Temporal Agent Programs

Jürgen Dix [a,1] Sarit Kraus [b,c,2] V.S. Subrahmanian [d,3]

[a] *The University of Manchester, Dept. of Computer Science, Oxford Road, Manchester, M13 9PL, UK*

[b] *Dept. of Computer Science, Bar-Ilan University, Israel*

[c] *Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20752, USA*

[d] *Dept. of Computer Science, University of Maryland, College Park, MD 20752, USA*

Abstract

The "agent program" framework introduced by Eiter, Subrahmanian and Pick (**Artificial Intelligence, 108(1-2), 1999**), supports developing agents on top of arbitrary legacy code. Such agents are continuously engaged in an *"event occurs→ think → act→ event occurs ..."* cycle. However, this framework has two major limitations: (1) all actions are assumed to have no duration, and (2) all actions are taken now, but cannot be *scheduled for the future*. In this paper, we present the concept of a "temporal agent program" (tap for short) and show that using taps, it is possible to build agents on top of legacy code that can reason about the past and about the future, and that can make temporal commitments for the future now. We develop a formal semantics for such agents, extending the concept of a status set proposed by Eiter et al., and develop algorithms to compute the status sets associated with temporal agent programs. Last, but not least, we show how taps support the decision making of collaborative agents.

Key words: Agents, Heterogenous Systems, Information Integration, Temporal Reasoning, Logic Programming.

---

[1] This work was carried out when the author was visiting the University of Maryland from January-October 1999.

[2] The author gratefully acknowledges support from NSF grant # IIS9907482.

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

1 Introduction

The success of an agent development infrastructure in the real world hinges on many aspects including, but not limited to: (i) the ability of the infrastructure to build agents on top of legacy/specialized code-bases without having to study the underlying source code, (ii) the ability to declaratively specify the behavior of the agent, (iii) the ability to guarantee that the agent will never cause certain "undesirable" situations to arise, (iv) the ability of agents to collaborate with each other, (v) the ability of agents to reason about time and make commitments for the future, (vi) the ability of agents to negotiate with each other and reason with (perhaps uncertain) beliefs about other agents, etc. In this paper, we focus on (v) in the context of a theory and implementation presented in (Eiter, Subrahmanian, and Pick 1999; Eiter and Subrahmanian 1999; Eiter, Subrahmanian, and Rogers 2000; Arisha, Ozcan, Ross, Subrahmanian, Eiter, and Kraus 1999) that encapsulates (i)–(iii) comprehensively, and supports many aspects of (iv) (Eiter and Subrahmanian 1999) and (vi) (Dix, Subrahmanian, and Pick 2000; Dix, Nanni, and Subrahmanian 2000).

In this paper, we focus on the following features supporting (v) above.

(1) Agents must be able to execute actions that have *duration*. For instance, a vehicular agent that is executing the action $go(A, B)$ can hardly do this instantaneously when A and B are two different locations.
(2) When agents execute actions with duration, the state of the agent might change during the execution of the action, rather than at the end. This then raises the question: should every tiny state change be recorded? Or should it be left to the agent developer to decide when such changes *during* action execution be incorporated into the state? We will support *both* options via a construct called a *checkpoint* defined in this paper.
(3) We introduce the concept of a *temporal agent program* (tap for short) which allows an agent developer to specify under what conditions and when an agent is permitted to take actions, forbidden from taking certain actions, obliged to take actions, etc. This is further complicated by the fact that actions may have effects over an extended time period. taps extend the notion of an agent program (Eiter, Subrahmanian, and Pick 1999) which we have implemented in a system called *IMPACT* (Eiter, Subrahmanian, and Rogers 2000; Subrahmanian, Bonatti, Dix, Eiter, Kraus, Özcan, and Ross 2000).
(4) In addition to the syntax of taps, we provide two semantics for taps which extend semantics for ordinary agent programs. In (Eiter, Subrahmanian, and Pick 1999), semantics of ordinary agent programs were shown to extend semantics of logic programs as well as the semantics of default and autoepistemic theories (Reiter 1980; Moore 1985). Thus we build on top of an extensively studied and solid semantical foundation.

(5) The above semantics are defined in terms of a semantical construct called a *temporal status set* for which we develop a compact representation.
(6) When an agent's state changes (which may occur through receipt of a message requesting a service or responding to a service request, or the tick of a clock, or a random number generator, etc.), the agent must find a new temporal status set that satisfies various semantical requirements. This tells the agent what it is obliged to do now (and in the future), what it is permitted to do (now and in the future), what it is forbidden from doing (now and in the future) and also allows it to determine what in fact it will do. We develop for positive taps algorithms to compute temporal status sets in accordance with both the semantics we developed.

The contributions listed above leverage from contributions in several areas of AI and computer science. We briefly overview areas of relevant research.

Temporal Reasoning: There has been extensive work in temporal reasoning in computer science. These fall into two broad categories—those that *numerically* represent time such as virtually all work in databases (Zaniolo, Ceri, Faloutsos, Snodgrass, Subrahmanian, and Zicari 1997, chaps. 5–7), multimedia systems (Subrahmanian 1998), operating systems and networks, and those that symbolically represent time (such as almost all work in temporal logic and AI (Fisher and Owens 1995)). Explicit numeric time has also been used (perhaps to a smaller extent) in AI (Allen and Ferguson 1994).

The closest work in temporal agents to ours is the work on *MetaTem* (Barringer, Fisher, Gabbay, Gough, and Owens 1989) and its successor, *Concurrent MetaTem* (Fisher 1994). While many of the rules in *MetaTem* can be expressed via taps and vice versa, not everything in *MetaTem* can be expressed via taps and vice versa. A detailed comparison will be presented in Section 7. However, all the rules expressed in *Concurrent MetaTem* as described in (Barringer, Fisher, Gabbay, Gough, and Owens 1989) are expressible via taps (see Section 7). Neither *MetaTem* nor *Concurrent MetaTem* appears to be able to easily express the following scenarios which are representative of a whole class of tap applications.

(1) We can express rules of the form *"If the maximal time previously taken to ship a package from location A to location B is T_1, and if package P is required to be at location B at time T, then ship package P sometime between time $T - T_1 - 10$ and $T - T_1$."* This is a very reasonable statement to make in any logistics application, but the time T might depend on the production schedule of the company at location B (which may be determined at run-time from a database), and T_2 likewise might depend on the identities of locations A, B (which may be instantiated at run time and whose locations might therefore need to be inferred at run-time from a database). In general, it appears hard to express in both *MetaTem* and *Concurrent MetaTem*, temporal constraints on events where the occurrence time of those events needs to be inferred

dynamically from databases or by using a software packages. In fact, the problem with classical temporal logic here is that it does not allow us to use arguments of atoms/literals to instantiate the number of occurrences of a modal connective like $\bigcirc$.

(2) Another example may say *"If a prediction package expects a stock to rise K% after T_K units of time and $K \geq 25$ then buy the stock at time $(\mathtt{X}_{now} + T_K - 2)$."* This cannot be easily expressed via *MetaTem* and *Concurrent MetaTem* even though this has obvious value.

(3) The use of deontic modalities in our framework is not covered in *MetaTem* and *Concurrent MetaTem*.

(4) Our mechanisms to access heterogeneous data structures and software packages is an immediate contribution that can be used by *MetaTem* and *Concurrent MetaTem*.

(5) Last, but not least, study of integrity constraints in *Concurrent MetaTem*, and ways of enforcing them appears to be an open issue.

(6) Conversely, *Concurrent MetaTem* has been extended in some interesting directions such as support for beliefs that we do not handle. In addition, rule heads may often contain a disjunction of literals which we also do not support in taps. These extensions have obvious value.

We show how (1) and (2) above can be expressed via taps just before the beginning of Section 4. We can also express Baudinet's temporal logic programming framework (Baudinet 1992) as instances of taps as well as part but not all of Abadi and Manna's *TEMPLOG* framework (Abadi and Manna 1987) (see Section 7). (1) and (2) above are just as hard to express in these other logics.

Active databases: There has also been extensive work on active databases (U. Dayal and E. Hanson and J. Widom 1995) via the notion of an event-condition-action (ECA) rule. ECA rules have the form *"If condition C is true in the current database state and actions $A_1, \ldots, A_n$ have been done and none of actions $B_1, \ldots, B_m$ have been done, then do action A."* As we will show in Section 7, ECA-rules are a special case of taps. ECA rules have been defined *only* for relational databases (Zaniolo, Ceri, Faloutsos, Snodgrass, Subrahmanian, and Zicari 1997, chaps. 2-4) and object bases (Fisher 1995). In contrast, taps are defined on top of arbitrary pieces of legacy code (not just relational and OO databases). Second, ECA-rules have no temporal component. In contrast, taps allow temporal indeterminacy in ECA-rules (e.g. *"If the stock dropped U_1 percent in the last day and U_2 units the day before, then sell the stock sometime in the next two hours."*) Rules such as these support temporal indeterminacy which ECA rules do not support. In addition, ECA-rules do not support rules of the form *"If person P was permitted to take action A and more than 5 time units have elapsed since then and he has not taken action A yet, then the system is obliged to take action B some time in the next 10 time units"*). For instance, action A might be *"send in tax return"* and action B might be *"send a reminder to person P about his/her tax return."* Rules of this kind cannot be expressed via ECA-rules.

Logic programs and deductive databases: Logic programs and deductive databases have extensively studied the problem of reasoning with rules of the form *"If $\langle$ condition$\rangle$ is true, then formula F"* is true as well. Variances in the syntax of condition C and F are studied (Lloyd 1987). In fact, this work may be viewed as a continuation of this trend. However, there are many differences: first and foremost, these systems assume either that all data is represented as logical atoms or in a relational database. In contrast, taps can be built on top of arbitrary bodies of software code. Second, these systems do not automatically allow us to evaluate conditions such as *"If state condition A and the obligation to do action B hold simultaneously at some time point between time* $\mathtt{X}_{now} - 5$ *and* $\mathtt{X}_{now}$ *and state condition C holds at time* $\mathtt{X}_{now}$ *and it is forbidden to perform action* α *now and state condition* $r(X)$ *holds, then perform action* β *at some time point in the next X time units."* In fact, we feel it is important to point out that this paper cleanly extends well-studied semantics for nonmonotonic logic programming such as the Herbrand model semantics and the minimal model semantics.

Deontic logic: We borrow from the field of deontic logic, the syntax of deontic statements; however we do not lay down the semantics of taps on the basis of one of the numerous deontic logical systems (e.g., Standard Deontic Logic (SDL), which amounts to the modal logic KD (Åquist 1984; Meyer and Wieringa 1993)). We mention that deontic logic has been plagued by paradoxical behavior of the logic (Åquist 1984; Meyer and Wieringa 1993). During the last 50 years, numerous systems of deontic logic based on modal logic have been proposed, but still most of these still suffer from paradoxical behavior. Another reason for not building upon existing deontic logic systems is that actions in deontic logic typically do not have effects – hence, the fact that a set of actions may all be individually permitted, but mutually impossible to be concurrently executed is not addressed in deontic logic. We address this. In addition, we explicitly support nonmonotonic negation in tap rules, and provide a framework for agent decision making on top of "real" legacy software packages. Last, but not least, we support temporal indeterminacy in deontic obligations, permissions, and forbidden atoms, and we provide algorithms to perform such computations.

The rest of this paper is organized as follows.

(1) Section 2 first presents a brief motivating example that will be expanded on as the paper proceeds. It then overviews the framework of (Eiter, Subrahmanian, and Pick 1999) and explains how legacy and specialized software code may be "agentized".
(2) Section 3 shows how to specify actions with *temporal duration* and introduces the syntax of a *temporal agent program* (tap).
(3) Section 4 develops two semantics for taps which extend the semantics introduced for non-temporal agents in (Eiter, Subrahmanian, and Pick

1999) based on the notion of a temporal status set ($\mathcal{TS}_{\mathsf{t_{now}}}$). The section ends with a description of a compact representation of $\mathcal{TS}_{\mathsf{t_{now}}}$'s.

(4) Section 5 presents an algorithm that computes the (compact representation of a) class of taps called *positive* taps.
(5) Finally, in Section 6, we present an application of taps. It is about individual agents collaboratively working together to satisfy a shared goal.
(6) Section 7 compares and contrasts our work with existing research.

We refer the reader to (Dix, Kraus, and Subrahmanian 2000), a technical report containing all the missing proofs of theorems and lemmas in this paper.

2 Preliminaries

Before giving an overview of the *IMPACT* framework defined in (Eiter, Subrahmanian, and Pick 1999; Subrahmanian, Bonatti, Dix, Eiter, Kraus, Özcan, and Ross 2000), we start with the following potential application.

Example 2.1 (Rescue-Scenario) Consider a simplistic rescue operation where a natural calamity (e.g. a flood) has stranded many people. Rescuing these people requires close coordination between helicopters and ground vehicles. For the sake of this example, we assume the existence of:

(1) A $\mathsf{helicopter}$ agent that conducts aerial reconnaissance and supports aerial rescues;
(2) A set $\mathsf{gv1}, \mathsf{gv2}, \mathsf{gv3}$ of ground vehicles that move along the ground to appropriate locations—such vehicles may include ambulances as well as earth moving vehicles.
(3) An immobile command center agent comc that coordinates between the helicopter and the ground vehicles.

In our *IMPACT* system (which this paper extends), each agent 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).

Definition 2.2 (Software Code) *We may characterize the code on top of which an agent* a *is built as a triple* $\mathcal{S}^{\mathsf{a}} =_{def} (\mathcal{T}^{\mathsf{a}}_{\mathcal{S}}, \mathcal{F}^{\mathsf{a}}_{\mathcal{S}}, \mathcal{C}^{\mathsf{a}}_{\mathcal{S}})$ *where:*

(1) $\mathcal{T}^{\mathsf{a}}_{\mathcal{S}}$ *is the set of all data types managed by* $\mathcal{S}$*,*
(2) $\mathcal{F}^{\mathsf{a}}_{\mathcal{S}}$ *is the set of predefined (API) functions over* $\mathcal{T}^{\mathsf{a}}_{\mathcal{S}}$ *through which external processes may access* a*'s data, and*
(3) $\mathcal{C}^{\mathsf{a}}_{\mathcal{S}}$ *is a set of type composition operations. A type composition operator is a partial n-ary function c which takes as input types* $\tau_1, \ldots, \tau_n$ *and yields*

as output a type $c(\tau_1, \ldots, \tau_n)$.

This characterization 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)). Without loss of generality, we will henceforth assume that $\mathcal{T}_{\mathcal{S}}$ is closed under the operations in $\mathcal{C}_{\mathcal{S}}$.

Each agent also has a message box having a well defined set of associated code calls that can be invoked by external programs.

Example 2.3 (Rescue Example) Consider the rescue mission described earlier. The $\mathtt{heli}$ agent may have the following data types and code calls.

- Data Types: *speed*, *bearing* of type `int`, *location* of type `point` (record containing x, y, z fields), *nextdest* of type `string`, and *inventory*—a relation having schema (`Item`, `Qty`, `Unit`).
- Functions:
  - $\mathtt{heli}: location()$: which specifies an (x, y, z) coordinate for the helicopter.
  - $\mathtt{heli}: inventory(item)$: returns a pair of the form $\langle Qty, Unit\rangle$. For instance, $\mathtt{heli}: inventory(blood)$ may return the pair $\langle 25, liters\rangle$ specifying that the helicopter currently has 25 units of blood available.

Definition 2.4 (State of an Agent) *The state of an agent at any given point t in time, denoted* $\mathcal{O}_{\mathcal{S}}(t)$*, consists of the set of all data objects in the data structures (consisting of types contained in* $\mathcal{T}_{\mathcal{S}}^{\mathfrak{a}}$*) of the agent.*

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 $\mathfrak{a}$'s mailbox, another agent $\mathfrak{b}$ cannot directly change $\mathfrak{a}$'s state. However, it might do so indirectly by shipping the other agent a message requesting a change.

Example 2.5 (Rescue: State) For instance, at a given instant of time, the state of the $\mathtt{heli}$ agent may consist of $location = \langle 45, 50, 9000\rangle$, and *inventory* containing the following four tuples: $\langle fuel, 125, gallons\rangle$, $\langle blood, 25, litres\rangle$, $\langle bandages, 50, -\rangle$, $\langle cotton, 20, lbs\rangle$.

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 below.

Definition 2.6 (Code Call/Code Call Atom) *If* $\mathcal{S}$ *is the name of a software package, f is a function defined in this package, and* $(d_1, \ldots, d_n)$ *is a tuple of arguments of the input type of f, then* $\mathcal{S}: f(\mathtt{d_1}, \ldots, \mathtt{d_n})$ *is called a* code call.

If $\texttt{cc}$ *is a code call, and* $\texttt{X}$ *is either a variable symbol, or an object of the output type of* $\texttt{cc}$*, then* $\mathbf{in}(\texttt{X}, \texttt{cc})$ *is called a* code call atom.

If $\texttt{X}$ is a variable over type τ and τ is a record structure with field f, then $\texttt{X}.f$ is a variable ranging over objects of the type of field f.

Definition 2.7 (Code Call Condition)

(1) Every code call atom is a code call condition.
(2) If $\texttt{s}, \texttt{t}$ *are either variables or objects, then* $\texttt{s} = \texttt{t}$ *is a code call condition.*
(3) If $\texttt{s}, \texttt{t}$ *are either integers/real valued objects, or are variables over the integers/reals, then* $\texttt{s} < \texttt{t}, \texttt{s} > \texttt{t}, \texttt{s} \geq \texttt{t}, \texttt{s} \leq \texttt{t}$ *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.*

For example, $\mathbf{in}(\texttt{X}, \texttt{heli}: \mathit{inventory}(\texttt{fuel})) \,\&\, \texttt{X.Qty} < 50$ is a code call condition that is satisfied whenever the helicopter has less than 50 gallons of fuel left.

Each agent has an associated action-base describing various actions that the agent is capable of executing. An action (whose behavior is that of a partial function from states to states) is implemented by a body of code in any suitable imperative (or declarative) programming language. The agent reasons about actions via a set of preconditions and effects defining the conditions an agent state must satisfy for the action to be considered executable, and the new state that results from such an execution. We assume that the preconditions and effects associated with an action correctly specify the behavior of the code implementing the action. Note, that in addition to changing the state of the agent, an action may change the state of other agents' `msgboxes`. In this paper, the actions preconditions will be expressed via code call conditions and the effects will be expressed by add/delete lists which are sets of ground code call atoms.

Each agent has (i) a set of **integrity constraints**—only states that satisfy these constraints are considered to be *valid* or *legal* states, (ii) a notion of **concurrency** specifying how to combine a set of actions into a single action, (iii) a set of **action constraints** that define the circumstances under which certain actions may be concurrently executed, and (iv) 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 2.8 (Status Atom/Status Set) *If* $\alpha(\vec{t})$ *is an action, and* $\mathsf{Op} \in$

Notation	Description	Definition
$\mathcal{S}^{\mathfrak{a}}$	software code on top of which $\mathfrak{a}$ is built	Def. 2.2
$\mathcal{T}^{\mathfrak{a}}_{\mathcal{S}}$	data types of software code $\mathcal{S}^{\mathfrak{a}}$	Def. 2.2
$\mathcal{F}^{\mathfrak{a}}_{\mathcal{S}}$	function of software code $\mathcal{S}^{\mathfrak{a}}$	Def. 2.2
$\mathcal{C}^{\mathfrak{a}}_{\mathcal{S}}$	type composition operations	Def. 2.2
$\mathcal{O}_{\mathcal{S}}(t)$	state of an agent at time t	Def. 2.4
$\mathbf{in}(\mathtt{X}, \mathtt{cc})$	code call atom indicating that $(X \in \mathtt{cc})$	Def. 2.6
χ	code call condition	Def. 2.7.
$\mathsf{Op}\alpha(\vec{t})$	status atom, e.g., $\mathbf{P}\alpha, \mathbf{O}\alpha, \mathbf{Do}\,\alpha, \mathbf{W}\alpha$	Def. 2.8
L_i	status literal, e.g., $\mathbf{P}\alpha$ and $\neg\mathbf{P}\alpha$	Def. 2.8
$\mathcal{IC}$	integrity constraints	Sec. 2
$\mathcal{AC}$	action constraints	Sec. 2

Table 1
Glossary 1: Basic Notation

$\{\mathbf{P}, \mathbf{F}, \mathbf{W}, \mathbf{Do}\,, \mathbf{O}\}$*, then* $\mathsf{Op}\alpha(\vec{t})$ *is called a* status atom*. If A is a status atom, then $A, \neg A$ are called* status literals*. A* status set *is a finite set of ground status atoms.*

Intuitively, $\mathbf{P}\alpha$ means α is permitted, $\mathbf{F}\alpha$ means α is forbidden, $\mathbf{O}\alpha$ means α is obligatory, $\mathbf{Do}\,\alpha$ means α is actually done, and $\mathbf{W}\alpha$ means that the obligation to perform α is waived.

Definition 2.9 (Agent Program) *An agent program $\mathcal{P}$ is a finite set of rules of the form $A \leftarrow \chi \,\&\, L_1 \,\&\, \ldots \,\&\, L_n$, where χ is a code call condition, L_i are status literals and A is a status atom.*

Several alternative semantics for agent programs are presented in (Eiter, Subrahmanian, and Pick 1999; Eiter and Subrahmanian 1999).

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, Subrahmanian, and

Notation	Description	Definition
$\mathsf{rel}:\{\mathtt{X} \mid \chi\}$ $\mathsf{rel}:\{i\}$	relative checkpoint expression, e.g., $\mathsf{rel}:\{20\}$, $\mathsf{rel}:\{X \mid \mathbf{in}(\mathtt{X}, \mathfrak{math}: compute(\frac{\mathtt{To-From}}{\mathtt{Speed}}))\}$	Def. 3.1
$\mathsf{abs}:\{\mathtt{X} \mid \chi\}$ $\mathsf{abs}:\{i\}$	absolute checkpoint expression, e.g., $\mathsf{abs}:\{20\}$ $\mathsf{abs}:\{\mathtt{T} \mid \mathbf{in}(\mathtt{T}, \mathfrak{clock}: time())\}$	Def. 3.1
tai	temporal annotation item, e.g., $5, \mathtt{X}_{\mathsf{now}} + 3$	Def. 3.7
$[\mathsf{tai}_1, \mathsf{tai}_2]$	temporal annotation, e.g., $[1, 7]$	Def. 3.8
$B(\varrho : \mathsf{ta})$	action status literals in ϱ	Def. 3.9
$B^-(\varrho : \mathsf{ta})$	negative literals in $B(\varrho : \mathsf{ta})$	Def. 3.9
$\neg.B^-(\varrho : \mathsf{ta})$	atoms that occur negatively in $B^-(\varrho : \mathsf{ta})$	Def. 3.9
$\mathcal{TP}$	temporal agent program	Def. 3.10
$\mathcal{T}S_{\mathsf{t_{now}}}$	temporal status set	Def. 4.1
$\mathsf{hist}_{\mathsf{t_{now}}}$	state history function	Def. 4.2
$\mathsf{acthist}_{\mathsf{t_{now}}}$	action history	Def. 4.5
$\mathcal{EO}(\mathsf{t})$	expected states at time t	Def. 4.8
tic	temporal interval constraint: $(5 \leq t \leq 10)$	Def. 4.19
$\mathsf{Op}\,\alpha : \mathsf{tic}$	interval constraint annotated status atom	Def. 4.20
$\mathsf{ic}\text{-}\mathcal{T}S$	interval constraint temporal status set	Def. 4.21
$\mathsf{CompTSS}$	temporal status sets compatible with $\mathsf{ic}\text{-}\mathcal{T}S$	Def. 4.22
$\mathsf{D}_{\mathcal{TP}}$	an operator that applies $\mathcal{TP}$ on its input $\mathsf{ic}\text{-}\mathcal{T}S$ once	Def. 5.1
H	constraint hitting set	Def. 5.8
$\mathsf{chs}(\mathsf{ic}\text{-}\mathcal{T}S)$	the set of all constraint hitting sets for $\mathsf{ic}\text{-}\mathcal{T}S$	Def. 5.8

Table 2
Glossary 2: Notions wrt. Temporal Approach.

Pick 1999; Eiter and Subrahmanian 1999), Table 2 points to the new notions introduced in this paper.

In addition, we note that agents always appear in the $\mathfrak{agent}$ font while functions and constants in software packages are written in italics. Variables and types come in typewriter font: $\mathbf{in}(\mathtt{X}, \mathfrak{agent}: function(const_1, \mathtt{Var}_2))$. Actions α are denoted by lower Greek letters. Calligraphic letters are used for meta objects, which are whole 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 **D-Cl**(), **A-Cl**(), and the deontic modalities **P**, **Do** , **O**, **P**, **W** .

The newly introduced temporal annotations and all things that have to do with time are put into a sans serif font to distinguish them from our base terminology: $\mathsf{t_{now}}$, $\mathsf{rel}: \{\mathtt{X} \mid \chi\}$, $\mathsf{duration}(\alpha)$, $\mathsf{checkpoints}(\alpha)$, tai, tasc, $\mathsf{hist}_{\mathsf{t_{now}}}$, tic, TSS.

3 Syntax of tap's

In this section, we introduce the syntax of temporal agent programs (taps for short), and provide an "intuitive" semantics for them—the formal semantics is deferred to Section 4. An important feature of a tap is that it makes statements about the status of *actions*. Consequently, Section 3.1, starts by extending the notion of action to *timed actions*.

3.1 Actions with Temporal Duration

Most real-world actions have a *duration*. Moreover, while the action takes place, it might be important to specify intermediate time points, *checkpoints* (Definition 3.1), and to update the current state incrementally at these pre-specified points. This updating of a state is specified in our framework by *timed effect triples* (Definition 3.3). Both notions are important ingredients for our definition of a *timed action* (Definition 3.5).

For example, the heli agent in our Rescue Example may execute the action fly("BigRag", "StonyPoint"). This action has a temporal duration during which the location of heli is changing continuously. More importantly, if we know the location of the plane *now* and we know the plane's velocity and climb angle, we can precisely compute its location in the future (assuming no change in these parameters). Thus, in order to specify a *timed action*, we must:

(1) Specify an estimate of the total amount of time it takes for the action to be "completed".
(2) Specify exactly how the state of the agent changes *while* the action is being executed.

It is worth noting that the duration of an action can be precisely specified in some cases, but not in others. For instance, saying that the action $drive(i95, south, 60)$ should be executed for 2 hours is a precise specification saying that the action "Drive south on Interstate I-95 at 60 mph" for 2 hours is a precise specification of action duration. However, it is hard to specify durations of actions such as $drive(washington, baltimore)$. In this case, the above definition requires an *estimate* to be provided.[4]

A further complication may arise when we consider the gv1, gv2, gv3 ground vehicle agents executing the action

$$drive(front\_royal, thornton, rte354)$$

saying that the vehicle in question is driving from *Front Royal* to *Thornton* along *Route 354*. Here, there may be no easy "formula" that allows us to specify where the vehicle is at a given instant of time, and furthermore, there may be no need to know that the vehicle has moved one mile further west along Interstate I-90 since the last report. The designer of the gv1 agent may be satisfied with knowing the location of the vehicle every 30 minutes.

3.1.1 Checkpoints

Thus, the notion of a *timed action* should allow the designer of an agent to specify the preconditions of an action, as well as *intermediate effects* that the action has prior to completion. That is, a given action α takes some time to execute. While α is executing, the state of the agent is changing. Checkpoints are used to model these intermediate changes. In particular, *Checkpoints* are time points when the agent's state is updated during execution of the action. For example, an action that takes 75 units of time starting at time 0 may have checkpoints every 15 units of time, i.e. at times 15,30,45,60, and 75. This means that every 15 time units, the state is updated. Note, that the execution of the action is not interrupted by the checkpoints. For example, consider the action $drive(front\_royal, thornton, rte354)$ which affects the location of the vehicle. Thus, in each checkpoint the state of the vehicle agent is changed, however, the agent does not stop its driving for doing so. Checkpoints bear a resemblance to "interrupts" in operating systems and microprocessors, as they require certain actions (i.e. state updates) to be taken when certain events (in this case, a clock tick indicating reaching a checkpoint) occur.

It is important to note that it is the *agent designer's responsibility* to specify checkpoints in a manner that satisfies his application's needs. For instance, if he needs to incorporate intermediate effects on a millisecond by millisecond basis, his checkpoints should be spaced out at each millisecond. The checkpoints can be specified using absolute times (indicated with abs) or relative to the beginning of the execution of the action (indicated with rel). We are now

[4] One could extend this to have uncertainty in duration. For instance, one might specify that it takes between 30 minutes and 120 minutes to drive from Washington to Baltimore and that a probability distribution δ gives us the probability, $\delta(t)$, that it will take exactly t minutes to do the trip. However, this would greatly complicate the framework proposed here and hence we defer this to future work.

ready to define checkpoint expressions.

Definition 3.1 (Checkpoint Expressions $\mathsf{rel}\!:\!\{\mathtt{X} \mid \chi\}, \mathsf{abs}\!:\!\{\mathtt{X} \mid \chi\}$**)**

- *If $i \in \mathbb{N}$ is a positive integer, then* $\mathsf{rel} : \{i\}$ *and* $\mathsf{abs} : \{i\}$ *are checkpoint expressions.*
- *If χ is a code call condition involving a non-negative, integer-valued variable* $\mathtt{X}$*, then* $\mathsf{rel}\!:\!\{\mathtt{X} \mid \chi\}$ *and* $\mathsf{abs}\!:\!\{\mathtt{X} \mid \chi\}$ *are* checkpoint expressions.

$\mathsf{rel}\!:\!\{i\}$ says that a checkpoint occurs every i units of time from the start of an action. $\mathsf{abs}\!:\!\{i\}$ says that a checkpoint occurs at time i. $\mathsf{rel}\!:\!\{\mathtt{X} \mid \chi\}$ says that for every possible value i of $\mathtt{X}$ that makes χ true in the current state, a checkpoint occurs every i units of time from the start of an action. Similarly, $\mathsf{abs}\!:\!\{\mathtt{X} \mid \chi\}$ says that a checkpoint occurs at every instance of $\mathtt{X}$ that makes $\mathtt{X} \mid \chi$ true. We use **cpe** as a metavariable for relative and absolute checkpoint expressions. The following example presents some simple checkpoint expressions.

Example 3.2 (Rescue: Checkpoints)

- $\mathsf{rel}\!:\!\{100\}$. This says that a checkpoint occurs at the time of the start of the action, 100 units later, 200 units later, and so on.
- $\mathsf{abs}\!:\!\{\mathtt{T} \mid \mathbf{in}(\mathtt{T}, \mathsf{clock}: \mathit{time}()) \,\&\, \mathbf{in}(\mathtt{0}, \mathsf{math}: \mathit{remainder}(\mathtt{T}, \mathtt{100})) \,\&\, \mathtt{T} > 5000\}$. This says that a checkpoint occurs at absolute times 5000, 5100, 5200, and so on.
- $\mathsf{abs}\!:\!\{\mathtt{T} \mid \mathbf{in}(\mathtt{T}, \mathsf{clock}: \mathit{time}()) \,\&\, \mathbf{in}(\mathtt{X}, \mathit{getMessage}(\mathsf{comc})) \,\&\, \mathtt{X.Time} - \mathtt{T} = 5\}$. This says that a checkpoint occurs at 5 time units after a message is received from the comc agent.

3.1.2 Timed Actions

While checkpoint expressions provide a convenient way of specifying a set of time points, timed effect triples (defined below) specify how to change a state when a checkpoint is encountered.

Definition 3.3 (Timed Effect Triple $\langle \mathbf{cpe}, \mathit{Add}, \mathit{Del} \rangle$**)** *A* timed effect triple *is a triple of the form* $\langle \mathbf{cpe}, \mathit{Add}, \mathit{Del} \rangle$ *where* **cpe** *is a checkpoint expression, and Add and Del are add lists and delete lists.*

Intuitively, if $\langle \mathbf{cpe}, \mathit{Add}, \mathit{Del} \rangle$ is associated with α, then the contents of the *Add* and *Del* lists are used to update the state of the agent at every time point specified by **cpe**. A couple of simple timed effect triples are shown below.

Example 3.4 (Rescue: Timed Effect Triples)

- The $\mathtt{truck}$ agent may use the following timed effect triple to update its fuel at absolute times 5000, 5100, 5200, and so on.

  1st arg :

  $\mathsf{abs}\colon\{\mathtt{T} \mid \mathbf{in}(\mathtt{T}, \mathtt{clock}\colon time()) \,\&\, \mathbf{in}(\mathtt{0}, \mathtt{math}\colon remainder(\mathtt{T}, \mathtt{100})) \,\&\, \mathtt{T} > 5000\}$

  2nd arg:$\{\mathbf{in}(\mathtt{NewFuelLevel}, \mathtt{truck}\colon fuelLevel(\mathtt{X}_{\mathsf{now}}))\}$

  3rd arg:$\{\mathbf{in}(\mathtt{OldFuelLevel}, \mathtt{truck}\colon fuelLevel(\mathtt{X}_{\mathsf{now}} - \mathsf{20}))\}$

We are now ready to define the concept of a timed action.

Definition 3.5 (Timed Action) *A* timed action α *consists of:*

Name: *A name, usually written* $\alpha(\mathtt{X}_1, \ldots, \mathtt{X}_n)$*, where the* $\mathtt{X}_i$*'s are root variables.*

Schema: *A schema, usually written as* $(\tau_1, \ldots, \tau_n)$*, of types. Intuitively, this says that the variable* $\mathtt{X}_i$ *must be of type* τ_i*, for all* $1 \leq i \leq n$*.*

Pre: *A code-call condition* χ*, called the* precondition *of the action, denoted by* $Pre(\alpha)$[5]

Dur: *An expression of the form* $\{i\}$ *or* $\{\mathtt{X} \mid \chi\}$*. Depending on the current object state, this expression determines a* duration $\mathsf{duration}(\alpha) \in \mathbb{N}$ *of* α*.* $\mathsf{duration}(\alpha)$ *is not used as an absolute time point but as a duration (length of a time interval).*

Tet: *A set* $\mathbf{Tet}(\alpha)$ *of timed effect triples such that if both* $\langle \mathbf{cpe}, Add, Del\rangle$ *and* $\langle \mathbf{cpe}', Add', Del'\rangle$ *are in* $\mathbf{Tet}(\alpha)$*, then* $\mathbf{cpe}$ *and* $\mathbf{cpe}'$ *have no common solution w.r.t. any object state. The set* $\mathbf{Tet}(\alpha)$ *together with* $\mathbf{Dur}(\alpha)$ *determines the set of checkpoints* $\mathsf{checkpoints}(\alpha)$ *for action* α *(as defined below).*

Intuitively, if α is an action that we start executing at $t^{\alpha}_{\text{start}}$, then $\mathbf{Dur}(\alpha)$ specifies how to compute the duration $\mathsf{duration}(\alpha)$ of α, and $\mathbf{Tet}(\alpha)$ specifies the checkpoints associated with action α. It is important to note that $\mathbf{Dur}(\alpha)$ and $\mathbf{Tet}(\alpha)$ may not specify the duration and checkpoint times explicitly (even if the associated checkpoints are of the form $\mathsf{abs}\colon\{\mathtt{X} \mid \chi\}$, i.e. absolute times). The method to compute $\mathsf{duration}(\alpha)$ is given below.

- If $\mathbf{Dur}(\alpha)$ is of the form $\{i\}$, then $\mathsf{duration}(\alpha) = i$.
- If $\mathbf{Dur}(\alpha)$ is of the form $\{\mathtt{X} \mid \chi\}$, then

[5] As in (Eiter, Subrahmanian, and Rogers 2000), we require that $Pre(\alpha)$ be *safe* modulo the variables $\mathtt{X}_1,\ldots,\mathtt{X}_n$, i.e., assuming the variables $\mathtt{X}_1,\ldots,\mathtt{X}_n$ are grounded, there must be some way in which the atoms in χ can be reordered so that the (reordered) version of χ can be evaluated from left to right. The formal definition of this is contained in (Eiter, Subrahmanian, and Rogers 2000) and is not required for this paper.

· If there is a solution is a solution of χ w.r.t. $\mathcal{O}_{\mathcal{S}}$ at time $t^{\alpha}_{\text{start}}$ then:

$$\mathsf{duration}(\alpha) = \min\{\|\mathtt{X}\theta - t^{\alpha}_{\text{start}}\| \mid \theta \text{ is a solution of } \chi \text{ w.r.t. } \mathcal{O}_{\mathcal{S}} \text{ at time } t^{\alpha}_{\text{start}} \text{ and } \mathtt{X}\theta \gneqq t^{\alpha}_{\text{start}}\}.$$

· Otherwise, $\mathsf{duration}(\alpha)$ is not defined with respect to $\mathcal{O}_{\mathcal{S}}$ at time $t^{\alpha}_{\text{start}}$.

Intuitively, the above definition says that we find solutions to χ which are greater than or equal to $t^{\alpha}_{\text{start}}$. Of such solutions, we pick the smallest—the duration of α is from α's start time, to the time point chosen in this way. If such a solution is not found, performing α is infeasible.

The set, $\mathsf{checkpoints}(\alpha)$, of checkpoints is the union of the following five sets:

- $\{t^{\alpha}_{\text{start}} + \mathsf{duration}(\alpha)\}$
- $\{\mathtt{X}\theta \mid \langle \mathsf{abs}\colon\{\mathtt{X} \mid \chi\}, \mathit{Add}, \mathit{Del}\rangle \in \mathbf{Tet}(\alpha) \text{ and } \theta \text{ is a solution of } \chi,\ \mathtt{X}\theta \gneqq t^{\alpha}_{\text{start}} \text{ and } \|\mathtt{X}\theta - t^{\alpha}_{\text{start}}\| \leq \mathsf{duration}(\alpha)\}$
- $\{i \mid \langle \mathsf{abs}\colon\{i\}, \mathit{Add}, \mathit{Del}\rangle \in \mathbf{Tet}(\alpha), i \gneqq t^{\alpha}_{\text{start}} \text{ and } \|i - t^{\alpha}_{\text{start}}\| \leq \mathsf{duration}(\alpha)\}$
- $\{t^{\alpha}_{\text{start}} + i \times j \mid \langle \mathsf{rel}\colon\{i\}, \mathit{Add}, \mathit{Del}\rangle \in \mathbf{Tet}(\alpha) \text{ and } i, j \in \mathbb{N}, i, j > 0 \text{ with } i \times j \leq \mathsf{duration}(\alpha)\ \}$
- $\{t^{\alpha}_{\text{start}} + i \times \mathtt{X}\theta \mid \langle \mathsf{rel}\colon\{\mathtt{X} \mid \chi\}, \mathit{Add}, \mathit{Del}\rangle \in \mathbf{Tet}(\alpha) \text{ and } \theta \text{ is a solution of } \chi \text{ and } i \in \mathbb{N}, i > 0 \text{ and } \|t^{\alpha}_{\text{start}} + i \times \mathtt{X}\theta\| \leq \mathsf{duration}(\alpha)\ \}.$

In other words, even though $\mathbf{Tet}(\alpha)$ may imply the existence of infinitely many checkpoints, only those that occur at or before the scheduled completion of the action α are considered to be valid checkpoints. In addition, the last time period of executing an action is always a checkpoint.

Example 3.6 (Rescue: Timed Actions) Returning to the Rescue example, we have the following timed action *drive*() of the truck agent which may be described via the following components:

- **Name:** $\mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway})$
- **Schema:** (String,String,String)
- **Pre:** $\mathbf{in}(\mathtt{From}, \mathtt{truck}\colon \mathit{location}())$
- **Dur:** $\{\mathtt{T} \mid \mathbf{in}(\mathtt{X}, \mathtt{math}\colon \mathit{distance}(\mathtt{From}, \mathtt{To})) \,\&\, \mathbf{in}(\mathtt{T}, \mathtt{math}\colon \mathit{compute}(\frac{60\mathtt{X}}{70}))\}$
- **Tet:**

  1st arg : $\mathsf{rel}\colon\{20\}$

  2nd arg :$\{\mathbf{in}(\mathtt{NewPosition}, \mathtt{truck}\colon \mathit{location}(\mathtt{X}_{\mathsf{now}}))\ \}$

  3rd arg :$\{\mathbf{in}(\mathtt{OldPosition}, \mathtt{truck}\colon \mathit{location}(\mathtt{X}_{\mathsf{now}} - 20))\ \}$

The **Tet** part says that the truck agent updates its location every 20 minutes (assuming a time period is equal to 1 minute) during the expected time it takes it to drive the distance between From to To at 70km per hour.

An agent may often base its actions (current and future) not only on its current/past state, but also on its current/past actions. Thus, we need to be able to specify temporal conditions involving an agent's state and actions. This is done via the concept of a temporal action state conjunct defined below.

Definition 3.7 (Temporal Annotation Item tai)

(1) Every integer is a temporal annotation item.
(2) The distinguished integer valued variable $\mathtt{X}_{now}$ is a temporal annotation item.
(3) Every integer valued variable is a temporal annotation item.
(4) If $\mathsf{tai}_1, \ldots, \mathsf{tai}_n$ are temporal annotation items, and $b_1, \ldots, b_n$ are integers (positive or negative), then $(b_1 \mathsf{tai}_1 + \ldots + b_n \mathsf{tai}_n)$ is a temporal annotation item.

For example, 1, $\mathtt{X}_{\mathsf{now}}$, $\mathtt{X}_{\mathsf{now}} + 3$, $\mathtt{X}_{\mathsf{now}} + 2v + 4$ are all temporal annotation items if v is an integer valued variable. Temporal annotation items, when ground, evaluate to time points. They are used to specify a time interval.

Definition 3.8 (Temporal Annotation $[\mathsf{tai}_1, \mathsf{tai}_2]$) *If $\mathsf{tai}_1, \mathsf{tai}_2$ are annotation items, then $[\mathsf{tai}_1, \mathsf{tai}_2]$ is a temporal annotation.*

For example, $[2, 5]$ is a temporal annotation item describing the set of time points between 2 and 5 (inclusive). $[2, 3\mathtt{X} + 4\mathtt{Y}]$ is a temporal annotation item. When $\mathtt{X} := 2, \mathtt{Y} := 3$, this defines the set of time points between 2 and 18. $[\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}} + 5]$ is a temporal annotation item. When $\mathtt{X}_{\mathsf{now}} := 10$, this specifies the set of time points between 10 and 15.

Definition 3.9 ((Temporal) Action State Condition) *Suppose χ is a (possibly empty) code call condition, $L_1, \ldots, L_n$ are action status literals, and* ta *is a temporal annotation. Then:*

(1) $(\chi \,\&\, L_1 \,\&\, \ldots \,\&\, L_n)$ *is called an* action state condition.
(2) $(\chi \,\&\, L_1 \,\&\, \ldots \,\&\, L_n) : \mathsf{ta}$ *is called a* temporal action state condition *(tasc).*
(3) If χ is empty, then $(\chi \,\&\, L_1 \,\&\, \ldots \,\&\, L_n) : \mathsf{ta}$ *is called a* state-independent tasc *. Otherwise, it is called* state-dependent tasc .

For any tasc $\varrho : \mathsf{ta}$, we denote by $B(\varrho : \mathsf{ta})$, the collection of action status literals in ϱ; by $B^-(\varrho : \mathsf{ta})$ we denote the negative literals in $B(\varrho : \mathsf{ta})$, and by $B^+(\varrho : \mathsf{ta})$ the positive literals in $B(\varrho : \mathsf{ta})$. Moreover, $\neg.B^-(\varrho : \mathsf{ta})$ denotes the status atoms whose negations occur in $B^-(\varrho : \mathsf{ta})$.

Intuitively, when $\varrho : \mathtt{ta}$ is ground for some action state condition ϱ, we may read this as "ϱ is true at some point in $\mathtt{ta}$". The following is a simple tasc.

- $(\mathbf{in}(\mathtt{X}, \mathtt{heli}: \mathit{inventory}(\mathit{fuel})) \,\&\, \mathtt{X.Qty} < 50) : [\mathtt{X}_{\mathsf{now}} - 10, \mathtt{X}_{\mathsf{now}}]$ Intuitively, this tasc is true if at some point in time t_i in the last 10 time units, the helicopter had less than 50 gallons of fuel left.

We are now ready to define the most important syntactic construct of this paper, a *temporal agent rule.*

Definition 3.10 (Temporal Agent Rule/Program $\mathcal{TP}$) *A* temporal agent rule *is an expression of the form* $\mathit{Op}\,\alpha : [\mathit{tai}_1, \mathit{tai}_2] \;\leftarrow\; \varrho_1 : \mathtt{ta}_1 \,\&\cdots\&\, \varrho_n : \mathtt{ta}_n$, *where* $\mathit{Op} \in \{\mathbf{P}, \mathbf{Do}, \mathbf{F}, \mathbf{O}, \mathbf{W}\}$, *and* $\varrho_1 : \mathtt{ta}_1, \ldots, \varrho_n : \mathtt{ta}_n$ *are* tascs*. A* temporal agent program *is a finite set of temporal agent rules.*

Intuitive Reading of Temporal Agent Rule

"If for all $1 \le i \le n$*, there exists a time point* t_i *such that* ϱ_i *is true at time* t_i *such that either*

(1) ϱ_i *is state independent and* $t_i \in \mathtt{ta}_i$*, or*

(2) ϱ_i *is state dependent and* $t_i \le \mathtt{t}_{\mathit{now}}$ *(i.e.* t_i *is now or is in the past) and* $t_i \in \mathtt{ta}_i$*,*

then $\mathit{Op}\,\alpha$ *is true at some point* $t \ge \mathtt{t}_{\mathit{now}}$ *(i.e. now or in the future) such that* $\mathit{tai}_1 \le t \le \mathit{tai}_2$*".*

This reading of a $\mathtt{tap}$ rule allows us to avoid several problems. The requirement that $t \ge \mathtt{t}_{\mathsf{now}}$ is to prevent cases in which something that becomes true now will lead to obligations in the past. In other words, the antecedent of a rule always refers to *past or current states of the world, and past action status atoms*, and the obligations, permissions, forbidden actions that are implied by rules apply to the *future.* Note that this framework is completely compatible with basing actions on *predictions* about the future, because such predictions are made *now* and hence are statements about the future in the current state!

The agent knows its current state and its states in the past. It is very difficult to predict its future states, which may depend on future actions of itself and others. Thus, we require that if ϱ_i is state dependent it will be true either now or in the past to make the rule fireable. On the other hand, agents are familiar with its future obligations, permissions etc., so it can evaluate state independent atoms in the future. Thus, we allow $t_i > \mathtt{t}_{\mathsf{now}}$ for $t_i \in \mathtt{ta}_i$.

We close this section by showing how $\mathtt{taps}$ can be used to express the two rules introduced in the introduction of this paper. We use two relational databases— one called `shipdata` containing at least the attributes `shiptime,orig,dest` (and perhaps other ones as well) which specifies data (such as shipping time) associated with past shipments. The other relational table is called `sched` which has at least the attributes `reqtime,place,item` specifying which items

are required at what time by what places.
$\mathbf{Do}\, ship(\mathtt{P},\mathtt{A},\mathtt{B}) : [\mathtt{T} - \mathtt{T_1} - 10, \mathtt{T} - \mathtt{T_1}] \leftarrow$

$$(\mathbf{in}(\mathtt{T_1}, \mathtt{db} : sql(\text{`}\mathtt{SELECTMAX\ shiptime\ FROM\ shipdata\ WHERE\ orig} = \mathtt{A}\&\, \mathtt{dest} = \mathtt{B}'))) \,\&$$

$$\mathbf{in}(\mathtt{T}, \mathtt{db} : sql(\text{`}\mathtt{SELECT\ reqtime\ FROM\ place\ WHERE\ item} = \mathtt{P}'))) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}].$$

The second example in the introduction may be modeled as follows. We assume a prediction package that given a stock uses (some stock expertise) to predict the change in the value of the stock at future time points. This function returns a set of pairs of the form (T, C). Intuitively, this says that T units from now, the stock price will change by C percent (positive or negative).

$$\mathbf{Do}\, buy(\mathtt{S}) : [\mathtt{X}_{\mathsf{now}} + \mathtt{X.T} - 2, \mathtt{X}_{\mathsf{now}} + \mathtt{X.T} - 2] \leftarrow$$

$$(\mathbf{in}(\mathtt{X}, \mathtt{pred} : dest(\mathtt{S})) \,\&\, \mathtt{X.C} \geq 25) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}].$$

4 Semantics of taps

In this section, we provide a formal semantics for taps, building upon the informal intuitions provided in the preceding section.

First and foremost, we reiterate that in our framework, we use the natural numbers to represent time. In classical temporal logics, a temporal interpretation associates a set of ground atoms with each time point. In our framework, things are somewhat more complex. This is because at any given point t in time, certain things are true in an agent's state, and certain action status atoms are true as well. Thus, we introduce *two* temporal structures: (1) a *temporal status set*, which captures actions, and (2) a *history*.

4.1 Temporal Status Set

A temporal status set extends the notion of a status set in much the same way as a temporal interpretation extends the classical logical notion of an interpretation (Lloyd 1987).

Definition 4.1 (Temporal Status Set $\mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}$) *A temporal status set $\mathcal{TS}_{\mathtt{t}_{now}}$ at time $\mathtt{t}_{now}$ is a mapping from natural numbers to ordinary status sets satisfying $\mathcal{TS}_{\mathtt{t}_{now}}(i) = \emptyset$ for all $i > i_0$ for some $i_0 \in \mathbb{N}$.*

Intuitively, if $\mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}(3) = \{\mathbf{O}\alpha, \mathbf{Do}\,\alpha, \mathbf{P}\alpha, \mathbf{F}\beta\}$, then this means that according to the temporal status set $\mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}$, at time instant 3, α is obligatory/done/permitted, while β is forbidden. Similarly, if $\mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}(4) = \{\mathbf{P}\alpha\}$ then according to the temporal status set $\mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}$, at time 4, α is permitted.

As an agent that reasons about time may need to reason about the current, as well as past states it was/is in, a notion of state history is needed.

Definition 4.2 (State History Function $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}$) *A state history function $\mathsf{hist}_{\mathsf{t}_{now}}$ at time $\mathtt{t}_{now}$ is a partial function from $\mathbb{N}$ to agent states such that $\mathsf{hist}_{\mathsf{t}_{now}}(\mathtt{t}_{now})$ is always defined and for all $i > \mathtt{t}_{now}$, $\mathsf{hist}_{\mathsf{t}_{now}}(i)$ is undefined.*

The definition of state history does not *require* that an agent store the entire past. For many agent applications, storing the entire past may be neither necessary nor desirable. The definition of state history function above merely requires that the agent stores the current agent state—which past agent states are to be stored is the choice of the agent designer. Furthermore, an agent cannot store future states, though it can schedule actions for the future (in its current state) and it may have beliefs (in its current state) about the future. Thus, the designer of an agent may make decisions such as those given below:

(1) He may decide to store no past information at all. In this case, $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}(i)$ is defined *if and only if* $i = \mathtt{t}_{\mathsf{now}}$.
(2) He may decide to store information only about the past i units of time. This means that he stores the agent's state at times $\mathtt{t}_{\mathsf{now}}, (\mathtt{t}_{\mathsf{now}} - 1), \ldots,$ $(\mathtt{t}_{\mathsf{now}} - i)$, i. e. $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}$ is defined for the following arguments: $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}(\mathtt{t}_{\mathsf{now}})$, $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}(\mathtt{t}_{\mathsf{now}} - 1), \ldots, \mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}(\mathtt{t}_{\mathsf{now}} - i)$ are defined.
(3) He may decide to store, in addition to the current state, the history every five time units. That is, $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}(\mathtt{t}_{\mathsf{now}})$ is defined and for each $0 \leq i \leq \mathtt{t}_{\mathsf{now}}$, if $i \mod 5 = 0$, then $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}(i)$ is defined. Such an agent may be specified by an agent designer when he believes that maintaining some (but not all) past snapshots is adequate for his application's needs.

Suppose we are given a temporal status set $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$ and a state history function, $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}$. We define below, what it means for a triple consisting of $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$, $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}$ and the current time, $\mathtt{t}_{\mathsf{now}}$, to *satisfy* a tap.

Definition 4.3 (Satisfaction, Closure of $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$ under tap Rules) *Suppose $\mathtt{t}_{now}$ is any integer. We present below, an inductive definition of satisfaction of formulas by $\langle \mathcal{T}S_{\mathsf{t}_{now}}, \mathsf{hist}_{\mathsf{t}_{now}}, \mathtt{t}_{now}\rangle$;*

(1) (a) State independent tasc:
$\langle \mathcal{T}S_{\mathsf{t}_{now}}, \mathsf{hist}_{\mathsf{t}_{now}}, \mathtt{t}_{now}\rangle \models^{\mathcal{O}}_{\mathsf{temp}} (L_1 \,\&\, \ldots \,\&\, L_n) : [\mathsf{tai}_1, \mathsf{tai}_2]$ where $L_1 \,\&\, \ldots \,\&\, L_n :$ $[\mathsf{tai}_1, \mathsf{tai}_2]$ is ground if there is an integer i, $\mathsf{tai}_1 \leq i \leq \mathsf{tai}_2$ such that $B^+((L_1 \,\&\, \ldots \,\&\, L_n) : [\mathsf{tai}_1, \mathsf{tai}_2]) \subseteq \mathcal{T}S_{\mathsf{t}_{now}}(i)$ and for every $L \in \neg.B^-((L_1 \,\&\, \ldots \,\&\, L_n) : [\mathsf{tai}_1, \mathsf{tai}_2])$ $L \notin \mathcal{T}S_{\mathsf{t}_{now}}(i)$. In this case, i is said to witness *the truth of this tasc.*
(b) General tasc:
$\langle \mathcal{T}S_{\mathsf{t}_{now}}, \mathsf{hist}_{\mathsf{t}_{now}}, \mathtt{t}_{now}\rangle \models^{\mathcal{O}}_{\mathsf{temp}} (\chi \,\&\, L_1 \,\&\, \ldots \,\&\, L_n) : [\mathsf{tai}_1, \mathsf{tai}_2]$ where the conjunction $\chi \,\&\, L_1 \,\&\, \ldots \,\&\, L_n : [\mathsf{tai}_1, \mathsf{tai}_2]$ is ground if there is an

integer i, $\mathit{tai}_1 \leq i \leq \mathit{tai}_2$ such that $\mathit{hist}_{\mathtt{t}_{now}}(i)$ is defined and χ is true in the agent state $\mathit{hist}_{\mathtt{t}_{now}}(i)$ and $B^+((L_1 \,\&\, \ldots \,\&\, L_n) : [\mathit{tai}_1, \mathit{tai}_2]) \subseteq \mathcal{TS}_{\mathtt{t}_{now}}(i)$ and for every $L \in \neg.B^-((L_1 \,\&\, \ldots \,\&\, L_n) : [\mathit{tai}_1, \mathit{tai}_2])$, $L \notin \mathcal{TS}_{\mathtt{t}_{now}}(i)$. In this case, i is said to witness *the truth of this* tasc.

(2) $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} \mathit{Op}\,\alpha : [\mathit{tai}_1, \mathit{tai}_2] \leftarrow \varrho_1 : \mathtt{ta}_1 \,\&\, \ldots \,\&\, \varrho_n : \mathtt{ta}_n$ *(where the rule is ground) if either:*

   (a) *there exists an $1 \leq i \leq n$ such that either (1) ϱ_i is state independent and for all $t_i \in \mathtt{ta}_i$, t_i is not a witness to the truth of $\varrho_i : \mathtt{ta}_i$ by $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle$, or (2) ϱ_i is state dependent and for all $t_i \leq \mathtt{t}_{now}$ and $t_i \in \mathtt{ta}_i$, t_i is not a witness to the truth of $\varrho_i : \mathtt{ta}_i$ by $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle$, or*

   (b) *there exists a $t_j \geq \mathtt{t}_{now}$ such that $t_j \in [\mathit{tai}_1, \mathit{tai}_2]$ and $\mathit{Op}\,\alpha \in \mathcal{TS}_{\mathtt{t}_{now}}(t_j)$.*

   If a temporal agent rule r is not ground, $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} r$ if for all ground instances of the rule r', $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} r'$.

(3) $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} (\forall x)\phi$ *if* $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} \phi[x/s]$ *for all ground terms s.*[6]

(4) $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} (\exists x)\phi$ *if* $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} \phi[x/s]$ *for some ground term s.*

(5) $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} \mathcal{TP}$ *where $\mathcal{TP}$ is a* tap *if for each temporal agent rule (*tar*) $r \in \mathcal{TP}$: $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} r$.*

Instead of $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathit{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle \models^{\mathcal{O}}_{\mathsf{temp}} \mathcal{TP}$ we also say "$\mathcal{TS}_{\mathtt{t}_{now}}$ is closed under the program rules of $\mathcal{TP}$".

The definition of satisfaction by $\langle \mathcal{TS}_{\mathtt{t}_{now}}, \mathsf{hist}_{\mathtt{t}_{now}}, \mathtt{t}_{now} \rangle$ is complex. In particular, item (2) in the above definition has subtle aspects to it. We illustrate some of these subtleties by revisiting the rescue example.

Example 4.4 (Rescue: Temporal Status Set) Consider the following very simple table, describing a temporal status set, $\mathcal{TS}_{\mathtt{t}_{now}}$ of the `truck` agent.

[6] Here $\phi[x/s]$ denotes the replacement of all free occurrences of x in ϕ by ground term s.

i	$\mathcal{T}S_{\mathsf{t_{now}}}(i)$
0	$\{\mathbf{F}\,drive(was, bal, hw95), \mathbf{F}\,drive(was, bal, hw295),$ $\mathbf{O}\,fill\_fuel(), \mathbf{Do}\,fill\_fuel()\}$
1	$\{\mathbf{P}\,drive(was, bal, hw95), \mathbf{F}\,drive(was, bal, hw295), \mathbf{F}\,fill\_fuel()\}$
2	$\{\mathbf{P}\,drive(was, bal, hw95), \mathbf{F}\,drive(was, bal, hw295)\}$
3	$\{\mathbf{O}\,drive(was, bal, hw95), \mathbf{Do}\ drive(was, bal, hw95),$ $\mathbf{F}\,drive(was, bal, hw295), \mathbf{P}\,drive(was, bal, hw95),$ $\mathbf{O}\,order\_item(fa\_bag), \mathbf{Do}\ order\_item(fa\_bag)\}$
4	$\{\mathbf{P}\,drive(was, bal, hw95), \mathbf{Do}\ drive(was, bal, hw95),$ $\mathbf{F}\,drive(was, bal, hw295), \mathbf{Do}\,fill\_fuel()\}$
$4 < i < 9$	$\{\mathbf{P}\,drive(was, bal, hw295), \mathbf{F}\,drive(was, bal, hw295)\}$
$i > 9$	$\emptyset$

Suppose we also consider the very simple table describing the state of the `truck` agent.

i	$\mathsf{hist}_{\mathsf{t_{now}}}(\mathbf{i})$
0	$\mathbf{in}(\mathtt{hw295}, \mathtt{msgbox}: gatherWarning(\mathtt{comc})), \mathbf{in}(\mathbf{true}, \mathtt{truck}: tank\_empty())$
1	$\mathbf{in}(\mathbf{false}, \mathtt{truck}: tank\_empty())$
2	$\mathbf{in}(\mathbf{false}, \mathtt{truck}: tank\_empty()), \mathbf{in}(2, \mathtt{truck}: inventory(fa\_bag)),$
3	$\mathbf{in}(\mathtt{1}, \mathtt{truck}: inventory(fa\_bag)), \mathbf{in}(\mathbf{false}, \mathtt{truck}: tank\_empty())$

Suppose $\mathsf{t_{now}} = 3$. Let us examine some simple ground formulas and see whether $\langle \mathcal{T}S_{\mathsf{t_{now}}}, \mathsf{hist}_{\mathsf{t_{now}}}, \mathsf{t_{now}} \rangle$ satisfies these formulas.

- $\big(\,\mathbf{in}(2, \mathtt{truck}: inventory(fa\_bag))\ \ \&$
  $\mathbf{in}(\mathbf{false}, \mathtt{truck}: tank\_empty())$
  $\big) : [\mathsf{t_{now}} - 3, \mathsf{t_{now}}]$.

  This formula is satisfied by $\langle \mathcal{T}S_{\mathsf{t_{now}}}, \mathsf{hist}_{\mathsf{t_{now}}}, \mathsf{t_{now}} \rangle$ because $i = 2$ is a witness to the satisfaction of this formula.
- The rule

  $\mathbf{F}\ drive(was, bal, hw295){:}[\mathsf{t_{now}}{,}\mathsf{t_{now}}\ {+}3] \leftarrow$
  $\quad \mathbf{in}(hw295, \mathtt{msgbox}: gatherWarning(\mathtt{comc})) : [\mathsf{t_{now}} - 3, \mathsf{t_{now}}]$

  is satisfied by $\langle \mathcal{T}S_{\mathsf{t_{now}}}, \mathsf{hist}_{\mathsf{t_{now}}}, \mathsf{t_{now}} \rangle$ because its antecedent is satisfied by it (witness $i = 0 < \mathsf{t_{now}}$) and its consequent is true at a future time instant, viz. at time $3 \geq \mathtt{t}_{\mathsf{now}}$.

- Consider the following tap

(1) $\mathbf{F}\, drive(was, bal, hw295) : [\mathtt{t}_{\mathsf{now}}, \mathtt{t}_{\mathsf{now}} + 2] \leftarrow$
$\quad\quad \mathbf{in}(hw295, \mathtt{msgbox} : gatherWarning(\mathtt{comc})) : [\mathtt{t}_{\mathsf{now}} - 3, \mathtt{t}_{\mathsf{now}}]$

(2) $\mathbf{Do}\, \mathit{fill\_fuel}() : [\mathtt{t}_{\mathsf{now}}, \mathtt{t}_{\mathsf{now}}] \leftarrow$
$\quad\quad \mathbf{in}(\mathbf{true}, \mathtt{truck} : tank\_empty()) : [\mathtt{t}_{\mathsf{now}} - 2, \mathtt{t}_{\mathsf{now}}]$

(3) $\mathbf{O}\, order\_item(fa\_bag) : [\mathtt{t}_{\mathsf{now}}, \mathtt{t}_{\mathsf{now}} + 4] \leftarrow$
$\quad\quad \mathbf{in}(1, \mathtt{truck} : inventory(fa\_bag))[\mathtt{t}_{\mathsf{now}} - 3, \mathtt{t}_{\mathsf{now}}]$

(4) $\mathbf{P}\, drive(was, bal, hw95) : [\mathtt{t}_{\mathsf{now}}, \mathtt{t}_{\mathsf{now}}] \leftarrow$
$\quad\quad \mathbf{in}(\mathbf{false}, \mathtt{truck} : tank\_empty()) : [\mathtt{t}_{\mathsf{now}}, \mathtt{t}_{\mathsf{now}}]\ \&$
$\quad\quad \mathbf{F}\, drive(was, bal, hw295) : [\mathtt{t}_{\mathsf{now}} + 1, \mathtt{t}_{\mathsf{now}} + 2]$

This tap is satisfied by $\langle \mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}, \mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}, \mathtt{t}_{\mathsf{now}} \rangle$ as all its temporal agent rules are satisfied. The first rule is satisfied by $\langle \mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}, \mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}, \mathtt{t}_{\mathsf{now}} \rangle$, because its antecedent is satisfied by a witness $i = 3 \leq \mathtt{t}_{\mathsf{now}}$, and its consequent is satisfied as $\mathbf{F}\, drive(was, bal, hw295) \in \mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(i \geq 3)$.

The second rule is satisfied by $\langle \mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}, \mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}, \mathtt{t}_{\mathsf{now}} \rangle$, because its antecedent is not satisfied—$\mathbf{in}(\mathbf{true}, \mathtt{truck} : tank\_empty()) \notin \mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}(2 \leq i \leq \mathtt{t}_{\mathsf{now}})$.

The third rule is satisfied by $\langle \mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}, \mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}, \mathtt{t}_{\mathsf{now}} \rangle$, because its antecedent is satisfied via witness i=3 $\leq \mathtt{t}_{\mathsf{now}}$ and its consequent is satisfied because $\mathbf{O}\, order\_item(fa\_bag) \in \mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(3)$.

Finally, the fourth rule is satisfied by $\langle \mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}, \mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}, \mathtt{t}_{\mathsf{now}} \rangle$ since its antecedent is satisfied and its consequent is satisfied. The first tasc of the antecedent, $\mathbf{in}(\mathbf{false}, \mathtt{truck} : tank\_empty()) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$, is satisfied via a witness i=3 $\leq \mathtt{t}_{\mathsf{now}}$. The second tasc, $\mathbf{F}\, drive(was, bal, hw295) : [\mathtt{t}_{\mathsf{now}}{+}1, \mathtt{t}_{\mathsf{now}}{+}2]$, is state independent and is satisfied as $\mathbf{F}\, drive(was, bal, hw295) \in \mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(4)$ The rule's consequent is satisfied as $\mathbf{P}\, drive(was, bal, hw95) \in \mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(3)$.

An agent may record not only its state history, but also 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 4.5 (Action History) *An* action history $\mathit{acthist}_{\mathtt{t}_{now}}$ *for an agent is a partial function from* $\mathbb{N}$ *to status sets satisfying* $\mathit{acthist}_{\mathtt{t}_{now}}(i) = \emptyset$ *for all* $i > i_0$ *for a* $i_0 \in \mathbb{N}$.

Intuitively, an action history specifies not only what the agent has done in the past, but also what an agent is *obliged/permitted* to or *forbidden from* doing in the future. In this respect, an action history is different from a state history.

An action history and a temporal status set both make statements about action status atoms. The following definition specifies what it means for the two to be compatible.

Definition 4.6 (History-Compatible Temporal Status Set) *Suppose the current time is* $\mathtt{t}_{now}$ *and* $\mathit{acthist}_{\mathtt{t}_{now}}(\cdot)$ *denotes the action history of an agent,* *and suppose* $\mathcal{TS}_{\mathtt{t}_{now}}$ *is a temporal status set.* $\mathcal{TS}_{\mathtt{t}_{now}}$ *is said to be* action history-compatible *at time* $\mathtt{t}_{now}$ *if for all* $i < \mathtt{t}_{now}$*, if* $\mathsf{acthist}_{\mathtt{t}_{now}}(i)$ *is defined, then* $\mathcal{TS}_{\mathtt{t}_{now}}(i) = \mathsf{acthist}_{\mathtt{t}_{now}}(i)$*, and for all* $i \geq \mathtt{t}_{now}$*, if* $\mathsf{acthist}_{\mathtt{t}_{now}}(i)$ *is defined, then* $\mathsf{acthist}_{\mathtt{t}_{now}}(i) \subseteq \mathcal{TS}_{\mathtt{t}_{now}}(i)$*.*

In other words, for a temporal status set 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. An example illustrating this kind of compatibility is given below.

Example 4.7 (Rescue: Action History)

$\mathbf{i}$	$\mathsf{acthist}_3(\mathbf{i})$
0	$\{\mathbf{F}\,drive(was, bal, hw95), \mathbf{F}\,drive(was, bal, hw295), \mathbf{O}\mathit{fill\_fuel}(), \mathbf{Do}\,\mathit{fill\_fuel}()\}$
1	$\{\mathbf{P}\,drive(was, bal, hw95), \mathbf{F}\,drive(was, bal, hw295), \mathbf{F}\mathit{fill\_fuel}()\}$
3	$\{\mathbf{O}\,drive(was, bal, hw95), \mathbf{Do}\ drive(was, bal, hw295), \mathbf{F}\,drive(was, bal, hw295),\}$
$4 \leq i < 9$	$\{\mathbf{F}\,drive(was, bal, hw295)\}$

The temporal status set, $\mathcal{TS}_{\mathtt{t}_{now}}$ presented in Example 4.4 is history-compatible with the above action history at time $\mathtt{t}_{\mathsf{now}} = 3$.

Given the agent's current temporal status set and its plans about the future, it has some expectation about how its future states will change over time. This leads to the notion of expected states.

Definition 4.8 (Expected States at time t: $\mathcal{EO}(\mathtt{t})$) *Suppose the current time is* $\mathtt{t}_{now}$*,* $\mathsf{hist}_{\mathtt{t}_{now}}$ *is the agent's state history function and* $\mathcal{TS}_{\mathtt{t}_{now}}$ *is a temporal status set. The agent's expected states are defined as follows:*

- $\mathcal{EO}(\mathtt{t}_{now}) = \mathsf{hist}_{\mathtt{t}_{now}}(\mathtt{t}_{now})$.
- *For all time points* $i > \mathtt{t}_{now}$*,* $\mathcal{EO}(i)$ *is the result of concurrently executing*

$$\{\alpha \mid \mathbf{Do}\,\alpha \in TS_{now}(i-1)\} \cup$$
$$\{\beta' \mid \mathbf{Do}\,\beta \in TS_{now}(j)\ \mathit{for}\ j \leq i-1 \text{ and } i-1 \text{ is a checkpoint for } \beta, \text{ and } \beta' \text{ denotes the action (non-timed) which has an empty precondition, and whose add and del lists are as specified by } \mathbf{Tet}(\beta)\}$$

in state $\mathcal{EO}(i-1)$.

We note that that from a certain $i_0 \in \mathbb{N}$ *onwards, we have* $\mathcal{EO}(i) = \emptyset$ *for all* $i > i_0$ *(this is because of the same property for the action history and the temporal status set).*

Example 4.9 (Rescue: Expected States) Suppose, $\mathtt{t}_{\mathsf{now}} = 1$,

$$\begin{aligned}
\mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}(0) &= \{\mathbf{in}(\mathit{was}, \mathtt{truck}: \mathit{location}()), \mathbf{in}(\mathbf{true}, \mathtt{truck}: \mathit{tank_empty}()),\\
&\quad \mathbf{in}(\mathit{empty}, \mathtt{truck}: \mathit{load}(\mathtt{0}))\\
\mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}(1) &= \{\mathbf{in}(\mathit{was}, \mathtt{truck}: \mathit{location}()), \mathbf{in}(\mathbf{true}, \mathtt{truck}: \mathit{tank_empty}()),\\
&\quad \mathbf{in}(\mathit{empty}, \mathtt{truck}: \mathit{load}(\mathtt{0}))
\end{aligned}$$

$$\begin{aligned}
\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(0) &= \{\mathbf{Do}\, \mathit{load_truck}(\mathtt{was})\}\\
\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(1) &= \{\mathbf{Do}\, \mathit{fill_fuel}()\}\\
\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(10) &= \{\mathbf{Do}\, \mathit{order_item}(\mathit{fa_bag})\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{EO}(2) &= \{\mathbf{in}(\mathtt{empty}, \mathtt{truck}: \mathit{load}(\mathtt{0})), \mathbf{in}(\mathit{was}, \mathtt{truck}: \mathit{location}()),\\
&\quad \mathbf{in}(\mathbf{false}, \mathtt{truck}: \mathit{tank_empty}())\}
\end{aligned}$$

For, $3 \leq i \leq 5$, $\mathcal{EO}(i) = \mathcal{EO}(2)$.

$$\begin{aligned}
\mathcal{EO}(6) = \{&\mathbf{in}(\mathit{half_loaded}, \mathtt{truck}: \mathit{load}(\mathtt{5})), \mathbf{in}(\mathit{was}, \mathtt{truck}: \mathit{location}()),\\
&\mathbf{in}(\mathbf{false}, \mathtt{truck}: \mathit{tank_empty}())\}
\end{aligned}$$

For, $7 \leq i \leq 10$, $\mathcal{EO}(i) = \mathcal{EO}(6)$.

$$\begin{aligned}
\mathcal{EO}(11) = \{\, &\mathbf{in}(\mathit{loaded}, \mathtt{truck}: \mathit{load}(\mathtt{10})), \mathbf{in}(\mathit{was}, \mathtt{truck}: \mathit{location}()),\\
&\mathbf{in}(\mathbf{false}, \mathtt{truck}: \mathit{tank_empty}()),\\
&\mathbf{in}(\langle \mathit{fa_bag}, \mathtt{10}\rangle, \mathtt{msgbox}: \mathit{supplier_to_be_notified}())\}.
\end{aligned}$$

For, $i > 11$, $\mathcal{EO}(i) = \mathcal{EO}(11)$.

It is apparent that given a temporal agent program, and a state/action history associated with that tap, temporal status sets must satisfy some "feasibility" requirements in order for them to be considered to represent the semantics of the tap in question. We are now ready to address the issue of what constitutes a feasible temporal status set.

Let us consider an agent $\mathfrak{a}$ that uses a temporal agent program tap to determine what actions it should take, and when it should take these actions. Let the current time be $\mathsf{t_{now}}$ and suppose $\mathsf{hist}_{\mathsf{t_{now}}}(\cdot), \mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$ represent the state and action histories associated with this agent at time $\mathsf{t_{now}}$.

Given a set S of action status atoms, let $\mathbf{D\text{-}Cl}(S)$ be the smallest superset S' of S such that $\mathbf{O}\alpha \in S' \to \mathbf{P}\alpha \in S'$. Likewise, let $\mathbf{A\text{-}Cl}(S)$ be the smallest superset S^* of S such that (i) $\mathbf{O}\alpha \in S^* \to \mathbf{Do}\,\alpha \in S^*$ and (ii) $\mathbf{Do}\,\alpha \in S^* \to \mathbf{P}\alpha \in S^*$. We say that set S is *deontically closed* iff $S = \mathbf{D\text{-}Cl}(S)$ and *action closed* iff $S = \mathbf{A\text{-}Cl}(S)$.

Definition 4.10 (Temporal Deontic Consistency) *Suppose $\mathit{hist}_{\mathsf{t}_{now}}$ is the agent's state history function. $\mathcal{TS}_{\mathsf{t}_{now}}$ is said to be* temporally deontically consistent *at time t_{now} if it satisfies the following conditions:*

- *For all time points i, (1) $\mathbf{O}\alpha \in \mathcal{TS}_{\mathsf{t}_{now}}(i) \to \mathbf{W}\alpha \notin \mathcal{TS}_{\mathsf{t}_{now}}(i)$; (2) $\mathbf{P}\alpha \in \mathcal{TS}_{\mathsf{t}_{now}}(i) \to \mathbf{F}\alpha \notin \mathcal{TS}_{\mathsf{t}_{now}}(i)$;*
- *For all $i \leq \mathsf{t}_{now}$, if $\mathbf{P}\alpha \in \mathcal{TS}_{\mathsf{t}_{now}}(i)$ and $\mathit{hist}_{\mathsf{t}_{now}}(i)$ is defined, then $\mathit{hist}_{\mathsf{t}_{now}}(i) \models Pre(\alpha)$ and* $\mathsf{duration}(\alpha)$ *is defined with respect to $\mathit{hist}_{\mathsf{t}_{now}}(i)$ and i.*
- *For all $i > \mathsf{t}_{now}$, if $\mathbf{P}\alpha \in \mathcal{TS}_{\mathsf{t}_{now}}(i)$, then $\mathcal{EO}(i) \models Pre(\alpha)$ and* $\mathsf{duration}(\alpha)$ *is defined with respect to $\mathcal{EO}(i)$ and i.*

Thus, if $\mathcal{TS}_{\mathsf{t_{now}}}(4) = \{\mathbf{Do}\,\alpha, \mathbf{F}\alpha\}$, then $\mathcal{TS}_{\mathsf{t_{now}}}$ cannot be deontically consistent. The following definition explains what it means for a temporal status set to be closed under the deontic modalities and under actions.

Definition 4.11 (Temporal Deontic/Action Closure) *$\mathcal{TS}_{\mathsf{t}_{now}}$ is said to be* temporally deontically closed *at time t_{now} if $\mathbf{D\text{-}Cl}(\mathcal{TS}_{\mathsf{t}_{now}}(i)) = \mathcal{TS}_{\mathsf{t}_{now}}(i)$ for all time points i.*

$\mathcal{TS}_{\mathsf{t}_{now}}$ is said to be temporally action closed *at time t_{now} if $\mathbf{A\text{-}Cl}(\mathcal{TS}_{\mathsf{t}_{now}}(i)) = \mathcal{TS}_{\mathsf{t}_{now}}(i)$ for all time points i.*

Action consistency ensures that action constraints are never violated.

Definition 4.12 (Action Consistency) *$\mathcal{TS}_{\mathsf{t}_{now}}$ is said to be* temporally action consistent *at time t_{now} if for all time points i such that $\mathit{acthist}_{\mathsf{t}_{now}}(i)$ and $\mathit{hist}_{\mathsf{t}_{now}}(i)$ are defined, $\mathbf{Do}_i = \{\mathbf{Do}\,\alpha \mid \mathbf{Do}\,\alpha \in \mathcal{TS}_{\mathsf{t}_{now}}(i)\}$ satisfies the action constraints with respect to the agent state $\mathit{hist}_{\mathsf{t}_{now}}(i)$.*[7]

[7] Note that for $i = \mathsf{t_{now}}$ both $\mathsf{acthist}_{\mathsf{t_{now}}}(i)$ and $\mathsf{hist}_{\mathsf{t_{now}}}(i)$ are defined.

In the above definition, the reader should note that action consistency is checked only at those time points for which the agent designer chose to save the agent state. The following example illustrates this definition.

Example 4.13 (Rescue: Action Consistency) Let the truck agent have the action constraint AC that intuitively saying that the tank agent cannot drive and fill fuel simultaneously. Furthermore, let $\mathsf{t}_{\mathsf{now}} = 3$, let $\mathcal{TS}_{\mathsf{t}_{\mathsf{now}}}$ and $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}$ be the temporal status set and the state history function from Example 4.4 respectively. Then $\mathcal{TS}_{\mathsf{t}_{\mathsf{now}}}$ is temporally action consistent since for all time points $i \leq 3$, $\mathbf{Do}_i$ satisfies AC w.r.t. $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}(i)$. Note that although $\mathbf{Do}_4$ does not satisfy AC, this does not alter the outcome since $\mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}(4)$ is not defined.

For a temporal status set to be feasible, whenever a checkpoint is encountered (and hence the state of the agent is updated), the new state must satisfy the integrity constraints. That is, the expected future states of the agent need to satisfy the integrity constraints. This requirement, called checkpoint consistency, is defined below.

Definition 4.14 (Checkpoint Consistency) *$\mathcal{TS}_{\mathsf{t}_{now}}$ is said to be* checkpoint consistent *at time t_{now} if for all $i > \mathsf{t}_{now}$, $\mathcal{EO}(i)$ satisfies the integrity constraints $\mathcal{IC}$.*

It is important to note that every time a checkpoint is encountered, we must ensure that all integrity constraints are satisfied. This means that at every checkpoint, we must ensure that the concurrent execution of all actions of the form $\mathbf{Do}\,\alpha$ at that time point does not lead to a state which is inconsistent.

For a temporal status set to be feasible, it must satisfy the additional requirement of state consistency.

Definition 4.15 (State Consistency) *$\mathcal{TS}_{\mathsf{t}_{now}}$ is said to be* state consistent at time $\mathsf{t}_{\mathsf{now}}$ *if for all $i \leq \mathsf{t}_{now}$ such that $\mathit{hist}_{\mathsf{t}_{now}}(i)$ is defined, the state obtained from $\mathit{hist}_{\mathsf{t}_{now}}(i)$ by concurrently applying all $\boldsymbol{Do}$ actions contained in $\mathcal{TS}_{\mathsf{t}_{now}}(i)$ satisfies the integrity constraints $\mathcal{IC}$.*

Definition 4.16 (Feasible Temporal Status Set) *Suppose the current time is t_{now}, $\mathcal{TP}$ is a tap, and $\mathit{hist}_{\mathsf{t}_{now}}$, $\mathit{acthist}_{\mathsf{t}_{now}}$ are the state/action history respectively. Further suppose that $\mathcal{IC}, \mathcal{AC}$ are sets of integrity constraints and actions constraints, respectively. A set $\mathcal{TS}_{\mathsf{t}_{now}}$ satisfying $\mathcal{TS}_{\mathsf{t}_{now}}(i) \neq \emptyset$ for only finitely many i is said to be a* feasible temporal status set *with respect to the above parameters if*

(1) $\mathcal{TS}_{\mathsf{t}_{now}}$ is closed under the rules of $\mathcal{TP}$,
(2) $\mathcal{TS}_{\mathsf{t}_{now}}$ is temporally deontically and action consistent at time t_{now},

(3) $\mathcal{TS}_{\mathtt{t}_{now}}$ *is temporally deontically and action closed at time* $\mathtt{t}_{now}$*,*
(4) $\mathcal{TS}_{\mathtt{t}_{now}}$ *is checkpoint consistent at time* $\mathtt{t}_{now}$*,*
(5) $\mathcal{TS}_{\mathtt{t}_{now}}$ *is state consistent at time* $\mathtt{t}_{now}$*,*
(6) $\mathcal{TS}_{\mathtt{t}_{now}}$ *is history compatible at time* $\mathtt{t}_{now}$*.*

4.3 Rational Temporal Status Sets

A feasible temporal status set may contain action status atoms that are not necessary for the temporal status set to be feasible. In this section, we identify a class of feasible status sets for which agents perform a minimal set of actions.

Definition 4.17 (Rational Feasible Temporal Status Set) *A temporal status set* $\mathcal{TS}_{\mathtt{t}_{now}}$ *is* grounded*, if there is no temporal status set* $\mathcal{TS}'_{\mathtt{t}_{now}} \neq \mathcal{TS}_{\mathtt{t}_{now}}$ *such that* $\mathcal{TS}'_{\mathtt{t}_{now}} \subseteq \mathcal{TS}_{\mathtt{t}_{now}}$ *and* $\mathcal{TS}'_{\mathtt{t}_{now}}$ *satisfies conditions (1)–(6) of a feasible temporal status set.*[8]

A temporal status set $\mathcal{TS}_{\mathtt{t}_{now}}$ *is a* rational temporal status set*, if* $\mathcal{TS}_{\mathtt{t}_{now}}$ *is a feasible status set and* $\mathcal{TS}_{\mathtt{t}_{now}}$ *is grounded.*

Note that when $\mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}$ is a feasible status set, every $\mathcal{TS}'_{\mathtt{t}_{\mathsf{now}}} \subseteq \mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}$ satisfies conditions (2), (5) in the definition of feasibility, and hence the above definition may be simplified to only require satisfaction of conditions (1), (3), (4) and (6). The notion of a rational temporal status set is illustrated via the following

Example 4.18 (Rescue: Rational Status Set) Consider the simple example where the `truck` agent has an action constraint as in Example 4.13, i.e., AC that intuitively saying that the `tank` agent cannot drive and fill fuel simultaneously. In addition it has one integrity constraint that intuitively says that the truck cannot simultaneously be at two different locations. The agent's tap contains exactly one rule specified at the end of Section 3. Suppose the agent has the following very simple state history ($\mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}$), and action history ($\mathsf{acthist}_{\mathtt{t}_{\mathsf{now}}}$) below:

$$\begin{aligned}
\mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}(0) &= \{\mathbf{in}(\mathbf{true}, \mathtt{truck}: \mathit{tank_empty}()), \mathbf{in}(\mathtt{was}, \mathtt{truck}: \mathit{location}())\}\\
\mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}(1) &= \{\mathbf{in}(\mathbf{true}, \mathtt{truck}: \mathit{tank_empty}()), \mathbf{in}(\mathtt{was}, \mathtt{truck}: \mathit{location}())\}\\
\mathsf{acthist}_{\mathtt{t}_{\mathsf{now}}}(8) &= \{\mathbf{O}\mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw95})\}
\end{aligned}$$

[8] $\mathcal{TS}'_{\mathtt{t}_{\mathsf{now}}} \subseteq \mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}$ if for all i, $\mathcal{TS}'_{\mathtt{t}_{\mathsf{now}}}(i) \subseteq \mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}(i)$.

Suppose, $\mathtt{t}_{\mathsf{now}} = 1$. The following temporal status sets are feasible sets, but only the first one is rational.

$$\begin{aligned}
\mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}(1) &= \{\mathbf{O}\mathit{fill_fuel}(), \mathbf{Do}\,\mathit{fill_fuel}(), \mathbf{P}\mathit{fill_fuel}()\} \\
\mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}(8) &= \{\mathbf{O}\,\mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw95}), \mathbf{P}\,\mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw95}), \\
&\quad \mathbf{Do}\,\mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw95})\} \\
\mathcal{TS}^2_{\mathtt{t}_{\mathsf{now}}}(1) &= \{\mathbf{O}\mathit{fill_fuel}(), \mathbf{Do}\,\mathit{fill_fuel}(), \mathbf{P}\mathit{fill_fuel}(), \\
&\quad \mathbf{O}\,\mathit{order_item}(\mathit{fa_bag}), \mathbf{Do}\,\mathit{order_item}(\mathit{fa_bag}), \\
&\quad \mathbf{P}\,\mathit{order_item}(\mathit{fa_bag}), \} \\
\mathcal{TS}^2_{\mathtt{t}_{\mathsf{now}}}(8) &= \{\mathbf{O}\,\mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw95}), \mathbf{P}\,\mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw95}), \\
&\quad \mathbf{Do}\,\mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw95})\}
\end{aligned}$$

$\mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}$ is temporally deontically consistent because (1) the only action which has a precondition is *drive* and it is easy to see that its precondition, namely $\mathbf{in}(\mathit{was}, \mathtt{truck}: \mathit{location}())$ is satisfied by $\mathcal{EO}(8)$; (2) for each $\mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}(i)$, there are no forbidden or waived actions. $\mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}$ is temporally action closed (and hence, temporally deontically closed) because for each $\mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}(i)$ where $\mathbf{O}\alpha \in \mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}(i)$, $\mathbf{Do}\,\alpha \in \mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}(i)$ and $\mathbf{P}\alpha \in \mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}(i)$. $\mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}$ is temporally action consistent because there is no $\mathbf{Do}_i$ where $\mathbf{Do}\,\mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway})$, and $\mathbf{Do}\,\mathit{fill\_fuel}() \in \mathbf{Do}_i$. $\mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}$ is checkpoint consistent as the only relevant action, $\mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw95})$, never violates the integrity constraints in $\mathcal{IC}$. Finally, it is easy to verify that $\mathcal{TS}^1_{\mathtt{t}_{\mathsf{now}}}$ is closed under the rule in $\mathcal{TP}$.

Note that $\mathcal{TS}^2_{\mathtt{t}_{\mathsf{now}}}$ is also a feasible temporal status set: however, it contains $\mathbf{Do}\,\mathit{order\_item}(\mathit{fa\_bag})$ even though no rule or previous commitment forces $\mathit{order\_item}(\mathit{fa\_bag})$ to be done. This prevents $\mathcal{TS}^2_{\mathtt{t}_{\mathsf{now}}}$ from being rational.

4.4 Compact Representation of Temporal Status Sets

Representing a feasible temporal status set explicitly is difficult because, for each time point i, $\mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}(i)$ must be explicitly represented. This is obviously problematic from an implementation point of view because i might be infinite, and representing actions for many such i's is difficult and cumbersome. To ameliorate this problem, we describe below, a *constrained representation* of a class of temporal feasible status sets.

Definition 4.19 (Temporal Interval Constraint tic) *An* atomic temporal interval constraint *is an expression of the form $\ell \leq \mathsf{t} \leq u$ where t is a variable ranging over natural numbers, and ℓ, u are natural numbers.*

Temporal Interval Constraints *are inductively defined as follows:*

(1) Atomic temporal interval constraints are temporal interval constraints.
(2) If $\mathsf{tic}_1, \mathsf{tic}_2$ *are temporal interval constraints involving the same variable* t*, then* $(\mathsf{tic}_1 \vee \mathsf{tic}_2)$*,* $(\mathsf{tic}_1 \,\&\, \mathsf{tic}_2)$ *and* $\neg\mathsf{tic}_1$ *are temporal interval constraints.*

For example, $(5 \leq t \leq 10)$ is an atomic temporal interval constraint. So is $(50 \leq t \leq 60)$. In addition, $(5 \leq t \leq 10) \vee (50 \leq t \leq 60)$ and $(5 \leq t \leq 10) \,\&\, (50 \leq t \leq 60)$ are temporal interval constraints.

As the concepts of constraints and solutions of constraints with variables ranging over the natural numbers are well known and well studied in the literature (Cormen, Leiserson, and Rivest 1989), we do not repeat them here.

Definition 4.20 (Interval Constraint Annotated Status Atom) *If* tic *is a temporal interval constraint, and* $\mathsf{Op}\,\alpha$ *is an action status atom, then* $\mathsf{Op}\,\alpha : \mathsf{tic}$ *is an interval constraint annotated status atom.*

Intuitively, the interval constraint annotated status atom $\mathsf{Op}\,\alpha : \mathsf{tic}$ may be read as *"*$\mathsf{Op}\,\alpha$ *is known to be true* ***at some time point*** *which is a solution of* tic*"*. For example, $\mathbf{O}\alpha : (500 \leq \mathsf{t} \leq 6000)$ says that an agent is obliged to do α at one of times $500, 501, \ldots, 6000$. If tic is an atomic temporal interval constraint, then we will sometimes write it as temporal annotation, e.g., instead of $\mathbf{O}\alpha : (500 \leq \mathsf{t} \leq 6000)$ we will write $\mathbf{O}\alpha : [500, 6000]$.

Notice that one single statement allows us to *implicitly* represent the obligation of this agent to do α at one of 5,501 time instances.

Definition 4.21 (Interval Constraint Temporal Status Set ic**-**$\mathcal{TS}$**)** *An* interval constraint temporal status set*, denoted* $\mathsf{ic}\text{-}\mathcal{TS}$*, is a set of interval constraint annotated status atoms.*

Such a set $\mathsf{ic}\text{-}\mathcal{TS}$ stands for a whole class of temporal status sets: all status sets that are compatible with it in the following sense:

Definition 4.22 (Temporal Status Sets Compatible with ic**-**$\mathcal{TS}$**)** *A temporal status set* $\mathcal{TS}_{\mathsf{t}_{now}}$ *is* compatible with $\mathsf{ic}\text{-}\mathcal{TS}$ *if for every* $\mathsf{Op}\,\alpha : \mathsf{tic}$ *in* $\mathsf{ic}\text{-}\mathcal{TS}$*, there is a solution* $\mathsf{t} = i$ *of* tic *such that* $\mathsf{Op}\,\alpha \in \mathcal{TS}_{\mathsf{t}_{now}}(i)$*.*

We use the notation $\mathsf{CompTSS}(\mathsf{ic}\text{-}\mathcal{TS})$ *to denote the set of all temporal status sets compatible with* $\mathsf{ic}\text{-}\mathcal{TS}$*.*

The following example illustrates the connection between interval constraint temporal status sets and temporal status sets.

Example 4.23 (Rescue: Compatibility) The status set $\mathcal{TS}_{\mathsf{t_{now}}}$ from Example 4.4 is compatible with the following ic-$\mathcal{TS}$:

$$\begin{aligned}\{&\mathbf{P}\,drive(was, bal, hw95) : (0 \le \mathsf{t} \le 4),\\ &\mathbf{O}\mathit{fill\_fuel}() : ((0 \le \mathsf{t} \le 2) \vee (4 \le \mathsf{t} \le 7)),\\ &\mathbf{F}\mathit{fill\_fuel}() : (0 \le \mathsf{t} \le 5)\}\end{aligned}$$

There is an infinite number of temporal status sets $\mathcal{TS}_{\mathsf{t_{now}}}$ that are compatible with this ic-$\mathcal{TS}$. For example, the temporal status set, $\mathcal{TS}'_{\mathsf{t_{now}}}$ defined by $\mathcal{TS}'_{\mathsf{t_{now}}}(0) = \{\mathbf{F}\mathit{fill\_fuel}()\}$, $\mathcal{TS}'_{\mathsf{t_{now}}}(4) = \{\mathbf{O}\mathit{fill\_fuel}(), \mathbf{P}\,drive(was, bal, hw95)\}$ is also compatible with it. On the other hand, there are infinitely many interval constraint temporal status sets that are compatible with the $\mathcal{TS}_{\mathsf{t_{now}}}$ from Example 4.4. The empty set is one example.

It is important to note that when we have two interval constraint annotated status atoms of the form $\mathsf{Op}\,\alpha : \mathsf{tic}_1$ and $\mathsf{Op}\,\alpha : \mathsf{tic}_2$ in ic-$\mathcal{TS}$, we cannot (in general) infer $\mathsf{Op}\,\alpha : \mathsf{tic}_1 \wedge \mathsf{tic}_2$.

We now define three important properties of an ic-$\mathcal{TS}$— these definitions will be used in the next section.

- ic-$\mathcal{TS}$ is *temporally deontically consistent if* there is a temporal status set $\mathcal{TS}_{\mathsf{t_{now}}}$ compatible with ic-$\mathcal{TS}$ which is temporally deontically consistent.
- ic-$\mathcal{TS}$ is *temporally deontically closed (resp. action closed) if* there is a temporal status set $\mathcal{TS}_{\mathsf{t_{now}}}$ compatible with ic-$\mathcal{TS}$ which is temporally deontically (resp. action) closed.

Example 4.24 Consider the ic-$\mathcal{TS}$ of Example 5.2, i.e.,

$$\begin{aligned}\{&\mathbf{Do}\,\mathit{fill\_fuel}() :\ (3 \le \mathsf{t} \le 3), \mathbf{P}\mathit{fill\_fuel}() :\ (3 \le \mathsf{t} \le 3),\\ &\mathbf{O}\,order\_item(fa\_bag) :\ (3 \le \mathsf{t} \le 7), \mathbf{P}\,order\_item(fa\_bag) :\ (3 \le \mathsf{t} \le 7),\\ &\mathbf{Do}\,order\_item(fa\_bag) :\ (3 \le \mathsf{t} \le 7)\ \ \}\end{aligned}$$

Suppose, $\mathsf{t_{now}} = 3$, $\mathsf{hist}_{\mathsf{t_{now}}}(0) = \mathsf{hist}_{\mathsf{t_{now}}}(1) = \mathsf{hist}_{\mathsf{t_{now}}}(2) = \emptyset$ and

$$\mathsf{hist}_{\mathsf{t_{now}}}(3) = \{\mathbf{in}(1, \mathtt{truck}\!: inventory(fa\_bag)), \mathbf{in}(\mathbf{true}, \mathtt{truck}: tank\_empty())\}$$

This ic-$\mathcal{TS}$ is temporally deontically consistent and temporally deontically closed and temporal action closed because the following $\mathcal{TS}_{\mathsf{t_{now}}}$ that is temporally deontically consistent and temporally deontically and action closed is compatible with it.

For all $i \neq 3$, $\mathcal{TS}_{\mathsf{t_{now}}}(i) = \emptyset$, and

$$\mathcal{TS}_{\mathsf{t_{now}}}(3) = \{\mathbf{Do}\, \mathit{fill\_fuel}(),\ \mathbf{P}\mathit{fill\_fuel}(),\ \ \mathbf{O}\mathit{order\_item}(\mathit{fa\_bag}),\\ \mathbf{P}\mathit{order\_item}(\mathit{fa\_bag}),\ \ \mathbf{Do}\ \mathit{order\_item}(\mathit{fa\_bag})\}$$

5 Status Set Computation Algorithm for Positive taps

The preceding section defines a formal semantics based on the concept of a feasible temporal status set. We now show how such sets can be constructed. Our main result is Theorem 5.11 stating that Algorithm 5.1 to compute such sets for positive taps (i.e. taps whose rules are negation-free) is correct and complete. As this result is complex and needs sophisticated technical machinery, we first present an overview of the proof.

In Section 5.1 we show that the main difficulty is to ensure that the set to be constructed is closed under the rules of $\mathcal{TP}$. This is because when a rule causes atoms to be added to $\mathcal{TS}_{\mathsf{t_{now}}}(i)$, for some $i \geq \mathsf{t_{now}}$, it may cause other rules to fire, which may cause other atoms to be added to $\mathcal{TS}_{\mathsf{t_{now}}}(i)$, for some $i \geq \mathsf{t_{now}}$. This in turn may cause additional rules to fire.

This difficulty suggests taking advantage of well-known methods from logic programming (Lloyd 1987), namely to construct a suitable monotone fixpoint operator (Definition 5.1) and to relate its least fixpoint $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ with feasible temporal status sets (Theorem 5.6). 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 5.1 applied to a set ic-$\mathcal{TS}$ gives us the result of applying all the rules at 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 is monotone.

In Section 5.2 we use the results of Section 5.1 to design Algorithm 5.1 to compute feasible temporal status sets. The main ingredient used in this algorithm is **ComputeTSS**, which computes status sets compatible with a given ic-$\mathcal{TS}$. The ic-$\mathcal{TS}$ we start with is the fixed point of the rules, $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$. To ensure important properties of **ComputeTSS** (Definition 5.7), we introduce *Constraint Hitting Sets* (Definition 5.8).

Suppose the current time, $\mathtt{t}_{\mathsf{now}}$ and the histories, $\mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}(\cdot)$ and $\mathsf{acthist}_{\mathtt{t}_{\mathsf{now}}}(\cdot)$ are arbitrary, but fixed.

(1) History compatibility uniquely specifies $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(i)$ for $i < \mathtt{t}_{\mathsf{now}}$.
(2) If there exists an $i < \mathtt{t}_{\mathsf{now}}$ for which state consistency does not hold, then no feasible temporal status set can exist.

Thus we are left with the construction of $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(i)$ for $i \geq \mathtt{t}_{\mathsf{now}}$ which should be based on the last condition for feasibility: the "closure under program rules". The question is whether we can satisfy this condition while making sure that all the other conditions are also satisfied.

When considering a program rule, we need to distinguish between two types of tasc that appear in the body of the rule: (i) state dependent tasc (ii) state independent tasc.

A state dependent tasc is always evaluated either in the *current or the past.* Thus, every state dependent tasc of the form $\varrho : [\mathsf{tai}_1, \mathsf{tai}_2]$ may implicitly be rewritten as $\varrho : [\mathsf{tai}_1, \min(\mathsf{tai}_2, \mathtt{t}_{\mathsf{now}})]$.

If $\mathsf{tai}_2 < \mathtt{t}_{\mathsf{now}}$, then evaluation of $\varrho : [\mathsf{tai}_1, \mathsf{tai}_2]$ boils down checking if ϱ is true at some time point $i < \mathtt{t}_{\mathsf{now}}$. Thus, all that is needed is to evaluate ϱ w.r.t. the state at time $i < \mathtt{t}_{\mathsf{now}}$ and $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(i)$ both of which are *fixed*! If the body is true and the head is of the form $\mathsf{Op}\alpha : \mathsf{ta}$, then $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(j)$ must contain $\mathsf{Op}\alpha$ for some time point $j \in \mathsf{ta}$ such that $j \geq \mathtt{t}_{\mathsf{now}}$.

Hence, it is only if $\mathsf{tai}_2 = \mathtt{t}_{\mathsf{now}}$ that we need to worry about evaluating the rule w.r.t. state dependent tascs. The main problem is therefore to compute the set $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(\mathtt{t}_{\mathsf{now}})$: Adding more and more action status atoms to $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(\mathtt{t}_{\mathsf{now}})$ forces us to reconsider program rules that already fired and thus to extend the set $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(\mathtt{t}_{\mathsf{now}})$! This is exactly where the fixpoint character of the computation comes in.

With respect to state independent tascs, we note that according to the intuitive reading after Definition 3.10 there is no requirement that rules only fire from up to now into the future (including now). Therefore rules of the form $\mathsf{Op}\,\alpha : [\mathtt{t}_{\mathsf{now}}, \mathtt{t}_{\mathsf{now}}] \leftarrow \mathsf{Op}\,\alpha : [\mathtt{t}_{\mathsf{now}} + 5, \mathtt{t}_{\mathsf{now}} + 5]$ are allowed. This, of course, adds another fixpoint flavor to the computation of temporal status sets.

Thus, in order to find a closure under program rules, we will define a fixpoint operator. In this definition we will distinguish between dependent and independent tasc's to reflect our discussion above. This definition assumes that implication of modalities is defined as follows: **O** implies **P** and **Do**, **Do** implies $\mathbf{P}$ and for every Op, Op implies Op.

Definition 5.1 (Operator $\mathbf{D}_{\mathcal{TP}}$) *Suppose $\mathcal{TP}$ is a tap, $hist_{t_{now}}(t_{now})$ is an agent state and ic-$\mathcal{TS}$ is an interval constraint temporal status set. Then we define $D_{\mathcal{TP}}(ic\text{-}\mathcal{TS})$ to be the set*

$\{Op'\,\alpha\colon tic \mid Op\,\alpha : [tai_1, tai_2] \;\leftarrow\; \varrho_1 : \mathsf{ta}_1 \,\&\ldots\&\, \varrho_n : \mathsf{ta}_n$

is a ground instance of a rule in $\mathcal{TP}$ and for all $1 \le i \le n$

(I) If ϱ_i is state independent

(we assume $\varrho_i = Op_{i_1}\alpha_{i_1} \wedge \ldots \wedge Op_{i_n}\alpha_{i_m}$)

If $m = 1$ then there exists $Op'_{i_1}\alpha_{i_1} : tic_{i_1}$ in ic-$\mathcal{TS}$ s. t.:

(1) Op'_{i_1} implies Op_{i_1},

(2) tic_{i_1} implies $t \in \mathsf{ta}_{i_1}$ (i.e. all solutions of tic_{i_1} are in ta_{i_1}).

If $m > 1$ then there exist $\mathsf{t}_i \in \mathsf{ta}_i$ and $Op'_{i_j}\alpha_{i_j} : [\mathsf{t_i}, \mathsf{t_i}]$ in ic-$\mathcal{TS}$ s. t. Op'_{i_j} implies Op_{i_j}.

(II) If ϱ_i is state dependent

(we assume $\varrho_i = \chi_i \wedge Op_{i_1}\alpha_{i_1} \wedge \ldots \wedge Op_{i_n}\alpha_{i_m}$)

If $m = 0$ then

(0) χ_i is true in the agent state $hist_{t_{now}}(\mathsf{t_i})$, for a $\mathsf{t_i} \le \mathsf{t}_{now}$ and

(1) $\mathsf{t_i}$ is a solution of ta_i.

If $m \ge 1$ then there exist $\mathsf{t}_i \le \mathsf{t}_{now}$ and $Op'_{i_j}\alpha_{i_j} : [\mathsf{t_i}, \mathsf{t_i}]$ in ic-$\mathcal{TS}$ s. t.:

(0) χ_i is true in the agent state $hist_{t_{now}}(\mathsf{t}_i)$,

(1) Op'_{i_j} implies Op_{i_j},

(2) t_i is a solution of ta_i.

and *$tic = [max\{tai_1, \mathsf{t}_{now}\}, tai_2]$ and Op implies Op'.*

$\}$

Intuitively, the input to this operator contains things we already know must be in a feasible temporal status set. The output computes what else must be added to the feasible temporal status set if all rules in $\mathcal{TP}$ are applied just once.

As discussed above, we distinguish between state dependent and state independent tasc's in the operator above because:

(1) For state independent tasc's, we do not need to ensure that the body of a rule is evaluated in the past (as we do not have a condition on the state which needs to be checked).
(2) However, state dependent tasc's have an associated state condition which can only be checked up to $\mathtt{t_{now}}$. Therefore, in this case, we only need to worry about the current time, $\mathtt{t_{now}}$.

In addition, in each category we distinguish between cases in which the tasc consists of only one disjunct, and cases in which the tasc consists of at least two disjuncts.

The first case is much easier to satisfy. If ϱ_i is a state-independent tasc, and if it consists of only one conjunct (i.e., $m = 1$), it is of the form $\mathsf{Op}_{i_1}\alpha_{i_1}$: ta_1, e.g., $\mathbf{P}\alpha : [\mathtt{t_{now}} + 1, \mathtt{t_{now}} + 5]$. For it to be satisfied, there must be an interval constraint annotated status atom, $\mathsf{Op}'_{i_1}\alpha_{i_1} : \mathsf{tic}_{i_1}$ that will lead to its satisfaction. As tic_{i_1} may be satisfied by several solutions, and we are not sure which of them will eventually be selected, we must verify that for every choice of a solution of tic_{i_1}, it will entail the satisfaction of ϱ_i. For this we require that all the solutions of tic_{i_1} will be members of ta_1. For $\mathbf{P}\alpha : [\mathtt{t_{now}} + 1, \mathtt{t_{now}} + 5]$, tic_{i_1} can be, for example, $\mathtt{t_{now}} + 2 \leq t \leq \mathtt{t_{now}} + 4$. On the other hand, we have some flexibility with respect to the operator Op'_{i_1} of the constraint atom. Since we aim at constructing a feasible temporal status set, we know that it will be temporally deontically and action closed, and thus we can allow that Op'_{i_1} will not be equal to Op_{i_1}, but only will imply it. In our example it can be $\mathbf{O}\alpha[\mathtt{t_{now}} + 2, \mathtt{t_{now}} + 4]$.

The case of in which ϱ_i is a state- dependent tasc and consists of only one conjunct (i.e., $m = 0$) is even more simple. It is of the form, $\chi_i : \mathsf{ta}_i$ and it *cannot* be satisfied in the future as we require in the definition.

The cases in which the tasc consists of at least two conjuncts is more complex. All the conjuncts must be satisfied in the same time period in order that the tasc will be satisfied. Therefore, we require that there is one time period $\mathsf{t}_i \in \mathsf{ta}_i$ such that there is an interval constraint annotated status atom of the form $\mathsf{Op}'_{i_j}\alpha_{i_j} : [\mathsf{t_i}, \mathsf{t_i}]$. For example, if $\varrho_i = (\mathbf{P}\alpha_1 \wedge \mathbf{O}\alpha_2) : [3, 7]$, then in order that it will be satisfied, $\mathsf{ic}\text{-}\mathcal{TS}$ may include $\mathbf{O}\alpha_1[4, 4]$ and $\mathbf{O}\alpha_2[4, 4]$.

In all the above cases, as our goal is to construct $\mathcal{TS}_{\mathtt{t_{now}}}(i)$ for $i \geq \mathtt{t_{now}}$, the lower bound of the time interval of the atom that we add, $\mathsf{Op}'\,\alpha : \mathsf{tic}$ is not smaller than $\mathtt{t_{now}}$ as stated in the last line of the definition. We demonstrate the usage of the operator in the next definition.

Example 5.2 (Rescue: $\mathbf{D}_{\mathcal{TP}}$) Suppose the `truck` agent's tap, $\mathcal{TP}$, contains rules (r2) and (r3) from Example 4.4 which are recapitulated below:

r2: $\mathbf{Do}\, \mathit{fill\_fuel}() : [\mathtt{t_{now}}, \mathtt{t_{now}}] \leftarrow$

$\mathbf{in}(\mathbf{true}, \mathtt{truck}: \mathit{tank\_empty}()) : [\mathsf{t_{now}} - 2, \mathsf{t_{now}}]$
r3: $\mathbf{O}\mathit{order\_item}(\mathit{fa\_bag}) : [\mathsf{t_{now}}, \mathsf{t_{now}} + 4] \leftarrow$
$\mathbf{in}(1, \mathtt{truck}: \mathit{inventory}(\mathit{fa\_bag}))[\mathsf{t_{now}} - 3, \mathsf{t_{now}}]$

Suppose, $\mathsf{t_{now}} = 3$ and

$\mathsf{hist}_{\mathsf{t_{now}}}(\mathsf{t_{now}}) = \{\mathbf{in}(1, \mathtt{truck}: \mathit{inventory}(\mathit{fa\_bag})), \mathbf{in}(\mathbf{true}, \mathtt{truck}: \mathit{tank\_empty}())\}$

$$
\begin{aligned}
\mathsf{D}_{\mathcal{TP}}(\emptyset) = \{&\mathbf{Do}\,\mathit{fill\_fuel}() : (3 \le \mathsf{t} \le 3), \mathbf{P}\mathit{fill\_fuel}() : (3 \le \mathsf{t} \le 3),\\
&\mathbf{O}\mathit{order\_item}(\mathit{fa\_bag}) : (3 \le \mathsf{t} \le 7),\\
&\mathbf{P}\mathit{order\_item}(\mathit{fa\_bag}) : (3 \le \mathsf{t} \le 7),\\
&\mathbf{Do}\,\mathit{order\_item}(\mathit{fa\_bag}) : (3 \le \mathsf{t} \le 7) \ \}
\end{aligned}
$$

In order to find a fixed point we need to iterate the operator. However, we do not start the operator at $\emptyset$: this is because part of the temporal status set we want to construct is already determined by $\mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$. Therefore we define

$$
\mathsf{ic}\text{-}\mathcal{T}S_{\text{start}} := \bigcup_{\{i \text{ s.t. } \mathsf{acthist}_{\mathsf{t_{now}}}(i) \text{ is defined}\}} \{\mathsf{Op}\alpha[i,i] : \ \mathsf{Op}\alpha \in \mathsf{acthist}_{\mathsf{t_{now}}}(i)\}.
$$

In most of the examples below, we assume without loss of generality that $\mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$ is empty and thus $\mathsf{ic}\text{-}\mathcal{T}S_{\text{start}} = \emptyset$.

Definition 5.3 (Iterations of $\mathsf{D}_{\mathcal{TP}}$) *Let $\mathcal{TP}$ be a positive* tap*, and $\mathsf{hist}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$ be an agent state. The* iterations *of $D_{\mathcal{TP}}$ are defined as follows:*

$$
\begin{aligned}
D_{\mathcal{TP}} \uparrow^{0} &= \mathit{ic}\text{-}\mathcal{T}S_{start}.\\
D_{\mathcal{TP}} \uparrow^{(j+1)} &= D_{\mathcal{TP}}(D_{\mathcal{TP}} \uparrow^{j}).\\
D_{\mathcal{TP}} \uparrow^{\omega} &= \bigcup_{j} D_{\mathcal{TP}} \uparrow^{j} .
\end{aligned}
$$

The following example demonstrates how we may iterate the $\mathsf{D}_{\mathcal{TP}}$ operator.

Example 5.4 (Rescue: $\mathsf{D}_{\mathcal{TP}}$) We continue with Example 5.2 and assume that the agent's state is as described there, $\mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$ is empty, and thus $\mathsf{ic}\text{-}\mathcal{T}S_{\text{start}} = \emptyset$. In addition to rules **r2** and **r3** mentioned there, we assume that we have two additional rules in the agent's tap $\mathcal{TP}$:

r0: $\mathbf{F}\mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{Highway}) : [\mathtt{X_{now}}, \mathtt{X_{now}} + 2] \leftarrow$
$\mathbf{Do}\,\mathit{fill\_fuel}()\ \&\ \mathbf{in}(\mathbf{true}, \mathtt{truck}: \mathit{tank\_empty}()) : [\mathtt{X_{now}}, \mathtt{X_{now}} + 2]$

r1: $\mathbf{O}\mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw95}) : [\mathtt{X_{now}} + 5, \mathtt{X_{now}} + 10] \leftarrow$
$\mathbf{O}\mathit{order\_item}(\mathit{fa\_bag}) : [\mathtt{X_{now}}, \mathtt{X_{now}} + 10]$

- $\mathsf{D}_{\mathcal{TP}} \uparrow^0 = \emptyset$ since ic-$\mathcal{T}S_{\text{start}} = \emptyset$.
- $\mathsf{D}_{\mathcal{TP}} \uparrow^{(1)} = \mathsf{D}_{\mathcal{TP}}(\mathsf{D}_{\mathcal{TP}} \uparrow^0) = \mathsf{D}_{\mathcal{TP}}(\emptyset)$. The additional two rules in $\mathcal{TP}$ didn't change the set $\mathsf{D}_{\mathcal{TP}}(\emptyset)$, and it is as in Example 5.2, i.e.,

$$\begin{aligned}\mathsf{D}_{\mathcal{TP}} \uparrow^{(1)} = \{ & \mathbf{Do}\, \mathit{fill\_fuel}()(3 \leq \mathsf{t} \leq 3), \mathbf{P}\mathit{fill\_fuel}()(3 \leq \mathsf{t} \leq 3), \\ & \mathbf{O}\, \mathit{order\_item}(\mathit{fa\_bag})(3 \leq \mathsf{t} \leq 7), \\ & \mathbf{P}\, \mathit{order\_item}(\mathit{fa\_bag})(3 \leq \mathsf{t} \leq 7), \\ & \mathbf{Do}\, \mathit{order\_item}(\mathit{fa\_bag})(3 \leq \mathsf{t} \leq 7) \ \}\end{aligned}$$

- Applying the rules **r0** and **r1**, in addition to **r2** and **r3**, we get:

$$\begin{aligned}\mathsf{D}_{\mathcal{TP}} \uparrow^{(2)} = & \ \mathsf{D}_{\mathcal{TP}} \uparrow^{(1)} \cup \\ & \{\mathbf{F}\, \mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw95}) : (3 \leq t \leq 5), \\ & \ \mathbf{F}\, \mathit{drive}(\mathit{was}, \mathit{bal}, \mathit{hw295}) : (3 \leq t \leq 5), \\ & \ \mathbf{O}\, \mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw95}) : (8 \leq t \leq 13), \\ & \ \mathbf{Do}\, \mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw95}) : (8 \leq t \leq 13), \\ & \ \mathbf{P}\, \mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw95}) : (8 \leq t \leq 13) \}\end{aligned}$$

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

We have proved that $\mathsf{D}_{\mathcal{TP}}$ is a monotone and continuous operator (Dix, Kraus, and Subrahmanian 2000) and that $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ is its least fixpoint. This is an application of the Knaster-Tarski theorem in universal algebra.

Theorem 5.5 *Suppose $\mathcal{TP}$ is a positive* tap*, and $\mathsf{hist}_{\mathsf{t}_{now}}(\mathsf{t}_{now})$ is an agent state. Then: $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ is the least fixpoint of $\mathsf{D}_{\mathcal{TP}}$.*

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

Theorem 5.6 *Suppose $\mathcal{TP}$ is a positive* tap*, and $\mathsf{hist}_{\mathsf{t}_{now}}(\mathsf{t}_{now})$ is an agent state. Then:*

(1) $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ is temporally deontically closed.
(2) $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ is temporally action closed.
(3) If $\mathcal{T}S_{\mathsf{t}_{now}}$ is a feasible temporal status set, then it is compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$.

The net result of the above theorem is that in order to find feasible temporal status sets we can (1) compute the least fixpoint of $\mathsf{D}_{\mathcal{TP}}$ and then (2) select from amongst the compatible temporal status sets, those that satisfy the other requirements for feasibility. Algorithm 5.1 uses a subroutine called **ComputeTSS** described later in Definition 5.7. Whenever an agent's state changes, this algorithm will be executed to find a new feasible temporal status set. The agent then concurrently executes all actions α such that $\mathbf{Do}\,\alpha$ is in the computed feasible temporal status set at time $\mathtt{t}_{\mathsf{now}}$. The algorithm works by iteratively modifying the set $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}$ (using the **ComputeTSS** subroutine) and checking for feasibility.

Algorithm 5.1 (Feasible Temporal Status Set Computation)

FTSS($\mathtt{t}_{\mathsf{now}}$,$\mathcal{TP}$,**hist**$_{\mathtt{t}_{\mathsf{now}}}$, **acthist**$_{\mathtt{t}_{\mathsf{now}}}$)

(⋆ *input is the current time, a positive tap* $\mathcal{TP}$ ⋆)
(⋆ *and the histories hist*$_{\mathtt{t}_{now}}$, *acthist*$_{\mathtt{t}_{now}}$ ⋆)
(⋆ *output is a feasible temporal status set if one exists* ⋆)
(⋆ *otherwise, the output is "No".* ⋆)

(1) ***if Check_trivial_part***(*hist*$_{\mathtt{t}_{now}}$, *acthist*$_{\mathtt{t}_{now}}$)= **false** ***then return*** *"No."*
(2) *done* := **false**;
(3) *Seen* := $\emptyset$;
(4) ***while*** $\neg$*done* ***do***
   (a) $\mathcal{T}S_{\mathtt{t}_{now}}$:= ***ComputeTSS***($\mathsf{D}_{\mathcal{TP}}\uparrow^{\omega}$,$\mathcal{TP}$, *Seen*);
   (b) ***if*** $\mathcal{T}S_{\mathtt{t}_{now}}$ = *"No"* ***then return*** *"No."*
   (c) ***if FeasTSS***($\mathcal{T}S_{\mathtt{t}_{now}}$) ***then*** *done* := **true** ***else*** *Seen* := *Seen* $\cup \{\mathcal{T}S_{\mathtt{t}_{now}}\}$;
(5) ***return*** $\mathcal{T}S_{\mathtt{t}_{now}}$.

The **FTSS** algorithm terminates as soon as a feasible temporal status set is found. The **Check_trivial_part** subroutine determines $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(i)$ for $i < \mathtt{t}_{\mathsf{now}}$ by history compatibility and then checks checkpoint consistency and state consistency for $\mathcal{T}S_{\mathtt{t}_{\mathsf{now}}}(i)$ for $i < \mathtt{t}_{\mathsf{now}}$. If either of them is not satisfied, it returns **false** and there is no compatible set (see the explanations at the beginning of the previous section).

The algorithm maintains a set, *Seen*, of compatible temporal status sets seen thus far—if the algorithm is "still running" this means that none of the compatible temporal status sets examined thus far is feasible, and hence, we need to continue trying to find a new compatible temporal status set that is feasible.

As **ComputeTSS** computes compatible status sets that are closed under the rules of the program, the **FeasTSS** algorithm only needs to check

(1) action consistency for $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$,
(2) whether all the operators $\mathsf{Op}\alpha$ in $\mathcal{TS}_{\mathsf{t_{now}}}$ where $\mathsf{Op} \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$ are executable in the current state $\mathsf{hist}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$,
(3) temporal deontic consistency for $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$. (As we require temporal status sets to be finite, we only need to check this for finitely many i.)
(4) checkpoint consistency for $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$ (As we require the number of checkpoints to be finite, we only need to check this for finitely many i.)

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

We now present a detailed algorithm for **ComputeTSS**. Before doing so, we first formulate the input-output behaviour of this function.

Definition 5.7 (Input and Output of ComputeTSS)
*The **ComputeTSS** function takes as input (1) an interval constraint temporal status set* $\mathsf{ic}\text{-}\mathcal{TS}$*, (2) a positive temporal program* $\mathcal{TP}$*, and (3) a set Seen of temporal status sets closed under the rules of* $\mathcal{TP}$*.*

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

We have to construct a temporal status set $\mathcal{TS}_{\mathsf{t_{now}}}$ compatible with $\mathsf{ic}\text{-}\mathcal{TS}$ and closed under the rules of $\mathcal{TP}$. How can we accomplish this? Obviously, if an atom of the form $\mathsf{Op}\alpha : [4, 4]$ is in $\mathsf{ic}\text{-}\mathcal{TS}$ we have to put $\mathsf{Op}\alpha$ into $\mathcal{TS}_{\mathsf{t_{now}}}(4)$. But all other atoms consisting of non-singleton tic's must also be satisfied. How can we assign them to $\mathcal{TS}_{\mathsf{t_{now}}}$? To obtain compatibility, for every such $\mathsf{Op}\alpha : \mathsf{tic}$ we must add at least one atom of the form $\mathsf{Op}\alpha : [i, i]$ where i is a solution of tic. However, we should choose i carefully so as to maintain closure under the program rules and to ensure minimality. For a precise description of how to make such a choice, let us introduce the concept of a constraint hitting set.

Definition 5.8 (Constraint Hitting Set) *Suppose* $\mathsf{ic}\text{-}\mathcal{TS}$ *is an interval constraint temporal status set. A* constraint hitting set*, H, for* $\mathsf{ic}\text{-}\mathcal{TS}$ *is a minimal set of ground annotated atoms of the form* $\mathsf{Op}\,\alpha\colon [i, i]$ *such that:*

> *For every interval constraint annotated status atom* $\mathsf{Op}\,\alpha\colon \mathsf{tic} \in \mathsf{ic}\text{-}\mathcal{TS}$*, there is an annotated atom of the form* $\mathsf{Op}\,\alpha\colon [i, i]$ *in H such that i is a solution of* tic*, and if* $i < \mathsf{t}_{now}$*, then* $\mathsf{Op}\,\alpha \in \mathsf{acthist}_{\mathsf{t}_{now}}(i)$*.*

We use $\mathsf{chs}(\mathsf{ic}\text{-}\mathcal{TS})$ *to denote the set of all constraint hitting sets for* $\mathsf{ic}\text{-}\mathcal{TS}$*.*

We will use a subroutine called *find_member_chs*$(\text{ic-}\mathcal{TS}, Seen)$ which finds a member of $\mathsf{chs}(\text{ic-}\mathcal{TS})$ that is not in *Seen*. If no such element exists, it returns "No solution." We do not specify the implementation of this algorithm as it can be easily implemented (using standard hitting set algorithms (Cormen, Leiserson, and Rivest 1989)).

Our algorithm for **ComputeTSS** systematically tries to satisfy the status atoms by first computing a hitting set. There are two possibilities:

(1) There is no such hitting set that does not appear already in *Seen*. If this happens, then the ic-$\mathcal{TS}$ we started with cannot be extended to a temporal status set closed under the rules of $\mathcal{TP}$ and still compatible with ic-$\mathcal{TS}$.
(2) There is such a hitting set H.

Such a hitting set H will serve us as a starting point to compute a temporal status set. Note that H can already be seen as a temporal status set compatible with ic-$\mathcal{TS}$. The problem is that it is not closed under the rules of $\mathcal{TP}$.

Before elaborating further on how **ComputeTSS** may be implemented, we need one more concept.

Definition 5.9 (Solution Closed) *A set F of interval constrained annotated atoms is said to be* solution-closed *iff:*

> *for all interval constrained annotated status atoms $\mathsf{Op}\alpha : \mathsf{tic} \in F$,*
> *the following holds: tic has a solution i and $\mathsf{Op}\alpha : [i, i] \in F$*

Intuitively, a set F is solution-closed *if* for every interval constrained annotated status atoms $\mathsf{Op}\alpha : \mathsf{tic}$ in F, some explicit $\mathsf{Op}\alpha : [i, i]$ is also in F where i is a solution of tic.

Given this requirement, another problem is that the ic-$\mathcal{TS}$ with which we started is the least fixpoint of the $\mathsf{D}_{\mathcal{TP}}$ operator. Thus it contains status atoms that violate the solution-closed requirement. We use the hitting set H to get rid of them. H chooses appropriate times for status atoms with annotations that include more than one time point. We add these atoms to the program and get a new program $\mathcal{TP}_{new}$ (see *(2)(d)(ii)(A)* in Algorithm 5.2). We then apply our operator $\mathsf{D}_{\mathcal{TP}}$ to $\mathcal{TP}_{new}$ (see *(2)(d)(ii)(B)* in Algorithm 5.2). Note that new atoms which violate the solution-closed requirement may still be generated. We repeat this process until either

(1) all constraint atoms have a solution in the current H^* (see *(2)(d)(ii)(C)* in Algorithm 5.2), or

(2) we reach a fixpoint H^*. This fixpoint yields a better candidate ic-$\mathcal{TS}_{new}$ and we have to re-iterate the whole process, by first computing a hitting set of ic-$\mathcal{TS}_{new}$ and then computing the iterations of our operator $\mathsf{D}_{\mathcal{TP}}$.

Algorithm 5.2 (ComputeTSS(ic-$\mathcal{TS}, \mathcal{TP}, Seen$))
ComputeTSS(**ic**-$\mathcal{TS}, \mathcal{TP}, Seen$)

(⋆ *input is a positive* **tap** $\mathcal{TP}$ ⋆)
(⋆ *an interval constraint temporal status set* ic-$\mathcal{TS}$ ⋆)
(⋆ *and a set Seen of temporal status sets* ⋆)
(⋆ *output is a compatible temporal status set not in Seen* ⋆)
(⋆ *which is closed under the rules of* $\mathcal{TP}$ ⋆)

(1) *done* := **false**; *found* := **false**; *Loc_Seen* := *Seen*;
ic-$\mathcal{TS}_{new}$:= ic-$\mathcal{TS}$; $H^* := \emptyset$, *done_inner* := **false**;
(2) ***while*** $\neg done \wedge \neg found$ ***do***
  (a) ***if*** *done_inner* ***then***
    (i) $H =$ *find_member_chs*(ic-$\mathcal{TS}$, *Loc_Seen*);
    (ii) ***if*** $H =$ "No" ***then*** *done*:= **true**;
    (iii) *done_inner*:= **false**
  (b) ***else***
    (i) $H =$ *find_member_chs*(ic-$\mathcal{TS}_{new}$, *Loc_Seen*);
    (ii) ***if*** $H =$ "No" ***then*** *done_inner*= **true**;
  (c) *Loc_Seen* := *Loc_Seen* $\cup \{H\}$;
  (d) ***if*** $H \neq$ "No" ***then***
    (i) $H^\star = H$; *changed* =**true**;
    (ii) ***while*** $\neg found \wedge changed$ ***do***
      (A) $\mathcal{TP}_{new} = \mathcal{TP} \cup H^\star$; $oldH^\star = H^\star$;
      (B) $H^\star = \mathsf{D}_{\mathcal{TP}_{new}}$;
      (C) ***if*** $H^\star \notin$ *Loc_Seen* $\wedge$ $H^\star$ *is solution closed*
        then *found* = **true**
        else *changed* = ($oldH^\star \neq H^\star$);
    (iii) *Loc_Seen* := *Loc_Seen* $\cup \{H^*\}$; ic-$\mathcal{TS}_{new} := H^*$
(3) ***if*** *found* ***then*** *return* $H^\star$ ***else*** *return* "No."

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 ic-$\mathcal{TS}$. The variable "done" is false as long as not all the hitting sets were considered.

For each hitting set of the original ic-$\mathcal{TS}$, the inner while loop (*2d.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 $\mathcal{TP}$ and assigning it to $\mathcal{TP}_{new}$ (*2d.ii.B*). It iteratively adds the result of applying $\mathsf{D}_{\mathcal{TP}}$ to $\mathcal{TP}$ (*2d.ii.A*). When H^* cannot be enlarged any more and still a solution wasn't found, a hitting set of H^* (which is assigned to ic-$\mathcal{TS}_{new}$ (*2d.iii*)) is found (*2b.i*) and the process continues until success or until it is clear that the chosen H can't be extended any more.

The following lemma states that the above implementation of Algorithm 5.2 satisfies the output conditions of Definition 5.7.

Lemma 5.10 *Suppose the find_member_chs(ic-$\mathcal{TS}_{new}$, Loc_Seen) algorithm is correctly implemented. Then:*

*(1) If algorithm **ComputeTSS** returns a temporal status set H^*, then H^* satisfies the output conditions of Definition 5.7.*
*(2) If algorithm **ComputeTSS** returns "No", then there is no temporal status set satisfying the output conditions of Definition 5.7.*

Theorem 5.11 (Algorithm 5.1 is Correct and Complete)
Algorithm 5.1 generates a feasible temporal status set (if one exists).

Proof: Suppose Algorithm 5.1 returns $\mathcal{TS}_{\mathsf{t_{now}}}$. In this case, $\mathcal{TS}_{\mathsf{t_{now}}}$ is compatible via step *(4)(a)*, and feasible via step *(4)(c)*.

Conversely, suppose Algorithm 5.1 returns "No." In this case, we know that **FindCompTSS** returned "No" which means that it was unable to find a temporal status set compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$. But this means that all temporal status set compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ are in *Seen* which means none of them is feasible. ∎

We note that the set generated by **ComputeTSS** is not minimal, i.e., there might be smaller sets closed under the rules which are compatible with ic-$\mathcal{TS}$. The reason is that the generated hitting set might not have been optimal. As an illustration, we consider the ic-$\mathcal{TS}$ consisting of $\mathbf{P}\alpha : [1,2]$, $\mathbf{P}\alpha : [3,4]$, and the program $\mathbf{P}\alpha : [1,1] \leftarrow \mathbf{P}\alpha : [2,2] \& \mathbf{P}\alpha : [4,4]$. If we start with the *wrong* hitting set, namely $\{\mathbf{P}\alpha : [2,2], \mathbf{P}\alpha : [4,4]\}$, **ComputeTSS** produces $H^* = \{\mathbf{P}\alpha : [1,1], \mathbf{P}\alpha : [2,2], \mathbf{P}\alpha : [4,4]\}$. But if we start with a subset of H^*, namely $\{\mathbf{P}\alpha : [1,1], \mathbf{P}\alpha : [4,4]\}$, then this is a minimal status set closed under the rules and compatible with ic-$\mathcal{TS}$. It is easy to modify Algorithm 5.2 to take this into account. We can either compute all solutions and check for minimality, or we can take a solution and compute a hitting set different from the one we started with. We then call **ComputeTSS** on this new hitting set. The result could be a smaller solution than originally obtained. This is stated in the next remark.

Remark 5.1 (Rational Temporal Status Set Computation) A slight modification of the **ComputeTSS** subroutine allows for the computation of *rational* status sets. Namely if we require the computed set of Algorithm 5.7

to be *minimal while being closed under $\mathcal{TP}$, and compatible with* ic-$\mathcal{T}S$, and change Definition 5.7 accordingly, then Algorithm 5.1 generates a rational feasible temporal status set (if one exists). The proof is literally the same. The fact that the outcome is rational follows from the minimality requirements in **ComputeTSS**.

Once the rational temporal status sets are known, the feasible status sets are easily obtained by adding new action atoms and then applying the **ComputeTSS** function. This is because each feasible status set is an extension of its underlying rational status set.

Remark 5.2 Note that we required a temporal status set to satisfy $\mathcal{T}S_{\mathsf{t_{now}}}(i) \neq \emptyset$ for only finitely many i. It is easy to write programs where **FindCompTSS** never terminates but yet such a finite temporal status set does not exist. For example, consider the simple loop $\mathbf{P}\alpha : [t+1, t+1] \;\leftarrow\;$ meaning that for all i: $\mathbf{P}\alpha \in \mathcal{T}S_{\mathsf{t_{now}}}(i)$.

It is obvious that in general, the complexity introduced by the subroutine **ComputeTSS** is very high. The underlying source is *reasoning by cases* which is known to be of high complexity. A possible remedy (which is beyond the scope of this paper) is to consider special classes of programs and to show that for these classes, the function **ComputeTSS** can be realized with low overall complexity.

6 Application of taps: Intention Reconciliation by Collaborative Agents

In this section, we present a multiagent application of temporal agent programs involving time. The example describes how different agents reconcile existing commitments to a group with new requests/opportunities. More applications can be found in (Dix, Kraus, and Subrahmanian 2000).

Many applications have been proposed that require individual agents to work collaboratively to satisfy a shared goal (Decker and Li 1998; Sen, Haynes, and Arora 1997; Sycara and Zeng 1996). In these settings, agents form teams to carry out actions, making commitments to their group's activity and to their individual actions in service of that activity. As rational agents, the individuals who form teams must be able to make individually rational decisions about their commitments and plans. As members of a team, they must be responsible to the team and, dually, be able to count on one another. In particular, there are many factors that an agent needs to take into consideration when confronted with a request to perform an individual action γ which conflicts with prior commitment to perform a group action β.

In (Sullivan, Glass, Grosz, and Kraus 1999; D. G. Sullivan and Kraus 2000) such factors were studied and tested in a simulation environment called SPIRE (SharedPlans Intention-Reconciliation Experiments). In this environment, a team of agents $(g_1, \ldots, g_n)$ work together on group activities, called GroupTasks, each of which consists of doing a set of tasks. Each task has a specified time period and is assigned to one of the agents in the group by a central scheduling function that has complete knowledge of agents' defaulting behavior. Agents receive income for the tasks they do which can be used in determining an agent's current utility and in estimating its future-expected utility.

Sometimes, agents, are offered the opportunity to do some action γ that conflicts with a task β they have been assigned. If the agent chooses the new opportunity, it defaults on the task β with which γ conflicts. If there is an available replacement agent that is capable of doing β, the task is given to that agent; otherwise, β goes undone. The group as a whole incurs a cost whenever an agent defaults, and this cost is divided equally among the group's members. The cost of a particular default depends on its impact on the group; it is larger if there is no replacement.

SPIRE currently uses a social-commitment policy (SCP) in which a portion of each agent's weekly tasks is assigned based on how "responsible" it has been over the course of the simulation. When an agent needs to make a decision whether to keep its commitment to the group (i.e., do β) or default and do the outside option, γ it weighs the impact of the choice on three factors: current income (CI), future expected income (FEI) given the SCP (i.e., effect on ranking and subsequent task assignment), and loss of good-guy stature in the community independent of effect on income (BP). For each option, it combines the three factors into an expected utility value using normalization methods of multiple attribute decision making theory (Yoon and Hwang 1995) and choose the option with the highest expected utility.

SPIRE is a simulation environment and only simulates agent decision making, but does not provide tools for creating and deploying such agents. In this section we demonstrate how such agents can be programmed using `tap`.

We associate with each agent, a specialized package called `utility` that supports the following functions in order to compute its expected utility. Such a package can be easily implemented in the *IMPACT* agent development environment (Eiter, Subrahmanian, and Rogers 2000) and agent developers can choose to use it if their agent needs to perform intention reconciliation of the sort described in SPIRE.

- `utility` : *current_income*$(\mathtt{A}, \mathtt{Rep}) \rightarrow$ `Real`
  current_income takes as input, an action and a flag. The flag indicates, in

case that the action is an outside option, whether there is a replacement if the agent defaults on its current assigned action in order to do the specified action. The function returns the income from the task or outside offer, as well as the agent's share of the group cost if it defaults.

- $\mathtt{utility}:\mathit{future\_e\_income}(\mathtt{A},\mathtt{Rep}) \rightarrow \mathtt{Real}$
  future_e_income takes as input, an action and a "replacement" flag and returns the agent's estimate of its income in the future, based on its current ranking if it will do A.
- $\mathtt{utility}:\mathit{brownie\_points}(\mathtt{A},\mathtt{Repl}) \rightarrow \mathtt{Real}$
  brownie_points takes as input, an action and a "replacement" flag and provides a measure of the agent's sense of its reputation as a responsible collaborator if it does A under the specified conditions.
- $\mathtt{utility}:\mathit{combine\_factors}(\mathtt{A},\mathtt{CI},\mathtt{FEI},\mathtt{BP}) \rightarrow \mathtt{Real}$
  combine_factors combines the three factors that the agent should take into consideration when making a decision, and computes the expected utility of the agents from doing A. The combination is done using normalization methods of multiple attribute decision making theory (Yoon and Hwang 1995).

We associate with each agent, an additional specialized package called schedule that supports the code call listed below and others that determine its schedule.

- $\mathtt{schedule}:\mathit{check}(\mathtt{T}) \rightarrow \mathtt{Actions}$
  check returns the action (to be precise the name of an action together with its arguments) that the agent is scheduled to do in the specified time, if such an action exist, and null, otherwise.

The *msgbox* package is discussed in depth in (Subrahmanian, Bonatti, Dix, Eiter, Kraus, Özcan, and Ross 2000). We now extend it by adding the following code calls:

- $\mathtt{msgbox}:\mathit{gather\_outside\_req}() \rightarrow \{\langle \mathit{Actions}, \mathit{Time}\rangle\}$
  gather_outside_req returns pairs of an action and a time period of outside offers.
- $\mathtt{msgbox}:\mathit{check\_replacement}(\mathtt{T}) \rightarrow \mathtt{Boolean}$
  check_replacement takes a time period as input, and returns whether there is an agent who is available at this time period.

Here are a few rules that can be used to program agents in SPIRE.

r1 (Schema for actions $\alpha \in$ Actions):

$$\mathbf{O}\alpha:[\mathtt{X}_{\mathsf{now}},\mathtt{X}_{\mathsf{now}}] \leftarrow \quad \mathbf{in}(\texttt{"}\alpha\texttt{"}, \mathtt{schedule}:\mathit{check}(\mathtt{X}_{\mathsf{now}})):[\mathtt{X}_{\mathsf{now}},\mathtt{X}_{\mathsf{now}}]$$

This represents a *schema* of rules: an instance is obtained by substituting the metavariable α with a particular action $act(\overline{x})$ (the tuple $\overline{x}$ represents the arguments) from a prespecified finite set Actions. We denote by "α"

resp. by "$act(\overline{x})$" the *string* consisting of the complete name of the action together with the arguments (we consider such strings as constants in the underlying language).

The agent is obliged to do an action α if it is in its schedule when the action time arrives.

r2: $\mathbf{O}\mathit{removeAction}(\mathtt{Y}):[\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow$
$(\,\mathbf{Do}\ \mathit{addAction}(\mathtt{Y},\mathtt{T})\ \&\ \mathbf{in}(\mathtt{Y}, \mathsf{schedule}: \mathit{check}(\mathtt{T}))\,):[\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$

The agent maintains the consistency of its schedule. It is obliged to remove Y from its schedule if it adds a conflicting action (in SPIRE, an agent cannot have more than one task *at a time* in its schedule).

r3: $\mathbf{P}\mathit{addAction}(\mathtt{Y},\mathtt{T}):[\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow$
$(\,\mathbf{in}(\langle Y, T\rangle, \mathsf{msgbox}: \mathit{gather\_outside\_req}())\quad \&$
$\mathbf{in}(\mathtt{null}, \mathsf{schedule}: \mathit{check}(\mathtt{T}))$
$):[\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$

The agent is permitted to add an action to its schedule if it was requested to do it and it does not conflict with its other scheduled actions.

r4: $\mathbf{P}\mathit{addAction}(\mathtt{Y},\mathtt{T}):[\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow$
$(\,\mathbf{in}(\langle \mathtt{Y}, \mathtt{T}\rangle, \mathsf{msgbox}: \mathit{gather\_outside\_req}())\quad \&$
$\mathbf{in}(\mathtt{Y1}, \mathsf{schedule}: \mathit{check}(\mathtt{T}))\quad \& \neq (Y1, Y)\quad \&$
$\mathbf{in}(\mathtt{X}, \mathsf{msgbox}: \mathit{check\_replacement}(\mathtt{T}))\quad \&$
$\mathbf{in}(\mathtt{CI}, \mathsf{utility}: \mathit{current\_income}(\mathtt{Y},\mathtt{X}))\quad \&$
$\mathbf{in}(\mathtt{FEI}, \mathsf{utility}: \mathit{future\_e\_income}(\mathtt{Y},\mathtt{X}))\quad \&$
$\mathbf{in}(\mathtt{BP}, \mathsf{utility}: \mathit{brownie\_points}(\mathtt{Y},\mathtt{X}))\quad \&$
$\mathbf{in}(\mathtt{U}, \mathsf{utility}: \mathit{combine\_factors}(\mathtt{Y},\mathtt{CI},\mathtt{FEI},\mathtt{BP}))\ \&$
$\mathbf{in}(\mathtt{CI}', \mathsf{utility}: \mathit{current\_income}(\mathtt{Y1}, \text{"na"}))\quad \&$
$\mathbf{in}(\mathtt{FEI}', \mathsf{utility}: \mathit{future\_e\_income}(\mathtt{Y1}), \text{"na"})\quad \&$
$\mathbf{in}(\mathtt{BP}', \mathsf{utility}: \mathit{brownie\_points}(\mathtt{Y1}, \text{"na"}))\quad \&$
$\mathbf{in}(\mathtt{U}', \mathsf{utility}: \mathit{combine\_factors}(\mathtt{Y1},\mathtt{CI}',\mathtt{FEI}',\mathtt{BP}'))\ \&$
$> (\mathtt{U},\mathtt{U}')$
$):[\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$

The agent is permitted to add a new action Y to its schedule, even if it conflicts with another action, $Y1$, if its expected utility from Y is larger than its expected utility from $Y1$. In order to compute the expected utility of an action the agent computes its current income, future expected income and its brownie points from doing the action.

r5: $\mathbf{O}\mathit{SendAnnouncement}(\mathsf{scheduler}, \text{"default"}, \langle X, T\rangle):[\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow$
$\mathbf{Do}\ \mathit{removeAction}(\mathtt{X},\mathtt{T}):[\mathtt{X}_{\mathsf{now}}+1, \mathtt{X}_{\mathsf{now}}+1]$

If the agent intends to remove a collaborative action from its schedule, it is obliged to announce the scheduler agent about its intention.

7 Related Work

As we have already described some relevant work in the introduction, in this section, we either describe details of (some of) that work and also present some other related work.

Temporal Reasoning: *MetaTem* and *Concurrent MetaTem* (Barringer, Fisher, Gabbay, Gough, and Owens 1989; Fisher 1994) start with the notion of literal in classical first order logic and define well formed formulas (wffs) inductively via usage of connectives such as $\bigcirc$ (next time), $\Diamond$ (eventually), $\Box$ (forever), $\bullet$ (previous time instance) plus some variants. It is known that all *MetaTem* rules can be encoded in a syntactic fragment consisting of the following four types of rules (cf. (Fisher 1994)).

(1) Initial $\Box$-rules have the form "Disjunction of *literals* is true at the start time." In taps, if we wish to express that a disjunction of status literals $(\mathsf{Op}_1 a_1 \vee \cdots \vee \mathsf{Op}_m a_m \vee \neg\mathsf{Op}_{m+1} a_{m+1} \vee \cdots \vee \neg\mathsf{Op}_{m+n} a_{m+n})$ is true, then we can use either the feasible (or rational) temporal status set semantics and write it as the rule $Op_1 a_1 : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow \neg\mathsf{Op}_2 a_2 : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \wedge \ldots \wedge \neg\mathsf{Op}_m a_m : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \wedge \mathsf{Op}_{m+1} a_{m+1} : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \wedge \ldots \wedge \mathsf{Op}_{m+n} a_{m+n} : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$. (Note that if the reasonable status set semantics of (Eiter, Subrahmanian, and Pick 1999) is used, then this would not be true).

(2) Global $\Box$-rules have the form "If a conjunction of literals was true at the previous time instance (and such a previous time instance existed), then a disjunction of *literals* is true now." This can be expressed using the same rule as above except that the condition is added to the rule body.

(3) Initial $\Box$-rules which say that in the initial state $\Box\ell$ is true (which of course means that that literal ℓ is true "now" and in all future states). When ℓ is positive, this can be expressed via the tap rules $\ell : [0, 0] \leftarrow$ and $\ell : [X + 1, X + 1] \leftarrow \ell : [X, X]$. When we have a negation literal $\ell = \neg a$ then we may express this via the rules (using a new "artificial" status atom a') $a' : [0, 0] \leftarrow$, $a' : [X + 1, X + 1] \leftarrow a' : [X, X]$ and $a : [X, Y] \leftarrow \neg a' : [X, Y]$ and using the rational temporal status set semantics (feasible will not work in this case).

(4) Global $\Diamond$-rules have the form "If a conjunction of literals was true at the previous time instance (and such a previous time instance existed), then $\Box\ell$ is true now." These can be expressed via taps as $\ell : [\mathtt{X}_{\mathsf{now}} + X, \mathtt{X}_{\mathsf{now}} + X] \leftarrow C : [\mathtt{X}_{\mathsf{now}} - 1, \mathtt{X}_{\mathsf{now}} - 1]$ where C is the desired conjunction and ℓ is assumed to be positive. If $\ell = \neg a$ is negative, then we use the rational temporal status set semantics and use the rules $a' : [\mathtt{X}_{\mathsf{now}} + X, \mathtt{X}_{\mathsf{now}} + X] \leftarrow C : [\mathtt{X}_{\mathsf{now}} - 1, \mathtt{X}_{\mathsf{now}} - 1]$ and $a : [X, Y] \leftarrow \neg a' : [X, Y]$.

As shown above, each of the four types of rules above in *MetaTem* can be expressed via taps.

Parts of ordinary agent programs (Eiter, Subrahmanian, and Pick 1999) can also be expressed via *MetaTem* under some conditions. For instance, in ordinary agent programming, $\mathbf{P}\alpha$ can be written as $\Box\neg\alpha$ *if there exists an action* $\neg\alpha$ *corresponding that reverses the effects of action* α. Likewise, $\mathbf{Do}\,\alpha$ can be expressed as $\bigcirc\alpha$ indicating that α is done at the next time instance. One may encode $\mathbf{O}\alpha$ as $\bigcirc obl_\alpha$ where obl_α is a special action with no preconditions and/or effects, saying that the "dummy" action α is obligatory. However, the syntax and techniques for accessing heterogeneous and/or legacy data sources in *IMPACT* are unique and are not included in *MetaTem*, though this is not impossible. In addition, the syntax of taps allows arguments to become temporal parameters as shown via the two examples in the Introduction of this paper. In general, speaking strictly in logical terms, it is hard in temporal logic to say things like "If $p(X, Y)$ holds, then do α $f(X, Y)$ time units from now." The reason is that in *MetaTem*, we cannot dynamically infer the number of occurrences of the $\bigcirc$ modality at run-time.

On the flip side, *MetaTem* and *Concurrent MetaTem* have studied things we have not studied, e.g. the interaction of temporal agents and belief structures (Fisher and Ghidini 1999) and the development of proof procedures for full fledged fragments of their calculus, while our methods compute status sets for positive taps only. This provides a rich avenue for future research.

In addition, there is the issue of determining when to use a *MetaTem* style language for temporal agent reasoning, and when to use a tap-style formalism. If heterogeneous data sources are involved, then taps clearly offer a principled way of dealing with them. Furthermore, if time computations depend upon the values of variables which are only available dynamically after deployment of the agent, as in the cases involving the shipping schedules mentioned in the Introduction, then taps again offer a clean solution. However, in cases where one models purely logical phenomena where data structures and legacy code bases are not an issue, then *MetaTem* may offer advantages. For instance, the Snow White example in (Fisher 1994) can be easily expressed both in *MetaTem* and via taps— which one is preferable depends on the user's level of comfort with each. In this example, Snow White hands out sweets to different dwarves, each of whom has a different strategy for asking for sweets. The dwarf eager asks for a sweet at time 0 and then asks for another whenever he receives one. This can be modeled by having $\mathtt{eager}$'s $\mathtt{tap}$ contain the rules:

$$\begin{aligned}
&\mathbf{Do}\, ask_for_sweet() : [0,0] && \leftarrow \\
&\mathbf{Do}\, ask_for_sweet() : [\mathtt{T}+1, \mathtt{T}+1] && \leftarrow (\mathbf{in}(\mathtt{X}, \mathtt{eager} : getmessage())\ \& \\
& && \quad = (\mathtt{X.content}, \text{"You get a sweet."})\ \& \\
& && \quad = (\mathtt{X.from}, \text{"Snow White"})\ \& \\
& && \quad = (\mathtt{X.to}, \text{`"eager"})) : [\mathtt{T}, \mathtt{T}].
\end{aligned}$$

Here, $\mathtt{eager}$ has a rule saying that if he receives a message from Snow White saying he has been awarded a sweet at time $\mathtt{T}$, then he immediately asks for another sweet at time $\mathtt{T}+1$. This model assumes (as does Fisher's example) that all dwarves see all messages being exchanged. Likewise, the dwarf $\mathtt{mimic}$ asks for a sweet whenever he knows $\mathtt{eager}$ asked for one. This can be modeled by having $\mathtt{mimic}$'s $\mathtt{tap}$ contain the rule:

$$\begin{aligned}
&\mathbf{Do}\, ask_for_sweet() : [\mathtt{T}+1, \mathtt{T}+1] \leftarrow (\mathbf{in}(\mathtt{X}, \mathtt{mimic} : getmessage())\ \& \\
& \quad = (\mathtt{X.content}, \text{"Request a sweet"}))\ \& \\
& \quad = (\mathtt{X.from}, \text{"eager"})\ \& \\
& \quad = (\mathtt{X.to}, \text{"Snow White"})) : [\mathtt{T}, \mathtt{T}]
\end{aligned}$$

In a similar vein, we can encode the behavior of all the dwarves and Snow White as well.

Brzoska (1995) introduced a temporal logic framework, *metric temporal logic*, and showed how this can be translated into Prolog programs with constraints, s.t. satisfiability checking of the constraints can be done in time linear in the variables. He uses modal operators like $\Box$ and $\Diamond$ and temporal annotations. Although we use only the $\Diamond$ operator, the $\Box$ operator can be easily expressed. We are more expressive in that our annotations can be arbitrary terms containing variables, whereas Brzoska's are fixed. In principle, our status sets can be seen as consisting of *successfully derived goals* (in Brzoska's terminology), but modulo the whole deontic machinery. We have in addition the compact representation of status sets, whereas nothing comparable is provided in Brzoska (1995), and the distinguished time point $\mathsf{t}_{\mathsf{now}}$ (to evaluate our rules by distinguishing between the past and the future). Similar remarks apply to (Frühwirth 1995).

Abadi and Manna (1987) proposed a language called *TEMPLOG* which was later extensively studied by another student of Manna, viz. Baudinet (1992). They take classical first-order logic and define a next atom to be of the form $\bigcirc^i A$ where $i \geq 0$ and A is an atom in first-order logic. As usual, $\bigcirc A$ is true at time t if A is true at time $t+1$. She defines two kinds of rules—initial program clauses (IPC) are of the form $N_0 \leftarrow N_1 \wedge \ldots \wedge N_k$

and permanent program clauses having the form $\Box C$ where C is an IPC. If $N_i = \bigcirc^{num_i} A_i$, then an IPC may be expressed as the `tap` rule

$$A_0 : [num_0, num_0] \leftarrow A_1 : [num_1, num_1] \ \wedge \ldots \wedge \ A_k : [num_k, num_k].$$

A permanent rule of the form $\Box C$ where C has the IPC form shown above can be expressed as:

$$A_0 : [\mathtt{X} + num_0, \mathtt{X} + num_0] \leftarrow A_1 : [\mathtt{X} + num_1, \mathtt{X} + num_1] \ \wedge \ldots \wedge \\ A_k : [\mathtt{X} + num_k, \mathtt{X} + num_k].$$

It is easy to see that *TEMPLOG* and its variants cannot express (at least easily) the two examples listed in the introduction.

In addition, all the above temporal frameworks described above provide no explicit way of building on top of arbitrary packages—something that `taps` certainly do.

Active databases: Active databases (Fisher 1995; U. Dayal and E. Hanson and J. Widom 1995; Gottlob, Moerkotte, and Subrahmanian 1996) use rules of the form "If condition C is true in the current database state and actions $A_1, \ldots, A_n$ have been done and none of actions $B_1, \ldots, B_m$ have been done, then do action A." This can be expressed via an ordinary agent program of Eiter, Subrahmanian, and Pick (1999) without even using the temporal features defined in this paper as follows:

$$\mathbf{Do}\, A \leftarrow C \ \wedge \ \mathbf{Do}\, A_1 \ \wedge \ldots \wedge \ \mathbf{Do}\, A_n \ \wedge \ \neg\mathbf{Do}\, B_1 \ \wedge \ \ldots \wedge \ \neg\mathbf{Do}\, B_m.$$

Theories of active databases—on which one of the authors has worked—only deal with relational and object databases, not with arbitrary packages. Furthermore, most theories of active databases have no temporal support whatsoever nor any deontic support.

Actions and time: have been extensively studied by many researchers in several areas of computing (e.g., (Manna and Pnueli 1992; Haddawy 1991; Morgenstern 1988; Allen 1984; Lamport 1994; Nirkhe, Kraus, Perlis, and Miller 1997; Dean and McDermott 1987; McDermott 1982; Rabinovich 1998; Singh 1998)). We present here the main differences between others' work and ours, and discuss work that combines time with deontic operators. Surveys of research on temporal reasoning include (Benthem 1991; Benthem 1995; Baker and Shoham 1995; Lee, Tannock, and Williams 1993; Vila 1994; Alur and Henzinger 1992; Chittaro and Montanari 1998).

- One of the main differences between our approach and the temporal logic approach is that we allow the history to be partially specified but in their approach, the entire history is defined for any time period. Allowing the history to be partially defined is needed when modeling bounded agents, as by Subrahmanian, Bonatti, Dix, Eiter, Kraus, Özcan, and Ross (2000).
- In our model, time can be expressed explicitly (as in, for example, (Thomas, Shoham, Schwartz, and Kraus 1991)). We do not use the modal temporal

logic approach where time periods cannot be expressed in the language.

- We use a simple interval based temporal logic (Allen 1984), and introduce a mechanism for specifying intermediate effects of an action. We focus on the semantics of temporal agent programs which specify the commitments of the agents. We do not have modal operators associated with time but only with the obligations, permissions, etc. of an agent. Other interval temporal logics (e.g., (Halpern and Shoham 1991; Allen and Ferguson 1994; Artale and Franconi 1998)) were developed for describing complex plans and/or the study of appropriate semantics and their complexity for interval based temporal logics.
- We presented a temporal interval constraint language in order to provide a compact way to represent temporal feasible status sets. Other attempts to use constraints to simplify temporal reasoning include Dechter, Meiri, and Pearl (1991) who were one of the first to apply general purpose constraint solving techniques (such as the Floyd-Warshall shortest path algorithm) to reason about temporal relationships, and Koehler and Treinen (1995) who use a translation of their interval-based temporal logic (LLP) into constraint logic (CPL) to obtain an efficient deduction system.
- In our framework, we allow temporal indeterminacy. For instance, a tax auditor may be obliged to notify the subject of the audit within 30 days of performing the audit, and an e-commerce billing system be forbidden from sending a client more than one bill per week. However, the auditor may send the notification on any one of 30 days leaving him with some choices. Likewise, our framework (via the notion of a temporal action state condition) allows us to specify and evaluate conditions that are true at some time point in a given set—this is similar to disjunctive reasoning.

Representation of Goals: The BDI agent architecture (Rao 1995; Rao and Georgeff 1991) has raised the point that agents might have goals and might need to reason with them. This has often been taken to mean that *all* agents must be able to construct plans. We take the position that the BDI model merely says that an *agent development environment* (such as the *IMPACT* system described in (Eiter, Subrahmanian, and Rogers 2000; Subrahmanian, Bonatti, Dix, Eiter, Kraus, Özcan, and Ross 2000)) should support the construction of agents that need to plan.

In *IMPACT*, once the agent program, integrity constraints, etc. are specified, the *IMPACT* system creates a mobile piece of Java code. As mobility is a major requirement in a large scale agent system (we'd like agents to move to a host with either the right data or with little load on it if needed) we'd like to keep agents "lightweight." They should not leave a large footprint on a target host. Forcing every agent to have planning software causes agents to cease being "lightweight." Thus, in our system, we support goal oriented reasoning (or planning) by having a diverse set of planning agents as part of the *IMPACT* system. These include:

(1) An agent developed by Dana Nau and his colleagues called SHOP that is a well known hierarchical task network style planner (Nau, Cao, Lotem,

and Muñoz-Avila 1999; Dix, Munoz-Avila, Nau, and Cao 2000)

(2) An agent that accesses MapQuest and can plan on-road routes between two geographic addresses;

(3) An agent that can create off-road route plans using a US military off-road route planner (Benton, Iyengar, Deng, Brener, and Subrahmanian 1996).

An agent requiring plan construction can ship a request to (the appropriate one of) the above agents, specifying the planning problem it needs solved.

Another advantage of this solution is that new planning agents can be seamlessly added. For instance, we are agentizing a planner based on difference constraint solving of the kind used extensively in scheduling multimedia presentations (Subrahmanian 1998).

Several researchers have combined logics of commitments and actions with time. Cohen and Levesque (1990a, Cohen and Levesque (1990b) define the notion of persistence goals (P-GOAL). They assume that if an agent has a P-GOAL toward a proposition, then the agent believes that this proposition is not true now, but that it will be true at some time in the future. The agent will drop a persistent goal only if it comes to believe that it is true or that it is impossible. In their logic, time doesn't explicitly appear in the proposition; thus, they cannot express a P-GOAL toward propositions that will be true at some specific time in the future or consider situations where a proposition is true now, but which the agent believes will become false later and therefore has a P-GOAL to make it true again after it becomes false. They do not have any notion of agent programs. Their logic is used for abstract specifications of agents behavior.

Sonenberg, Tidhar, Werner, Kinny, Ljungberg, and Rao (1992) use a similar approach to that of Cohen and Levesque. However, they provide detailed specifications of various plan-constructs that may be used in the development of collaborative agents. Shoham (1993)'s Agents0 has programs with commitments and a very simple mechanism to express time points.

Fiadeiro and Maibaum (1991) provide a complex temporal semantics to the deontic concepts of permission and obligation in order to be able to reason about the temporal properties of systems whose behavior has been specified in a deontic way. They are interested in the normative behaviour of a system, while we focus on decision making of agents over time.

Deontic Logic: Horty (1996) proposes an analysis of what an agent ought to do. It is based on a loose parallel between action in indeterministic time (branching time) and choice under uncertainty, as it is studied in decision theory. Intuitively, a particular preference ordering is adapted from a study of choice under uncertainty; it is then proposed that an agent ought to see to it that A occurs whenever the agent has an available action which guarantees the truth of A, and which is not dominated by another action that does not guarantee the truth of A. The obligations of our agents are influenced by their programs and we do not use decision theory. An agent's obligations is determined using its status set and we provide a language for writing agents program with time.

Dignum and Kuiper (1997, Dignum, Weigand, and Verharen (1996) combine temporal logic with deontic logic. Their semantics is based on Kripke models with implicit time while ours is based on status sets where time can be explicitly expressed. They focus on modeling deadlines and we focus on programming agents. They admit that automatic reasoning with specifications written in their language is not yet possible.

J.J. Ch-Meyer's group's interesting work on deontic logic (Hindriks, de Boer, van der Hoek, and Meyer 1997; Meyer and Wieringa 1993) for building agents is closely related. However, his work does not build explicitly on top of heterogeneous data structures, and no explicit support is present for modeling actions with intermediate effects and with constructing agents that can reason with past/future commitments (though his work does apply to reasoning logically about agents over time).

Their work on dynamic of commitments (Meyer, van der Hoek, and van Linder 1999) is also important and relevant. They present an expressive formalization of motivational attitudes such as wishes, goals and commitments. They study the important issue of acts associated with selecting between wishes and with (un)commiting to action, In this work agents can reason about the change in their commitments, but there is no explicit support for reasoning about time or build explicitly on top of heterogeneous data structures.

An important aspect of deontic logic is what an agent should do when it faces "moral dilemmas" or conflicts. This occurs (for instance) when an agent has two or more obligations, but cannot do them all. In (Eiter, Subrahmanian, and Pick 1999), we showed how any of the semantics of ordinary agent programs (e.g. feasible status set, rational status set, etc.) can be modified to handle this problem and we introduced (Eiter, Subrahmanian, and Pick 1999) *weak* (feasible status set, rational status set, etc.) semantics that accounted for moral dilemmas. The same method can be straightforwardly applied to weaken the feasible and rational temporal status set semantics defined in this paper, but for space reasons, we do not do this.

8 Conclusions

There has been intensive work over the years on the topic of software agents. By now, the idea that agents are entities that have a "state" and that autonomously (and hopefully intelligently) react to changes in the state, has taken firm root (Rosenschein and Zlotkin 1994; Shoham 1993). Important aspects of how to build agents and reason about them logically have been studied by many researchers—(Huhns and Singh 1997) provides an excellent overview.

In (Eiter, Subrahmanian, and Pick 1999), the authors proposed a formal methodology for building agents "on top" of heterogeneous data structures and/or legacy software. In the formalism proposed in (Eiter, Subrahmanian, and Pick 1999), instant t, the state of the agent was assumed to be "acceptable" (in the terms of this paper, "acceptable" is synonymous with "satisfying the integrity constraints"). However, the agent's state would be "disturbed" by an external event at time t. The receipt of a message by the agent (e.g. from a sensing device, from a clock agent recording a "tick", from another agent, or a human) is one way the agent's state can change. When such a state change occurs, the agent uses its associated structures (primarily the agent program) to find a feasible (or rational) status set S. It then concurrently and *immediately* executes all actions in $\mathbf{Do}\,(S)$ to obtain a new state that satisfies the integrity constraints. This cycle is repeated *ad infinitum* or till the agent is "killed."

A key limitation in the (Eiter, Subrahmanian, and Pick 1999) framework is that agents must act *immediately.* In addition, all actions are assumed to be *instantaneous* (i.e. actions take no time for execution). However, in the real world, both these assumptions are restrictive. After all, human beings routinely make commitments for the future. Such commitments are made based on their goals, and on their resources, and their obligations. Similarly, most actions that agents execute take time—and during this time, there may be a need to update the agent's state to record the fact that an action need not finish executing for it to have intermediate effects.

In this paper, we first propose a syntax for "timed actions" that allows actions to have duration and that allows actions to have intermediate effects while executing. We then extend the concept of agent programs proposed in (Eiter, Subrahmanian, and Pick 1999) to handle temporal aspects of agent decision making. Specifically, this paper allows an agent to schedule actions now, as well as in the future. It allows agents to determine that certain actions are forbidden/obligatory/permitted/to be done at future instances of time, based on conditions known to be true at the time the actions are executed (including predictions *currently held* to be true). We propose a formal syntax and semantics for agents of this kind, and provide (in the case of positive temporal agent programs only), algorithms that the agent might use to compute such semantics.

Acknowledgments

We thank an anonymous reviewer for excellent insightful comments that helped improve the quality of the paper.

References

Abadi, M. and Z. Manna (1987). Temporal Logic Programming. In *Proc. of the 1987 IEEE Symposium on Logic Programming*, San Francisco, CA, pp. 4–16.

Allen, J. (1984). Towards a General Theory of Action and Time. *Artificial Intelligence 23*(2), 123–144.

Allen, J. F. and G. Ferguson (1994). Actions and Events in Interval Temporal Logic. *Journal of Logic and Computation 4*(5), 531–579.

Alur, R. and T. A. Henzinger (1992, June). Logics and Models of Real Time: A Survey. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg (Eds.), *Proceedings of Real-Time: Theory in Practice*, Volume 600 of *Lecture Notes in Computer Science*, pp. 74–106. Berlin, Germany: Springer-Verlag.

Åquist, L. (1984). Deontic Logic. In D. Gabbay and F. Guenthner (Eds.), *Handbook of Philosophical Logic, Volume II*, Chapter II.11, pp. 605–714. D. Reidel Publishing Company.

Arisha, K., F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus (1999, March/April). IMPACT: A Platform for Collaborating Agents. *IEEE Intelligent Systems 14*, 64–72.

Artale, A. and E. Franconi (1998). A Temporal Description Logic for Reasoning about Actions and Plans. *Journal of Artificial Intelligence Research 9*, 463–506.

Baker, A. B. and Y. Shoham (1995). Nonmonotonic Temporal reasoning. In D. Gabbay, C. Hogger, and J. Robinson (Eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming.* Oxford University Press.

Barringer, H., M. Fisher, D. Gabbay, G. Gough, and R. Owens (1989, June). MetateM: A framework for programming in temporal logic. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness, Lecture Notes in Computer Science 430.* Springer-Verlag.

Baudinet, M. (1992). A simple proof of the completeness of temporal logic programming. In L. F. del Cerro and M. Penttonen (Eds.), *Intensional Logics for Programming*, pp. 51–84. Oxford Univ. Press.

Benthem, J. v. (1991). *The logic of time.* Kluwer Academic Publishers.

Benthem, J. v. (1995). Temporal logic. In D. Gabbay, C. Hogger, and J. Robinson (Eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming*, pp. 241–350. Oxford University Press.

Benton, J., S. Iyengar, W. Deng, N. Brener, and V. Subrahmanian (1996). Tactical Route Plandheing, New Algorithms for Decomposing the Map. *Intl. Journal of Tools in Artificial Intelligence 5*(1 and 2), 199–218.

Brzoska, C. (1995). Temporal Logic Programming with Metric and Past Operators. In M. Fisher and R. Owens (Eds.), *Executable Modal and Temporal Logics*, Volume 897 of *Lecture Notes in Artificial Intelligence*, pp. 21–39. Springer-Verlag.

Cattell, R. G. G., et al. (1997). *The Object Database Standard: ODMG-93.*

Morgan Kaufmann.

Chittaro, L. and A. Montanari (1998). Editorial: Temporal representation and reasoning. *Annals of Mathematics and Artificial Intelligence 22*, 1–4.

Cohen, P. and H. Levesque (1990a). Intention is Choice with Commitment. *Artificial Intelligence 42*, 263–310.

Cohen, P. R. and H. Levesque (1990b). Rational Interaction as the Basis for Communication. In P. R. Cohen, J. L. Morgan, and M. E. Pollack (Eds.), *Intentions in Communication*, pp. 221–256. Cambridge, MA: MIT Press.

Cormen, T. H., C. E. Leiserson, and R. L. Rivest (1989). *Introduction to Algorithms*. McGraw-Hill.

D. G. Sullivan, B. J. G. and S. Kraus (2000, July). Intention reconciliation by collaborative agents. In *Proc. of ICMAS-2000*, Boston, pp. 293–300.

Dean, T. and D. McDermott (1987). Temporal data base management. *Artificial Intelligence 32*(1), 1–55.

Dechter, R., I. Meiri, and J. Pearl (1991). Temporal Constraint Networks. *Artificial Intelligence 49*, 61–95.

Decker, K. and J. Li (1998). Coordinated hospital patient scheduling. In *Proc. of ICMAS '98*, Paris, pp. 104–111.

Dignum, F. and R. Kuiper (1997). Combining Deontic Logic and Temporal logic for Specification of Deadlines. In *Proceedings of Hawaii International Conference on System Sciences, HICSS-97*, Maui, Hawaii.

Dignum, F., H. Weigand, and E. Verharen (1996). Meeting the deadline: on the formal specification of temporal Deontic constraints. *Lecture Notes in Computer Science 1079*, 243–.

Dix, J., S. Kraus, and V. Subrahmanian (2000, August). Temporal agent programs. Technical Report CS-TR-4055, Dept. of CS, University of Maryland, College Park, MD 20752.

Dix, J., H. Munoz-Avila, D. Nau, and Y. Cao (2000, January). IMPACTing Shop: Foundations for Integrating HTN Planning and Multi Agency. Technical Report CS-TR-4100, Dept. of CS, University of Maryland, College Park, MD 20752. To appear at the CLIMA '2000 Workshop within CL 2000, London, July 2000.

Dix, J., M. Nanni, and V. S. Subrahmanian (2000). Probabilistic agent reasoning. *Transactions of Computational Logic 1*(2), 207–245.

Dix, J., V. S. Subrahmanian, and G. Pick (2000). Meta Agent Programs. *Journal of Logic Programming 46*(1-2), 1–60.

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

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

Eiter, T. and V. S. Subrahmanian (1999). Heterogeneous Active Agents, II: Algorithms and Complexity. *Artificial Intelligence 108*(1-2), 257–307.

Fiadeiro, J. and T. Maibaum (1991). Temporal Reasoning over Deontic

Specifications. *Journal of Logic and Computation 1*(3), 357–395.

Fisher, M. (1994, July). A survey of Concurrent MetateM — the language and its applications. In D. M. Gabbay and H. J. Ohlbach (Eds.), *Temporal Logic — Proceedings of the First International Conference*. Springer-Verlag.

Fisher, M. (1995). Towards a semantics for concurrent metatem. In M. Fisher and R. Owens (Eds.), *Executable Modal and Temporal Logics*. Springer Verlag LNAI Vol. 897.

Fisher, M. and C. Ghidini (1999). Programming resource-bounded deliberative agents. In *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence*, pp. 200–205. Morgan Kaufmann.

Fisher, M. and R. Owens (Eds.) (1995). *Executable Modal and Temporal Logics*. Springer Verlag LNAI Vol. 897.

Frühwirth, T. (1995). Temporal Logic and Annotated Constraint Logic Programming. In M. Fisher and R. Owens (Eds.), *Executable Modal and Temporal Logics*, Volume 897 of *Lecture Notes in Artificial Intelligence*, pp. 58–68. Springer-Verlag.

Gottlob, G., G. Moerkotte, and V. Subrahmanian (1996, March). The park semantics for active rules. In *Proc. 1996 Intl. Conf. on Extending Database Technology*, Avignon, France. Springer-Verlag.

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

Halpern, J. and Y. Shoham (1991). A propositional modal interval logic. *Journal of the ACM 38*(4), 935–962.

Hindriks, K. V., F. S. de Boer, W. van der Hoek, and J. J. C. Meyer (1997). Formal Semantics of an Abstract Agent Programming Language. In *International Workshop on Agent Theories, Architectures, and Languages*, Providence, RI, pp. 204–218.

Horty, J. F. (1996). Agency and obligation. *Synthese 108*, 269–307.

Huhns, M. and M. Singh (1997). *Readings in Agents*. Morgan Kaufmann.

Koehler, J. and R. Treinen (1995). Constraint Deduction in an Interval-based Temporal Logic. In M. Fisher and R. Owens (Eds.), *Executable Modal and Temporal Logics*, Volume 897 of *Lecture Notes in Artificial Intelligence*, pp. 103–117. Springer-Verlag.

Lamport, L. (1994, May). The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems 16*(3), 872–923.

Lee, H., J. Tannock, and J. S. Williams (1993). Logic-based reasoning about actions and plans in artificial intelligence. *The Knowledge Engineering Review 11*(2), 91–105.

Lloyd, J. W. (1987). *Foundations of Logic Programming*. Berlin: Springer. 2nd edition.

Manna, Z. and A. Pnueli (1992). *Temporal Logic of Reactive and Concurrent Systems*. Addison Wesley.

McDermott, D. (1982, December). A temporal logic for reasoning about processes and plans. *Cognitive Science 6*, 101–155.

Meyer, J.-J. C., W. van der Hoek, and B. van Linder (1999). A logical approach to the dynamics of commitments. *Artificial Intelligence journal 113*, 1–40.

Meyer, J.-J. C. and R. Wieringa (Eds.) (1993). *Deontic Logic in Computer Science*. Chichester: John Wiley & Sons.

Moore, R. C. (1985). Semantical considerations on non-monotonic logic. *Artificial Intelligence 25*, 75–94.

Morgenstern, L. (1988). *Foundations of a Logic of Knowledge, Action, and Communication*. Ph. D. thesis, New York University.

Nau, D., Y. Cao, A. Lotem, and H. Muñoz-Avila (1999). Shop: Simple hierarchical ordered planner. In *Proceedings of IJCAI-99*.

Nirkhe, M., S. Kraus, D. Perlis, and M. Miller (1997). How to (plan to) meet a deadline between Now and Then. *Journal of Logic and Computation 7*(1), 109–156.

Rabinovich, A. (1998). Expressive Completeness of Temporal Logic of Action. *Lecture Notes in Computer Science 1450*, 229–242.

Rao, A. S. (1995). Decision Procedures for Propositional Linear-Time Belief-Desire-Intention Logics. In M. Wooldridge, J. Müller, and M. Tambe (Eds.), *Intelligent Agents II – Proceedings of the 1995 Workshop on Agent Theories, Architectures and Languages (ATAL-95)*, Volume 890 of *LNAI*, pp. 1–39. Berlin, Germany: Springer-Verlag.

Rao, A. S. and M. Georgeff (1991). Modeling Rational Agents within a BDI-Architecture. In J. F. Allen, R. Fikes, and E. Sandewall (Eds.), *Knowledge Representation and Reasoning*, Cambridge, MA, pp. 473–484. Morgan Kaufmann.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence 13*, 81–132.

Rosenschein, J. S. and G. Zlotkin (1994). *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. Boston: MIT Press.

Sen, S., T. Haynes, and N. Arora (1997). Satisfying user preferences while negotiating meetings. *Int. Journal on Human-Computer Studies 47*(3), 407–27.

Shoham, Y. (1993). Agent Oriented Programming. *Artificial Intelligence 60*, 51–92.

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

Singh, M. P. (1998). Toward a model theory of actions: How Agents do it in branching time. *Computational Intelligence 14*(3), 287–305.

Sonenberg, E., G. Tidhar, E. Werner, D. Kinny, M. Ljungberg, and A. Rao (1992). Planned Team Activity. Technical Report 26, Australian Artificial Intelligence Institute, Australia.

Subrahmanian, V. (1998). *Principles of Multimedia Database Systems*. Morgan Kaufmann.

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

Sullivan, D. G., A. Glass, B. J. Grosz, and S. Kraus (1999). Intention reconciliation in the context of teamwork: an initial empirical investigation. In M. Klusch, O. Shehory, and G. Weiss (Eds.), *Cooperative Information Agents III*, Lecture Notes in Artificial Intelligence, Vol. 1652, pp. 138–151. Berlin et al.: Springer-Verlag.

Sycara, J. and D. Zeng (1996). Coordination of multiple intelligent software agents. *International Journal of Intelligent and Cooperative Information Systems 5*, 181–211.

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

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

U. Dayal and E. Hanson and J. Widom (1995). Active database systems. In W. Kim (Ed.), *Modern Database Systems*, pp. 434–456. Addison Wesley.

Vila, L. (1994, March). A Survey on Temporal Reasoning in Artificial Intelligence. *AI Communications 7*(1), 4–28.

Yoon, K. and C. Hwang (1995). *Multiple attribute decision making: an introduction*. Thousand Oaks: Sage.

Zaniolo, C., S. Ceri, C. Faloutsos, R. Snodgrass, V. Subrahmanian, and R. Zicari (1997). *Advanced Database Systems*. Morgan Kaufmann.