}) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}], \delta_u\rangle] \leftarrow \ \ X > P^{max}$$

This rule says that the generator is forbidden to bid a price that is higher than P^{max} now.

Updates. *Let us say that on day d, the energy demand estimator says that there is a 99%*

probability that the demand for energy in Baltimore tomorrow (day $d+1$) is $m+2$. In this case, on day d, the agent is permitted to bid the highest price on the energy. However, suppose the agent does not actually place the bid on day d. Now, on day $d+1$, the energy demand estimator, as a consequence of updates to market data, may estimate that there is only an 80% probability that the demand for energy in Baltimore some time between day $d+1$ and $d+6$ is $m+2$. On day $d+1$ therefore, the agent can no longer infer that it is permitted to bid the maximum amount on the energy. So it cannot place such a bid on day $d+1$. Example 5.20 later in the paper will show how this kind of update is handled by our semantics.

Incorporating Third Party Updating Mechanisms. Both the above examples explain, intuitively, how updates are handled in our framework. We only handle updates to data rather than updates to the agent's rules themselves. We believe the vast majority of updates will be updates in state, rather than in rules and hence we do not focus on updates to the rules themselves. Remember that an update to an agent state occurs every time the agent receives a message, as well as at any time one of the data structures associated with the agent changes.

Note that if beliefs of an agent are represented in an agent, they must be represented as *data* stored in some data structure the agent has. In this case, these beliefs can be updated using any classical mechanism for updating beliefs (e.g. [Friedman and Halpern, 1997], Jeffrey's revision rule, etc.). For example, suppose $updatebel$ is a function (with two arguments—a set of beliefs to be updated and a set of updates) that implements some belief updating strategy, and suppose BEL is a set of beliefs stored in some data structure of the agent and suppose U is a set of updates to the agent's state. Then the desired belief updating mechanism can be easily encoded in an agent via a rule similar to:

$$\mathbf{Do}\ updatebel(\mathtt{BEL},\mathtt{u}) \quad \leftarrow \quad \mathbf{in}(\mathtt{X},\ \mathtt{agent}:getcurrbeliefs())\ \&\\ \mathbf{in}(\mathtt{U},\ \mathtt{agent}:getcurrupdates()).$$

A major strength of our approach is that we do NOT force all programmers to some prefixed set of actions. They can add whatever actions they want to the agent. In the situation above, they could make $updatebel()$ be any classical mechanism to update beliefs.

Incorporating Third Party Planners. Suppose an agent developer wants to invoke a third party method for decision making using a Markov Decision Process (under some conditions). In this case, this *MDP* code can be invoked directly within our framework. As in the case of updating mechanisms discussed above, we can simply add a rule of the form

$$\mathbf{Do}\ mdp(\langle \mathtt{arguments} \rangle) \quad \leftarrow \quad \langle \textit{condition} \rangle.$$

5 Semantics of HTP Agents

Though the reader may be tempted to infer that the rules given above are read in terms of usual logical inference, it will soon be clear that things are somewhat more complex.

5.1 Agent Decision Cycle

The primary contribution of this section is the concept of a *feasible temporal probabilistic status interpretation* (FTPSI for short). Informally speaking, an FTPSI is a semantical structure that specifies what the agent is permitted to do, forbidden from doing, obliged to do, and actually plans to do at any time instance i. Thus, any given FTPSI says what the agent will do at time 1, 2, 3, etc. Note however that even though $\mathcal{I}$ may be an FTPSI at time $\mathtt{t}$, if some updates occur, $\mathcal{I}$ may no longer be an FTPSI at time $\mathtt{t}+1$. At any given point $\mathtt{t}$ in time, the agent must find an FTPSI and take the actions (now) that FTPSI says it should take. As time passes, the agent continues to take the prescribed actions until an externally caused state change (update) occurs. At this point, the agent computes a *new* FTPSI and acts according to that. This cycle is repeated forever by the agent. The agent's decision cycle may be summed up as follows:

HTP Agent Decision Cycle

1. $\mathtt{t} = 0$;
2. $\text{FTPSI}_{\mathtt{t}}$ = compute initial FTPSI w.r.t. original state;
3. **while** $true$ **do**
   (a) execute all actions to be executed by $\text{FTPSI}_{\mathtt{t}}(\mathtt{t})$;
   (b) $\mathtt{t}++$;
   (c) **if** an externally triggered state change occurred between time $\mathtt{t}$ and $\mathtt{t}+1$ **then** compute a new $\text{FTPSI}_{\mathtt{t}}$;
   (d) **else** $\text{FTPSI}_{\mathtt{t}} = \text{FTPSI}_{\mathtt{t}-1}$;
4. **end**

In the above decision cycle an externally triggered action is one that the agent did not execute. Examples of externally triggered actions include receipt of messages, clock ticks, updates to a database by some third party and so on.
Note that our HTP agent syntax does not *explicitly* allow for stochastic actions. However, such actions can be easily programmed within our framework. For example, suppose the agent developer has written some set S of rules for his or her $\mathtt{agent}$. One way to introduce stochastic actions is as follows. Introduce a new piece of code called

stochastic with two functions: one function takes an action α and a list of ground arguments for it as input, and returns a unique code for that action. This can be done by any one of many standard encoding mechanisms. The other function takes a code as input and randomly assigns 0 or 1 to it. Then, for each action α, the agent developer can automatically add the following two rules to S. A special action called $cache()$ stores action codes in a cache called *tempcache*.

$$\begin{aligned} \mathbf{Do}\, cache(\texttt{Code}) &\leftarrow \mathbf{P}\alpha(\vec{X}) \wedge \\ &\quad\ \mathbf{in}(\texttt{Code}, \texttt{agent} : stochastic\_encode(\alpha, \tilde{\texttt{X}})). \\ \mathbf{Do}\, \alpha(\texttt{X}) &\leftarrow \mathbf{in}(\texttt{Code}, \texttt{agent} : tempcache()) \wedge \\ &\quad\ \mathbf{in}(1, \texttt{agent} : stochastic\_toss(\texttt{Code})). \end{aligned}$$

Note that many variations of this kind of stochastic action are possible.
The main aim of this section is to formally define what constitutes an FTPSI.

5.2 Satisfaction of TP ccc's and TP asc's

Definition 5.1 ((Possible answer situations)) *Consider an agent state $\mathcal{O}$, a code call cc, a set T of time points, and an object o whose type is the same as cc's output type. The* possible answer situations *of cc w.r.t. T and o, denoted* ***pas***(cc, o, T) *is the set*

$$\{(\mathtt{t}, Obj, \wp) \mid \mathtt{t} \in \mathsf{T} \text{ and } (Obj, \wp) \in cc^{\mathrm{TP}}(\mathtt{t}) \text{ and } o \in Obj\}.$$

In other words, the possible answer situations w.r.t. T and o are all random variables that contain o, indexed by all time points in T. It is easy to see that when T is a singleton, because of the coherence requirement on TPCC's, **pas**(cc, o, T) is either the empty set or a singleton.

Example 5.2 ((Energy example: possible answer situations))
Consider the temporal probabilistic code call $\texttt{energy} : demand(\texttt{bal})$. *Suppose it returns:*

- $\{\{m+1, m+2\}, p_0\}$ *for the time point* $\mathtt{t}_{now}$ *where* $p_0(m+1) = 0.25$ *and* $p_0(m+2) = 0.75$.
- $\{\{m+1, m+2\}, p_1\}$ *for time point* $\mathtt{t}_{now}$ *+1 where* $p_1(m+1) = 0.35$ *and* $p_1(m+2) = 0.65$
- $\{\{m+2\}, p_2\}$ *for the time point* $\mathtt{t}_{now}$ *+2 where* $p_2(m+2) = 1$.

In this case:

$$\begin{aligned} &\textbf{\textit{pas}}(\texttt{energy} : demand(\texttt{bal}), m+1, \{\mathtt{t}_{now}, \mathtt{t}_{now}+1, \mathtt{t}_{now}+2\}) = \\ &\quad \{(\mathtt{t}_{now}, \{m+1, m+2\}, p_0), (\mathtt{t}_{now}+1, \{m+1, m+2\}, p_1)\} \end{aligned}$$

Definition 5.3 ((Answer time probabilities)) *Consider an agent state $\mathcal{O}$, a code call cc, a time point* t*, and an object o whose type is the same as cc's output type. The probability that o is in the answer of cc at time* t*, denoted* $\mathsf{prob}(o, cc, \mathtt{t})$*, is given by:*

$$\mathsf{prob}(o, cc, \mathtt{t}) = \begin{cases} \wp(o) & \textit{if } \textbf{\textit{pas}}(cc, o, \{\mathtt{t}\}) = \{(\mathtt{t}, Obj, \wp)\} \\ 0 & \textit{otherwise}. \end{cases}$$

Intuitively, $\mathsf{prob}(o, cc, \mathtt{t})$ specifies the probability that object o is in the result of code call cc at time t. Considering the cc of Example 5.2, $\mathsf{prob}(m+1, \mathtt{energy} : demand(\mathtt{bal}), \mathtt{t}_{\mathsf{now}}) = 0.25$.

We have now formally defined what it means for an object o to be contained in a temporal probabilistic code call. Note that in the basic case (no time, no probability), this is simple membership relation. However in the temporal probabilistic case, things are far more complex because we have to (1) check whether there is a random variable containing o in the answer to the TPCC, and (2) determine the probability of o w.r.t. such a random variable.

The above definition specifies the probability that an object o is in the answer to a code call at time t. However, in order to give a semantics for HTP agent programs, we must specify what it means for a TP-CCC to be satisfied in an agent state. Clearly, we can only define a *probability* of satisfaction of a TP-CCC by an agent state. However, as TP-CCC's involve conjunctions, we need to consider probabilistic conjunction strategies in such a definition. This is because the probability with which a conjunction of code call atoms is satisfied is dependent upon the dependencies, if any, between the events represented by them.

Definition 5.4 ((Satisfaction of TP-CCC's at fixed time t**:** $\mathsf{prob}(\chi, \mathcal{O}, \mathtt{t})$**))** *The probability with which a state $\mathcal{O}$ satisfies a code call condition χ at time* t *under conjunction strategy $\otimes$, denoted* $\mathsf{prob}(\chi, \mathcal{O}, \mathtt{t})$ *is given by* $\otimes_{\mathbf{in}(\mathtt{o}, cc) \in \chi} \mathsf{prob}(o, cc, \mathtt{t})$*, where $\otimes$ is the conjunction strategy associated with χ.*

Recall that the state of an agent can change frequently. When such changes (or updates) occur, TP-CCC*'s that were satisfied prior to the change may no longer be satisfied after the change (and vice versa).*

Example 5.5 ((Energy: (Satisfaction of TP-CCC's at fixed time))
Consider an agent whose state is $\mathcal{O}$. Suppose $\mathtt{energy} : demand(\mathtt{bal})$ *returns* $\{\{m+1, m+2\}, p_0\}$ *for the time point* $\mathtt{t}_{now}$ *where* $p_0(m+1) = 0.25$ *and* $p_0(m+2) = 0.75$.

Consider $\mathtt{energy} : demand(\mathtt{ann})$. *Suppose it returns* $\{\{m+1, m+2\}, q_0\}$ *for the time period* $\mathtt{t}_{now}$ *where* $q_0(m+1) = 0.85$ *and* $q_0(m+2) = 0.15$.

Consider $\chi = \mathbf{in}(\mathtt{m}+2, \mathtt{energy} : demand(\mathtt{bal})) \,\&\, \mathbf{in}(\mathtt{m}+2, \mathtt{energy} : demand(\mathtt{ann}))$ *where the conjunction strategy used is that of independence ($\otimes_{ind}$). Then,* $\mathsf{prob}(\chi, \mathcal{O}, \mathtt{t}_{now}) = 0.75 * 0.15$.

When we consider a ground TP code call of the form $\mathbf{in}(\mathtt{o}, cc) : [\otimes, \langle \mathsf{ti}, \delta \rangle, [\ell, \ell']]$, we may need to consider multiple time points. For example, ti may have two solutions,

$\mathtt{t} = t_1$ and $\mathtt{t} = t_2$. The probability that o is in the answer returned by cc is the probability that o is in the answer returned by cc at time $\mathtt{t}_1$ (event e_1) **or** at time $\mathtt{t}_2$ (event e_2). However, we have no information about dependencies between these two events. Are they independent? Is there some kind of correlation between these events? Are we completely ignorant about the relationship between these two events? In general, we are attempting to evaluate the probability of a disjunction of two events, given information about the probability of the individual events involved. To do this, we assume that every agent has an arbitrary but fixed probabilistic disjunction strategy $\oplus$ that it uses.

We now define two notions of satisfaction of TP-CCC's by an agent state. We use $\mathcal{O} \models tpccc$ where $tpccc$ is a TP-CCC to denote that $tpccc$ is true now or in the past, while $\mathcal{O} \models^{\mathsf{now}} tpccc$ is used to denote the fact that it is satisfied at the current time, now.

Definition 5.6 ((Satisfaction of TPCCC's)) *Suppose $\mathcal{O}$ is an agent state and $\oplus$ is a fixed probabilistic disjunction strategy. Our definition of satisfaction is by induction on the structure of the* TP-CCC.

1. *$\mathcal{O}$ satisfies a ground* TP-CCC $\chi : [\otimes, \langle \mathit{ti}, \delta\rangle, [\ell, \ell']]$, *denoted* $\mathcal{O} \models \chi : [\otimes, \langle \mathit{ti}, \delta\rangle, [\ell, \ell']]$ *iff* $\ell \leq \oplus\{\mathsf{prob}(\chi, \mathcal{O}, \mathtt{t}) \mid \mathtt{t} \in \mathit{ti}\} \leq \ell'$,
2. *$\mathcal{O}$ satisfies a ground* TP-CCC $\chi : [\otimes, \langle \mathit{ti}, \delta\rangle, [\ell, \ell']]$ *with respect to* $\mathtt{t}_{\mathit{now}}$ *and the past, denoted* $\mathcal{O} \models^{\mathsf{now}} \chi : [\otimes, \langle \mathit{ti}, \delta\rangle, [\ell, \ell']]$ *iff* $\ell \leq \oplus\{\mathsf{prob}(\chi, \mathcal{O}, \mathtt{t}) \mid \mathtt{t} \in \mathit{ti}, \mathtt{t} \leq \mathtt{t}_{\mathit{now}}\} \leq \ell'$,
3. *$\mathcal{O}$ satisfies a conjunction $tpccc_1 \wedge tpccc_2$ of* TP-CCC*'s iff $\mathcal{O} \models tpccc_1$ and $\mathcal{O} \models tpccc_2$.*
4. *$\mathcal{O}$ satisfies $(\forall x)F$ iff $\mathcal{O} \models F[x/c]$ where c is a constant of the same type as x and $F[x/c]$ denotes the replacement of all free[3] occurrences of x in F by c.*

The definition of the last two cases for $\models^{\mathsf{now}}$ is identical to that of $\models$.

Example 5.7 ((Energy: Satisfaction of TPCCC's))
Consider the agent state $\mathcal{O}$ of Example 5.5. Suppose $\mathtt{energy} : demand(\mathtt{bal})$ *returns:*

- $\{\{m{+}1, m{+}2\}, p_0\}$ *for time* $\mathtt{t}_{\mathit{now}}$ *where* $p_0(m{+}1) = 0.25$ *and* $p_0(m{+}2) = 0.75$
- $\{\{m{+}1, m{+}2\}, p_1\}$ *for time* $\mathtt{t}_{\mathit{now}}{+}1$ *where* $p_1(m{+}1) = 0.35$ *and* $p_1(m{+}2) = 0.65$.

In this case

$$\mathcal{O} \models \mathbf{in}(\mathtt{m} + 2, \mathtt{energy} : demand(\mathtt{bal})) : [\oplus_{ind}, \langle [\mathtt{X}_{\mathit{now}}, \mathtt{X}_{\mathit{now}} + 1], \delta\rangle, [0.4, 0.7]].$$

This is because:

[3] We do not formally define free occurrences here but refer the reader to the standard definition which may be readily adapted to our case.

- $[\mathtt{X}_{now}, \mathtt{X}_{now} + 1] = \{\mathtt{X}_{now}, \mathtt{X}_{now} + 1\}$*;*
- $\mathsf{prob}(m + 1, \mathtt{energy} : demand(\mathtt{bal}), \mathtt{t}_{now}) = 0.25$*;*
- $\mathsf{prob}(m + 1, \mathtt{energy} : demand(\mathtt{bal}), \mathtt{t}_{now} + 1) = 0.35$*;*
- *Applying* $\oplus_{ind}$ *to these events we obtain:* $0.25 + 0.35 - 0.25 * 0.35 = 0.5125$ *which is between 0.4 and 0.7.*

However,

$$\mathcal{O} \not\models^{\mathsf{now}} \mathbf{in}(\mathtt{m} + 2, \mathtt{energy} : demand(\mathtt{bal})) : [\oplus_{ind}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now} + 1], \delta\rangle, [0.4, 0.7]].$$

Here, we need to consider only $\mathtt{t}_{now}$ *and we see that*

$$\mathsf{prob}(m + 1, \mathtt{energy} : demand(\mathtt{bal}), \mathtt{t}_{now}) = 0.25$$

which is less than 0.4*.*

5.3 HTP status models

We now define the formal semantics of HTP agents. The formal semantics is given via a concept of a *feasible* TP status interpretation. These are mathematical structures which we will define in this section. Intuitively, a feasible TP status interpretation specifies, for each time point, a set of status atoms which are true at that time point. Of course, not any such set will suffice—rather the set must satisfy various "feasibility" requirements. The goal of this section is to provide such a definition of a feasible TP status interpretation.

We start by noting that in the preceding section, we have already provided a semantics for TP-CCC's. In order to give a formal semantics for HTP agents, we need to define a semantics for TP-ASC's as well. The notion of a TP-status interpretation given below is used to define a semantics for TP-ASC's.

Definition 5.8 ((TP-status interpretation (TPSI)))
A TP-status interpretation *(*TPSI *for short) is a mapping* ρ *that maps natural numbers to status sets.*

Thus, given any time point t, $\rho(\mathtt{t})$ is a status set. Intuitively, if $\rho(3) = \{\mathbf{O}\alpha, \mathbf{Do}\,\alpha, \mathbf{P}\alpha, \mathbf{F}\beta\}$, then this means that according to the TP-status interpretation ρ, at time instant 3, α is obligatory/done/permitted, while β is forbidden. Similarly, if $\rho(4) = \{\mathbf{P}\alpha\}$ then according to the temporal status set, at time 4, α is permitted.

We now define what it means for a TP status interpretation to satisfy an TP action status condition.

Definition 5.9 ((Satisfaction of TP-ASC's)) *Suppose* ρ *is a* TP*-status interpretation. To define the meaning of a* TP-ASC*, we proceed by induction on the structure of a* TP-ASC *and distinguish between a status atom and a conjunction of status atoms.*

- *The probability with which ρ satisfies a status atom* $\mathsf{Op}\alpha$ *at time* t *is*

$$\mathsf{prob}(\mathsf{Op}\alpha, \rho, \mathsf{t}) = \begin{cases} 1 & \textit{if } \mathsf{Op}\alpha \in \rho(\mathsf{t}) \\ 0 & \textit{otherwise.} \end{cases}$$

- *The probability with which ρ satisfies a status condition $\varrho = (A_1, \wedge \ldots \wedge\ A_n)$ w.r.t. a given conjunction strategy $\otimes$ and at a given time* t*, denoted* $\mathsf{prob}_{\otimes}(\varrho, \rho, \mathsf{t})$ *is given by:*

$$\mathsf{prob}(A_1, \rho, \mathsf{t}) \otimes \mathsf{prob}(A_2, \rho, \mathsf{t}) \otimes \cdots \otimes \mathsf{prob}(A_n, \rho, \mathsf{t}).$$

- *Suppose* $(A_1\ \wedge \ldots \wedge\ A_n) : [\otimes, \langle \mathit{ti}, \delta \rangle]$ *is a status condition.*

$$\begin{array}{c} \rho \ \textit{satisfies}\ \ (A_1\ \wedge \ldots \wedge\ A_n) : [\otimes, \langle \mathit{ti}, \delta \rangle]\ \ \textit{at time}\ \ \mathsf{t}\ \ \textit{w.r.t.}\ \oplus \\ \textit{iff} \\ \oplus\{\mathsf{prob}_{\otimes}(A_1\ \wedge \ldots \wedge\ A_n, \rho, \mathsf{t}) \mid \mathsf{t} \in \mathit{ti}\} = 1. \end{array}$$

Note that for status *atoms*, a TPSI either satisfies it (probability 1) or not (probability 0). For status conditions however, other probabilities can arise as well. The reason for this is that status atoms are either true or not.

Definition 5.10 *Suppose* $SA_0 \leftarrow tpccc_1 \wedge \ldots \wedge tpccc_n \wedge asc_1 \wedge \ldots \wedge asc_n$ *is a* ground TP-*rule, ρ is a* TP-*status interpretation and $\mathcal{O}$ is the current agent state.*

ρ satisfies *the above ground rule in state $\mathcal{O}$ iff either:*

1. *$\mathcal{O}$ does not satisfy $tpccc_1 \wedge \ldots \wedge tpccc_n$ with respect to now and the past, or*
2. *ρ does not satisfy $asc_1 \wedge \ldots \wedge asc_n$ or*
3. *ρ satisfies SA_0.*

ρ satisfies a rule r iff it satisfies all ground instances of r. ρ satisfies a TP *program $\mathcal{TPP}$ iff for each temporal probabilistic agent rule $r \in \mathcal{TPP}$, ρ satisfies r.*

Example 5.11 ***(Energy:* TP-*status interpretation)*** *Consider the following very simple table describing a* TP-*status interpretation ρ of a generator agent.*[4]

[4]For simplicity of the examples, we do not consider the precondition and the add and delete lists of the actions in the energy example.

i	$\rho(i)$
0	$\{\mathbf{F}Bid(P^{max}, bal)$, $\mathbf{P}Produce(m, bal)$, $\mathbf{O}Produce(m, bal), \mathbf{Do}\ Produce(m, bal)\}$
1	$\{\ \mathbf{P}Produce(m+1, bal)$, $\mathbf{P}Bid(c+5, bal)$, $\mathbf{Do}\ Produce(m+1, bal)$, $\mathbf{Do}\ Bid(c+5, bal)$, $\}$
2	$\{\ \mathbf{Do}\ Produce(m+1, bal)\ \}$
3	$\{\mathbf{O}Produce(m+1, bal), \mathbf{Do}\ Produce(m+1, bal)$, $\mathbf{P}\ Produce(m+1, bal)$, $\mathbf{P}\ Bid(c+7, bal)$, $\mathbf{P}\ Bid(P^{max}, bal)\ \}$
4	$\{\mathbf{F}Bid(P^{max}+1, bal)$, $\mathbf{P}\ Produce(m+1, bal)$, $\mathbf{Do}\ Produce(m, ann)\ \}$
$4 < i < 9$	$\{\ \mathbf{F}Bid(P^{max}+1, bal), \}$
$i > 9$	$\emptyset$

Suppose we also consider a simple description of the state $\mathcal{O}$ of the generator agent. As in the previous examples $\texttt{energy} : demand(\texttt{A})$ *returns a mapping from time points (periods) to random variables of the form $\{\{m+1, m+2\}, p\}$, where $p(m+1) = \phi$ and $p(m+2) = 1 - \phi$. The actual values are specified in the following table:*

Time	*bal*		*ann*	
	$p(m+1)$	$p(m+2)$	$q(m+1)$	$q(m+2)$
0	*0.30*	*0.70*	*0.60*	*0.40*
1	*0.40*	*0.60*	*0.60*	*0.40*
3	*0.25*	*0.75*	*0.85*	*0.15*
4	*0.35*	*0.65*	*0.90*	*0.10*

Suppose $\texttt{t}_{now} = 3$ *and the agent uses $\oplus_{ind}$ as its fixed probabilistic disjunction strategy. ρ satisfies the following rules in $\mathcal{O}$:*

- $\mathbf{P}Bid(\texttt{P}^{\texttt{max}}, \texttt{bal}) : [\otimes_{ig}, \langle[\texttt{X}_{now}, \texttt{X}_{now}+1], \delta_u\rangle] \leftarrow$
  $\mathbf{in}(\texttt{m}+2, \texttt{energy} : demand(\texttt{bal})) : [\otimes_{ig}, \langle[\texttt{X}_{now}, \texttt{X}_{now}+5], \delta_u\rangle, [0.9, 1]]$

  ρ satisfies the rule because it satisfies $\mathbf{P}Bid(\texttt{P}^{\texttt{max}}, \texttt{bal}) : [\otimes_{ig}, \langle[\texttt{X}_{now}, \texttt{X}_{now}+1], \delta_u\rangle]$.

- $\mathbf{O}Bid(\texttt{c}+5, \texttt{bal}) : [\otimes_{ig}, \langle[\texttt{X}_{now}, \texttt{X}_{now}+1], \delta_u\rangle] \leftarrow$
  $\mathbf{P}Bid(\texttt{c}+5, \texttt{bal}) : [\otimes_{ig}, \langle[\texttt{X}_{now}, \texttt{X}_{now}], \delta_u\rangle]$

  ρ satisfies the rule because it does not satisfy $\mathbf{P}Bid(\texttt{c}+5, \texttt{bal}) : [\otimes_{ig}, \langle[\texttt{X}_{now}, \texttt{X}_{now}], \delta_u\rangle]$.

ρ does not satisfy the following rule:

$$\mathbf{O}Bid(\mathtt{P}^{\mathtt{max}}, \mathtt{bal}) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+1], \delta_u\rangle] \leftarrow$$
$$\mathbf{in}(\mathtt{m}+2, \mathtt{energy} : demand(\mathtt{bal})) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+1], \delta_u\rangle, [0.9, 1]]$$

ρ does not satisfy the rule because $\mathcal{O}$ satisfies

$$\mathbf{in}(\mathtt{m}+2, \mathtt{energy} : demand(\mathtt{bal})) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+1], \delta_u\rangle, [0.9, 1]]$$

but does not satisfy

$$\mathbf{O}Bid(\mathtt{P}^{\mathtt{max}}, \mathtt{bal}) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+1], \delta_u\rangle].$$

An agent may record the actions it took (or was obliged to take, forbidden from taking etc.) in the past. This leads to the notion of an *action history*.

Definition 5.12 ((Action History acthist)) *An* action history *acthist for an agent is a partial function from $\{0, 1, ..., \mathtt{t}_{now}\}$ to status sets.*

Intuitively, an action history specifies what the agent has done in the past. As action histories are partial (rather than total) functions, the agent developer has the flexibility to choose to store none, some or all status atoms associated with the agent's past.

Example 5.13 ***(Energy: Action History)***

i	*acthist*(i)
0	$\{\mathbf{F}Bid(P^{max}, bal),$ $\mathbf{P}\ Produce(m, bal),$ $\mathbf{O}Produce(m, bal), \mathbf{Do}\ Produce(m, bal)\}$
1	$\mathbf{P}\ Produce(m+1, bal), \mathbf{P}Bid(c+5, bal),$ $\mathbf{Do}\ Produce(m+1, bal), \mathbf{Do}\ Bid(c+5, bal), \}$
2	$\emptyset$

An action history and a TP*-status interpretation both make statements about action status atoms. Therefore they need to be* compatible*.*

Definition 5.14 ((History-Compatible TPSI)) *Suppose the current time is $\mathtt{t}_{now}$ and acthist$(\cdot)$ denotes the action history of an agent, and suppose ρ is a* TPSI*. ρ is said to be* action history-compatible *at time $\mathtt{t}_{now}$ iff for all $\mathtt{t} < \mathtt{t}_{now}$, if acthist$(\mathtt{t})$ is defined, then $\rho(\mathtt{t}) =$ acthist$(\mathtt{t})$.*

In other words, for a TPSI to be compatible with an action history, it must be consistent with the past history of actions taken by the agent and with commitments to do things in the future that were made in the past by the agent. The action history of example 5.13 and the TP-status interpretation of example 5.11 are compatible.

5.4 Feasible Temporal Probabilistic Interpretations

Let us consider an agent $\mathfrak{a}$ that uses a TP-interpretation ρ to determine what actions it should take, and when it should take these actions. Not all such interpretations make sense. For example we must impose conditions that ensure an action is not both permitted and forbidden at the same time (deontic conditions as formalised in Definitions 5.17, 5.16). Furthermore, such an interpretation ρ must be closed under the rules of the TP program. The main results of this section are Definition 5.19, which states the conditions we need, and Theorem 5.21, which determines the complexity of verifying them for a given TP-interpretation.

An important requirement about feasibility is that actions scheduled for the future need to be executable. For example, it does not make sense for an agent to decide to execute the action $\mathbf{Do}\, sell(ibm, 100)$ (i.e. sell 100 shares of IBM) 5 days from now if the agent does not have 100 shares to sell at that time. In order for conditions such as this to be incorporated into our definition of feasibility, we need to have a concept of *expected* future states. This notion is defined below.

Definition 5.15 ((Expected States at time $\mathtt{t}$: $\mathcal{EO}(\mathtt{t})$)) *Suppose the current time is $\mathtt{t}_{now}$, $\mathcal{O}$ is the agent state and ρ is a* TP*-status interpretation. The agent's expected states are defined as follows: $\mathcal{EO}(\mathtt{t}_{now}) = \mathcal{O}$,*

- *For all time points $\mathtt{t} > \mathtt{t}_{now}$, $\mathcal{EO}(\mathtt{t})$ is the result of concurrently executing $\{\alpha \mid \mathbf{Do}\, \alpha \in \rho(\mathtt{t}-1)\}$ in state $\mathcal{EO}(\mathtt{t}-1)$.*

As described earlier in Section 5.1 on the agent decision cycle, at any given point i in time, the agent has a "current" feasible temporal probabilistic status interpretation ρ_i. The expected future states of the agent at times $\mathtt{t} \geq i$ are defined in terms of the current FTPSI ρ_i. The state $\mathcal{EO}(i+1)$ at time $i+1$ is obtained by concurrently executing all actions of the form $\mathbf{Do}\, \alpha \in \rho(i)$ in the state of the agent at time i. The state $\mathcal{EO}(i+2)$ is obtained by concurrently executing all actions of the form $\mathbf{Do}\, \alpha \in \rho(i+1)$ in the state of the agent at time $i+1$, i.e. in the state $\mathcal{EO}(i+1)$, and so on. Thus, at any given time point i, the set of possible future states of the agent is *fixed*. There is no nondeterminism.

In contrast, in many existing mechanisms like *Markov Decision Processes* (*MDP*), the system reasons about all possible things that could happen in the future [Boutilier et al., 1999]. This is impractical to do in many applications. For example, let us return to the case of a stock database agent. In the real world, this agent is likely to support a code call called $\mathtt{stock} : sql()$ which ships any *SQL* query or update to the underlying *DBMS*. Virtually any such update could arise. As a consequence, if we used a state space based approach (as a Markov decision process might do), there would be an infinite branching factor for each node in the tree as the set of potential events that could occur is infinite.

This clearly explains the difference between *MDP*'s and our framework. We do not attempt to reason about all possible things that could happen in the world in future.

Rather we reason about what changes *we* plan to bring about in the world in the future. This may be shaped by our *predictions* about what the future might bring. E.g. the predictive component of the stock example predicts that the stock price for a given stock will exceed \$50 in 10-20 days with probability higher than 0.8. It may also predict that it will only reach \$40 in 10-20 days with probability 0.2. Our decision on whether to buy the given stock may depend on both prediction. Likewise, the predictive component of our energy example predicts what the demand for energy in Baltimore might be in the future, and our decision on what to bid is based on that prediction.

The definition of TP deontic consistency below requires, for example, that at all times $\mathtt{t}$, we cannot have deontic conflicts. Furthermore, for time points $\mathtt{t} \geq \mathtt{t}_{\mathsf{now}}$ in the past, if an action is permitted at that time, then the precondition of that action must be true in the state *expected* at that time. A similar condition holds for the past. This is the gist of the following definition.

Definition 5.16 ((TP Deontic Consistency)) *Suppose $\mathcal{O}$ is the agent state. A* TP-*status interpretation ρ is said to be* TP deontically consistent *at time $\mathtt{t}_{now}$ iff it satisfies the following conditions for all time points* $\mathtt{t}$ *(in the following, $Pre(\alpha)$ stands for the preconditions of the action α):*

(1) $\mathbf{O}\alpha \in \rho(\mathtt{t}) \rightarrow \mathbf{W}\alpha \notin \rho(\mathtt{t})$*;*
(2) $\mathbf{P}\alpha \in \rho(\mathtt{t}) \rightarrow \mathbf{F}\alpha \notin \rho(\mathtt{t})$*;*

(3) if $\mathtt{t} \leq \mathtt{t}_{now}$*,* $\mathbf{P}\alpha \in \rho(\mathtt{t})$*, then* $\mathsf{prob}(Pre(\alpha), \mathcal{O}, \mathtt{t}) = 1$*,*

(4) if $\mathtt{t} > \mathtt{t}_{now}$*,* $\mathbf{P}\alpha \in \rho(\mathtt{t})$*, then* $\mathsf{prob}(Pre(\alpha), \mathcal{EO}(\mathtt{t}), \mathtt{t}) = 1$*.*

The first two conditions express the underlying meaning of the deontic atoms: at any given point in time, and for any action, if the agent is obliged to perform the action, then it means that the obligation to perform the action has not been waived. Conditions (3) and (4) formalise the intuition that a permitted action should have a precondition that is true (with probability 1). While (3) talks about current or past time points, condition (4) refers to future time points (and thus has to use the notion of expected states). Thus, if $\rho(4) = \{\mathbf{P}\alpha, \mathbf{F}\alpha\}$, then ρ cannot be TP-deontically consistent.

The reader would have noticed that conditions (3) and (4) in the preceding definition only apply to actions which are permitted. Surely, the same conditions should also apply to actions that are to be done or obligatory (i.e. when $\mathbf{Do}\,\alpha$ or $\mathbf{O}\alpha$ are true at time i) ? We define below, the notion of TP-deontic closure which accomplishes this. Given a set S of status atoms, let $\mathbf{D\text{-}Cl}(S)$ be the smallest superset S' of S such that $\mathbf{O}\alpha \in S' \rightarrow \mathbf{P}\alpha \in S'$. Likewise, let $\mathbf{A\text{-}Cl}(S)$ be the smallest superset S^* of S such that (i) $\mathbf{O}\alpha \in S^* \rightarrow \mathbf{Do}\,\alpha \in S^*$ and (ii) $\mathbf{Do}\,\alpha \in S^* \rightarrow \mathbf{P}\alpha \in S^*$. We say that set S is *deontically closed iff* $S = \mathbf{D\text{-}Cl}(S)$ and *action closed iff* $S = \mathbf{A\text{-}Cl}(S)$.

The following definition explains what it means for a TP-status interpretation to be closed under the deontic modalities.

Definition 5.17 ((TP Deontic/Action Closure)) *ρ is said to be* TP deontically closed *at time $\mathtt{t}_{now}$ iff for all time points* t*:* $\mathbf{D\text{-}Cl}(\rho(\mathtt{t})) = \rho(\mathtt{t})$. *$\rho$ is said to be* TP action closed *at time $\mathtt{t}_{now}$ iff for all time points* t*:* $\mathbf{A\text{-}Cl}(\rho(\mathtt{t})) = \rho(\mathtt{t})$.

It is easy to see that the TP-interpretation ρ of example 5.11 is both TP-deontically consistent and deontically and action closed.

For a temporal probabilistic status set to be feasible, it must satisfy the additional requirement of TP-state consistency. We assume that an agent has a set of integrity constraints $\mathcal{IC}$ of the form: $\psi \Rightarrow \chi$ where ψ is a code call condition, and χ is an atomic code call. These constraints are evaluated in the state. They say that an agent must never transition to a state that violates any of these constraints. [5]

Definition 5.18 ((TP State Consistency)) *ρ is said to be* TP-state consistent at time $\mathtt{t}_{\mathsf{now}}$ *iff for each integrity constraint $\psi \Rightarrow \chi$ in $\mathcal{IC}$ for all $\mathtt{t} \leq \mathtt{t}_{now}$ for every legal assignment of objects from $\mathcal{O}$ to the variables of ψ and χ either* $\mathsf{prob}(\psi, \mathcal{O}, i) \neq 1$ *or* $\mathsf{prob}(\chi, \mathcal{O}, \mathtt{t}) = 1$

A feasible TPSI is like a model of a classical logic theory.

Definition 5.19 ((Feasible TPSI (FTPSI))) *Suppose the current time is $\mathtt{t}_{now}$, $\mathcal{TPP}$ is a* TP *program, $\mathcal{O}$ is an agent state,* acthist *is an action history and $\mathcal{IC}$ is a set of integrity constraints. A* TP*-status interpretation ρ satisfying $\rho(i) \neq \emptyset$ for only finitely many i is said to be* feasible *with respect to the above parameters, denoted by* FTPSI*, iff*

(1) ρ satisfies all rules in $\mathcal{TPP}$,

(2) ρ is TP *deontically consistent at time $\mathtt{t}_{now}$,*

(3) ρ is TP *deontically and action closed at time $\mathtt{t}_{now}$,*

(4) ρ is TP*-state consistent at time $\mathtt{t}_{now}$,*

(5) ρ is action history compatible at time $\mathtt{t}_{now}$.

Recall again that we would like the agent's decision cycle to always ensure that a valid state (one where the integrity constraints are satisfied) is reached by in accordance with the agent's agent program. The key idea behind FTPSI's is that whenever a state change occurs (e.g. when the agent receives a message, or when the clock ticks, or when an update is made), the agent computes a new feasible TPSI ρ and concurrently executes all actions α such that $\mathbf{Do}\,\alpha \in \rho(\mathtt{t}_{\mathsf{now}})$. ρ also specifies what actions are to be executed at time $\mathtt{t}_{\mathsf{now}}+1, \mathtt{t}_{\mathsf{now}}+2, \ldots$ and so on. However if an update occurs between time $\mathtt{t}_{\mathsf{now}}$ and $\mathtt{t}_{\mathsf{now}}+1$, then the agent computes a *new* FTPSI ρ' and the actions it takes at time

[5] [Eiter et al., 1999] also allows agents to have a set, possibly empty, of *action constraints*. It has been shown [Subrahmanian et al., 2000] that action constraints can be expressed as integrity constraints, and hence, we do not consider them in this paper.

$\mathtt{t}_{\mathsf{now}} + 1$ will be those that are specified by $\rho'(\mathtt{t}_{\mathsf{now}} + 1)$. Thus, updates are evaluated and handled on an ongoing basis via the computation of new FTPSI's that specify what actions must be taken when an update occurs. We note that an agent may have zero, one, or many FTPSI's in a given state. Later (Section 7), we give a mechanism to choose between multiple FTPSI's using an objective function.

We demonstrate the updates using the following example.

Example 5.20 *Suppose the generator's* TP *program, $\mathcal{TPP}$, contains the following rule:*

$$\mathbf{Do}\ Produce(m, ann)[\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now} + 1], \delta_u\rangle] \leftarrow$$
$$\mathbf{in}(\mathtt{m} + 1, \mathtt{energy} : demand(\mathtt{ann})) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now} + 1], \delta_u\rangle, [0.95, 1]]$$

Suppose that originally the agent's state is as specified in example 5.11, $\mathtt{t}_{now} = 3$ and the agent uses $\oplus_{ind}$ as its fixed probabilistic disjunction strategy. Suppose that in order to satisfy the rule above, $\mathbf{Do}\ Produce(m, ann) \in \mathrm{FTPSI}_3(4)$*, as specified in the* TP*-status interpretation of example 5.11.*

Suppose that an external update of the agent state occurred (e.g., new data about weather condition arrived) and the actual values of $\{\{m + 1, m + 2\}, p\}$, and $\{\{m + 1, m + 2\}, q\}$, are as specified in the following table (the new and modified values are emphasised):

Time	*bal*		*ann*	
	$p(m+1)$	$p(m+2)$	$q(m+1)$	$q(m+2)$
0	*0.30*	*0.70*	*0.60*	*0.40*
1	*0.40*	*0.60*	*0.60*	*0.40*
3	*0.25*	*0.75*	*0.85*	*0.15*
4	*0.35*	*0.65*	**0.85**	*0.15*
5	**0.35**	**0.65**	**0.60**	**0.10**

After the update, when $\mathtt{t}_{now} = 4$ the agent will compute FTPSI_4*. In the new state,*

$$\mathbf{in}(\mathtt{m} + 1, \mathtt{energy} : demand(\mathtt{ann})) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now} + 1], \delta_u\rangle, [0.95, 1]]$$

is not satisfied. Thus, $\mathbf{Do}\ Produce(m, ann)$ *will not belong to* $\mathrm{FTPSI}_4(4)$.

Theorem 5.21 ((Complexity)) *Suppose the current time is $\mathtt{t}_{now}$, $\mathcal{TPP}$ is a* TP *program, $\mathcal{O}$ is an agent state, acthist is an action history and $\mathcal{IC}$ is a set of integrity constraints.*

1. *The problem of checking whether a given* TPSI ρ *is feasible or not, can be done in polynomial time.*
2. *The problem of checking whether a feasible* TPSI ρ *exists, is **NP**-complete.*

Proof: These complexity results rely on several assumptions about the underlying software code and the active domains of objects considered. These assumptions, as well as the effect of having no integrity constraints, are discussed in [Subrahmanian et al., 2000, Eiter et al., 2000] and we refer the interested reader to these papers.

Checking feasibility of a given TPSI ρ reduces to checking all the conditions in Definition 5.19. Obviously, action history compatibility, deontic consistency, deontic and action closedness, as well as TP state consistency can all be checked in linear time. So it remains to check closedness under the rules. But this is also a linear check, since we fix the underlying temporal probabilistic agent program. Therefore, due to our assumptions about the underlying software code, we get at most polynomial complexity overall.

By what we have just shown, the existence problem is certainly in **NP** (we can guess a candidate and then verify it in polynomial time). **NP** completeness follows immediately from the respective result for ordinary (non TP) programs as first proved in [Eiter et al., 2000]. ∎

6 FTPSI Computation

The preceding section defines a formal semantics based on the concept of a feasible TP-status interpretation. We now show how FTPSI's can be constructed for finite strict TPP's.

The method and techniques we are applying here extend those used in [Dix et al., 2001]. The key part of the algorithm is a fixpoint operator. A huge difference is that the fixpoint operator is completely different from that in [Dix et al., 2001]. As a consequence, though many of the results in this section are similar in statement as those in [Dix et al., 2001], the proofs are completely different.

Since we are considering only strict rules, for simplicity, we assume that an annotation associated with a status condition is of the form ti where ti is a temporal interval. In addition, if ti is a singleton $[\mathsf{t}, \mathsf{t}]$, we will use the annotation $[\mathsf{t}]$.

In the following, we will consider sets of status atoms of the form $\mathsf{Op}(\alpha) : [\mathsf{t}, \mathsf{t}']$ and refer to them as temporally constrained status atoms. We also write $\mathsf{Op}(\alpha) : \mathsf{ti}$ if the special form of ti is not important. The fixpoint operator to be defined below is operating on such sets.

Definition 6.1 ((Temporally constrained status sets tc-$\mathcal{T}S$)) *A temporally constrained status set* tc-$\mathcal{T}S$ *consists of status atoms of the form* $\mathsf{Op}(\alpha) : \mathsf{ti}$*. We also call status atoms of the form* $\mathsf{Op}(\alpha) : [\mathsf{t}]$*, where* t *is an integer,* singleton TP status atoms.

It is straightforward to construct a TP-status interpretation from a set of singleton TP status atoms tc-$\mathcal{T}S$:

$$\mathsf{Op}(\alpha) \in \rho(\mathsf{t}) \textit{ iff } \mathsf{Op}(\alpha) : [\mathsf{t}] \in \text{tc-}\mathcal{T}S.$$

In this case, ρ and tc-$\mathcal{TS}$ are *compatible*.

However, if tc-$\mathcal{TS}$ is a temporally constrained TP status atoms set, there are several TP-status interpretations that can be constructed based on it and the notion of compatibility is defined as follows.

Definition 6.2 ((TPSI Compatible with tc-$\mathcal{TS}$)) *A* TP-*status interpretation* ρ *is* compatible with *tc-$\mathcal{TS}$ iff for every* $\mathsf{Op}\,\alpha$: *ti in tc-$\mathcal{TS}$, there is a solution* $i \in$ *ti such that* $\mathsf{Op}\,\alpha \in \rho(i)$.

We denote the set of all TP-*status interpretations that are compatible with tc-$\mathcal{TS}$ by* Comp-*tc-$\mathcal{TS}$*.

There are an infinite number of TP-status interpretations that are compatible with a given tc-$\mathcal{TS}$. For example, the TP-status interpretation of Example 5.11 is compatible with

$$\begin{array}{ll}\{ & \mathbf{F}\,Bid(\mathtt{P}^{\mathtt{max}}, bal)[0,6], \\ & \mathbf{Do}\; Produce(m+1, bal)[0,3], \\ & \mathbf{F}\,Bid(P^{max}+1, bal)[4,\infty], \\ \} & \end{array}$$

In addition, there could be infinitely many temporally constrained TP status atom sets that are compatible with a given ρ. We denote the subset of the singleton status atoms of tc-$\mathcal{TS}$ by $Singl$(tc-$\mathcal{TS}$).

Furthermore, we will say that tc-$\mathcal{TS}$ has a given property, e.g., it is feasible, *iff* its compatible ρ is feasible. Thus, given a program, $\mathcal{TPP}$, an agent state, $\mathcal{O}$, and an action history acthist, our goal is to construct a feasible tc-$\mathcal{TS}$. We will use tc-$\mathcal{TS}$ and its corresponding ρ synonymously.

In Subsection 6.1 we introduce our main technical machinery for constructing feasible tc-$\mathcal{TS}$: a fixpoint operator which possesses a least fixpoint. All feasible tc-$\mathcal{TS}$ are compatible with this fixpoint (Theorem 6.8) and thus we have reduced our problem to finding all compatible sets. This requires further technical notions which are introduced in Subsection 6.2 and finally lead to Algorithm 6.9 which is shown to be sound and complete (Theorem 6.16).

6.1 The Fixed point operator

In this section, we consider the problem of constructing a feasible tc-$\mathcal{TS}$ based on a TP program, $\mathcal{TPP}$, an agent state, $\mathcal{O}$, and an action history acthist.

A cursory examination of the definition of a feasible tc-$\mathcal{TS}$ (Definition 5.19) reveals that most of the conditions are simple checks that can be easily incorporated in the construction process. The main exception to this happy state of affairs is the first condition—that of closure under the rules in the TP program. This is because when a rule causes atoms to be added to $\rho(i)$, for some $i \geq \mathtt{t}_{\mathsf{now}}$, it may cause other rules to fire, which in turn may cause other atoms to be added to $\rho(i)$, for some $i \geq \mathtt{t}_{\mathsf{now}}$. This in turn may cause additional rules to fire, and so on.

We therefore take advantage of well-known methods from logic programming [Lloyd, 1987], and we construct a suitable monotone fixpoint operator and relate its least fixpoint $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ with feasible TP-status interpretations.

The iterative construction of this fixpoint is nothing but a mathematical description of the well known "loop" construct in programming languages. These methods allow us to mathematically model the transitive closure of forward chaining rules in an elegant way. For readers not familiar with this approach, we add some explanations. The fixpoint operator of Definition 6.3 below, when applied to a set $\mathsf{tc}\text{-}\mathcal{T}S$, gives us the result of simultaneously applying all the rules in our TP program once. Thus, to get the transitive closure, we have to iterate this operator. We are therefore interested in its least fixpoint, if it exists: this fixpoint would then constitute the transitive closure of all the rules. The existence of this fixpoint follows immediately, using the famous theorem of Knaster/Tarski [Tarski, 1955], because the operator itself will turn out to be monotone.

Thus, we first define a fixpoint operator. The definition uses the notion of modalities implying modalities defined as follows: $\mathbf{O}$ implies both $\mathbf{Do}$ and $\mathbf{P}$, $\mathbf{Do}$ implies $\mathbf{P}$ and all modalities imply themselves.

We emphasize the fact that we are only considering strict programs (so all probabilities in action status conditions are equal to 1). However, arbitrary probability intervals can occur in TP-CCC's.

We also assume in the following, w.l.o.g., that all intervals $\langle \mathsf{ti}, \delta \rangle$ are such that δ is nonzero in the whole interval (otherwise we can write it as a finite union of intervals in the required form).

Definition 6.3 ((Operator $\mathsf{D}_{\mathcal{TP}}$)) *Let $\mathcal{TPP}$ be a* TP *program, $\mathcal{O}$ a state and $\mathsf{tc}\text{-}\mathcal{T}S$ a set of temporally constrained* TP *status atoms. Then we define $\mathsf{D}_{\mathcal{TP}}(\mathsf{tc}\text{-}\mathcal{T}S)$ to be the following set of temporally constrained* TP *status atoms:*

$\{\mathsf{Op}'\,\alpha : [\mathsf{ti}'] \mid \mathsf{Op}\,\alpha : [\otimes, \langle \mathsf{ti}, \delta \rangle] \leftarrow tpccc_1 \wedge \ldots \wedge tpccc_n \wedge \varrho_1 : [\otimes, \langle \mathsf{ti}_1, \delta_1 \rangle] \wedge \ldots \wedge \varrho_m : [\otimes, \langle \mathsf{ti}_m, \delta_m \rangle]$
is a ground instance of a rule in $\mathcal{TPP}$ and
(I) for all $1 \leq i \leq n$***:*** $\mathcal{O} \models^{\mathsf{now}} tpccc_i$ ***and***
(II) for all $1 \leq i \leq m$*:*
If $\varrho_i = \mathsf{Op}_i \alpha_i$
then *there exists* $\mathsf{Op}'_i \alpha_{i_1} : [\mathsf{ti}'_i]$ *in* $\mathsf{tc}\text{-}\mathcal{T}S$ *s. t.:*
(1) Op'_i *implies* Op_i*,*
(2) $\mathsf{ti}'_i \subseteq \mathsf{ti}_i$ ***and***
If $\varrho_i = \mathsf{Op}_{i_1} \alpha_{i_1} \wedge \ldots \wedge \mathsf{Op}_{i_k} \alpha_{i_k}$
then *there exist* $\mathsf{t}_i \in \mathsf{ti}_i$
and for $1 \leq j \leq k$*:* $\mathsf{Op}'_{i_j} \alpha_{i_j} : [\mathsf{t_i}]$ *in* $\mathsf{tc}\text{-}\mathcal{T}S$
s. t. Op'_{i_j} *implies* Op_{i_j}***.and***
(III) $\mathsf{ti}' = [\mathsf{ti} \wedge t \geq \mathsf{t}_{now}]$ *and* Op *implies* Op'*.* $\}$

As the above definition is quite complex, we provide a simple example below to show how it works.

Example 6.4 ((Energy example: Operator $\mathsf{D}_{\mathcal{TP}}$)) *Suppose the generator's* TP *program, $\mathcal{TPP}$, contains the following rules:*

r1: $\mathbf{Do}\, Bid(c+5, bal)[\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+3], \delta_u\rangle] \leftarrow$
$\mathbf{in}(\mathtt{m}+2, \mathtt{energy} : demand(\mathtt{bal})) : [\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+1], \delta_u\rangle, [0.9, 1]]$

r2: $\mathbf{O}\, Produce(m, bal)[\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+2], \delta_u\rangle] \leftarrow$
$\mathbf{P}\, Bid(c+5, bal)[\otimes_{ig}, \langle[\mathtt{X}_{now}, \mathtt{X}_{now}+5], \delta_u\rangle]$

Suppose $\mathtt{t}_{now} = 3$ and the agent's state is as specified in example 5.11 and there are no integrity constraints.

$$\mathsf{D}_{\mathcal{TP}}(\emptyset) = \{ \begin{array}{l} \mathbf{P}\, Bid(c+5, bal)[3,6], \\ \mathbf{Do}\, Bid(c+5, bal)[3,6]\} \end{array}$$

In order to find a fixed point we need to iterate the operator. This is traditionally done by first starting out with the TP-interpretation that assigns the empty set to each time point, and then iteratively firing rules and adding to these sets. However, we do not start the operator by assigning $\emptyset$ to all time points t. This is because part of the TP-status interpretation we want to construct is already determined by acthist. Therefore we define tc-$\mathcal{TS}_{\text{start}}$ to be the set of singleton TP status atoms that corresponds to acthist. In some situations we will add to tc-$\mathcal{TS}_{\text{start}}$ status atoms that may belong to the feasible TP-status interpretation. We now define the iterations of $\mathsf{D}_{\mathcal{TP}}$.

Definition 6.5 ((Iterations of $\mathsf{D}_{\mathcal{TP}}$)) *Let $\mathcal{TPP}$ be a strict* TP *program, and $\mathcal{O}$ be an agent state. The* iterations *of $\mathsf{D}_{\mathcal{TP}}$ are defined as follows:*

$$\begin{aligned} \mathsf{D}_{\mathcal{TP}} \uparrow^0 &= \text{tc-}\mathcal{TS}_{start}. \\ \mathsf{D}_{\mathcal{TP}} \uparrow^{(j+1)} &= \mathsf{D}_{\mathcal{TP}}(\mathsf{D}_{\mathcal{TP}} \uparrow^j). \\ \mathsf{D}_{\mathcal{TP}} \uparrow^\omega &= \bigcup_j \mathsf{D}_{\mathcal{TP}} \uparrow^j . \end{aligned}$$

When tc-$\mathcal{TS}_{\text{start}}$ is not clear from the context, we will use the notation $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega (\text{tc-}\mathcal{TS}_{\text{start}})$ to explicitly specify it.

Example 6.6 ((Energy example: Iterations of $\mathsf{D}_{\mathcal{TP}}$)) *We continue with example 6.4 and assume that tc-$\mathcal{TS}_{start} = \emptyset$.*

- $\mathsf{D}_{\mathcal{TP}} \uparrow^0 = \emptyset$ *since* tc-$\mathcal{TS}_{start} = \emptyset$.
- $\mathsf{D}_{\mathcal{TP}} \uparrow^{(1)} = \mathsf{D}_{\mathcal{TP}}(\mathsf{D}_{\mathcal{TP}} \uparrow^0) = \mathsf{D}_{\mathcal{TP}}(\emptyset)$. *This was computed in Example 6.4:*

$$\mathsf{D}_{\mathcal{TP}} \uparrow^{(1)} = \{ \begin{array}{l} \mathbf{P}\, Bid(c+5, bal)[3,6], \\ \mathbf{Do}\, Bid(c+5, bal)[3,6]\} \end{array}$$

- *Applying the rules again we get:*

$$\begin{aligned} \mathsf{D}_{\mathcal{TP}} \uparrow^{(2)} \quad = \quad & \mathsf{D}_{\mathcal{TP}} \uparrow^{(1)} \cup \\ & \{\, \mathbf{O}\mathit{Produce}(m, \mathit{bal})[3,5], \\ & \ \mathbf{Do}\ \mathit{Produce}(m, \mathit{bal})[3,5], \\ & \ \mathbf{P}\mathit{Produce}(m, \mathit{bal})[3,5], \} \end{aligned}$$

- *For all $j > 2$, $\mathsf{D}_{\mathcal{TP}} \uparrow^{(j)} = \mathsf{D}_{\mathcal{TP}} \uparrow^{(2)}$.*
- *$\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega} = \mathsf{D}_{\mathcal{TP}} \uparrow^{(2)}$*

The following result tells us that $\mathsf{D}_{\mathcal{TP}}$ is monotone and continuous w.r.t. subset inclusion and hence, by the Tarski-Knaster theorem, is guaranteed to have a least fixpoint which equals $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$.

Theorem 6.7 ((Least fixpoint of $\mathsf{D}_{\mathcal{TP}}$)) *$\mathsf{D}_{\mathcal{TP}}$ is a monotone and continuous operator. Hence, $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ is its least fixpoint.*

Proof: We first show monotonicity:

$$\mathsf{tc}\text{-}\mathcal{T}S_1 \subseteq \mathsf{tc}\text{-}\mathcal{T}S_2 \Rightarrow \mathsf{D}_{\mathcal{TP}}(\mathsf{tc}\text{-}\mathcal{T}S_1) \subseteq \mathsf{D}_{\mathcal{TP}}(\mathsf{tc}\text{-}\mathcal{T}S_2).$$

Suppose $\mathsf{Op}\,\alpha : \mathsf{ti} \in \mathsf{D}_{\mathcal{TP}}(\mathsf{tc}\text{-}\mathcal{T}S_1)$. Then there is a rule in $\mathcal{TPP}$ having a ground instance of the form

$$\mathsf{Op}\,\alpha : \mathsf{ti} \ \leftarrow \ \ tpccc_1 \wedge \ldots \wedge tpccc_n \ \ \wedge \varrho_1 : \mathsf{ti}_1 \wedge \ldots \wedge \ \varrho_m : \mathsf{ti}_m$$

such that for all $1 \leq i \leq m$, there exist $\mathsf{Op}'_{i_j}\alpha_{i_j} : \mathsf{ti}_{i_j}$ in $\mathsf{tc}\text{-}\mathcal{T}S_1$ such that conditions (I)-(III) hold and ti has the required form. But in this case, as $\mathsf{tc}\text{-}\mathcal{T}S_1 \subseteq \mathsf{tc}\text{-}\mathcal{T}S_2$, all the $\mathsf{Op}'_{i_j}\alpha_{i_j} : \mathsf{ti}_{i_j}$ are also in $\mathsf{tc}\text{-}\mathcal{T}S_2$ and hence, the conditions for $\mathsf{Op}\,\alpha : \mathsf{ti} \in \mathsf{D}_{\mathcal{TP}}(\mathsf{tc}\text{-}\mathcal{T}S_2)$ hold.

We now consider continuity. Let $\mathsf{tc}\text{-}\mathcal{T}S_1 \subseteq \mathsf{tc}\text{-}\mathcal{T}S_2 \subseteq \cdots \subseteq \mathsf{tc}\text{-}\mathcal{T}S_n \subseteq \mathsf{tc}\text{-}\mathcal{T}S_{n+1} \subseteq \cdots$ be an ascending chain of temporally constrained temporal status sets. Then:

$$\mathsf{D}_{\mathcal{TP}}(\bigcup_i \mathsf{tc}\text{-}\mathcal{T}S_i) = \bigcup_i \mathsf{D}_{\mathcal{TP}}(\mathsf{tc}\text{-}\mathcal{T}S_i).$$

>From monotonicity of $\mathsf{D}_{\mathcal{TP}}$, it is immediate that

$$\mathsf{D}_{\mathcal{TP}}(\bigcup_i \mathsf{tc}\text{-}\mathcal{T}S_i) \supseteq \bigcup_i \mathsf{D}_{\mathcal{TP}}(\mathsf{tc}\text{-}\mathcal{T}S_i).$$

Hence, we only need to show that $\mathsf{D}_{\mathcal{TP}}(\bigcup_i \mathsf{tc}\text{-}\mathcal{T}S_i) \subseteq \bigcup_i \mathsf{D}_{\mathcal{TP}}(\mathsf{tc}\text{-}\mathcal{T}S_i)$. Suppose $\mathsf{Op}'\alpha : \mathsf{ti} \in \mathsf{D}_{\mathcal{TP}}(\bigcup_i \mathsf{tc}\text{-}\mathcal{T}S_i)$. Then there is a rule in $\mathcal{TPP}$ having a ground instance of the form

$$\mathsf{Op}\,\alpha : \mathsf{ti} \ \leftarrow \ \ tpccc_1 \wedge \ldots \wedge tpccc_n \ \ \wedge \varrho_1 : \mathsf{ti}_1 \wedge \ldots \wedge \ \varrho_m : \mathsf{ti}_m$$

such that for all $1 \leq i \leq m$, there exist $\mathsf{Op}'_{i_j}\alpha_{i_j} : \mathsf{ti}_{i_j}$ in $\bigcup_i$ tc-$\mathcal{T}S_i$ such that conditions (I)-(III) hold and ti has the required form. As m is finite, there must exist an integer r such that the above conditions are satisfied by tc-$\mathcal{T}S_r$. In this case, we have $\mathsf{Op}'_{i_j}\alpha_{i_j} : \mathsf{ti}_{i_j} \in \mathsf{D}_{\mathcal{TP}}$(tc-$\mathcal{T}S_r$) and we are done.

The statement is now an immediate consequence of the Tarski-Knaster theorem [Lloyd, 1987] that states that if f is a continuous function on a complete lattice, then $f \uparrow^\omega$ is the least fixpoint of f. Here, $\mathsf{D}_{\mathcal{TP}}$ is a continuous function, and the set of all TP status sets is a complete lattice under set inclusion. ∎

We are now ready to show that $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$ has the properties of TP deontic and action closure, and also that all feasible temporal probabilistic status sets must be compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$. These properties will later help us in computing feasible temporal probabilistic status sets.

Theorem 6.8 *Let $\mathcal{TPP}$ be a strict temporal probabilistic agent program, and $\mathcal{O}$ a state.*

1. *There is a* TPSI *ρ compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$ which is* TP *deontically closed and temporally action closed.*

2. *If ρ is a feasible* TPSI*, then it is compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$.*

Proof:

1. Let $X_{\mathbf{O}}$ be the set of all ti status atoms of the form $\mathbf{O}\alpha : \mathsf{ti}$ in $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$. Let $X_{\mathbf{P}}$ be the set of all ti status atoms of the form $\mathbf{P}\alpha : \mathsf{ti}$ in $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$. Each atom $\mathbf{O}\alpha : \mathsf{ti}$ in $X_{\mathbf{O}}$ must be in $\mathsf{D}_{\mathcal{TP}} \uparrow^j$ for some integer j, but then, as $\mathbf{O}$ implies $\mathbf{P}$, the same rule used to place $\mathbf{O}\alpha : \mathsf{ti}$ in $\mathsf{D}_{\mathcal{TP}} \uparrow^j$ must also have been used to insert $\mathbf{P}\alpha : \mathsf{ti}$ into $\mathsf{D}_{\mathcal{TP}} \uparrow^j$ which means $\mathbf{P}\alpha : \mathsf{ti} \in X_{\mathbf{P}}$. One may now construct a ti status set ρ' as follows: for each $\mathbf{O}\alpha : \mathsf{ti} \in \mathsf{D}_{\mathcal{TP}} \uparrow^\omega$, insert $\mathbf{O}\alpha, \mathbf{P}\alpha$ into $\rho'(j)$ where j is the smallest integer which is contained in ti. For all other modalities $\mathsf{Op} \neq \mathbf{O}$, if $\mathsf{Op}\alpha : \mathsf{ti}$ in $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$, insert $\mathsf{Op}\alpha$ into $\rho'(j)$ where j is the smallest integer which is in ti. It is easy to see that ρ' is compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$ and ρ' is temporally deontically consistent.

2. Similar to the proof of the previous item.

3. Suppose ρ is a feasible temporal status set which is not compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$. We will attempt to derive a contradiction. We call a ti status atom $\mathsf{Op}\alpha : \mathsf{ti}$ a *rogue* atom *iff* $\mathsf{Op}\alpha \notin \rho(i)$ for all i's that are in ti. Let Rogues be the set of all rogue atoms associated with $\rho(i)$, and let

$$j = \min\{r \mid \mathsf{Op}\alpha : \mathsf{ti} \in \mathsf{D}_{\mathcal{TP}} \uparrow^r \text{ and } \mathsf{Op}\alpha : \mathsf{ti} \in \text{Rogues}\}.$$

   We proceed by induction on j.

j=0: In the base case, we obtain an absurdity immediately as $\mathsf{D}_{\mathcal{TP}} \uparrow 0 = \emptyset$ and hence cannot contain any rogue atoms.

j=s+1: As no rogues occur in $\mathsf{D}_{\mathcal{TP}} \uparrow^s$, we know that ρ is compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^s$.As $\mathsf{Op}\alpha \in \mathsf{D}_{\mathcal{TP}} \uparrow^{s+1}$, there must exist a rule in $\mathcal{TPP}$ having a ground instance of the form shown in Definition 6.3 and satisfying the conditions stated there. Clearly, each interval ti status atom in the body of this rule is satisfied by ρ, i.e. for each ti status atom $\mathsf{Op}_i\alpha_i : \mathsf{ti}_i$ in the body of this rule, $rho(t_i)$ contains $\mathsf{Op}_i\alpha_i$ where $t_i \in \mathsf{ti}_i$. As ρ is feasible, by the "closure under program rules" condition in the definition of feasibility, it must satisfy the head of this rule, but to do this, it must satisfy the constraint attached to the rule head, which coincides with ti. This contradicts our assumption.

6.2 Feasible Temporal Probabilistic Interpretation Algorithm

Given the above theorem, it may seem that a TPSI that is compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^\omega$ is a good candidate for constructing a feasible TP-status interpretation. Thus, in order to find a feasible TP-status interpretation, we could:

1. first compute the least fixpoint of $\mathsf{D}_{\mathcal{TP}}$ and then
2. select from amongst the compatible TP-status interpretations, those that satisfy the other requirements for feasibility.

We follow this intuition in Algorithm 6.9 below. This algorithm uses a subroutine called **ComputeTPI** described later in Definition 6.11. Whenever an agent's state changes, this algorithm will be executed to find a new feasible TP-status interpretation. The agent then concurrently executes all actions α such that $\mathbf{Do}\,\alpha$ is in the computed feasible temporal probabilistic status interpretation at time $\mathsf{t_{now}}$ using the notion of concurrency described in [Subrahmanian et al., 2000] (these details are omitted as they are not relevant for the purposes of this paper).

The **FTPSS** algorithm terminates as soon as a feasible TP-interpretation is found. It maintains a set, $Seen$, of compatible TP interpretations seen thus far—if the algorithm is "still running" this means that none of the compatible TP interpretations examined thus far is feasible, and hence, we need to continue trying to find a new compatible TP interpretation that is feasible. More precisely, the algorithm works by:

(1) Iteratively modifying a set tc-$\mathcal{TS}$ This is a set of temporally constrained TP atoms. Initially, it is determined by $\mathsf{acthist}$ and set to

$$\mathsf{tc\text{-}}\mathcal{TS}_{\text{start}} := \bigcup_{\{i \text{ s.t. } \mathsf{acthist}(i) \text{ is defined}\}} \{\mathsf{Op}\alpha[i] \mid \mathsf{Op}\alpha \in \mathsf{acthist}(i)\},$$

in accordance with Theorem 6.8.

(2) Checking for feasibility Once the **ComputeTPI** procedure has generated a new candidate (to be described below), this set is checked for feasibility. If it is feasible, a solution has been found. If not, this set is added to the set $Seen$ and the procedure goes on.

Algorithm 6.9 ((Feasible Temporal Probabilistic Status Set Computation))

FTPSS$(\mathcal{TPP}$***,acthist,***$\mathcal{O})$

(⋆ ***Input:*** *(1) a strict* $\mathcal{TPP}$, ⋆)
(⋆ *(2) the history* acthist*, and* ⋆)
(⋆ *(3) the state* $\mathcal{O}$ ⋆)
(⋆ ***Output:*** *(1) a feasible temporal probabilistic status set, if one exists* ⋆)
(⋆ *(2) "No" otherwise* ⋆)

1. tc-$\mathcal{TS}_{new} = D_{\mathcal{TP}} \uparrow^{\omega}$ (tc-$\mathcal{TS}_{start}$).
2. *done* := **false**;
3. $Seen := \emptyset$;
4. ***while*** ¬*done* ***do***
   (a) TPSI := ***ComputeTPI***(tc-$\mathcal{TS}_{new}$,$\mathcal{TPP}$, $\mathcal{O}$,$Seen$);
   (b) ***if*** TPSI = *"No"* ***then return*** *"No"*.
   (c) ***if FeasTPI***$(Singl(\text{TPSI}))$ ***then*** *done* := **true** ***else*** $Seen := Seen \cup \{\text{TPSI}\}$;
5. ***return*** $Singl(\text{TPSI})$.

Before turning to the **ComputeTPI** procedure, we describe the feasibility check in more detail.

Lemma 6.10 ((Feasibility Check FeasTPI)) *Given a finite set of singleton* TP *status atoms* tc-$\mathcal{TS}$ *that is closed under the program rules of* $\mathcal{TPP}$*, it is possible to check whether it is feasible. We will therefore assume that there is a* ***FeasTPI*** *algorithm that checks whether* tc-$\mathcal{TS}$ *is*

1. TP *deontically consistent at time* $\mathtt{t}_{now}$,
2. TP *deontically and action closed at time* $\mathtt{t}_{now}$,
3. TP *state consistent at time* $\mathtt{t}_{now}$.

It returns **true** *if all these requirements are met—otherwise it returns* false.

Proof: It is possible to develop such an algorithm since we require TPSI's to be finite. Thus, we only need to check these conditions for finitely many time points t. Note that we do not have to check for history compatibility (as required in Definition A.7) nor for closure under the rules of $\mathcal{TPP}$: these conditions are guaranteed by the **ComputeTPI** procedure (to be explained below). ∎

Before stating our main procedure **ComputeTPI** in full detail, we first describe the required input-output behaviour to ensure that Algorithm 6.9 is correct and complete.

Definition 6.11 ((Input and Output of ComputeTPI))
*The **ComputeTPI** function takes as input (1) a set* tc-$\mathcal{TS}$ *of temporally constrained* TP *status atoms, (2) a strict temporal probabilistic program $\mathcal{TPP}$, (3) a state $\mathcal{O}$ and (4) a set $Seen$ of* TP*-status interpretations which are closed under the rules of $\mathcal{TPP}$.*

It either returns a TP *status interpretation closed under the rules of $\mathcal{TPP}$, compatible with* tc-$\mathcal{TS}$*, and different from the sets in $Seen$, if such a set exists, or "No" (if no such* TP *status set exists).*

Thus, we are confronted with the problem of implementing **ComputeTPI**. In order to do so, we need to construct a TP status set TPSI which is compatible with tc-$\mathcal{TS}$ and which is closed under the rules of $\mathcal{TPP}$.

The reader may wonder why we cannot simply take the least fixpoint $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ (tc-$\mathcal{TS}_{\text{start}}$) for this purpose. Unfortunately, we cannot do so because $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ (tc-$\mathcal{TS}_{\text{start}}$) is in general not solution-closed (as defined below).

Definition 6.12 ((Solution Closed)) *A set F of temporally constrained* TP *status atoms is said to be* solution-closed *iff*

for all Opα : ti $\in F$*: there is* $i \in$ ti *and* Op$\alpha : [i] \in F$

For example, $\mathbf{Do}\,\alpha_1[\mathsf{ti}_1]$ and $\mathbf{Do}\,\alpha_2[\mathsf{ti}_2]$ may both be such that no status atom of the form $\mathbf{Do}\,\alpha_1[\mathsf{t}_1]$ and $\mathbf{Do}\,\alpha_2[\mathsf{t}_2]$ are present in tc-$\mathcal{TS}_{new}$ where $\mathsf{t}_1, \mathsf{t}_2$ are contained in $\mathsf{ti}_1, \mathsf{ti}_2$ respectively. $\mathsf{ti}_1, \mathsf{ti}_2$ may have lots of solutions but we can pick a *hitting set* of their set of solutions and add those temporally constrained TP status atoms to tc-$\mathcal{TS}_{new}$ to remove the "reasons" for solution closure to fail. This process may in turn cause new status atoms to be derivable (i.e. tc-$\mathcal{TS}_{new}$ may not be closed under program rules after the addition of these atoms) and hence the process must be repeated.

Definition 6.13 ((TP Hitting Set)) *Suppose tc-$\mathcal{TS}$ is a set of temporally constrained* TP *status atoms. A* TP hitting set*, H, for tc-$\mathcal{TS}$ is a minimal set of singleton ground* TP *status atoms of the form* Op$\,\alpha : [i]$ *such that:*

> *For every* Op$\,\alpha :$ ti $\in$ tc-$\mathcal{TS}$*, there is an* TP *atom of the form* Op$\,\alpha : [i]$ *in H such that $i \in$* ti*, and if $i <$* t_{now}*, then* Op$\,\alpha \in$ acthist(i).

We use $\mathsf{chs}(\mathsf{tc}\text{-}\mathcal{T}S)$ *to denote the set of all* TP *hitting sets for* $\mathsf{tc}\text{-}\mathcal{T}S$.

We will use a subroutine called $find\_member\_chs(\mathsf{tc}\text{-}\mathcal{T}S)$ which finds a member of $\mathsf{chs}(\mathsf{tc}\text{-}\mathcal{T}S)$ that is not a subset of $\mathsf{tc}\text{-}\mathcal{T}S$. If no such element exists, it returns "No". We do not specify the implementation of this algorithm as it can be easily implemented (using standard hitting set algorithms [Cormen et al., 1989]).

Unfortunately, blindly adding elements of a hitting set to the current candidate set may destroy the closure under the rules condition: this is because there may be more TP atoms available and thus more rules could fire and entail still more new TP atoms. This problem can be taken care of by adding these atoms to the program and getting a new program $\mathcal{TPP}_{new}$ (see *(2)(d)(ii)(A)* in Algorithm 6.14). We then apply our operator $\mathsf{D}_{\mathcal{TP}}$ to $\mathcal{TPP}_{new}$ (see *(2)(d)(ii)(B)* in Algorithm 6.14). Note that new atoms which violate the solution-closed requirement may still be generated. We repeat this process until either

1. all TP atoms have a solution in the current H^* (see *(2)(d)(ii)(C)* in Algorithm 6.14), or
2. we reach a fixpoint H^*. This fixpoint yields a better candidate $\mathsf{tc}\text{-}\mathcal{T}S_{new}$ and we have to re-iterate the whole process, by first computing a hitting set of $\mathsf{tc}\text{-}\mathcal{T}S_{new}$ and then computing the iterations of our operator $\mathsf{D}_{\mathcal{TP}}$.

Algorithm 6.14 (ComputeTPI($\mathsf{tc}\text{-}\mathcal{T}S, \mathcal{TPP}, \mathcal{O}, Seen$))
ComputeTPI$(\mathsf{tc}\text{-}\mathcal{T}S, \mathcal{TPP}, \mathcal{O}, Seen)$

(⋆ ***Input****: (1) a set of temporally constrained* TP *status atoms* $\mathsf{tc}\text{-}\mathcal{T}S$ *⋆)*
(⋆ (2) a strict $\mathcal{TPP}$*, (3) the state* $\mathcal{O}$ *and ⋆)*
(⋆ (4) a set $Seen$ *of* TP *status sets, ⋆)*
(⋆ ***Output****: (1) a compatible* TP *status set not in* $Seen$ *which is ⋆)*
(⋆ closed under the rules of $\mathcal{TPP}$ *(if one exists) ⋆)*
(⋆ (2) "No" otherwise ⋆)

1. *done* $:=$ **false***; found* $:=$ **false***;* $Loc\_Seen := Seen$*;*
   $\mathsf{tc}\text{-}\mathcal{T}S_{new} := \mathsf{tc}\text{-}\mathcal{T}S$*;* $H^* := \emptyset$*, done_inner* $:=$ **false***;*
2. ***while*** $\neg done \wedge \neg found$ ***do***
   (a) ***if*** *done_inner* ***then***
      i. $H = find\_member\_chs(\mathsf{tc}\text{-}\mathcal{T}S, Loc\_Seen)$*;*
      ii. ***if*** $H =$ *"No"* ***then*** *done*$:=$ **true***;*
      iii. *done_inner*$:=$ **false**
   (b) ***else***
      i. $H = find\_member\_chs(\mathsf{tc}\text{-}\mathcal{T}S_{new}, Loc\_Seen)$*;*

ii. ***if*** $H =$ *"No"* ***then*** *done_inner*$= \mathbf{true}$*;*

(c) $Loc\_Seen := Loc\_Seen \cup \{H\}$*;*

(d) ***if*** $H \neq$ *"No"* ***then***

i. $H^\star = H$*; changed* $= \mathbf{true}$*;*

ii. ***while*** $\neg found \wedge changed$ ***do***

A. $\mathcal{TPP}_{new} = \mathcal{TPP} \cup H^\star$*;* $oldH^\star = H^\star$*;*

B. $H^\star = \mathsf{D}_{\mathcal{TPP}_{new}}$*;*

C. ***if*** $H^\star \notin Loc\_Seen \wedge H^\star$ *is solution closed* ***then*** *found* = $\mathbf{true}$ ***else*** *changed* $= (oldH^\star \neq H^\star)$*;*

iii. $Loc\_Seen := Loc\_Seen \cup \{H^*\}$*;* tc-$\mathcal{TS}_{new} := H^*$

3. ***if*** $found$ ***then*** *return* $H^\star$ ***else*** *return "No".*

The following lemma states that the above implementation of Algorithm 6.14 satisfies the output conditions of Definition 6.11.

Lemma 6.15 *Suppose the* $find\_member\_chs(\text{tc-}\mathcal{TS}_{new}, Loc\_Seen)$ *algorithm is correctly implemented. Then:*

1. *If algorithm* ***ComputeTPI*** *returns a temporal status set* H^**, then* H^* *satisfies the output conditions of Definition 6.11.*
2. *If algorithm* ***ComputeTPI*** *returns "No", then there is no temporal status set satisfying the output conditions of Definition 6.11.*

Proof: There are two while loops in the algorithm above. The outer loop (step 2 of the algorithm), considers the possible hitting sets of the original tc-$\mathcal{TS}$. The variable "done" is false as long as not all the hitting sets were considered.

For each hitting set of the original tc-$\mathcal{TS}$, the inner while loop (*2(d)ii* of the algorithm), tries to find a solution-closed superset H^* of the hitting set. This is done using $\mathsf{D}_{\mathcal{TP}}$ after adding the hitting set to $\mathcal{TPP}$ and assigning it to $\mathcal{TPP}_{new}$ (*2(d)iiB*). It iteratively adds the result of applying $\mathsf{D}_{\mathcal{TP}}$ to $\mathcal{TPP}$ (*2(d)iiA*). When H^* cannot be enlarged any more and still a solution wasn't found, a hitting set of H^* (which is assigned to tc-$\mathcal{TS}_{new}$ (*2(d)iii*)) is found (*2(b)i*) and the process continues until success or until it is clear that the chosen H can't be extended any more. ∎

Theorem 6.16 ((Algorithm 6.9 is Correct and Complete))
Algorithm 6.9 generates a feasible TP *status set (if one exists).*

Proof: Suppose Algorithm 6.9 returns tc-$\mathcal{TS}$. In this case, tc-$\mathcal{TS}$ is compatible via step *(4)(a)*, and feasible via step *(4)(c)*.

Conversely, suppose Algorithm 6.9 returns "No". In this case, we know that **ComputeTPI** returned "No" which means that it was unable to find a TP status set compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$. But this means that all TP status set compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ are in $Seen$ which means none of them is feasible. ∎

7 Optimal Feasible Temporal Probabilistic Status Sets

As the reader has undoubtedly noticed by now, in a given state, an agent can have zero, one, or many FTPSI's. The agent is required to choose one and act according to the status atoms in that FTPSI. Thus far, we have proceeded under the assumption that the agent will arbitrarily choose one. In this section, we suggest that the agent choose one based on an objective function.

Definition 7.1 ((Objective function (objf))) *Suppose* $\mathfrak{a}$ *is a given agent. An objective function* objf *for* $\mathfrak{a}$ *is a mapping that takes as input, a state (for agent* $\mathfrak{a}$*) and an* FTPSI*, and returns as output, a non-negative real number.*

Intuitively, we will think of an objective function as assigning a *cost* to the choice of a given FTPSI. This cost has two components—the actions in FTPSI and the undesirability of the state that results if we choose to act in accordance with the FTPSI in question.

Definition 7.2 ((Optimal FTPSI)) *Suppose* $\mathfrak{a}$ *is a given agent and* objf *is an objective function for* $\mathfrak{a}$*. Given any* FTPSI S *w.r.t. agent* $\mathfrak{a}$ *and state* $\mathcal{O}$*, we use* ***result***$(S, \mathcal{O})$ *to denote the new state that results by executing all the* **Do** *actions in* $S(\mathsf{t}_{now})$ *w.r.t. the current state* $\mathcal{O}$*.*

An FTPSI S *is* optimal *w.r.t. a given agent state* $\mathcal{O}$ *and* objf *iff there is no* other FTPSI S' *for agent* $\mathfrak{a}$ *in state* $\mathcal{O}$ *such that* objf(***result***$(S', \mathcal{O}), S') <$ objf(***result***$(S, \mathcal{O}), S)$.

Intuitively, as we are thinking of minimising the cost associated with the choice of an FTPSS over other possible FTPSS's, the above definition requires us to minimise objf.

The following algorithm modifies the **FTPSS** algorithm so as to compute an optimal feasible temporal probabilistic status interpretation w.r.t. a given agent and a given objective function.

Algorithm 7.3 ((Optimal Feasible Temporal Probabilistic Status Int. Computation))
OptFTPSS$(\mathcal{TPP}$***,acthist,***$\mathcal{O}$***,objf)***

(⋆ ***Input:*** *(1) a strict* $\mathcal{TPP}$*, ⋆)*
(⋆ (2) the history acthist*, ⋆)*
(⋆ (3) the state $\mathcal{O}$*, and ⋆)*

(⋆ (4) objective function objf *⋆)*
(⋆ ***Output:*** *(1) an optimal* FTPSI *if one exists ⋆)*
(⋆ (2) "No" otherwise ⋆)

1. tc-$\mathcal{TS}_{new}$ = $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ (tc-$\mathcal{TS}_{start}$).
2. *done* := **false**;
3. $Seen := \emptyset$;
4. $bestSol = NIL$; $bestCost = \infty$;
5. ***while*** ¬*done* ***do***
   (a) TPSI := ***ComputeTPI***(tc-$\mathcal{TS}_{new}$,$\mathcal{TP}$, $\mathcal{O}$,$Seen$);
   (b) ***if*** TPSI = *"No"* ***then*** *done* := **true**;
   (c) ***else***
      i. $Seen := Seen \cup \{\text{TPSI}\}$;
      ii. ***if FeasTPI***$(Singl(\text{TPSI}))$ ***then***
         A. ***if*** objf(***result***$(\mathcal{O}, \text{TPSI})) <$ objf(***result***$(\mathcal{O}, bestSol))$ ***then***
            - $bestSol :=$ TPSI
            - $bestCost :=$ objf(***result***$(\mathcal{O}, \text{TPSI})$

return $Singl(\text{TPSI})$.

Theorem 7.4 ((Correctness of OptFTPSS)) *Suppose an agent* a *has a temporal probabilistic agent program* $\mathcal{TP}$*, action history* acthist *and current state* $\mathcal{O}$*. Suppose* objf *is an objective function Then:*

1. *If* a *has at least one* FTPSI*, then algorithm* ***OptFTPSS***$(\mathcal{TP}, \mathsf{acthist}, \mathcal{O}, \mathsf{objf})$ *will find an optimal* FTPSI *wrt.* objf*.*
2. *If* ***OptFTPSS***$(\mathcal{TP}, \mathsf{acthist}, \mathcal{O}, \mathsf{objf})$ *returns "No" then the agent has no* FTPSI *in the current state.*

Proof: (1) Suppose a has at least one FTPSI. Every FTPSI will be eventually generated by Step 5(a) via the **ComputeTPI** invocation. When a feasible FTPSI is found, step 5(c)(i)(A) checks if the value of the objective function on this FTPSI is less than that of the current best solution. If so, it updates both the bestCost variable and the bestSol value. Otherwise this FTPSI is not as good as the current best solution. Hence, when the algorithm terminates in Step 5(b), all FTPSI's have been examined and the bestSol variable has the best FTPSI.

(2) Immediate from the above. If "NIL" is returned, this means that all FTPSI's compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ (tc-$\mathcal{TS}_{\text{start}}$) were examined and none was found to be feasible. ∎

8 Related Work

The work reported in this paper focuses on decision making in the presence of uncertainty in time. Over the years, there has been extensive work on uncertainty management and temporal reasoning. To put our work in the correct context, we list below various issues associated with time and uncertainty that were considered in previous research. To demonstrate the main issue of each category, we discuss shortly a few of the works that belong to it. Several of the works mentioned below consider more than one issue. We categorise a paper according to the main issue it considered. The main issue of our paper is the development of agent mechanisms to determine how to take actions now or in the future, and thus it belongs to the last category in the list. However, in each category we will mention how we addressed the category's issue in the context of building our mechanism for agent decision making.

Reasoning about the interactions between time and uncertainty

For example, [Lehmann and Shelah, 1982] developed a probabilistic temporal logic. [Dubois et al., 1991] have studied the integration of uncertainty and time – they extend the well-known possibilistic logic theory [Dubois and Prade, 1994, Dubois et al., 1991, Dubois and Prade, 1989] to a "timed possibilistic logic".

[Halpern and Tuttle, 1992] study the semantics of reasoning about distributed systems when uncertainty is present. They develop a logic where a process has knowledge about the probability of events which facilitates decision-making by the process.

[Kifer and Subrahmanian, 1992] show how uncertainty (including point, as well as interval based fuzzy logics) and time can be integrated via their annotated logics. In particular, they establish that various forms of temporal reasoning due to Shoham [Shoham, 1988] can be captured in their framework.

The basic blocks provided to our agent to reason on time and probability is the temporal probabilistic code calls. The advantage of using this technique is the ability to build agents on top of legacy code.

Reasoning about the interactions between time and beliefs

For example, [Fagin et al., 1995, Fagin et al., 1990] proposed modal logics of time and beliefs that can be used to model the behaviour of different types of distributed systems, e.g., systems that have perfect recall about the past, and those that have bounded recall.

[Thomas et al., 1991] have developed a framework for integrating beliefs, time, commitment, desires and multiple agents.

[Gmytrasiewicz and Durfee, 1992] have developed a logic of knowledge and belief to model multiagent coordination. Their framework permits an agent to reason not only about the world and its own actions, but also to simulate and

model the behaviour of other agents in the environment. In a separate paper [Gmytrasiewicz et al., 1991], they show how one agent can reason with a probabilistic view of the behaviour of other agents so as to achieve coordination.

[Friedman and Halpern, 1997] introduced a semantic based framework to model belief change. This framework combines temporal and epistemic modalities with a notion of plausibility, allowing to examine the changes of beliefs over time. They show belief revision and belief update fit into their framework [Friedman and Halpern, 1999].

In this paper we do not model the beliefs of an agent, but rather provides the agent with a mechanism to use probabilistic estimations of the legacy code the agent is built on top off. Updates are made to the data, which in turn yields new estimations.

Reasoning about time, action and change

For example, [E. Sandewall, 1994] developed techniques based on the concept of fluents for reasoning about actions and change. In his framework, existing and new logics can be described, compared and analyzed.

Karlsson and his colleagues that were inspired and motivated by Sandewall's work developed a class of narrative-based temporal action logics (TAL) [Doherty et al., 1998].

Dean and Kanazawa have studied the integration of probability and time with a view to developing efficient planning techniques [Dean and Kanazawa, 1988, Kanazawa, 1991]. Their main interest is in how probabilities of facts and events change over time. Similarly, [Haddawy, 1991, Haddawy et al., 1995] develops a logic for reasoning about actions, probabilities, and time and use it as a basis foe efficient planning.

[Hanks and McDermott, 1994] presented a system that uses probabilistic model to reason about the effects of agent's proposed actions. The system is able to answer questions of the form "is the probability that ϕ will hold in the world at time t is greater than r?"

[Baral et al., 2002] presented a language to reason about actions in a probabilistic setting. The main feature of their model is its use of static and dynamic causal laws, and use of unknown (or background) variables – whose values are determined by factors beyond their model.

We adopted *IMPACT*'s approach to model actions in which their is no uncertainty with respect to how an action will change the agent's state. It is specified by the add and delete lists and the action's code. However, we extended the add and the delete lists of an action to include temporal probabilistic code calls. We do not consider planning in this paper, but rather focuses on the agents programming and agents' decision making. However, our framework could be used to develop agents that can do planning as is demonstrated in [Dix et al., 2003].

Analyzing and modelling uncertain temporal data

This research includes techniques such as Kalman filtering that addresses the question of how does one update a "best" estimate for the state of a system as new, but still inaccurate, data arrived [Scientific, 2003]. It also includes techniques for modelling the evolution of a system from data such as time series analysis and forecasting techniques [Arsham, 2003].

We do not consider the problem of data analysis, but allow agents to be built on top of any software modules that perform such analysis.

Building action policies across time in the presence of uncertainty

The main goal of our paper has been to develop a framework that provides an agent with mechanisms to determine how to take actions now or in the future. The decision is based on the uncertain data the agent currently has access to via zero or more legacy data sources and based on leveraging existing software programs. The decision of what to do may change as the data changes based on external events. Thus, in the rest of this section we will compare our work with other works that belong to this category.

The work on *MetaTem* [Barringer et al., 1990] and its successor, *Concurrent MetaTem* [Fisher, 1994] are closely related to our work as they developed a logical framework to provide agents with temporal rules to decide on what to do. No probabilities are processed in their work—when probabilities are ignored, the relationship of their work to our framework has been cleanly described in [Dix et al., 2001].

The Independent Choice Logic (*ICL*) of Poole [Poole, 1997] is another very interesting approach. ICL is a semantic framework that allows for independent choices and a logic program that gives the consequence of choices. Various techniques such as influence diagrams and structured Markov decision processes can be embedded in ICL. However, the goal of ICL, as explicitly stated by Poole, is to be used as a specification for agents that act in a world and make observations of that world and as a modelling tool for dynamic environments with uncertainty. It does not discuss the computation issues related to building agents nor provides tools to be used by agents acting in uncertain environments.

MDP's can be applied for the construction of optimal or approximately optimal policies under uncertainty. In order to model a problem using *MDP*'s there is a need to define: (i) a state space; (ii) a set of possible actions (iii) a state transaction (iii) in case of POMDPs–observations (iv) a reward function.

[Boutilier et al., 1999] describe a number of ways in which intensional representations can be exploited to solve *MDP*'s effectively without enumeration all of the state space (e.g., the structured policy iteration algorithm [Boutilier et al., 1995], or explicitly specifying all the transaction and reward functions. However, even when using these methods, the modelling process is highly time consuming. Using our approach a system designer does not need to model the agent state. In our framework, the state of an agent consists of whatever data is stored inside the data structures of the code on top of

which the agent is built. Prediction about the future could be done using any available software packages. Instead of specifying a transaction function, in our framework the designer provides the agent with temporal probabilistic rules. If necessary, the designer can provide the agent with an objective function that it will use when choosing between several possible set of actions that are supported by the rules.

[Boutilier et al., 2000] propose the *DTGolog* model for robot programming on top of *MDP*'s. *DTGolog* allows the agents designer to partially specify a control program in a high-level, logical language. This designer's program directs the search of the *MDP* module. The joint goal of our paper and that of the Boutilier et al. is to provide agent's designer with a suitably high-level language to program its agent while allowing the agent some latitude in choosing its actions in real time. However, while their agent can be built only on top of *MDP*'s, we provide a framework to build agents on top of any legacy code.

Finally, [Dix et al., 2000] developed an extension of *IMPACT* to make decisions in the presence of uncertainty alone, and [Dix et al., 2001] extend agent programs to handle purely temporal reasoning. This paper extends the above papers in the following significant ways.

1. There is a great difference between the syntax of both [Dix et al., 2000] and [Dix et al., 2001] and this paper. The notion of a TP-annotation provided in this paper is new, as are the definitions of TP-CCC's and TP-ASC's. This in turn causes the syntax of TP agent programs to be very different from those in both [Dix et al., 2000, Dix et al., 2001].

2. As a consequence, the semantics of HTP agents uses structures that are very different from those in either of the above two papers.

3. Our use of a fixpoint operator to characterise the semantics of TP agents is derived from a long body of work in logic programming. The fact that a fixpoint operator is used in this paper provides a superficial similarity to the papers [Dix et al., 2001, Dix et al., 2000]. However, as the syntax and semantical structures s of HTP agents are different from the frameworks in [Dix et al., 2000, Dix et al., 2001], it follows that our fixpoint operator is also very different. As a consequence, the proofs of our main theorems are very different from [Dix et al., 2001, Dix et al., 2000], although the statements in the main theorems and lemmas resemble those in classical logic programming.

4. We have also introduced in this paper, the notion of *optimal* feasible temporal probabilistic status sets, a notion that is not dealt with in [Dix et al., 2001, Dix et al., 2000].

References

[Arsham, 2003] Arsham, H. (2003). Time Series Analysis and Forecasting Techniques. http://obelia.jde.aca.mmu.ac.uk/resdesgn/arsham/opre330Forecast.htm.

[Baral et al., 2002] Baral, C., Tran, N., and Tuan, L. (2002). Reasoning about actions in a probabilistic setting. In *Proc. of AAAI'02*, pages 507–512, Edmonton, Alberta, Canada. AAAI Press.

[Barringer et al., 1990] Barringer, H., Fisher, M., Gabbay, D., Gough, G., and Owens, R. (1990). METATEM: A framework for programming in temporal logic. In de Bakker, J. W., de Roever, W. P., and Rozenberg, G., editors, *REX Workshop*, volume 430 of *Lecture Notes in Computer Science*. Springer.

[Boole, 1854] Boole, G. (1854). *The Laws of Thought*. Macmillan, London.

[Boutilier et al., 1999] Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

[Boutilier et al., 1995] Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting structure in policy construction. In Mellish, C., editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, San Francisco. Morgan Kaufmann.

[Boutilier et al., 2000] Boutilier, C., Reiter, R., Soutchanski, M., and Thrun, S. (2000). Decision-theoretic, high-level agent programming in the, situation calculus. In *Proc. of AAAI-00*, pages 355–362.

[Cattell, R. G. G., et al., 1997] Cattell, R. G. G., et al. (1997). *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Mareo, California.

[Chalupsky et al., 2001] Chalupsky, H., Gil, Y., Knoblock, C., Lerman, K., Oh, J., Pynadath, D., Russ, T., and Tambe, M. (2001). Electric elves: Applying agent technology to support human organizations. In Hirsh, H. and Chien, S., editors, *IAAI*. AAAI.

[Cormen et al., 1989] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1989). *Introduction to Algorithms*. MIT Press and McGraw-Hill Book, Cambridge, Mass.

[Dean and Kanazawa, 1988] Dean, T. and Kanazawa, K. (1988). Probabilistic Temporal Reasoning. In *Proceedings AAAI*, pages 524–529, St. Paul, MN, USA. AAAI Press / The MIT Press.

[Dix et al., 2001] Dix, J., Kraus, S., and Subrahmanian, V. (2001). Temporal agent reasoning. *Artificial Intelligence*, 127(1):87–135.

[Dix et al., 2003] Dix, J., Munoz-Avila, H., Nau, D., and Zhang, L. (2003). IMPACTing SHOP: Putting an AI planner into a Multi-Agent Environment. *Annals of Mathematics and AI*, 4(37):381–407.

[Dix et al., 2000] Dix, J., Nanni, M., and Subrahmanian, V. (2000). Probabilistic agent reasoning. *ACM Transactions of Computational Logic*, 1(2):201–245.

[Doherty et al., 1998] Doherty, P., Gustafsson, J., Karlsson, L., and Kvarnstrom, J. (1998). (TAL) temporal action logics: Language specification and tutorial. *Electronic Transactions on Artificial Intelligence*, 2:3-4:273–306.

[Dubois et al., 1991] Dubois, D., Land, J., and Prade, H. (1991). Towards Possibilistic Logic Programming. In *Proceedings of the Eigth International Conference on Logic Programming*, pages 581–595, Paris, France. MIT Press.

[Dubois and Prade, 1989] Dubois, D. and Prade, H. (1989). Processing Fuzzy Temporal Knowledge. *IEEE Transactions on Systems, Man and Cybernetics*, 19(4):729–744.

[Dubois and Prade, 1994] Dubois, D. and Prade, H. (1994). Possibilistic logic. In Gabbay, D., Hogger, C., and Robinson, J., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming Vol. 3, Nonmonotonic and Uncertain Reasoning*, pages 439–513. Oxford University Press.

[E. Sandewall, 1994] E. Sandewall (1994). Features and Fluents: The Representation of Knowledge about Dynamical Systems.

[Eiter et al., 2001] Eiter, T., Lukasiewicz, T., and Walter, M. (2001). A Data Model and Algebra for Probabilistic Complex Values. *Annals of Mathematics and Artificial Intelligence*, 33(2-4):205–252.

[Eiter et al., 1999] Eiter, T., Subrahmanian, V., and Pick, G. (1999). Heterogeneous Active Agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255.

[Eiter et al., 2000] Eiter, T., Subrahmanian, V., and Rogers, T. (2000). Heterogeneous Active Agents, III: Polynomially Implementable Agents. *Artificial Intelligence*, 117(1):107–167.

[Fagin et al., 1995] Fagin, R., Halpern, J., Moses, Y., and Vardi, M. (1995). *Reasoning about Knowledge*. MIT Press, Cambridge, Massachusetts. 2nd printing.

[Fagin et al., 1990] Fagin, R., Halpern, J. Y., and Megiddo, N. (1990). A logic for reasoning about probabilities. *Information and Computation*, 87(1/2):78–128.

[Fisher, 1994] Fisher, M. (1994). A survey of Concurrent METATEM —, the language and its applications. In Gabbay, D. M. and Ohlbach, H. J., editors, *Temporal Logic — Proceedings of the, First International Conference*, volume 827 of *Lecture Notes in Computer Science*. Springer.

[Friedman and Halpern, 1997] Friedman, N. and Halpern, J. Y. (1997). Modeling belief in dynamic systems, part I: Foundations. *Artificial Intelligence Journal*, 95(2):257–316.

[Friedman and Halpern, 1999] Friedman, N. and Halpern, J. Y. (1999). Modeling belief in dynamic systems, part II: Revision and update. *Journal of Artificial Intelligence Research*, 10:117–167.

[Gmytrasiewicz and Durfee, 1992] Gmytrasiewicz, P. and Durfee, E. (1992). A Logic of Knowledge and Belief for Recursive Modeling. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 628–634, San Jose, CA. AAAI Press/MIT Press.

[Gmytrasiewicz et al., 1991] Gmytrasiewicz, P., Durfee, E., and Wehe., D. (1991). A Decision-Theoretic Approach to Coordinating Multiagent Interactions. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 62–68, Sydney, Australia. Morgan Kaufmann.

[Haddawy, 1991] Haddawy, P. (1991). *Representing Plans under Uncertainty: A Logic of Time, Chance and Action*. PhD thesis, University of Illinois. Technical Report UIUCDCS-R-91-1719.

[Haddawy et al., 1995] Haddawy, P., Doan, A., and Goodwin, R. (1995). Efficient Decision-Theoretic Planning: Techniques and Empirical Analysis. In Besnard, P. and Hanks, S., editors, *UAI*, pages 229–236. Morgan Kaufmann.

[Halpern and Tuttle, 1992] Halpern, J. Y. and Tuttle, M. (1992). Knowledge, Probability and Adversaries. Technical report, IBM. IBM Research Report.

[Hanks and McDermott, 1994] Hanks, S. and McDermott, D. (1994). Modeling a dynamic and uncertain world I: Symbolic and, probabilistic reasoning about change. *Artificial Intelligence*, 65(2):1–55.

[Kanazawa, 1991] Kanazawa, K. (1991). A Logic and Time Nets for Probabilistic Inference. In *Proceedings AAAI-91*, pages 360–365, Anaheim. AAAI Press / The MIT Press.

[Kifer and Subrahmanian, 1992] Kifer, M. and Subrahmanian, V. (1992). Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming*, 12(4):335–368.

[Lakshmanan et al., 1997] Lakshmanan, V., Leone, N., Ross, R., and Subrahmanian, V. (1997). ProbView: A Flexible Probabilistic Database System. *ACM Transactions on Database Systems*, 22(3):419–469.

[Lehmann and Shelah, 1982] Lehmann, D. and Shelah, S. (1982). Reasoning with time and chance. *Information and Control*, 53:165–198.

[Lloyd, 1987] Lloyd, J. (1984, 1987). *Foundations of Logic Programming*. Springer-Verlag, Berlin, Germany.

[Mittu and Ross, 2003] Mittu, R. and Ross, R. (2003). Building upon the coalitions agent experiment (coax) - integration of multimedia information in gccs-m using impact. In *Proceedings 9th International Workshop on Multimedia Information Systems*, pages 35–44, Ischia, Italy.

[Poole, 1997] Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56.

[Puterman, 1994] Puterman, M. (1994). *Markov Decision Processes: Discrete Dynamic Programming*. Wiley & Sons, Chicester, New York, Brisbane.

[Reiter, 1998] Reiter, R. (1998). Sequential temporal golog. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 547–556, Trento, Italy. Morgan Kaufman.

[Scientific, 2003] Scientific, T. (2003). Kalman Filter References. http://www.taygeta.com/kalrefs.html.

[Shoham, 1988] Shoham, Y. (1988). *Reasoning about Change*. MIT Press, Cambridge, Mass.

[Siegal, 1996] Siegal, J. (1996). *CORBA Fundementals and Programming*. John Wiley & Sons, New York.

[Subrahmanian et al., 2000] Subrahmanian, V., Bonatti, P., Dix, J., Eiter, T., Kraus, S., Özcan, F., and Ross, R. (2000). *Heterogenous Active Agents*. MIT-Press, Cambridge, Mass.

[T. Hammel and Rogers, 2003] T. Hammel, B. Y. and Rogers, T. (2003). Fusing live sensor data into situational multimedia views. In *Proceedings 9th International Workshop on Multimedia Information Systems*, pages 145–156, Ischia, Italy.

[Tarski, 1955] Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309.

[Thomas et al., 1991] Thomas, B., Shoham, Y., Schwartz, A., and Kraus, S. (1991). Preliminary Thoughts on an Agent Description Language. *International Journal of Intelligent Systems*, 6(5):497–508.

[Wolfram, 1998] Wolfram, C. D. (1998). Strategic bidding in a multi-unit auction: An empirical analysis of bids to supply electricity. *RAND Journal of Economics*, 29(4):703–72.

A Core Theory of *IMPACT*

In *IMPACT*, each agent $\mathtt{a}$ is built on top of a body of software code (built in any programming language) that supports a well–defined application programmer interface (either part of the code itself, or developed to augment the code). Hence, associated with each agent $\mathtt{a}$ is a body of software code defined as follows.

Definition A.1 ((Software Code)) *We may characterise the code on top of which an agent is built as a triple $\mathcal{S} = (\mathcal{T}_\mathtt{a}, \mathcal{F}_\mathtt{a}, \mathcal{C}_\mathtt{a})$ where*

1. *$\mathcal{T}_\mathtt{a}$ is the set of all data types managed by $\mathcal{S}$,*
2. *$\mathcal{F}_\mathtt{a}$ is a set of predefined functions which makes access to the data objects managed by the agent available to external processes, and*
3. *$\mathcal{C}_\mathtt{a}$ is a set of type composition operations. A type composition operator is a partial n-ary function c which takes types $\tau_1, \ldots, \tau_n$ as input, and yields a type $c(\tau_1, \ldots, \tau_n)$ as output. As c is a partial function, c may only be defined for certain arguments $\tau_1, \ldots, \tau_n$, i.e., c is not necessarily applicable on arbitrary types.*

When $\mathtt{a}$ is clear from context, we will often drop the subscript $\mathtt{a}$. Intuitively, $\mathcal{T}_\mathtt{a}$ is the set of all data types managed by $\mathtt{a}$. $\mathcal{F}_\mathtt{a}$ is the set of all function calls supported by the application programmer interface (*API*) of the agent's legacy code. $\mathcal{C}_\mathtt{a}$ is the set of ways of creating new data types from existing data types. This characterisation of a piece of software code is widely used (cf. the *Object Data Management Group*'s *ODMG* standard [Cattell, R. G. G., et al., 1997] and the *CORBA* framework [Siegal, 1996]). Each agent also has a message box having a well–defined set of associated code calls that can be invoked by external programs.

Definition A.2 ((State of an Agent)) *The state of an agent $\mathtt{a}$ at any given point $\mathtt{t}$ in time, denoted $\mathcal{O}_\mathtt{a}(t)$, consists of the set of all instantiated data objects of types contained in $\mathcal{T}_\mathtt{a}$.*

An agent's state may change because it took an action, or because it received a message. Throughout this paper we will assume that except for appending messages to an agent $\mathtt{a}$'s mailbox, another agent $\mathtt{b}$ cannot directly change $\mathtt{a}$'s state. However, it might do so indirectly by shipping the other agent a message requesting a change.

Queries and/or conditions may be evaluated w.r.t. an agent state using the notion of a code call atom and a code call condition defined in Section 3 (Definitions 3.1 and 3.2). An integrity constraint is an implication whose consequent is a code call atom, and whose antecedent is a code call condition. Appendix A contains a detailed definition.

Each agent has an action-base describing various actions that the agent is capable of executing. Actions change the state of the agent and perhaps the state of other agents' `msgboxes`. An action has five components: *(i)* a name $\alpha(\mathtt{X}_1, \ldots, \mathtt{X}_n)$ where $n \geq 0$ and

$\mathtt{X}_1, \ldots, \mathtt{X}_n$ are variables, *(ii)* a *precondition* which is a code call condition, *(iii)* an add list which is a set of code call atoms, *(iv)* a delete list which is a set of code call atoms, and *(v)* an action *method* (in the strict sense of object–oriented programming) which is a possibly imperative body of code that implements the action. An *instance* of an action is obtained by applying a substitution to components *(i)–(iv)* of an action which causes all these components to become *ground* (i.e., variable free). As usual, an action instance can be executed when the appropriate instance of the precondition is true in the current agent state, and the new state that results is just like the current state except that the ground atoms in the add list instance become true, while the ground atoms in the delete list instance become false. For more detailed definitions, the reader is referred to [Eiter et al., 1999].

Each agent has an associated "notion of concurrency" **conc**, which takes a set of actions and an agent state as input, and produces as output, a single action that reflects the combination of all the input actions. [Eiter et al., 1999] provide examples of three different notions of concurrency. We will sometimes abuse notation and write $\mathbf{conc}(S, \mathcal{O})$ to denote the new state obtained by concurrently executing the actions in S in state $\mathcal{O}$.

Each agent has an associated set of *action constraints* that define the circumstances under which certain actions may be concurrently executed. As at any given point $\mathtt{t}$ in time, many sets of actions may be concurrently executable, each agent has an *Agent Program* that determines what actions the agent can take, what actions the agent cannot take, and what actions the agent must take. Agent programs are defined in terms of status atoms defined below.

Definition A.3 ((Agent Program)) *An agent program $\mathcal{P}$ is a finite set of rules of the form*

$$A \quad \leftarrow \quad \chi \,\&\, L_1 \,\&\, \ldots \,\&\, L_n$$

where χ is a code call condition and $L_1, \ldots, L_n$ are status literals.

Various semantics of agent programs are well described in [Eiter et al., 1999] as well as in [Subrahmanian et al., 2000]. They are not needed in the sequel, as all newly introduced semantics will be discussed at length. But understanding the framework of ordinary agent programs obviously helps to get a better picture of the extension described in this paper.

Definition A.4 ((Deontic and Action Consistency)) *A status set S is called* deontically consistent*, iff it satisfies the following rules for any ground action α:*

- *If $\mathbf{O}\alpha \in S$, then $\mathbf{W}\alpha \notin S$*
- *If $\mathbf{P}\alpha \in S$, then $\mathbf{F}\alpha \notin S$*
- *If $\mathbf{P}\alpha \in S$, then $\mathcal{O}_{\mathcal{S}} \models \exists^* Pre(\alpha)$, where $\exists^* Pre(\alpha)$ denotes the existential closure of $Pre(\alpha)$, i.e., all free variables in $Pre(\alpha)$ are governed by an existential quantifier. This condition means that the action α is in fact executable in the state $\mathcal{O}_{\mathcal{S}}$.*

A status set S is called action consistent*, if $S, \mathcal{O}_\mathcal{S} \models \mathcal{AC}$ holds.*

Besides consistency, we also wish that the presence of certain atoms in S entails the presence of other atoms in S. For example, if $\mathbf{O}\alpha$ is in S, then we expect that $\mathbf{P}\alpha$ is also in S, and if $\mathbf{O}\alpha$ is in S, then we would like to have $\mathbf{Do}\,\alpha$ in S. This is captured by the concept of deontic and action closure.

Definition A.5 ((Deontic and Action Closure)) *The* deontic closure *of a status S, denoted $\mathbf{D\text{-}Cl}(S)$, is the closure of S under the rule*

If $\mathbf{O}\alpha \in S$, then $\mathbf{P}\alpha \in S$

where α is any ground action. We say that S is deontically closed*, if $S = \mathbf{D\text{-}Cl}(S)$ holds.*

The action closure *of a status set S, denoted $\mathbf{A\text{-}Cl}(S)$, is the closure of S under the rules*

If $\mathbf{O}\alpha \in S$, then $\mathbf{Do}\,\alpha \in S$

If $\mathbf{Do}\,\alpha \in S$, then $\mathbf{P}\alpha \in S$

where α is any ground action. We say that a status S is action-closed, if $S = \mathbf{A\text{-}Cl}(S)$ holds.

The following straightforward results shows that status sets that are action-closed are also deontically closed, i.e.,

Definition A.6 ((Operator $\mathbf{App}_{\mathcal{P},\mathcal{O}_\mathcal{S}}(S)$)) *Suppose $\mathcal{P}$ is an agent program, and $\mathcal{O}_\mathcal{S}$ is an agent state. Then, $\mathbf{App}_{\mathcal{P},\mathcal{O}_\mathcal{S}}(S)$ is defined to be the set of all ground action status atoms A such that there exists a rule in P having a ground instance of the form $r : A \leftarrow L_1 \& \ldots \& L_n$ such that*

1. *$B^+_{as}(r) \subseteq S$ and $\{L : \neg L \in B^-_{as}(r)\} \cap S = \emptyset$, and*
2. *every code call $cc \in B^+_{cc}(r)$ succeeds in $\mathcal{O}_\mathcal{S}$, and*
3. *every code call $cc \in \{L : \neg L \in B^-_{cc}(r)\}$ does not succeed in $\mathcal{O}_\mathcal{S}$, and*
4. *for every atom $\mathsf{Op}(\alpha) \in B^+(r) \cup \{A\}$ such that $\mathsf{Op} \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$, the action α is executable in state $\mathcal{O}_\mathcal{S}$.*

Note that part (4) of the above definition only applies to the "positive" modes $\mathbf{P}, \mathbf{O}, \mathbf{Do}$. It does not apply to atoms of the form $\mathbf{F}\alpha$, as such actions are not executed, nor does it apply to atoms of the form $\mathbf{W}\alpha$, because execution of an action might be (vacuously) waived, if its prerequisites are not fulfilled.

Our approach is to base the semantics of agent programs on consistent and closed status sets. However, we have to take into account the rules of the program as well as integrity constraints. This leads us to the notion of a feasible status set.

Definition A.7 ((Feasible Status Set)) *Let $\mathcal{P}$ be an agent program, and let $\mathcal{O}_\mathcal{S}$ be an agent state. Then, a status set S is a* feasible status set *for $\mathcal{P}$ on $\mathcal{O}_\mathcal{S}$, if the following conditions hold:*

(S1) *(closure under the program rules)* $\boldsymbol{App}_{\mathcal{P},\mathcal{O}_\mathcal{S}}(S) \subseteq S$;

(S2) *(deontic/action consistency)* S *is deontically and action consistent;*

(S3) *(deontic/action closure)* S *is action closed and deontically closed;*

(S4) *(state consistency)* $\mathcal{O}'_\mathcal{S} \models \mathcal{IC}$, *where* $\mathcal{O}'_\mathcal{S} = apply(\mathbf{Do}\,(S), \mathcal{O}_\mathcal{S})$ *is the state which results after taking all actions in* $\mathbf{Do}\,(S)$ *on the state* $\mathcal{O}_\mathcal{S}$.

Definition A.8 ((Groundedness; Rational Status Set)) *A status set S is* grounded*, if there exists no status set $S' \neq S$ such that $S' \subseteq S$ and S' satisfies conditions $(S1)$–$(S3)$ of a feasible status set.*

A status set S is a rational status set*, if S is a feasible status set and S is grounded.*

Definition A.9 ((Reasonable Status Set)) *Let $\mathcal{P}$ be an agent program; let $\mathcal{O}_\mathcal{S}$ be an agent state, and let S be a status set.*

1. *If $\mathcal{P}$ is a positive agent program, then S is a* reasonable status set *for $\mathcal{P}$ on $\mathcal{O}_\mathcal{S}$, if and only if S is a rational status set for $\mathcal{P}$ on $\mathcal{O}_\mathcal{S}$.*
2. *The reduct of $\mathcal{P}$ w.r.t. S and $\mathcal{O}_\mathcal{S}$, denoted by $red^S(\mathcal{P}, \mathcal{O}_\mathcal{S})$, is the program which is obtained from the ground instances of the rules in $\mathcal{P}$ over $\mathcal{O}_\mathcal{S}$ as follows.*
   (a) *First, remove every rule r such that $B^-_{as}(r) \cap S \neq \emptyset$.*
   (b) *Remove all atoms in $B^-_{as}(r)$ from the remaining rules.*

   Then S is a reasonable status set *for $\mathcal{P}$ w.r.t. $\mathcal{O}_\mathcal{S}$, if it is a reasonable status set of the program $red^S(\mathcal{P}, \mathcal{O}_\mathcal{S})$ with respect to $\mathcal{O}_\mathcal{S}$.*

Notational Conventions.

As this paper involves heterogeneous data sources, deontic modalities, actions, logical methods, and temporal reasoning, all of which are complex subjects of research in their own right, it is inevitable that the paper is heavy on notation. We end this section with two tables listing the terminology used. While Table 1 contains the basic notation already introduced in [Eiter et al., 1999, Subrahmanian et al., 2000], Table 2 points to the new notions introduced in this paper.

In addition, we note that agents always appear in the `agent` font while functions and constants in software packages are written in italics. Variables and types come in typewriter font: **in**(`X`, `agent`:$function(const_1, \texttt{Var}_2)$). Actions α are denoted by lower Greek letters. Calligraphic letters are used for meta objects, which are whole

Notation	Description	Definition
$\mathcal{S}^{\mathfrak{a}}$	software code on top of which $\mathfrak{a}$ is built	Def. A.1
$\mathcal{T}_{\mathcal{S}}^{\mathfrak{a}}$	data types of software code $\mathcal{S}^{\mathfrak{a}}$	Def. A.1
$\mathcal{F}_{\mathcal{S}}^{\mathfrak{a}}$	function of software code $\mathcal{S}^{\mathfrak{a}}$	Def. A.1
$\mathcal{C}_{\mathcal{S}}^{\mathfrak{a}}$	type composition operations	Def. A.1
$\mathcal{O}_{\mathcal{S}}(t)$	state of an agent at time t	Def. A.2
$\mathbf{in}(\mathtt{X}, \mathtt{cc})$	code call atom indicating that ($X \in \mathtt{cc}$)	Def. 3.1
χ	code call condition	Def. 3.2
$\mathsf{Op}\alpha(\vec{t})$	status atom, e.g., $\mathbf{P}\alpha, \mathbf{O}\alpha, \mathbf{Do}\,\alpha, \mathbf{W}\alpha$	Def. 3.7
L_i	status literal, e.g., $\mathbf{P}\alpha$ and $\neg\mathbf{P}\alpha$	Def. 3.7
$\mathcal{IC}$	integrity constraints	Sec. 1
$\mathcal{AC}$	action constraints	Sec. 1

Table 1: Glossary 1: Basic Notation

collections of objects: $\mathcal{T}_{\mathcal{S}}^{\mathfrak{a}}$, $\mathcal{O}_{\mathcal{S}}(t)$, $\mathcal{IC}$, $\mathcal{TP}$. Boldface is also used for meta-theoretic notions: operators like $\mathsf{D}_{\mathcal{TP}}$, closures like $\mathbf{D\text{-}Cl}()$, $\mathbf{A\text{-}Cl}()$, and the deontic modalities $\mathbf{P}$, $\mathbf{Do}$, $\mathbf{O}$, $\mathbf{P}$, $\mathbf{W}$.

The newly introduced temporal expressions and all things that have to do with time are put into a sans serif font to distinguish them from our base terminology: $\mathsf{t}_{\mathsf{now}}$, te, ti, $\mathsf{acthist}$, tc-$\mathcal{T}S$, $[\mathsf{te}_1, \mathsf{te}_2]$.

Probabilistic items are written in a curly font: ℓ_1, ℓ_2, δ, $[\ell_1, \ell_2]$. Sets of objects are again written in boldface: **pas**, T.

Both time and probability come together in the notion of a TP annotation: $[\otimes, \langle[\mathsf{t}_1, \mathsf{t}_2], \delta\rangle, [\ell_1, \ell_2]]$.

Notation	Description	Definition
$\mathbf{RV}(Obj, \wp)$	random variable	Def. 3.3
S	*coherent* set of $\mathbf{RV}$'s	Def. 3.3
τ	type of a code call	Def. 3.3
cc^{TP}	temporal probabilistic code call based on cc	Def. 3.5
$\otimes$ ($\oplus$)	probabilistic conjunction (disjunction) strategy	Def. 3.6
te	temporal expression, e.g., $5, \mathtt{x}_{\mathsf{now}} + 3$	Def. 4.1
ti	temporal interval, e.g., $[\mathsf{te}_1, \mathsf{te}_2]$	Def. 4.1
ℓ	probabilistic item, e.g. $\frac{\mathbf{x}+1}{2}$	Def. 4.2
pi	probabilistic interval, e.g., $[\ell_1, \ell_2]$	Def. 4.2
δ	probability distribution function (**pdf**)	Def. 4.3
$[\otimes, \langle \mathsf{ti}, \delta \rangle, [\ell, \ell']]$	TP-annotation	Def. 4.4
$\chi : [\otimes, \langle \mathsf{ti}, \delta \rangle, [\ell, \ell']]$	TP-CCC	Def. 4.5
$(A_1 \wedge \ldots \wedge A_n) : [\otimes, \langle \mathsf{ti}, \delta \rangle, [\ell, \ell']]$	TP-ASC	Def. 4.5
$[\otimes, \langle \mathsf{ti}, \delta \rangle]$	TP-annotation for strict programs	Def. 4.6
$\mathcal{TPP}$	TP agent program	Def. 4.7
$\mathbf{pas}(cc, o, \mathsf{T})$	set of possible answer situations	Def. 5.1
FTPSI	feasible TP status interpretation	Def. 5.19
acthist	action history	Def. 5.12
$\mathcal{EO}(\mathsf{t})$	expected states at time t	Def. 5.15
tc-$\mathcal{TS}$	temporally constrained status set	Def. 6.1
Comp-tc-$\mathcal{TS}$	TP-status interpr. compatible with tc-$\mathcal{TS}$	Def. 6.2
$\mathsf{D}_{\mathcal{TP}}$	operator applying $\mathcal{TPP}$ on tc-$\mathcal{TS}$ once	Def. 6.3
H	TP hitting set	Def. 6.13
chs(tc-$\mathcal{TS}$)	set of temp. constr. hitting sets for tc-$\mathcal{TS}$	Def. 6.13

Table 2: Glossary 2: Notions wrt. Temporal Probabilistic Approach.

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Wojciech Jamroga
Contact: wjamroga@in.tu-clausthal.de
URL: http://www.in.tu-clausthal.de/~wjamroga/techreports/

The IfI Review Board

Prof. Dr. Jürgen Dix (Theoretical Computer Science/Computational Intelligence)
Prof. Dr. Klaus Ecker (Applied Computer Science)
Prof. Dr. habil. Torsten Grust (Databases)
Prof. Dr. Barbara Hammer (Theoretical Foundations of Computer Science)
Prof. Dr. Kai Hormann (Computer Graphics)
Dr. Michaela Huhn (Economical Computer Science)
Prof. Dr. Gerhard R. Joubert (Practical Computer Science)
Prof. Dr. Ingbert Kupka (Theoretical Computer Science)
Prof. Dr. Wilfried Lex (Mathematical Foundations of Computer Science)
Prof. Dr. Jörg Müller (Agent Systems)
Prof. Dr.-Ing. Dr. rer. nat. habil. Harald Richter (Technical Computer Science)
Prof. Dr. Gabriel Zachmann (Virtual Reality)