Temporal Agent Programs

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

[a] *University of Koblenz, Dept. of Computer Science, D-56075 Koblenz, Germany*

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

[c] *Dept. of CS, 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 classical negotiation methods (as well as some new ones) and classical auction methods (as well as some new ones).

Key words: Temporal Reasoning, Logic Programming, Multi-Agent Reasoning

---

[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

Over the last few years, there has been intense work in the area of intelligent agents (Huhns and Singh 1997; Wooldridge and Jennings 1995). Applications of agents range from intelligent news and mail filtering agents (Maes 1994), agents that monitor the state of the stock market and detect trends in stock prices, intelligent web search agents (Etzioni and Weld 1994), digital battlefield agents that monitor and merge information gathered from multiple heterogeneous information sources (Arens, Chee, Hsu, and Knoblock 1993; Labrou and Finin 1994; Labrou and Finin 1997; Subrahmanian 1994; Wiederhold 1993). More recently, we have seen an increase in the number of agents that automatically interact with one another. Such agents can negotiate with each other, participate in auctions, make group consensus decisions, and the like (Kraus 1997; Rosenschein and Zlotkin 1994; Sullivan, Glass, Grosz, and Kraus 1999).

In previous work (Arisha, Ozcan, Ross, Subrahmanian, Eiter, and Kraus 1999; Eiter, Subrahmanian, and Pick 1999; Eiter and Subrahmanian 1999), we have developed a framework (called *IMPACT*) for building agents on top of specialized data structures and/or legacy code bases. Each such agent has a "state" and provides a set of services to other agents. Such services include data retrieval services (answering database queries, retrievals from geographic information systems, etc.) as well as computational services (e.g. creating a plan, recognizing features in imagery, finding a route, etc.). The developer of an *IMPACT* agent can augment a body of software code in any ordinary programming language with a set of rules called an *agent program*. These rules encode the "operating principles" of the agent. In (Eiter, Subrahmanian, and Pick 1999; Eiter and Subrahmanian 1999), the semantics of an agent is characterized via a basic construct called a *feasible status set* which specifies what the agent is permitted to do, forbidden from doing, obligated to do, and actually does in a manner that complies with its operating principles. Eiter, Subrahmanian, and Pick (1999, Eiter and Subrahmanian (1999) propose various more refined semantics by selecting certain feasible status sets—different semantics Sem choose different feasible status sets.

IMPACT agents are engaged in a continuous cycle of *"evaluate state changes* $\rightarrow$ *compute a* Sem*-status set* $\rightarrow$ *take actions* $\rightarrow$ *evaluate state changes* $\rightarrow$ *. . . ."* When an agent receives a message, its state changes. The agent computes an appropriate Sem-status set, finds out what actions are to be performed, and executes them immediately. This framework therefore has two major drawbacks:

- An agent always takes actions *now*. It cannot decide (now) that it is obligated to do something tomorrow, and forbidden from doing something else day after tomorrow.

- All actions are assumed to take zero time to perform. However, in the real world, agents may take actions that have a temporal duration—for example, actions such as $drive(boston, ny)$ have a temporal extent during which the agent's location (hence its state) is changing.

The primary aim of this paper is to endow agents with the ability to execute actions which have a temporal duration, and to make commitments for the future. The paper's contributions are organized as follows:

(1) Section 2 contains a brief motivating example that will be expanded on as the paper proceeds.
(2) Section 3 overviews the framework of (Eiter, Subrahmanian, and Pick 1999) and explains how legacy and specialized software code may be "agentized" via agent programs and other related structures.
(3) Section 4 shows how to specify actions with *temporal duration*.
(4) Section 5 introduces the syntax of a *temporal agent program* (`tap` for short).
(5) Section 6 develops two semantics for `tap`s which extend the semantics introduced for non-temporal agents in (Eiter, Subrahmanian, and Pick 1999).
(6) Section 7 describes a compact representation of the semantic structures associated with `tap`s.
(7) Section 8 presents an algorithm that computes the (compact representation of a) class of `tap`s called *positive* `tap`s.
(8) Finally, in Section 9, we present three applications of `tap`s—the first application is about individual agents collaboratively working together to satisfy a shared goal. The second implementation shows that auction mechanisms can be logically modeled via `tap`s and in fact that certain useful auction mechanisms that have not been studied before are supported by `tap`s. As a third application, we show that `tap`s may be used to support well known (as well as some new) forms of negotiation within the contract net (Smith and Davis 1983) paradigm.
(9) Section 10 compares and contrasts our work with existing research by others.

2 Motivating Example

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 the following agents:

(1) A `heli`copter 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.

We will periodically revisit this simple scenario to illustrate the basic definitions used in this paper.

3 Preliminaries

In *IMPACT*, each agent $\mathfrak{a}$ 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). In general, we will assume that the piece of software $\mathcal{S}^{\mathfrak{a}}$ associated with an agent $\mathfrak{a} \in \mathsf{A}$ is represented by a triple $\mathcal{S}^{\mathfrak{a}} =_{def} (\mathcal{T}_{\mathcal{S}}^{\mathfrak{a}}, \mathcal{F}_{\mathcal{S}}^{\mathfrak{a}}, \mathcal{C}_{\mathcal{S}}^{\mathfrak{a}})$:

Definition 3.1 (Software Code) *We may characterize the code on top of which an agent $\mathfrak{a}$ is built as a triple $\mathcal{S}^{\mathfrak{a}} =_{def} (\mathcal{T}_{\mathcal{S}}^{\mathfrak{a}}, \mathcal{F}_{\mathcal{S}}^{\mathfrak{a}}, \mathcal{C}_{\mathcal{S}}^{\mathfrak{a}})$ where:*

(1) $\mathcal{T}_{\mathcal{S}}^{\mathfrak{a}}$ is the set of all data types managed by $\mathcal{S}$,

(2) $\mathcal{F}_{\mathcal{S}}^{\mathfrak{a}}$ is a set of predefined functions over $\mathcal{T}_{\mathcal{S}}^{\mathfrak{a}}$—these functions typically are those through which external processes may access the data objects managed by the agent, and

(3) $\mathcal{C}_{\mathcal{S}}^{\mathfrak{a}}$ 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 $c(\tau_1, \ldots, \tau_n)$. As c is a partial function, c may only be defined for certain arguments $\tau_1, \ldots, \tau_n$, i.e., c is not necessarily applicable on arbitrary types.

When $\mathfrak{a}$ is clear from context, we will often drop the superscript $\mathfrak{a}$. Intuitively, $\mathcal{T}_{\mathcal{S}}$ is the set of all data types managed by $\mathfrak{a}$, $\mathcal{F}_{\mathcal{S}}$ is the set of all function calls supported by $\mathcal{S}$'s application programmer interface (*API*). $\mathcal{C}_{\mathcal{S}}$ is the set of ways of creating new data types from existing data types. 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. Appendix B.1 describes the details of the message box.

Example 3.2 (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:
  - $\mathtt{heli}:speed()$: which specifies the current speed of the helicopter.
  - $\mathtt{heli}:location()$: which specifies an (x, y, z) coordinate for the helicopter.
  - $\mathtt{heli}:inventory(\mathtt{Item})$: returns a pair of the form $\langle Qty, Unit\rangle$. For instance, $\mathtt{heli}:inventory(\mathtt{blood})$ may return the pair $\langle 25, liters\rangle$ specifying that the helicopter currently has 25 units of blood available.
  - $\mathtt{heli}:bearing()$: returns an angular bearing of the helicopter.
  - $\mathtt{heli}:nextdest()$: specifies the next destination (i.e. a string) of the helicopter.

Definition 3.3 (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 $\mathcal{T}^{\mathfrak{a}}_{\mathcal{S}}$) 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 3.4 (Rescue: State) For instance, returning to the $\mathtt{heli}$ agent, at a given instant of time, the state of the agent may be:

- $speed = 50$ (mph).
- $location = \langle 45, 50, 9000\rangle$.
- *inventory* contains the following four tuples:
  $\langle fuel, 125, gallons\rangle$, $\langle blood, 25, litres\rangle$, $\langle bandages, 50, -\rangle$, $\langle cotton, 20, lbs\rangle$.
- $bearing = 56$.
- $nextdest =$ "big-rag".

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

It is important to note that for each data type τ managed by an agent, we assume the existence of a set of variable symbols ranging over objects of type τ. When τ is a complex type (e.g. a record), we assume the use of variables ranging over τ's components. Thus, if $\mathtt{X}$ is a variable over type τ and τ is a record structure with field f, then $\mathtt{X}.f$ is a variable ranging over objects of the type of field f. In this case, $\mathtt{X}$ is called a *root* variable. This "intuitive" definition of a root variable suffices for this paper—for a formal definition, the reader is referred to (Eiter, Subrahmanian, and Rogers 1999).

Definition 3.6 (Code Call Condition) *A* code call condition χ *is defined as follows:*

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

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

Example 3.7 (Rescue: Code Call Conditions) Returning to the Rescue example, we have the following simple code call conditions:

- $\mathbf{in}(\mathtt{X}, \mathtt{heli}: \mathit{inventory}(\mathtt{fuel})) \,\&\, \mathtt{X.Qty} < 50.$
  This code call condition is satisfied whenever the helicopter has less than 50 gallons of fuel left.
- $\mathbf{in}(\mathtt{X}, \mathtt{heli}: \mathit{inventory}(\mathtt{bandages})) \,\&\, \mathtt{X.Qty} < 50 \,\&$
  $\mathbf{in}(\mathtt{Y}, \mathtt{heli}: \mathit{inventory}(\mathtt{cotton})) \,\&\, \mathtt{Y.Qty} < 100 \,\&\, \mathtt{Y.Unit} = \mathit{lbs}.$
  This code call condition is satisfied whenever the helicopter has less than 50 bandages and less than 100 pounds of cotton.

Each agent has an action-base describing various actions that the agent is capable of executing. Actions change the state of the agent and perhaps the state of other agents' `msgboxes`. As usual actions have preconditions (code call condition) and add/delete lists (sets of ground code call atoms).

Each agent has an associated set of **integrity constraints**—only states that satisfy these constraints are considered to be *valid* or *legal* states. Each agent also has an associated notion of **concurrency** specifying how to combine a set of actions into a single action—this is a function that maps a set of actions and an agent state into one action. (Eiter, Subrahmanian, and Pick 1999) describes three alternative notions of concurrency. Each agent has an associated set of **action constraints** that define the circumstances under which certain actions may be concurrently executed. As at any given point

t in time, many sets of actions may be concurrently executable, each agent has an *Agent Program* that determines what actions the agent can take, what actions the agent cannot take, and what actions the agent must take. Agent programs are defined in terms of status atoms defined below.

Definition 3.8 (Status Atom/Status Set) *If $\alpha(\vec{t})$ is an action, and $\mathsf{Op} \in \{\mathbf{P}, \mathbf{F}, \mathbf{W}, \mathbf{Do}, \mathbf{O}\}$, then $\mathsf{Op}\alpha(\vec{t})$ is called a* status atom*. If A is an action 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 3.9 (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 and $L_1, \ldots, L_n$ are status literals.

Several alternative semantics for agent programs are presented in (Eiter, Subrahmanian, and Pick 1999; Eiter and Subrahmanian 1999)—due to space reasons, we do not explicitly recapitulate them here, though Appendix A contains a brief overview of the semantics. Appendix B contains a description of the agents (code calls, actions, integrity and action constraints, etc.) in the Rescue Example.

4 Actions with Temporal Duration

In classical AI (Nilsson 1986), an action contains a precondition, an add-list, and a delete-list. The idea is that for the action to be executable in a given state, the precondition should be true in the state, and the new state that results after executing the action differs from the previous state in that it no longer satisfies the code call atoms in the delete list, but satisfies the atoms in the add list. This is also the notion of action used in works on reasoning with actions by Baral (Baral and Gelfond 1993; Baral, Gelfond, and Provetti 1995; Baral and Gelfond 1994; Baral and Lobo 1996), Baldoni (Baldoni, Giordano, Martelli, and Patti 1998), and Gelfond and Lifschitz (Gelfond and Lifschitz 1993; Gelfond and Lifschitz 1998; Lifschitz 1997).

We would like to extend this general definition, to allow an action to have a *duration* or *temporal extent*. For example, consider the `heli` agent in our Rescue

Example. Here, the agent may execute the action fly("BigRag", "StonyPoint"). It is immediately apparent that this action is one that has a temporal duration—going from *Big Rag* to *Stony Point* may take some time, during which the location of the `heli` agent 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 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. Most traditional AI planning frameworks (Nilsson 1980) assume that an action's effects are realized only *after* the entire action is successfully executed.

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.

4.1 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. *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.

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. If he needs to incorporate intermediate effects on a millisecond by millisecond basis, his checkpoints should be spaced out at each millisecond (assuming the time unit is not larger than a millisecond). If on the other hand, the designer of the `gv1` agent feels that checkpoints are needed on an hourly basis (assuming the time unit of the time line is not larger than an hour), then he has implicitly decided that incorporating the effects of the *drive* action on an hourly basis is good enough for his (assuming the time unit of the time line is not larger than an hour) agent. We are now ready to define checkpoint expressions.

Definition 4.1 (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.

We will use the symbol **cpe** *as a metavariable for relative and absolute 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. If χ is a code call condition involving a non-negative, integer-valued variable $\mathtt{X}$, then $\mathsf{rel}:\{\mathtt{X} \mid \chi\}$ says that for every possible value i of $\mathtt{X}$ that makes χ true in the current object 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 time point which is a value $\mathtt{X}$ that makes $\mathtt{X} \mid \chi$ true in the current object state.

The following example presents some simple checkpoint expressions.

Example 4.2 (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.
- $\mathsf{abs}:\{\mathtt{T} \mid \mathbf{in}(\mathtt{T}, \mathtt{clock}: \mathit{time}())\ \&$
  $\mathbf{in}(\mathtt{0}, \mathtt{math}: \mathit{remainder}(\mathtt{T}, \mathtt{100}))\ \&$
  $\mathtt{T} > 5000\}$.
  This says that a checkpoint occurs at absolute times 5000, 5100, 5200, and so on.
- $\mathsf{abs}:\{\mathtt{T} \mid \mathbf{in}(\mathtt{T}, \mathtt{clock}: \mathit{time}())\ \&$
  $\mathbf{in}(\mathtt{X}, \mathit{getMessage}(\mathtt{comc}))\ \&$
  $\mathtt{X.Time} - \mathtt{T} = 5\}$.
  This says that a checkpoint occurs at 5 time units after a message is received from the $\mathtt{comc}$ agent.

4.2 Timed Actions

Checkpoint expressions provide a convenient way of specifying a set of time points. In this section, we will show how definitions of actions in classical AI may be extended with checkpoint expressions so as to syntactically extend actions to have duration and intermediate effects over time (i.e. while they are being executed). Our first definition is the concept of a timed effect triple.

Definition 4.3 (Timed Effect Triple $\langle \mathbf{cpe}, Add, Del\rangle$) *A* timed effect triple *is a triple of the form $\langle \mathbf{cpe}, Add, Del\rangle$ where* **cpe** *is a checkpoint expression, and Add and Del are add lists and delete lists.*

Intuitively, when we associate a triple of the form $\langle \mathbf{cpe}, Add, Del\rangle$ with an action α, we are saying that 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 4.4 (Rescue: Timed Effect Triples)

- The heli agent may use the following timed effect triple to update its location every 30 seconds;

$$\begin{aligned}
\text{1st arg} &: \ \mathsf{rel}\!:\!\{30\}\\
\text{2nd arg} &: \{\mathbf{in}(\mathtt{NewLocation}, \mathtt{heli}\!:\!\mathit{location}(\mathtt{X}_{\mathsf{now}}))\}\\
\text{3rd arg} &: \{\mathbf{in}(\mathtt{OldLocation}, \mathtt{heli}\!:\!\mathit{location}(\mathtt{X}_{\mathsf{now}} - 30))\}
\end{aligned}$$

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

$$\text{1st arg}:$$

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

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

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

We are now ready to define the concept of a timed action—an action whose effects are incorporated into a state at the checkpoints specified by the designer of the agent.

Definition 4.5 (Timed Action) *A* timed action α *consists of five components:*

Name: *A name, usually written $\alpha(X_1, \ldots, X_n)$, where the 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 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 $Pre(\alpha)$* [4]

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
  · 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:

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

[4] As in (Eiter, Subrahmanian, and Rogers 1999), 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 1999) and is not required for this paper.

- $\{t^{\alpha}_{\text{start}} + i \times \mathtt{X}\theta \mid \langle \mathsf{rel} : \{\mathtt{X} \mid \chi\}, Add, Del\rangle \in \mathbf{Tet}(\alpha)$ and θ is a solution of χ and $i \in \mathbb{N}, i > 0$ 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 4.6 (Rescue: Timed Actions) Returning to the Rescue example, we have the following timed actions.

The action *drive*() of the `truck` agent may be described via the following components;

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

$$\begin{aligned}
&\text{1st arg} : \mathsf{rel} : \{20\} \\
&\text{2nd arg} : \{\mathbf{in}(\mathtt{NewPosition}, \mathtt{truck} : \mathit{location}(\mathtt{X}_{\mathsf{now}}))\} \\
&\text{3rd arg} : \{\mathbf{in}(\mathtt{OldPosition}, \mathtt{truck} : \mathit{location}(\mathtt{X}_{\mathsf{now}} - 20))\}
\end{aligned}$$

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.

The action *load_truck*() of the `truck` agent may be described via the following components:

- **Name:** $\mathit{load\_truck}(\mathtt{Loc})$
- **Schema:** (`String`)
- **Pre:** $\mathbf{in}(\mathtt{Loc}, \mathtt{truck} : \mathit{location}())$
- **Dur:** $\{10\}$
- **Tet:**

$$\begin{aligned}
&\text{1st arg} : \mathsf{rel} : \{5\} \\
&\text{2nd arg} : \{\mathbf{in}(\mathtt{NewStatus}, \mathtt{truck} : \mathit{load}(\mathtt{X}_{\mathsf{now}}))\} \\
&\text{3rd arg} : \{\mathbf{in}(\mathtt{OldStatus}, \mathtt{truck} : \mathit{load}(\mathtt{X}_{\mathsf{now}} - 5))\}
\end{aligned}$$

The **Tet** part says that the `truck` agent updates its load status every 5 minutes during the expected time it takes it to load the truck.

The action *fill_fuel*() of the $\mathtt{truck}$ agent may be described via the following components:

- **Name:** *fill_fuel*()
- **Pre:**
- **Dur:** $\{1\}$
- **Tet:**

$$
\begin{aligned}
\text{1st arg} &: \ \mathsf{rel}:\{1\}\\
\text{2nd arg} &: \{\mathbf{in}(\mathbf{false}, \mathtt{truck}: \mathit{tank\_empty}())\}\\
\text{3rd arg} &: \{\mathbf{in}(\mathbf{true}, \mathtt{truck}: \mathit{tank\_empty}())\}
\end{aligned}
$$

The **Tet** part says that after filling the tank, the agent updates its state by indicating that the tank is not empty anymore.

The action *monitorHeli*($\mathtt{Heli}$) of the $\mathtt{comc}$ agent may be described as follows;

- **Name:** *monitorHeli*($\mathtt{Heli}$)
- **Schema:** ($\mathtt{String}$)
- **Pre:** $\mathbf{in}(\mathtt{danger}, \mathtt{comc}: \mathit{checkStatus}(\mathtt{Heli}))$
- **Dur:** $\{30\}$
- **Tet:**

$$
\begin{aligned}
\text{1st arg}: \ \mathtt{abs}:\{ \ & \mathtt{X}_{\mathsf{now}} \mid \mathbf{in}(\mathtt{X}_{\mathsf{now}}, \mathtt{clock}: \mathit{time}())\ \&\\
& \mathbf{in}(\mathtt{danger}, \mathtt{comc}: \mathit{checkStatus}(\mathtt{Heli}))\&\\
& \mathbf{in}(\mathtt{0}, \mathtt{math}: \mathit{remainder}(\mathtt{X}_{\mathsf{now}}, \mathtt{2}))\\
\}\ &\\
\text{2nd arg}: \qquad \{ \ & \mathbf{in}(\langle \mathtt{Heli}, \mathtt{danger}, \mathtt{X}_{\mathsf{now}}\rangle, \mathtt{comc}: \mathit{vehicle\_to\_be\_notified}())\&\\
& \mathbf{in}(\mathtt{X}_{\mathsf{now}}, \mathtt{clock}: \mathit{time}())\\
\}\ &\\
\text{3rd arg}: \qquad \{\}\ &
\end{aligned}
$$

Whenever there is a helicopter which is in danger, the $\mathtt{comc}$ agent executes the *monitorHeli* timed action. The *monitorHeli* action is executed for 30 minutes, and a notification is sent to the helicopter every 2 minutes.

5 Syntax of taps

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 6. The first concept we define is that of a temporal annotation item.

Definition 5.1 (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. A temporal annotation, defined below, uses temporal annotation items to specify a time interval.

Definition 5.2 (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.*

Examples of temporal annotations are given below.

Example 5.3 (Rescue: Temporal Annotations)

- $[2, 5]$ 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.
- $[\mathtt{X} + \mathtt{Y}, \mathtt{X} + 4\mathtt{y}]$ is a temporal annotation item. When $\mathtt{X} := 2, \mathtt{Y} := 3$, this defines the set of time points between 5 and 18.
- $[\mathtt{X} + 4\mathtt{Y}, \mathtt{X} - \mathtt{Y}]$ is a temporal annotation item. When $\mathtt{X} := 2, \mathtt{Y} := 3$, this defines the empty set of time points.
- $[\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}+5]$ is a temporal annotation item. When $\mathtt{X}_{\mathsf{now}} := 10$, this specifies the set of time points between 10 and 15.
- $[\mathtt{X}_{\mathsf{now}} - 5, \mathtt{X}_{\mathsf{now}} + 5]$ is a temporal annotation item. When $\mathtt{X}_{\mathsf{now}} := 10$, this specifies the set of time points between 5 and 15; when $\mathtt{X}_{\mathsf{now}} := 3$ this specifies the set of time points between 0 and 8.

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

(1) $(\chi \,\&\, L_1 \,\&\, \dots \,\&\, L_n)$ *is called an* action state condition.
(2) $(\chi \,\&\, L_1 \,\&\, \dots \,\&\, L_n) : \mathsf{ta}$ *is called a* temporal action state conjunct *(tasc).*
(3) If χ is empty, then $(\chi \,\&\, L_1 \,\&\, \dots \,\&\, L_n) : \mathsf{ta}$ *is called a* state-independent 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 $\varrho : \mathsf{ta}$ is ground for some action state condition ϱ, we may read this as "ϱ is true at some point in ta". For example, the following are simple tascs.

- $\mathbf{in}(\mathtt{danger}, \mathtt{comc} : \mathit{checkStatus}(\mathtt{heli})) : [6, 10]$
  This tasc is true if if at some point between times 6 and 10 (inclusive) the heli agent was in danger according to the comc agent.
- $(\mathbf{in}(\mathtt{X}, \mathtt{heli} : \mathit{inventory}(\mathtt{fuel})) \,\&\, \mathtt{X.Qty} < 50) : [\mathtt{X}_{\mathsf{now}} - 10, \mathtt{X}_{\mathsf{now}}]$ 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, viz. a temporal agent rule.

Definition 5.5 (Temporal Agent Rule/Program $\mathcal{TP}$) *A* temporal agent rule *is an expression of the form:*

$$\mathit{Op}\,\alpha : [\mathit{tai}_1, \mathit{tai}_2] \leftarrow \varrho_1 : \mathsf{ta}_1 \,\&\, \cdots \,\&\, \varrho_n : \mathsf{ta}_n \tag{1}$$

where $\mathit{Op} \in \{\mathbf{P}, \mathbf{Do}, \mathbf{F}, \mathbf{O}, \mathbf{W}\}$, and $\varrho_1 : \mathsf{ta}_1, \dots, \varrho_n : \mathsf{ta}_n$ are tascs.

A temporal agent program *is a finite set of temporal agent rules.*

Intuitively, a ground temporal agent rule of the form shown in (1) above may be read as:

Intuitive Reading of Temporal Agent Rule (First Cut)
"If ϱ_1 was true at some time point in the interval ta_1 and ... and ϱ_n was true at some time point in the interval ta_n, then $\mathit{Op}\,\alpha$ should be true at some time point in the interval $[\mathit{tai}_1, \mathit{tai}_2]$".

Example 5.6 (Rescue: Temporal Agent Rules) The following rules may be used by agents in the rescue example:

r1: $\mathbf{O}\mathit{delete\_msg}(Msg.Id) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow$
$\quad\mathbf{Do}\,\mathit{process\_request}(\mathtt{Msg.Id}, \mathtt{Agent}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$.

This rule says that the agent immediately deletes all messages that it has processed from its message box.

r2 (Agent heli):
$\mathbf{O}\mathit{order\_item}(\mathtt{Item}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}} + 5] \leftarrow$
$\quad(\,\mathbf{in}(\mathtt{Item}, \mathtt{heli} : \mathit{emergency\_items}())\ \&$
$\quad\ \mathbf{in}(\mathtt{X}, \mathtt{heli} : \mathit{inventory}(\mathtt{Item}))\ \&$
$\quad\ \mathbf{in}(\mathtt{Minimal\_amount}, \mathtt{heli} : \mathit{minimal\_inventory}(\mathtt{Item}))\ \&$
$\quad\ \mathtt{X.Qty} \leq \mathtt{Minimal\_amount}$
$\quad) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$

If the helicopter's supply of an item which is needed in an emergency is below the minimal required quantity, it is obliged to order this item within 5 time units.

r3 (Agent comc):
$\mathbf{O}\mathit{notifyAgent}(\mathtt{Agnt}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow$
$\quad\mathbf{in}(\mathtt{danger}, \mathtt{comc} : \mathit{checkStatus}(\mathtt{Agnt})) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$

The comc agent is obliged to notify an agent immediately if it believes the agent is in danger.

r4 (Agent truck):
$\mathbf{F}\,\mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow$
$\quad\mathbf{Do}\,\mathit{process\_request}(\mathtt{Msg.Id}, \mathtt{Agent}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]\ \&$
$\quad=(\mathtt{Msg.body.call}, \texttt{"}\mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway})\texttt{"}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]\ \&$
$\quad\mathbf{in}(\mathtt{Highway}, \mathtt{msgbox} : \mathit{gatherWarning}(\mathtt{comc})) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}} - 10]$

If, in the last 10 time units, the truck agent received a warning message from the comc agent concerning a given highway, it is forbidden to drive through this highway.

The intuitive "First Cut" reading of the rules is not quite correct because it is possible for the body of a temporal agent rule to become true *now*, even though it was not true before. Thus, we may be obliged (now) by such a rule to do something in the past—something that is clearly infeasible. The following example rule says that a truck agent is obliged to fill its tank before driving more than 30km.

$$\begin{aligned}
&\mathbf{O}\mathit{fill_fuel}() : [\mathtt{X}_{\mathsf{now}} - 1, \mathtt{X}_{\mathsf{now}} - 1] \leftarrow \\
&\quad (\, \mathbf{Do}\ \mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway})\ \& \\
&\quad\ \ \mathbf{in}(\mathtt{X}, \mathtt{math} : \mathit{distance}(\mathtt{From}, \mathtt{To}))\ \& \\
&\quad\ \ \mathtt{X} > 30 \\
&\quad) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]
\end{aligned}$$

This rule may not have fired "yesterday" as the action status atom

$$\mathbf{Do}\ \mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway})$$

may not have been true yesterday, yet if it becomes true today, it imposes (today) an obligation on us to have done something yesterday which clearly makes no sense !

An alternative reading of the temporal agent rule $\mathsf{Op}\,\alpha : [\mathsf{tai}_1, \mathsf{tai}_2] \leftarrow \varrho_1 : \mathsf{ta}_1\ \& \ldots \&\ \varrho_n : \mathsf{ta}_n$ is:

Intuitive Reading of Temporal Agent Rule (Second Cut)
"If for all $1 \leq i \leq n$, there exists a time point t_i such that ϱ_i is true at time t_i and such that $t_i \leq \mathtt{t}_{now}$ (i.e. t_i is now or is in the past) and $t_i \in \mathsf{ta}_i$ (i.e. t_i is true at one of designated time points), then $\mathsf{Op}\,\alpha$ is true at some point $t \geq \mathtt{t}_{now}$ (i.e. now or in the future) such that $\mathsf{tai}_1 \leq t \leq \mathsf{tai}_2$."

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!

However, the second reading above still has a flaw. For instance, consider the rule:

$$\mathbf{P}\gamma : [\mathtt{t}_{\mathsf{now}} + 9, \mathtt{t}_{\mathsf{now}} + 9] \leftarrow \mathbf{P}\alpha : [\mathtt{t}_{\mathsf{now}} + 1, \mathtt{t}_{\mathsf{now}} + 10] \wedge \mathbf{P}\beta : [\mathtt{t}_{\mathsf{now}} + 1, \mathtt{t}_{\mathsf{now}} + 8].$$

According to the above reading, even if we know that α is permitted during the interval $[\mathtt{t}_{\mathsf{now}}+1, \mathtt{t}_{\mathsf{now}}+10]$ and β is permitted during the interval $[\mathtt{t}_{\mathsf{now}}+1, \mathtt{t}_{\mathsf{now}}+8]$, this rule cannot be fired because the body of the rule is supposed to be true at some time point in the past ! Thus, we are led to a third reading of a temporal agent rule—this reading distinguishes between state independent and arbitrary tasc's.

Intuitive Reading of Temporal Agent Rule (Third Cut)
"If for all $1 \leq i \leq n$, 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 not state independent and $t_i \leq \mathtt{t}_{\mathit{now}}$ (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 $t \geq \mathtt{t}_{\mathit{now}}$ (i.e. now or in the future) such that $\mathit{tai}_1 \leq t \leq \mathit{tai}_2$".

This reading of a tap rule allows us to avoid the problems encountered earlier.

For example, the truck agent is obliged to fill fuel now if it's tank is empty and it is obliged to drive somewhere in 15 minutes.

$$
\begin{aligned}
&\mathbf{O}\mathit{fill\_fuel}() : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow \\
&\qquad \mathbf{in}(\mathbf{true}, \mathtt{truck} : \mathit{tank\_empty}()) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]\ \& \\
&\qquad \mathbf{O}\mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}} + 15]
\end{aligned}
$$

Since, $\mathbf{O}\mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway})$ is a state-independent tasc the above rule makes sense.

6 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*.

6.1 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 6.1 (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 this means that 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 by an agent.

Definition 6.2 (State History Function $\mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}$) *A* state history function *$\mathsf{hist}_{\mathtt{t}_{now}}$ at time $\mathtt{t}_{now}$ is a partial function from $\mathbb{N}$ to agent states such that $\mathsf{hist}_{\mathtt{t}_{now}}(\mathtt{t}_{now})$ is always defined and for all $i > \mathtt{t}_{now}$, $\mathsf{hist}_{\mathtt{t}_{now}}(i)$ 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, $\mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}(i)$ 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 $\mathtt{t}_{\mathsf{now}}, (\mathtt{t}_{\mathsf{now}} - 1), \ldots,$

$(\mathsf{t_{now}}-i)$, i. e. $\mathsf{hist}_{\mathsf{t_{now}}}$ is defined for the following arguments: $\mathsf{hist}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$, $\mathsf{hist}_{\mathsf{t_{now}}}(\mathsf{t_{now}}-1),\ldots,\ \mathsf{hist}_{\mathsf{t_{now}}}(\mathsf{t_{now}}-i)$ are defined.

(3) He may decide to store, in addition to the current state, the history every five time units. That is, $\mathsf{hist}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$ is defined and for each $0 \le i \le \mathsf{t_{now}}$, if $i \mod 5 = 0$, then $\mathsf{hist}_{\mathsf{t_{now}}}(i)$ 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 the application's needs.

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

Definition 6.3 (Satisfaction, Closure of $\mathcal{TS}_{\mathsf{t_{now}}}$ under tap Rules) *Suppose t_{now} is any integer. We present below, an inductive definition of satisfaction of formulas by $\langle \mathcal{TS}_{\mathsf{t}_{now}}, \mathit{hist}_{\mathsf{t}_{now}}, \mathsf{t}_{now}\rangle$;*

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

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

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

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

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

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

(3) $\langle \mathcal{TS}_{\mathsf{t}_{now}}, \mathit{hist}_{\mathsf{t}_{now}}, \mathsf{t}_{now}\rangle \models^{\mathcal{O}}_{\mathsf{temp}} (\forall x)\phi$ if $\langle \mathcal{TS}_{\mathsf{t}_{now}}, \mathit{hist}_{\mathsf{t}_{now}}, \mathsf{t}_{now}\rangle \models^{\mathcal{O}}_{\mathsf{temp}} \phi[x/s]$ for all ground terms s.[5]

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

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

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

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

The definition of satisfaction by $\langle \mathcal{TS}_{\mathsf{t}_{\mathsf{now}}}, \mathsf{hist}_{\mathsf{t}_{\mathsf{now}}}, \mathsf{t}_{\mathsf{now}} \rangle$ 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 6.4 (Rescue: Temporal Status Set) Suppose we consider the following very simple table, describing a temporal status set, $\mathcal{TS}_{\mathsf{t}_{\mathsf{now}}}$ of the truck agent.

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

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

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

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

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

  This formula is satisfied by $\langle \mathcal{T}S_{\mathtt{t_{now}}}, \mathsf{hist}_{\mathtt{t_{now}}}, \mathtt{t_{now}} \rangle$ because $i = 2$ is a witness to the satisfaction of this formula.
- $($ $\mathbf{in}(2, \mathsf{truck}: \mathit{inventory}(\mathtt{fa\_bag}))$ &
  $\mathbf{in}(\mathbf{false}, \mathsf{truck}: \mathit{tank\_empty}())$ &
  $\mathbf{F}\ \mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw295})$
  $) : [\mathtt{t_{now}} - 3, \mathtt{t_{now}}]$.

  This formula is satisfied by $\langle \mathcal{T}S_{\mathtt{t_{now}}}, \mathsf{hist}_{\mathtt{t_{now}}}, \mathtt{t_{now}} \rangle$ because $i = 2$ is a witness to the satisfaction of this formula. Notice that at time 1, the action status atom $\mathbf{F}\mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw295}) \in \mathcal{T}S_{\mathtt{t_{now}}}(2)$ and the other $\mathbf{in}(,)$ atoms are also true in the agent state at time 2.
- $($ $\mathbf{in}(\mathbf{true}, \mathsf{truck}: \mathit{tank\_empty}())$ &
  $\mathbf{O}\mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw95})$
  $) : [\mathtt{t_{now}} - 3, \mathtt{t_{now}} - 3]$.

  This formula is not satisfied by $\langle \mathcal{T}S_{\mathtt{t_{now}}}, \mathsf{hist}_{\mathtt{t_{now}}}, \mathtt{t_{now}} \rangle$ because the action status atom $\mathbf{O}\mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw95}) \notin \mathcal{T}S_{\mathtt{t_{now}}}(0)$.
- The rule

  $\mathbf{O}\mathit{fill\_fuel}() : [\mathtt{t_{now}}, \mathtt{t_{now}}] \leftarrow$
  $($ $\mathbf{in}(\mathbf{true}, \mathsf{truck}: \mathit{tank\_empty}())$ &
  $\mathbf{O}\ \mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw95})$
  $) : [\mathtt{t_{now}} - 3, \mathtt{t_{now}} - 3]$

  is satisfied by $\langle \mathcal{T}S_{\mathtt{t_{now}}}, \mathsf{hist}_{\mathtt{t_{now}}}, \mathtt{t_{now}} \rangle$ because its antecedent is not satisfied via a witness $i \leq \mathtt{t_{now}}$ by $\langle \mathcal{T}S_{\mathtt{t_{now}}}, \mathsf{hist}_{\mathtt{t_{now}}}, \mathtt{t_{now}} \rangle$.
- The rule

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

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

(1) $\mathbf{F}\mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw295}) : [\mathtt{t_{now}}, \mathtt{t_{now}} + 2] \leftarrow$
$\mathbf{in}(\mathtt{hw295}, \mathsf{msgbox}: \mathit{gatherWarning}(\mathsf{comc})) : [\mathtt{t_{now}} - 3, \mathtt{t_{now}}]$
(2) $\mathbf{Do}\,\mathit{fill\_fuel}() : [\mathtt{t_{now}}, \mathtt{t_{now}}] \leftarrow$
$\mathbf{in}(\mathbf{true}, \mathsf{truck}: \mathit{tank\_empty}()) : [\mathtt{t_{now}} - 2, \mathtt{t_{now}}]$
(3) $\mathbf{O}\mathit{order\_item}(\mathtt{fa\_bag}) : [\mathtt{t_{now}}, \mathtt{t_{now}} + 4] \leftarrow$
$\mathbf{in}(1, \mathsf{truck}: \mathit{inventory}(\mathtt{fa\_bag}))[\mathtt{t_{now}} - 3, \mathtt{t_{now}}]$
(4) $\mathbf{P}\mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw95}) : [\mathtt{t_{now}}, \mathtt{t_{now}}] \leftarrow$
$\mathbf{in}(\mathbf{false}, \mathsf{truck}: \mathit{tank\_empty}()) : [\mathtt{X_{now}}, \mathtt{X_{now}}]$ &

$\mathbf{F}\, drive(\texttt{was}, \texttt{bal}, \texttt{hw295}) : [\mathsf{t_{now}} + 1, \mathsf{t_{now}} + 2]$

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

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

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

Finally, the fourth rule is satisfied by $\langle \mathcal{TS}_{\mathsf{t_{now}}}, \mathsf{hist}_{\mathsf{t_{now}}}, \mathsf{t_{now}} \rangle$ since its antecedent is satisfied and its consequent is satisfied. The first tasc of the antecedent, $\mathbf{in}(\mathbf{false}, \texttt{truck} : tank\_empty()) : [\mathtt{X_{now}}, \mathtt{X_{now}}]$, is satisfied via a witness i=3 $\leq \mathsf{t_{now}}$. The second tasc, $\mathbf{F}\, drive(\texttt{was}, \texttt{bal}, \texttt{hw295}) : [\mathsf{t_{now}}{+}1, \mathsf{t_{now}}{+}2]$, is state independent and is satisfied as $\mathbf{F}\, drive(\texttt{was}, \texttt{bal}, \texttt{hw295}) \in \mathcal{TS}_{\mathsf{t_{now}}}(4)$ The rule's consequent is satisfied as $\mathbf{P}\, drive(\texttt{was}, \texttt{bal}, \texttt{hw95}) \in \mathcal{TS}_{\mathsf{t_{now}}}(3)$.

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 6.5 (Action History) *An* action history $\mathit{acthist}_{\mathsf{t_{now}}}$ *for an agent is a partial function from* $\mathbb{N}$ *to status sets satisfying* $\mathit{acthist}_{\mathsf{t_{now}}}(i) = \emptyset$ *for all* $i > i_0$ *for a* $i_0 \in \mathbb{N}$.

Intuitively, an action history specifies not only what an 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. Here is an example of an action history:

i	$\mathsf{acthist}_2(\mathbf{i})$
0	$\{\mathbf{P}\alpha_1, \mathbf{O}\alpha_2, \mathbf{Do}\,\alpha_2, \mathbf{P}\alpha_2, \mathbf{F}\alpha_3\}$
1	$\{\mathbf{O}\alpha_2, \mathbf{Do}\,\alpha_2, \mathbf{P}\alpha_2\}$
2	$\{\mathbf{P}\alpha_3, \mathbf{Do}\,\alpha_3\}$
3	$\{\mathbf{F}\alpha_3\}$
4	$\{\mathbf{O}\alpha_1, \mathbf{Do}\,\alpha_1\}$
$i \geq 5$	$\{\}$

If the current time is $\mathsf{t_{now}} = 2$, then this says the agent is forbidden from doing α_3 at time 3 and is obliged to do action α_1 at time 4. What this means is that when this agent receives new requests from other agents at time 3, it must consider the fact that it is forbidden from doing α_3 at time 3 and doing α_4 at time 4, i.e. it must find a new temporal status set, based on commitments made in the past (even if these commitments involve the future). The new temporal status set, in turn, may add more commitments or forbidden actions to the future and the $\mathsf{acthist}_{\mathtt{t}_{\mathsf{now}}}$ with respect to $\mathtt{t}_{\mathsf{now}}$ and the future may change.

Intuitively, an action history specifies the *intention* of an agent, as far as its future actions are concerned. For example, according to the above example, at time 3, the agent *intends* to $\mathbf{Do}\,\alpha_3$, even though it is not obliged to do so. An external event at time 3 may well cause it to change its mind.

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 6.6 (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* $\mathit{acthist}_{\mathtt{t}_{now}}(i)$ *is defined, then* $\mathcal{TS}_{\mathtt{t}_{now}}(i) = \mathit{acthist}_{\mathtt{t}_{now}}(i)$*, and for all* $i \geq \mathtt{t}_{now}$*, if* $\mathit{acthist}_{\mathtt{t}_{now}}(i)$ *is defined, then* $\mathit{acthist}_{\mathtt{t}_{now}}(i) \subseteq \mathcal{TS}_{\mathtt{t}_{now}}(i)$*.*

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 6.7 (Rescue: Action History) Consider the following simple action history of the `truck` agent.

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

The temporal status set, $\mathcal{TS}_{\mathtt{t}_{\mathsf{now}}}$ presented in Example 6.4 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 6.8 (Expected States at time $\mathtt{t}$: $\mathcal{EO}(\mathtt{t})$) *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*

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

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).

We demonstrate the computation of the expected states using the following example.

Example 6.9 (Rescue: Expected States) Suppose, $\mathtt{t}_{\mathsf{now}} = 1$,

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

and

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

Then,

$$\mathcal{EO}(2) = \{\mathbf{in}(\mathtt{empty}, \mathtt{truck}: load(\mathtt{0})), \mathbf{in}(\mathtt{was}, \mathtt{truck}: location()),$$
$$\mathbf{in}(\mathbf{false}, \mathtt{truck}: tank\_empty())\}$$

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

$$\mathcal{EO}(6) = \{\mathbf{in}(\mathtt{half\_loaded}, \mathtt{truck}: load(\mathtt{5})), \mathbf{in}(\mathtt{was}, \mathtt{truck}: location()),$$
$$\mathbf{in}(\mathbf{false}, \mathtt{truck}: tank\_empty())\}$$

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

$$\mathcal{EO}(11) = \{\mathbf{in}(\mathtt{loaded}, \mathtt{truck}: load(\mathtt{10})), \mathbf{in}(\mathtt{was}, \mathtt{truck}: location()),$$
$$\mathbf{in}(\mathbf{false}, \mathtt{truck}: tank\_empty()),$$
$$\mathbf{in}(\langle \mathtt{fa\_bag}, \mathtt{10}\rangle, \mathtt{msgbox}: supplier\_to\_be\_notified())\}.$$

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.

6.2 Feasible Temporal Status Sets

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 $\mathtt{t_{now}}$ and suppose $\mathsf{hist}_{\mathtt{t_{now}}}(\cdot)$, $\mathsf{acthist}_{\mathtt{t_{now}}}(\cdot)$ represent the state and action histories associated with this agent at time $\mathtt{t_{now}}$.

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

Definition 6.10 (Temporal Deontic Consistency) *Suppose $\mathsf{hist}_{\mathtt{t}_{now}}$ is the agent's state history function. $\mathcal{TS}_{\mathtt{t}_{now}}$ is said to be* temporally deontically consistent *at time $\mathtt{t}_{now}$ if it satisfies the following conditions:*

- *For all time points i, (1) $\mathbf{O}\alpha \in \mathcal{TS}_{\mathsf{t}_{now}}(i) \rightarrow \mathbf{W}\alpha \notin \mathcal{TS}_{\mathsf{t}_{now}}(i)$; (2) $\mathbf{P}\alpha \in \mathcal{TS}_{\mathsf{t}_{now}}(i) \rightarrow \mathbf{F}\alpha \notin \mathcal{TS}_{\mathsf{t}_{now}}(i)$;*
- *For all $i \leq \mathsf{t}_{now}$, if $\mathbf{P}\alpha \in \mathcal{TS}_{\mathsf{t}_{now}}(i)$ and $\mathsf{hist}_{\mathsf{t}_{now}}(i)$ is defined, then $\mathsf{hist}_{\mathsf{t}_{now}}(i) \models Pre(\alpha)$ and $\mathsf{duration}(\alpha)$ is defined with respect to $\mathsf{hist}_{\mathsf{t}_{now}}(i)$ 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 $\mathcal{TS}_{\mathsf{t}_{\mathsf{now}}}(4) = \{\mathbf{Do}\,\alpha, \mathbf{F}\alpha\}$, then $\mathcal{TS}_{\mathsf{t}_{\mathsf{now}}}$ 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 6.11 (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.*

The definition of action consistency ensures that action constraints are never violated.

Definition 6.12 (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 $\mathsf{acthist}_{\mathsf{t}_{now}}(i)$ and $\mathsf{hist}_{\mathsf{t}_{now}}(i)$ 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 $\mathsf{hist}_{\mathsf{t}_{now}}(i)$.*[6]

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 6.13 (Rescue: Action Consistency) Let the `truck` agent have the following action constraint $AC = \{drive(\mathtt{From}, \mathtt{To}, \mathtt{Highway}), \mathit{fill\_fuel}()\} \hookleftarrow$, 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 6.4 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

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

satisfy the integrity constraints. This requirement, called checkpoint consistency, is defined below.

Definition 6.14 (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 6.15 (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 **Do** actions contained in $\mathcal{TS}_{\mathsf{t}_{now}}(i)$ satisfies the integrity constraints $\mathcal{IC}$.*

Definition 6.16 (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) $\mathcal{TS}_{\mathsf{t}_{now}}$ is temporally deontically and action closed at time t_{now},
(4) $\mathcal{TS}_{\mathsf{t}_{now}}$ is checkpoint consistent at time t_{now},
(5) $\mathcal{TS}_{\mathsf{t}_{now}}$ is state consistent at time t_{now},
(6) $\mathcal{TS}_{\mathsf{t}_{now}}$ is history compatible at time t_{now}.

6.3 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 6.17 (Rational Feasible Temporal Status Set) *A temporal status set $\mathcal{TS}_{\mathsf{t}_{now}}$ is* grounded*, if there is no temporal status set $\mathcal{TS}'_{\mathsf{t}_{now}} \neq \mathcal{TS}_{\mathsf{t}_{now}}$ such that $\mathcal{TS}'_{\mathsf{t}_{now}} \subseteq \mathcal{TS}_{\mathsf{t}_{now}}$ and $\mathcal{TS}'_{\mathsf{t}_{now}}$ satisfies conditions (1)–(6) of a feasible temporal status set.*

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.

Example 6.18 (Rescue: Rational Status Set) Consider the simple example where the `truck` agent has an action constraint as in Example 6.13, i.e.,

$$AC = \{\mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway}), \mathit{fill\_fuel}()\} \hookleftarrow\}$$

and one integrity constraint

$$\begin{aligned}\mathcal{IC} = \{\ & \mathbf{in}(\mathtt{Loc1}, \mathtt{truck} : \mathit{location}())\& \\ & \neq (\mathit{Loc}1, \mathit{Loc}2) \Rightarrow \mathbf{not\_in}(\mathtt{Loc2}, \mathtt{truck} : \mathit{location}())\}\end{aligned}$$

intuitively saying that the truck cannot simultaneously be in two different locations.

The agent's tap includes only one rule specified at the end of Section 5, i.e.,

$$\begin{aligned}& \mathbf{O}\mathit{fill\_fuel}() : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow \\ & \quad \mathbf{in}(\mathbf{true}, \mathtt{truck} : \mathit{tank\_empty}()) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]\ \& \\ & \quad \mathbf{O}\mathit{drive}(\mathtt{From}, \mathtt{To}, \mathtt{Highway}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}} + 15]\end{aligned}$$

Suppose the agent has the following very simple state history ($\mathsf{hist}_{\mathtt{t}_{\mathsf{now}}}$), and action history ($\mathsf{acthist}_{\mathtt{t}_{\mathsf{now}}}$):

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

$$\mathsf{acthist}_{\mathtt{t}_{\mathsf{now}}}(8) = \{\mathbf{O}\mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{hw95})\}$$

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

$$\begin{aligned}\mathcal{TS}^1_{\mathtt{t_{now}}}(1) &= \{\mathbf{O}\mathit{fill\_fuel}(), \mathbf{Do}\,\mathit{fill\_fuel}(), \mathbf{P}\mathit{fill\_fuel}()\}\\ \mathcal{TS}^1_{\mathtt{t_{now}}}(8) &= \{\mathbf{O}\,\mathit{drive}(\mathtt{was},\mathtt{bal},\mathtt{hw95}), \mathbf{P}\,\mathit{drive}(\mathtt{was},\mathtt{bal},\mathtt{hw95}),\\ &\quad \mathbf{Do}\,\mathit{drive}(\mathtt{was},\mathtt{bal},\mathtt{hw95})\}\end{aligned}$$

$$\begin{aligned}\mathcal{TS}^2_{\mathtt{t_{now}}}(1) &= \{\mathbf{O}\mathit{fill\_fuel}(), \mathbf{Do}\,\mathit{fill\_fuel}(), \mathbf{P}\mathit{fill\_fuel}(),\\ &\quad \mathbf{O}\,\mathit{order\_item}(\mathtt{fa\_bag}), \mathbf{Do}\,\mathit{order\_item}(\mathtt{fa\_bag}),\\ &\quad \mathbf{P}\,\mathit{order\_item}(\mathtt{fa\_bag}),\}\\ \mathcal{TS}^2_{\mathtt{t_{now}}}(8) &= \{\mathbf{O}\,\mathit{drive}(\mathtt{was},\mathtt{bal},\mathtt{hw95}), \mathbf{P}\,\mathit{drive}(\mathtt{was},\mathtt{bal},\mathtt{hw95}),\\ &\quad \mathbf{Do}\,\mathit{drive}(\mathtt{was},\mathtt{bal},\mathtt{hw95})\}\end{aligned}$$

$\mathcal{TS}^1_{\mathtt{t_{now}}}$ is temporally deontically consistent because (1) the only action which has a precondition is *drive* and it is easy to see that its precondition

$$\mathbf{in}(\mathtt{was}, \mathtt{truck} : \mathit{location}())$$

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

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

7 Compact Representation of Temporal Status Sets

Representing a feasible temporal status set explicitly is difficult because, for each time point i, $\mathcal{TS}_{\mathtt{t_{now}}}(i)$ 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 7.1 (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 those concepts here.

Definition 7.2 (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 7.3 (Interval Constraint Temporal Status Set ic-$\mathcal{TS}$) *An* interval constraint temporal status set*, denoted ic-$\mathcal{TS}$, is a set of interval constraint annotated status atoms.*

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

Definition 7.4 (Temporal Status Sets Compatible with ic-$\mathcal{TS}$ (I)) *A temporal status set $\mathcal{TS}_{\mathsf{t}_{now}}$ is* compatible with *ic-$\mathcal{TS}$ if for every $\mathsf{Op}\,\alpha : \mathsf{tic}$ in ic-$\mathcal{TS}$, 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 $\mathsf{CompTSS}(\textit{ic-}\mathcal{T}S)$ *to denote the set of all temporal status sets compatible with* $\mathsf{ic}\text{-}\mathcal{T}S$*.*

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

Example 7.5 (Rescue: Compatibility) The status set $\mathcal{T}S_{\mathsf{t_{now}}}$ from Example 6.4 is compatible with the following $\mathsf{ic}\text{-}\mathcal{T}S$:

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

There are an infinite number of temporal status sets $\mathcal{T}S_{\mathsf{t_{now}}}$ that are compatible with this $\mathsf{ic}\text{-}\mathcal{T}S$. For example, the following temporal status set, $\mathcal{T}S'_{\mathsf{t_{now}}}$ is also compatible with it:

$$\begin{aligned}\mathcal{T}S'_{\mathsf{t_{now}}}(0) &= \{\mathbf{F}\mathit{fill\_fuel}()\}\\ \mathcal{T}S'_{\mathsf{t_{now}}}(4) &= \{\mathbf{O}\mathit{fill\_fuel}(), \mathbf{P}\,\mathit{drive}(\mathtt{was},\mathtt{bal},\mathtt{hw95})\}\end{aligned}$$

On the other hand, there are infinitely many interval constraint temporal status sets that are compatible with the $\mathcal{T}S_{\mathsf{t_{now}}}$ from Example 6.4. 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 $\mathsf{ic}\text{-}\mathcal{T}S$, we cannot (in general) infer $\mathsf{Op}\,\alpha : \mathsf{tic}_1 \wedge \mathsf{tic}_2$.

The following section shows how to use interval constraint status sets to compute feasible status sets.

8 Status Set Computation Algorithm for Positive taps

In this section, we provide an iterative fixpoint algorithm to compute a feasible temporal status set when all the rules in our temporal agent program are *negation free*—such taps are called *positive* taps. Suppose the current time, $\mathsf{t_{now}}$ and the histories, $\mathsf{hist}_{\mathsf{t_{now}}}(\cdot)$ and $\mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$ are arbitrary, but fixed. When computing a temporal feasible status set, we need to address the following issues.

(1) History compatibility uniquely specifies $\mathcal{TS}_{\mathsf{t_{now}}}(i)$ for $i < \mathsf{t_{now}}$.
(2) Checkpoint consistency constrains $\mathcal{TS}_{\mathsf{t_{now}}}(i)$ for $i \leq \mathsf{t_{now}}$.
(3) If there exists an $i < \mathsf{t_{now}}$ for which state consistency does not hold, then no feasible temporal status set can exist. But if for all i with $i < \mathsf{t_{now}}$ state consistency holds, then Checkpoint consistency imposes an additional restriction on $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$.
(4) Let us now consider the "closure under program rules" condition for feasibility. As rule bodies are always evaluated either in the *current or the past* (if the tasc is state dependent) every state dependent tasc of the form $\varrho : [\mathsf{tai}_1, \mathsf{tai}_2]$ may implicitly be rewritten as $\varrho : [\mathsf{tai}_1, \min(\mathsf{tai}_2, \mathsf{t_{now}})]$. The state independent tasc's stay unchanged. If $\mathsf{tai}_2 < \mathsf{t_{now}}$, then evaluation of $\varrho : [\mathsf{tai}_1, \mathsf{tai}_2]$ boils down to either (1) (if ϱ is state dependent) checking if ϱ is true at some time point $i < \mathsf{t_{now}}$, or (2) (if ϱ is state independent) checking if ϱ is true at some time point in $[\mathsf{tai}_1, \mathsf{tai}_2]$. Now (2) is a check that does not involve the state, but just the temporal status set $\mathcal{TS}_{\mathsf{t_{now}}}$ being constructed. Thus satisfying the rule adds a constraint on $\mathcal{TS}_{\mathsf{t_{now}}}$ (for those i with $i \geq \mathsf{t_{now}}$, for $i < \mathsf{t_{now}}$ the sets $\mathcal{TS}_{\mathsf{t_{now}}}(i)$ are fixed). In case (1), all that is needed is to evaluate ϱ w.r.t. the state at time i and $\mathcal{TS}_{\mathsf{t_{now}}}(i)$ both of which are *fixed*! If the body is true and the head is of the form $\mathsf{Op}\alpha : \mathsf{ta}$, then $\mathcal{TS}_{\mathsf{t_{now}}}(j)$ must contain $\mathsf{Op}\alpha$ for some time point $j \in \mathsf{ta}$ such that $j \geq \mathsf{t_{now}}$.

    Hence, it is only if $\mathsf{tai}_2 = \mathsf{t_{now}}$ that we need to worry about evaluating the rule wrt state dependent tascs (using modus ponens). State independent tascs in rules add constraints on the sets $\mathcal{TS}_{\mathsf{t_{now}}}(i)$ for $i \geq \mathsf{t_{now}}$.

The problem is therefore to compute the set $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$! Adding more and more action status atoms to $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$ leads to more and more program rules that evaluate to true! However, adding an action status atom $\mathbf{Do}\,\beta$ to $\mathcal{TS}_{\mathsf{t_{now}}}(i)$ where $i > \mathsf{t_{now}}$ influences the set $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$! For instance, suppose the checkpoint consistency constraints are satisfied, so that $\mathbf{Do}\,\beta$ can be put into $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}} + 1)$. As these constraints never ever change, $\mathbf{Do}\,\beta$ can stay there. But later we might be forced to make $\mathbf{F}\,\beta : [\mathsf{t_{now}}, \mathsf{t_{now}} + 1]$ true. As both $\mathbf{F}\,\beta$ and $\mathbf{Do}\,\beta$ cannot be in $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}} + 1)$ (deontic consistency), we are forced to include $\mathbf{F}\,\beta$ into $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$.

Therefore, when constructing the feasible temporal status set bottom-up, we have to be very cautious about adding action status atoms to $\mathcal{TS}_{\mathsf{t_{now}}}(i)$ for $i \geq \mathsf{t_{now}}$. When a rule body fires (as in item 4 above) and we have a choice of which $\mathcal{TS}_{\mathsf{t_{now}}}(j)$ to insert $\mathsf{Op}\alpha$ for $j \geq \mathsf{t_{now}}$, some choices might lead to success and others might lead to failure. Our algorithms reflect these choices.

Remark 8.1 As mentioned above, the feasible temporal status sets to be constructed are already determined for time points strictly smaller than $\mathsf{t_{now}}$. In particular, we assume from now on that $\mathsf{hist}_{\mathsf{t_{now}}}(\cdot)$ and $\mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$ are arbitrary but fixed. We also assume that $\mathsf{hist}_{\mathsf{t_{now}}}(\cdot)$ and $\mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$ are *consistent* with the existence of a temporal status set: by this, we mean that state consistency (point *(3)* on the previous page) holds for all time points $t < \mathsf{t}_{\mathsf{now}}$. Obviously, it can be easily decided if this condition holds or not.

Our algorithm to compute temporal feasible status sets is based on two broad steps.

(1) Find an interval constraint temporal status set ic-$\mathcal{T}S$ s.t. CompTSS(ic-$\mathcal{T}S$) is a superset of the set of all feasible temporal status sets. This step does not involve explicitly computing CompTSS(ic-$\mathcal{T}S$).
(2) Manipulate ic-$\mathcal{T}S$ so as to identify a member of CompTSS(ic-$\mathcal{T}S$) that is feasible.

Algorithms to implement Step (1) are described in Section 8.1, while algorithms to implement step (2) are described in Section 8.2.

8.1 Fixpoint Operator acting on ic-$\mathcal{T}S$'s

We refer the reader to the *intuitive reading of a temporal agent rule (third cut)* at the end of Section 5:

(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 $\mathsf{t}_{\mathsf{now}}$. Therefore, in this case, we only need to worry about the current time, $\mathsf{t}_{\mathsf{now}}$. Thus, if we derive new facts of the form $\mathsf{Op}'\,\alpha: \mathsf{tic}$ and we want to use these facts in the bodies of rules, we only have to look for occurrences of the form $\mathsf{Op}'\,\alpha : \mathsf{tic}'$ where tic' has $\mathsf{t}_{\mathsf{now}}$ as a solution.

We now associate with a temporal agent program $\mathcal{TP}$, an operator $\mathsf{D}_{\mathcal{TP}}$ which maps interval constraint temporal status sets into themselves. This definition assumes that implication of modalities is defined as follows: **O** implies **P** and **Do** , **Do** implies **P** and for every Op, Op implies Op.

Definition 8.1 (Operator $\mathsf{D}_{\mathcal{TP}}$) *Suppose $\mathcal{TP}$ is a tap, $hist_{\mathsf{t}_{now}}(\mathsf{t}_{now})$ is an agent state and ic-$\mathcal{T}S$ is an interval constraint temporal status set. Then we define $\mathsf{D}_{\mathcal{TP}}$(ic-$\mathcal{T}S$) to be the set*

$\{\mathit{Op}'\,\alpha\colon \mathit{tic} \,|\, \mathit{Op}\,\alpha : [\mathit{tai}_1, \mathit{tai}_2] \;\leftarrow\; \varrho_1 : \mathsf{ta}_1 \,\&\ldots\&\, \varrho_n : \mathsf{ta}_n$

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 $\varrho_i = \mathit{Op}_{i_1}\alpha_{i_1} \wedge \ldots \wedge \mathit{Op}_{i_n}\alpha_{i_m}$*)*

If $m = 1$ *then there exists* $\mathit{Op}'_{i_1}\alpha_{i_1} : \mathit{tic}_{i_1}$ *in* ic-$\mathcal{T}S$ *s. t.:*

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

(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* $\mathit{Op}'_{i_j}\alpha_{i_j} : [\mathsf{t_i}, \mathsf{t_i}]$ *in* ic-$\mathcal{T}S$

s. t. Op'_{i_j} *implies* Op_{i_j} *and* Op *implies* Op'*.*

(II) If ϱ_i ***is not state independent***

(we assume $\varrho_i = \chi_i \wedge \mathit{Op}_{i_1}\alpha_{i_1} \wedge \ldots \wedge \mathit{Op}_{i_n}\alpha_{i_m}$*)*

If $m = 0$ *then*

(0) χ_i *is true in the agent state* $\mathit{hist}_{\mathsf{t}_{now}}(\mathsf{t_i})$*, 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* $\mathit{Op}'_{i_j}\alpha_{i_j} : [\mathsf{t_i}, \mathsf{t_i}]$ *in* ic-$\mathcal{T}S$ *s. t.:*

(0) χ_i *is true in the agent state* $\mathit{hist}_{\mathsf{t}_{now}}(\mathsf{t}_i)$*,*

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

(2) t_i *is a solution of* ta_i*.*

and $\mathit{tic} = [\mathit{max}\{\mathit{tai}_1, \mathsf{t}_{now}\}, \mathit{tai}_2]$.

$\}$

Remark 8.2

- The overall idea of the operator above is to construct an ic-$\mathcal{T}S$ by iterating 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 $\mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$ (see Note 8.1). Therefore we define

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

  In most of the examples below, we assume without loss of generality that $\mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$ is empty and thus ic-$\mathcal{T}S_{\text{start}} = \emptyset$.
- There may be situations in which $\max\{\mathsf{tai}_1, \mathsf{t_{now}}\} > \mathsf{tai}_2$ for one of the rules

$\mathsf{Op}\,\alpha : [\mathsf{tai}_1, \mathsf{tai}_2] \leftarrow \varrho_1 : \mathsf{ta}_1 \,\&\ldots\&\, \varrho_n : \mathsf{ta}_n$ in $\mathcal{TP}$. In this case, there is no feasible temporal status set with respect to $\mathcal{TP}$, and our algorithm will report failure.

The following example illustrates the use of the $\mathsf{D}_{\mathcal{TP}}$ operator in the context of our Rescue Example.

Example 8.2 (Rescue: $\mathbf{D}_{\mathcal{TP}}$) Suppose the truck agent's tap, $\mathcal{TP}$, contains rules (r2) and (r3) from Example 6.4 which are recapitulated for convenience below:

r2: $\mathbf{Do}\,\textit{fill\_fuel}() : [\mathsf{t_{now}}, \mathsf{t_{now}}] \leftarrow$
$\quad\mathbf{in}(\mathbf{true}, \mathtt{truck} : \textit{tank\_empty}()) : [\mathsf{t_{now}} - 2, \mathsf{t_{now}}]$
r3: $\mathbf{O}\,\textit{order\_item}(\mathtt{fa\_bag}) : [\mathsf{t_{now}}, \mathsf{t_{now}} + 4] \leftarrow$
$\quad\mathbf{in}(1, \mathtt{truck} : \textit{inventory}(\mathtt{fa\_bag}))[\mathsf{t_{now}} - 3, \mathsf{t_{now}}]$

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

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

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

We will later apply this operator to construct an $\mathsf{ic}\text{-}\mathcal{T}S$ which represents a set of potential candidate feasible temporal status sets. This $\mathsf{ic}\text{-}\mathcal{T}S$ will be the least fixpoint of $\mathsf{D}_{\mathcal{TP}}$. To show that a least fixpoint of $\mathsf{D}_{\mathcal{TP}}$ exists, we have to show that the operator is monotone (Theorem 8.3). In fact, $\mathsf{D}_{\mathcal{TP}}$ is also continuous (Theorem 8.6) so that the least fixpoint is reached after ω-iterations (Theorem 8.9).

Theorem 8.3 (Inclusion Monotonicity) *Suppose $\mathcal{TP}$ is a positive tap, $\mathcal{O}$ is an agent state and $\mathsf{ic}\text{-}\mathcal{T}S$ is an interval constraint temporal status set. Then: $D_{\mathcal{TP}}$ is monotone, i.e.*

$$\mathit{ic}\text{-}\mathcal{T}S_1 \subseteq \mathit{ic}\text{-}\mathcal{T}S_2 \Rightarrow D_{\mathcal{TP}}(\mathit{ic}\text{-}\mathcal{T}S_1) \subseteq D_{\mathcal{TP}}(\mathit{ic}\text{-}\mathcal{T}S_2).$$

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

$$\mathsf{Op}\,\alpha : [\mathsf{tai}_1, \mathsf{tai}_2] \leftarrow \rho_1 : \mathsf{ta}_1 \,\&\ldots\&\, \rho_n : \mathsf{ta}_n$$

such that for all $1 \leq i \leq n$, there exist $\mathsf{Op}'_{i_j} \alpha_{i_j} : \mathsf{tic}_{i_j}$ in ic-$\mathcal{T}S_1$ such that conditions (1)-(3) hold (condition (0) if ρ_i is state independent) and tic has the required form. But in this case, as ic-$\mathcal{T}S_1 \subseteq$ ic-$\mathcal{T}S_2$, all the $\mathsf{Op}'_{i_j} \alpha_{i_j} : \mathsf{tic}_{i_j}$ are also in ic-$\mathcal{T}S_2$ and hence, the conditions for $\mathsf{Op}\, \alpha : \mathsf{tic} \in \mathsf{D}_{\mathcal{TP}}(\text{ic-}\mathcal{T}S_2)$ hold.

∎

The following result proves a different kind of monotonicity result. Given two interval constraint temporal status sets ic-$\mathcal{T}S_1$, ic-$\mathcal{T}S_2$, we say that ic-$\mathcal{T}S_1$ is *less vague* that ic-$\mathcal{T}S_2$, denoted ic-$\mathcal{T}S_1 \leq_v$ ic-$\mathcal{T}S_2$, *if* there is a surjective function f from ic-$\mathcal{T}S_1$ to ic-$\mathcal{T}S_2$ such that:

$f(\mathsf{Op}\alpha : \mathsf{tic}) = \mathsf{Op}\alpha : \mathsf{tic}'$ and every solution of tic is a solution of tic'.

To see why this captures the meaning of *less vague*, consider the following example.

Example 8.4 (Rescue: Vagueness) Suppose ic-$\mathcal{T}S_1 = \{\mathbf{P}\alpha : 5 \leq \mathsf{t} \leq 6\}$ and ic-$\mathcal{T}S_2 = \{\mathbf{P}\alpha : 3 \leq \mathsf{t} \leq 7\}$. The first interval constraint temporal status set says $\mathbf{P}$ is permitted sometime between time 5 and 6, while the second says it is permitted sometime between time 3 and 7—the former statement is certainly less vague than the latter.

In contrast to the previous monotonicity result, the following result shows that $\mathsf{D}_{\mathcal{TP}}$ is anti-monotonic w.r.t. to vagueness.

Theorem 8.5 (Vagueness Anti-Monotonicity) *Suppose $\mathcal{TP}$ is a positive tap, $\mathcal{O}$ is an agent state and ic-$\mathcal{T}S$ is an interval constraint temporal status set. Then $\mathsf{D}_{\mathcal{TP}}$ is $\leq_v$-anti-monotone, i.e.*

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

Proof: Suppose $\mathsf{Op}\, \alpha : \mathsf{tic} \in \mathsf{D}_{\mathcal{TP}}(\text{ic-}\mathcal{T}S_2)$. Then there is a rule in $\mathcal{TP}$ having a ground instance of the form

$$\mathsf{Op}\, \alpha : [\mathsf{tai}_1, \mathsf{tai}_2] \leftarrow \rho_1 : \mathsf{ta}_1 \,\&\, \ldots \,\&\, \rho_n : \mathsf{ta}_n$$

such that for all $1 \leq i \leq n$, there exist $\mathsf{Op}'_{i_j} \alpha_{i_j} : \mathsf{tic}_{i_j}$ in ic-$\mathcal{T}S_1$ such that conditions (1)-(3) hold (condition (0) if ρ_i is state independent) and tic has the required form.

As ic-$\mathcal{T}S_1 \leq_v$ ic-$\mathcal{T}S_2$, it follows that for each $1 \leq i \leq n$, ic-$\mathcal{T}S_1$ contains an interval constraint annotated status atom $\mathsf{Op}'_i\alpha : \mathsf{tic}^\star_i$ such that every solution of $\mathsf{tic}^\star_i$ is also a solution of tic_i. Hence, for al $1 \leq i \leq n$, $\mathsf{tic}^\star_i$ implies tic_i and therefore it also implies $\mathsf{t} \in \mathsf{ta}_i$. Hence, $\mathsf{Op}\,\alpha : \mathsf{tic}_i^\star$ is in $\mathsf{D}_{\mathcal{TP}}(\mathsf{ic}\text{-}\mathcal{T}S_1)$ and it is easy to see that $\mathsf{Op}\,\alpha : \mathsf{tic}_i^\star$ is less vague than $\mathsf{Op}\,\alpha : \mathsf{tic}$. ∎

The following theorem guarantees that $\mathsf{D}_{\mathcal{TP}}$ is continuous w.r.t. inclusion.

Theorem 8.6 (Continuity) *Suppose $\mathsf{ic}\text{-}\mathcal{T}S_1 \subseteq \mathsf{ic}\text{-}\mathcal{T}S_2 \subseteq \cdots \subseteq \mathsf{ic}\text{-}\mathcal{T}S_n \subseteq \mathsf{ic}\text{-}\mathcal{T}S_{n+1} \subseteq \cdots$ is an ascending chain of interval constraint temporal status sets. Then:*

$$D_{\mathcal{TP}}(\bigcup_i \mathsf{ic}\text{-}\mathcal{T}S_i) = \bigcup_i D_{\mathcal{TP}}(\mathsf{ic}\text{-}\mathcal{T}S_i).$$

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

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

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

$$\mathsf{Op}\,\alpha : [\mathsf{tai}_1, \mathsf{tai}_2] \leftarrow \rho_1 : \mathsf{ta}_1 \,\&\, \ldots \,\&\, \rho_n : \mathsf{ta}_n$$

such that for all $1 \leq i \leq n$, there exist $\mathsf{Op}'_{i_j}\alpha_{i_j} : \mathsf{tic}_{i_j}$ in $\bigcup_i \mathsf{ic}\text{-}\mathcal{T}S_i$ such that conditions (1)-(3) hold (condition (0) if ρ_i is state independent) and tic has the required form.

As n is finite, there must exist an integer r such that the above conditions are satisfied by $\mathsf{ic}\text{-}\mathcal{T}S_r$. In this case, we have $\mathsf{Op}'_{i_j}\alpha_{i_j} : \mathsf{tic}_{i_j} \in \mathsf{D}_{\mathcal{TP}}(\mathsf{ic}\text{-}\mathcal{T}S_r)$ and we are done. ∎

One major consequence of the above theorem is that if we iteratively apply the $\mathsf{D}_{\mathcal{TP}}$ operator, starting from the empty interval constraint temporal status set, then we will be guaranteed to terminate. In order to state this result, we must first define the *iterations* of the $\mathsf{D}_{\mathcal{TP}}$ operator.

Definition 8.7 (Iterations of $\mathsf{D}_{\mathcal{TP}}$) *Suppose $\mathcal{TP}$ is a positive* tap*, and $\mathcal{O}$ is an agent state. The* iterations *of $D_{\mathcal{TP}}$ are defined as follows:*

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

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

Example 8.8 (Rescue: $\mathbf{D}_{\mathcal{TP}}$) We will continue with Example 8.2 and will assume that the agent state is as described there, $\mathsf{acthist}_{\mathsf{t_{now}}}(\cdot)$ is empty and thus $\mathsf{ic}\text{-}\mathcal{T}S_{\mathrm{start}} = \emptyset$. 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: $\mathbf{F}\mathit{drive}(\mathtt{was}, \mathtt{bal}, \mathtt{Highway}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}} + 2] \leftarrow$
$\mathbf{Do}\,\mathit{fill\_fuel}()\ \&\ \mathbf{in}(\mathbf{true}, \mathtt{truck} : \mathit{tank\_empty}()) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}} + 2]$

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

- $\mathsf{D}_{\mathcal{TP}} \uparrow^{0} = \emptyset$ since $\mathsf{ic}\text{-}\mathcal{T}S_{\mathrm{start}} = \emptyset$.
- $\mathsf{D}_{\mathcal{TP}} \uparrow^{(1)} = \mathsf{D}_{\mathcal{TP}}(\mathsf{D}_{\mathcal{TP}} \uparrow^{0}) = \mathsf{D}_{\mathcal{TP}}(\emptyset)$. The additional two rules in $\mathcal{TP}$ didn't change the set $\mathsf{D}_{\mathcal{TP}}(\emptyset)$, and it is as in Example 8.2, i.e.,

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

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

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

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

The following result specifies that $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ is the least fixpoint of $\mathsf{D}_{\mathcal{TP}}$.

Theorem 8.9 *Suppose $\mathcal{TP}$ is a positive `tap`, and $\mathcal{O}$ is an agent state. Then: $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ is the least fixpoint of $\mathsf{D}_{\mathcal{TP}}$.*

Proof: Immediate consequence of the Tarski-Knaster theorem (Lloyd 1987) that states that if f is a continuous function on a complete lattice, then $f \uparrow^{\omega}$

is the least fixpoint of f. Here, $\mathsf{D}_{\mathcal{TP}}$ is a continuous function, and the set of all interval constraint temporal status sets is a complete lattice under set inclusion. ∎

We are now ready to show that $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ has various desirable properties that will later help us in computing feasible temporal status sets. First, we need to define three important properties of an interval constraint temporal status set.

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

The following example illustrates these concepts.

Example 8.10 Consider the ic-$\mathcal{T}S$ of Example 8.2, i.e.,

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

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

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

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

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

$$\begin{aligned}\mathcal{T}S_{\mathsf{t_{now}}}(3) = \{&\mathbf{Do}\,\mathit{fill\_fuel}(),\ \mathbf{P}\mathit{fill\_fuel}(),\ \ \mathbf{O}\mathit{order\_item}(\mathtt{fa\_bag}),\\ &\mathbf{P}\mathit{order\_item}(\mathtt{fa\_bag}),\ \ \mathbf{Do}\,\mathit{order\_item}(\mathtt{fa\_bag})\}\end{aligned}$$

The following result shows 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 $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$.

Theorem 8.11 *Suppose $\mathcal{TP}$ is a positive* tap*, and $\mathcal{O}$ 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 must be compatible with* $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$.

Proof:

(1) Let $X_{\mathbf{O}}$ be the set of all interval constraint annotated status atoms of the form $\mathbf{O}\alpha : \mathsf{tic}$ in $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$. Let $X_{\mathbf{P}}$ be the set of all interval constraint annotated status atoms of the form $\mathbf{P}\alpha : \mathsf{tic}$ in $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$. Each atom $\mathbf{O}\alpha : \mathsf{tic}$ in $X_{\mathbf{O}}$ must be in $\mathsf{D}_{\mathcal{TP}} \uparrow^{j}$ for some integer j, but then, as $\mathbf{O}$ implies $\mathbf{P}$, the same rule used to place $\mathbf{O}\alpha : \mathsf{tic}$ in $\mathsf{D}_{\mathcal{TP}} \uparrow^{j}$ must also have been used to insert $\mathbf{P}\alpha : \mathsf{tic}$ into $\mathsf{D}_{\mathcal{TP}} \uparrow^{j}$ which means $\mathbf{P}\alpha : \mathsf{tic} \in X_{\mathbf{P}}$. One may now construct a temporal status set $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$ as follows: for each $\mathbf{O}\alpha : \mathsf{tic}$ in $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$, insert $\mathbf{O}\alpha, \mathbf{P}\alpha$ into $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}(j)$ where j is the smallest integer which is a solution of tic. For all other modalities $\mathsf{Op} \neq \mathbf{O}$, if $\mathsf{Op}\alpha : \mathsf{tic}$ in $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$, insert $\mathsf{Op}\alpha$ into $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}(j)$ where j is the smallest integer which is a solution of tic. It is easy to see that $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$ is compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ and $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$ is temporally deontically consistent.
(2) Similar to the proof of the previous item.
(3) Suppose $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$ is a feasible temporal status set which is not compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$. We will attempt to derive a contradiction.

We call an interval constraint annotated status atom $\mathsf{Op}\alpha : \mathsf{tic}$ a *rogue* atom *if* $\mathsf{Op}\alpha \notin \mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}(i)$ for all i's that are solutions of tic. Let Rogues be the set of all rogue atoms associated with $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$, and let

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

We proceed by induction on j.

j=0: In the base case, $\mathsf{D}_{\mathcal{TP}} \uparrow^{0} = \mathsf{ic}\text{-}\mathcal{T}S_{\text{start}}$. Since $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$ is a feasible temporal status set, it is history compatible at time $\mathsf{t}_{\mathsf{now}}$ and hence cannot contain any rogue atoms.

j=s+1: As no rogues occur in $\mathsf{D}_{\mathcal{TP}} \uparrow^{s}$, we know that $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$ is compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{s}$. As $\mathsf{Op}\alpha : \mathsf{tic} \in \mathsf{D}_{\mathcal{TP}} \uparrow^{s+1}$, there must exist a rule in $\mathcal{TP}$ having a ground instance of the form shown in Definition 8.1 and satisfying the conditions stated there. Clearly, each interval constraint annotated status atom in the body of this rule is satisfied by $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$, i.e. for each interval constraint annotated status atom $\mathsf{Op}_i\alpha_i : \mathsf{tic}_i$ in the body of this rule, $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}(t_i)$ contains $\mathsf{Op}_i\alpha_i$ where t_i is a solution of tic_i. As $\mathcal{T}S_{\mathsf{t}_{\mathsf{now}}}$ is feasible, by the "closure under program rules" condition in the definition of feasibility, it must satisfy the head of this rule, but to do this, it must satisfy the constraint attached to the rule head, which coincides with tic. This contradicts our assumption.

The following example demonstrates how the feasible temporal status sets of a tap must be compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$.

Example 8.12 (Rescue: Compatibility with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$) Suppose the $\mathtt{truck}$ agent's tap, $\mathcal{TP}$, includes the rules **r2** and **r3** as in Examples 8.2 and 8.10, $\mathsf{hist}_{\mathsf{t_{now}}}$ is as in Example 8.10, $\mathcal{IC}$ and AC are as in Example 6.18, and for all $i \geq 0$, $\mathsf{acthist}_{\mathsf{t_{now}}}(i) = \emptyset$, and $\mathsf{t_{now}} = 3$. It is easy to see that $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ is equal to the ic-$\mathcal{T}S$ of Example 8.10, i.e.,

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

The following is a feasible temporal status state with respect to the above parameters. It is easy to see that it is compatible with $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$. For all $i \neq 3$, $\mathcal{T}S_{\mathsf{t_{now}}}(i) = \emptyset$, and

$$\begin{aligned}\mathcal{T}S_{\mathsf{t_{now}}}(3) = \{ & \mathbf{Do}\, \mathit{fill\_fuel}(), \ \mathbf{P} \mathit{fill\_fuel}(), \ \ \mathbf{O}\, \mathit{order\_item}(\mathtt{fa\_bag}), \\ & \mathbf{P}\, \mathit{order\_item}(\mathtt{fa\_bag}), \ \ \mathbf{Do}\, \mathit{order\_item}(\mathtt{fa\_bag}) \}\end{aligned}$$

What exactly is the role of $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$? $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ serves as a starting point to compute feasible temporal status sets. But note that $\mathsf{D}_{\mathcal{TP}} \uparrow^{\omega}$ is not, by itself, a feasible temporal status set. This is because our notion of "closure under rules" (Definition 6.16) admits *reasoning by cases*. Let us illustrate this.

Example 8.13 ($\mathsf{D}_{\mathcal{TP}}$ and Closure under the Rules)
Consider the following temporal program

$$\begin{aligned}&\mathbf{P}\alpha_1 : [10, 10] \leftarrow \mathbf{P}\beta : [1, 5] \\ &\mathbf{P}\alpha_2 : [10, 10] \leftarrow \mathbf{P}\beta : [6, 10] \\ &\mathbf{P}\beta : [1, 10] \quad \leftarrow\end{aligned}$$

The $\mathsf{D}_{\mathcal{TP}}$ operator is very cautious: as $[1, 10]$ neither implies $[1, 5]$ nor $[6, 10]$, the least fixpoint is simply $\{\mathbf{P}\beta : [1, 10]\}$. However, any temporal status set closed under the rules of this program contains either $\mathbf{P}\alpha_1 : [10, 10]$ or $\mathbf{P}\alpha_2 : [10, 10]$: this is because $\mathbf{P}\beta$ must be contained in at least one of $\mathcal{T}S_{\mathsf{t_{now}}}(1)$, $\mathcal{T}S_{\mathsf{t_{now}}}(2), \ldots, \mathcal{T}S_{\mathsf{t_{now}}}(10)$ and thus at least $\mathbf{P}\alpha_1$ or $\mathbf{P}\alpha_2$ must be contained in $\mathcal{T}S_{\mathsf{t_{now}}}(10)$.

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 8.1 describes this procedure. It uses three major subroutines: one to find compatible temporal status sets, called **FindCompTSS** described in Section 8.2.1, one to compute temporal status sets satisfying certain conditions called **ComputeTSS** described in Section 8.2.2, and one to compute certain changes to temporal status sets called **Modified_Set** described in Section 8.2.3. (In addition, it uses some relatively minor subroutines that we will describe as needed).[7]

Algorithm 8.1 (Feasible Temporal Status Set Computation)

FTSS$(\mathcal{O}_{\mathcal{S}}, \mathcal{TP}, \mathsf{hist}_{\mathsf{t_{now}}}, \mathsf{acthist}_{\mathsf{t_{now}}})$

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

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

The **FTSS** algorithm terminates as soon as a feasible temporal status set is found. The **Check_trivial_part** subroutine determines $\mathcal{T}S_{\mathsf{t_{now}}}(i)$ for $i < \mathsf{t_{now}}$ by history compatibility and then checks checkpoint consistency and state

[7] In order to simplify the procedures we will sometime refer to a temporal constraint set ic-$\mathcal{T}S$ as a temporal status set. The intended temporal status set $\mathcal{T}S_{\mathsf{t_{now}}}$ is defined as follows: for each time point i, $\mathsf{Op}\,\alpha \in \mathcal{T}S_{\mathsf{t_{now}}}(i)$ iff $\mathsf{Op}\,\alpha : [i,i] \in \mathsf{ic}$-$\mathcal{T}S$ or $\mathsf{Op}\,\alpha : (i \leq t \leq i) \in \mathsf{ic}$-$\mathcal{T}S$.

consistency for $\mathcal{TS}_{\mathsf{t_{now}}}(i)$ for $i < \mathsf{t_{now}}$. 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 must find a new compatible temporal status set that is feasible.

As **FindCompTSS** computes compatible status sets that are closed under the rules of the program, the **FeasTSS** algorithm only needs to check (for a given $\mathcal{TS}_{\mathsf{t_{now}}}$)

(1) checkpoint consistency for $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$,
(2) action consistency for $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$,
(3) 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 $\mathsf{hist}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$,
(4) temporal deontic consistency for $\mathcal{TS}_{\mathsf{t_{now}}}(\mathsf{t_{now}})$. (As we require our temporal status sets to be finite, we only need to check this for finitely many i.)

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

New compatible temporal status sets are generated by the **FindCompTSS** subroutine. This algorithm, which we will define shortly, uses the notion of a culprit (Definition 8.14). Culprits reflect ways of generating compatible temporal status sets different from those generated before.

Definition 8.14 (Culprit) *Suppose* $\mathsf{ic}\text{-}\mathcal{TS}$ *is an interval constraint temporal status set, and Seen is a set of temporal status sets closed under the rules of* $\mathcal{TP}$*. A set CAA of constraint annotated atoms is said to be a* culprit set *of* $\mathsf{ic}\text{-}\mathcal{TS}$ *with respect to Seen and* $\mathcal{TP}$ *if*

(1) If $\mathsf{Op}\alpha : \mathsf{ta} \in CAA$*, then* ta *is of the form* $[\mathsf{tai}, \mathsf{tai}]$ *(i.e. the annotation represents a single time point) and*
(2) If $\mathsf{Op}\alpha : [\mathsf{tai}, \mathsf{tai}] \in CAA$*, then there exists an* $\mathsf{Op}\alpha : \mathsf{tic}$ *in* $\mathsf{ic}\text{-}\mathcal{TS}$ *such that* tai *is a solution of* tic *and*
(3) For every $\mathcal{TS}_{\mathsf{t_{now}}} \in Seen$*, there exists an* $\mathsf{Op}\alpha : \mathsf{tic}$ *in* $\mathsf{ic}\text{-}\mathcal{TS}$ *such that* tic *has* i *as a solution and* $\mathsf{Op}\alpha : [i, i] \in CAA$ *and* $\mathsf{Op}\alpha \notin \mathcal{TS}_{\mathsf{t_{now}}}(i)$*. In this case,* $\langle \mathsf{Op}\alpha : \mathsf{tic}, \mathcal{TS}_{\mathsf{t_{now}}} \rangle$ *is called a* culprit-TSS pair.

Example 8.15 (Rescue: Culprit) Suppose the $\mathtt{truck}$ agent's tap, $\mathcal{TP}$, includes rules **r2** and **r3** as in Examples 8.2 and 8.10, and $\mathsf{ic}\text{-}\mathcal{TS}$ is as in Example 8.12, and *Seen* includes the $\mathcal{TS}_{\mathsf{t_{now}}}$ of Example 8.12 and the following $\mathcal{TS}'_{\mathsf{t_{now}}}$

For all $i < 3$ and $i > 5$, $\mathcal{TS}'_{\mathsf{t_{now}}}(i) = \emptyset$, and

$$\mathcal{TS}'_{\mathsf{t_{now}}}(3) = \{\mathbf{Do}\, \mathit{fill\_fuel}(),\ \mathbf{P}\mathit{fill\_fuel}()\}$$
$$\mathcal{TS}'_{\mathsf{t_{now}}}(4) = \{\mathbf{O}\mathit{order\_item}(\mathtt{fa\_bag}), \mathbf{P}\mathit{order\_item}(\mathtt{fa\_bag})\}$$

$$CAA = \{\mathbf{O}\mathit{order\_item}(\mathtt{fa\_bag})\colon[4,4]\}$$
$$\mathbf{P}\mathit{order\_item}(\mathtt{fa\_bag})\colon[5,5]\}$$

is a culprit set of ic-$\mathcal{TS}$ with respect to *Seen* and $\mathcal{TP}$.

$$\langle \mathbf{O}\mathit{order\_item}(\mathtt{fa\_bag})\colon[4,4], \mathcal{TS}_{\mathsf{t_{now}}}\rangle, \langle \mathbf{P}\mathit{order\_item}(\mathtt{fa\_bag})\colon[5,5], \mathcal{TS}'_{\mathsf{t_{now}}}\rangle$$

are culprit-TSS pairs.

An algorithm, **FindCulprit**(ic-$\mathcal{TS}$,*Seen*), to find a culprit is relatively simple. Pick a member of *Seen* and then find an $\mathsf{Op}\alpha$: tic in ic-$\mathcal{TS}$ that satisfies condition *(3)* in the definition of culprits. Insert $\mathsf{Op}\alpha : [i,i]$ into the culprit set. Repeat this for each member of *Seen*. The resulting temporal status set is compatible with ic-$\mathcal{TS}$, but is different from all the members of *Seen*. The algorithm for culprit identification is easily seen to run in time $O(|\text{ic-}\mathcal{TS}| \cdot |\mathit{Seen}|)$. We extend this function by another argument H: H is a set of CAA's which were already considered.

8.2.1 Algorithm to Find Compatible Temporal Status Sets

Intuitively, when we have a set ic-$\mathcal{TS}$ of interval constraint temporal status atoms, and we have a set *Seen* of temporal status sets compatible with ic-$\mathcal{TS}$, the culprit set associated with ic-$\mathcal{TS}$ and *Seen* specifies ways of modifying the elements of *Seen* so as to get a new temporal status set not in *Seen* that is compatible with ic-$\mathcal{TS}$. This may be done as follows.

Algorithm 8.2 (Find Compatible TSS ($\mathcal{O}_\mathcal{S}$,ic-$\mathcal{TS}$,*Seen*))
FindCompTSS($\mathcal{O}_\mathcal{S}$,ic-$\mathcal{TS}$,Seen)

($\star$ input is an agent state $\mathcal{O}_\mathcal{S}$, a positive tap $\mathcal{TP}$ *$\star$)*
($\star$ an interval constraint temporal status set ic-$\mathcal{TS}$ *$\star$)*
($\star$ and a set Seen of temporal status sets $\star$)
($\star$ output is a compatible temporal status set not in Seen $\star$)
($\star$ if such a set exists and "No" otherwise $\star$)

(1) ***if*** *Seen* $= \emptyset$ ***then return ComputeTSS***(ic-$\mathcal{TS}$,$\mathcal{TP}$,*Seen*);

(2) $H := \emptyset$;
(3) $CAA :=$ ***FindCulprit***(ic-$\mathcal{TS}$,*Seen*, H);
(4) ***if*** $CAA = \emptyset$ ***then return ComputeTSS***(ic-$\mathcal{TS}$,$\mathcal{TP}$,*Seen*);
(5) *CTSSP = set of all culprit-TSS pairs of* ic-$\mathcal{TS}$,*Seen w.r.t* CAA;
(6) ***while*** $CTSSP \neq \emptyset$ ***do***
  (a) Select a pair $\langle \mathsf{Op}\alpha : \mathsf{tic}, \mathcal{TS}_{\mathtt{t}_{now}} \rangle$ *from* $CTSSP$;
  (b) Set $CTSSP := CTSSP \setminus \{\langle \mathsf{Op}\alpha : \mathsf{tic}, \mathcal{TS}_{\mathtt{t}_{now}} \rangle\}$;
  (c) Set $\mathcal{TS}'_{\mathtt{t}_{now}} :=$ ***Modified_Set***(ic-$\mathcal{TS}$,$\mathcal{TP}$,$\mathcal{TS}_{\mathtt{t}_{now}}$, $\mathsf{Op}\alpha : \mathsf{tic}$).
  (d) ***if*** $\mathcal{TS}'_{\mathtt{t}_{now}} \neq$ *"No"* $\wedge$ $\mathcal{TS}'_{\mathtt{t}_{now}} \notin$ *Seen* ***then return*** $\mathcal{TS}'_{\mathtt{t}_{now}}$.
(7) $H := H \cup \{CAA\}$;
(8) ***Go to 3***;

Remark 8.3 The simplest algorithm for **FindCompTSS(**$\mathcal{O}_{\mathcal{S}}$**,ic-**$\mathcal{TS}$**,*Seen*)** is the one consisting solely of calling the function **ComputeTSS**(ic-$\mathcal{TS}$,$\mathcal{TP}$,*Seen*). In fact, if ic-$\mathcal{TS}$ is trivial (e.g. empty), then CAA is empty and Algorithm 8.2 comes down to line *(4)*. However, in many cases ic-$\mathcal{TS}$ is more structured and different temporal status sets can be distinguished by a $\mathsf{Op}\alpha : \mathsf{tic}$ in ic-$\mathcal{TS}$ according to condition *(3)* of Definition 8.14. Thus the culprits give us in many cases a much more precise instrument for determining different status sets, rather than just calling the function **ComputeTSS**(ic-$\mathcal{TS}$,$\mathcal{TP}$,*Seen*).

Let us elaborate on the two subroutines **ComputeTSS** and **Modified_Set**.

Definition 8.16 (Input and Output of ComputeTSS)
The ***ComputeTSS*** *function takes as input (1) an interval constraint temporal status set* ic-$\mathcal{TS}$*, (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* ic-$\mathcal{TS}$*, and different from the sets in Seen and minimal wrt. these properties, if such a set exists, or "No" (if no such temporal status set exists).*

Definition 8.17 (Input and Output of Modified_Set)
The ***Modified_Set*** *function takes as inputs (1) an interval constraint temporal status set* ic-$\mathcal{TS}$*, (2) a positive temporal program* $\mathcal{TP}$*, (3) a temporal status set* $\mathcal{TS}_{\mathtt{t}_{now}}$ *closed under the rules of* $\mathcal{TP}$*, and (4) a temporal atom* $\mathsf{Op}\alpha : [i, i]$ *such that* $\mathsf{Op}\alpha \notin \mathcal{TS}_{\mathtt{t}_{now}}(i)$*.*

It either returns a new temporal status set $\mathcal{TS}^{new}_{\mathtt{t}_{now}}$ *which is closed under the rules of* $\mathcal{TP}$*, compatible with* ic-$\mathcal{TS}$ *and satisfies* $\mathsf{Op}\alpha \in \mathcal{TS}_{\mathtt{t}_{now}}(i)$*, and minimal wrt. these properties if such a set exists, or "No" otherwise. In addition,* $\mathcal{TS}_{\mathtt{t}_{now}}$ *is not contained in* $\mathcal{TS}^{new}_{\mathtt{t}_{now}}$*.*

The following theorem states that algorithm **FindCompTSS** is correct if the subroutines it uses are correctly implemented. We will prove correctness of those subroutines later.

Theorem 8.18 *Suppose the subroutines invoked by algorithm **FindCompTSS** behave correctly as defined. Then:*

*(1) If Algorithm **FindCompTSS** returns a temporal status set $\mathcal{TS}'_{\mathsf{t}_{now}}$ on the inputs $(\mathcal{O}, \mathsf{ic}\text{-}\mathcal{TS}, Seen)$, then $\mathcal{TS}'_{\mathsf{t}_{now}}$ is compatible with $\mathsf{ic}\text{-}\mathcal{TS}$ in state $\mathcal{O}$ and $\mathcal{TS}'_{\mathsf{t}_{now}} \notin Seen$.*
*(2) If Algorithm **FindCompTSS** returns "No" then there is no temporal status set compatible with $\mathsf{ic}\text{-}\mathcal{TS}$ in state $\mathcal{O}$ which is not in Seen.*

Proof:

(1) Suppose algorithm **FindCompTSS** returns a temporal status set $\mathcal{TS}'_{\mathsf{t}_{\mathsf{now}}}$ on the inputs $(\mathcal{O}, \mathsf{ic}\text{-}\mathcal{TS}, Seen)$. If this is returned in step 1, then we know by definition of the **ComputeTSS** algorithm, that the output returned is compatible with $\mathsf{ic}\text{-}\mathcal{TS}$. The only other step in which $\mathcal{TS}'_{\mathsf{t}_{\mathsf{now}}}$ can be returned is Step 6d inside the while loop of Step 6. This step immediately ensures that $\mathcal{TS}'_{\mathsf{t}_{\mathsf{now}}}$ is not in *Seen*. By part (4) of the definition of **Modified_Set** (Definition 8.17), it follows that the output returned is compatible with $\mathsf{ic}\text{-}\mathcal{TS}$.
(2) The only way that algorithm **FindCompTSS** returns "No" is when **ComputeTSS**($\mathsf{ic}\text{-}\mathcal{TS}$,$\mathcal{TP}$,*Seen*) returns "No". Thus the claim follows from the correctness of **ComputeTSS**.

8.2.2 Algorithm ComputeTSS

We now present a detailed algorithm for **ComputeTSS**—later we will explain how this function may be used to implement the **Modified_Set** function.

Formally, the function **ComputeTSS** is used directly only at the beginning of Algorithm 8.2 when *Seen* is still empty (later, in Section 8.2.3, we will see that it is also used in the **Modified_Set** algorithm). We have to construct a temporal status set $\mathcal{TS}_{\mathsf{t}_{\mathsf{now}}}$ 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 $\mathcal{TS}_{\mathsf{t}_{\mathsf{now}}}(4)$.

But all other atoms consisting of non-singleton tic's must also be satisfied. How can we assign them to $\mathcal{TS}_{\mathsf{t}_{\mathsf{now}}}$? For a precise description, let us introduce the concept of a constraint hitting set.

Definition 8.19 (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 $\mathit{Op}\,\alpha\colon \mathit{tic} \in \mathit{ic}\text{-}\mathcal{TS}$, there is an annotated atom of the form $\mathit{Op}\,\alpha\colon [i,i]$ in H such that i is a solution of tic, and if $i < t_now$, then $\mathit{Op}\,\alpha \in \mathit{acthist}_{\mathsf{t}_{now}}(i)$.

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

We will use a subroutine called *find_member_chs*($\mathsf{ic}\text{-}\mathcal{TS}, \mathit{Seen}$) which finds a member of $\mathsf{chs}(\mathsf{ic}\text{-}\mathcal{TS})$ 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)) in time proportional to the product of

(1) the cardinality of $\mathsf{ic}\text{-}\mathcal{TS}$,
(2) $\max\{card(X) \mid \mathsf{Op}\alpha : \mathsf{tic} \in \mathsf{ic}\text{-}\mathcal{TS}$ and X is the set of all solutions of $\mathsf{tic}\}$, and
(3) cardinality of *Seen*, and
(4) $\max\{\mathrm{card}(Y) \mid Y \in \mathit{Seen}\}$.

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. If this happens, then the $\mathsf{ic}\text{-}\mathcal{TS}$ we started with cannot be extended to a temporal status set closed under the rules of $\mathcal{TP}$ and still compatible with $\mathsf{ic}\text{-}\mathcal{TS}$.
(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 $\mathsf{ic}\text{-}\mathcal{TS}$. 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 8.20 (Solution closed) *A set H of interval constrained annotated atoms is said to be* solution-closed *iff:*

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

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

Given this requirement, another problem is that the $\mathsf{ic}\text{-}\mathcal{T}S$ with which we started is obtained as the least fixpoint of the D 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 includes 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 8.3). We then apply our operator D to $\mathcal{TP}_{new}$ (see *(2)(d)(ii)(B)* in Algorithm 8.3). 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 8.3), or
(2) we reach a fixpoint H^*. This fixpoint yields a better candidate $\mathsf{ic}\text{-}\mathcal{T}S_{new}$ and we have to re-iterate the whole process, by first computing a hitting set of $\mathsf{ic}\text{-}\mathcal{T}S_{new}$ and then computing the iterations of our operator D.

We may now implement **ComputeTSS** as follows.

Algorithm 8.3 (ComputeTSS($\mathsf{ic}\text{-}\mathcal{T}S, \mathcal{TP}, Seen$))
ComputeTSS($\mathsf{ic}\text{-}\mathcal{T}S, \mathcal{TP}, Seen$)

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

(1) $done$:= **false**; $found$:= **false**; $Loc\_Seen := Seen$;
$\quad$ $\mathsf{ic}\text{-}\mathcal{T}S_{new} := \mathsf{ic}\text{-}\mathcal{T}S$; $H^* := \emptyset$, $done\_inner$:= **false**;
(2) ***while*** $\neg done \wedge \neg found$ ***do***
$\quad$ *(a)* ***if*** $done\_inner$ ***then***
$\quad\quad$ *(i)* $H = find\_member\_chs(\mathsf{ic}\text{-}\mathcal{T}S, Loc\_Seen)$;
$\quad\quad$ *(ii)* ***if*** $H =$ *"No"* ***then*** $done$:= **true**;
$\quad\quad$ *(iii)* $done\_inner$:= **false**
$\quad$ *(b)* ***else***
$\quad\quad$ *(i)* $H = find\_member\_chs(\mathsf{ic}\text{-}\mathcal{T}S_{new}, Loc\_Seen)$;
$\quad\quad$ *(ii)* ***if*** $H =$ *"No"* ***then*** $done\_inner$= **true**;
$\quad$ *(c)* $Loc\_Seen := Loc\_Seen \cup \{H\}$;
$\quad$ *(d)* ***if*** $H \neq$ *"No"* ***then***
$\quad\quad$ *(i)* $H^\star = H$; $changed$ =**true**;
$\quad\quad$ *(ii)* ***while*** $\neg found \wedge changed$ ***do***
$\quad\quad\quad$ *(A)* $\mathcal{TP}_{new} = \mathcal{TP} \cup H^\star$; $oldH^\star = H^\star$;
$\quad\quad\quad$ *(B)* $H^\star = \mathsf{D}_{\mathcal{TP}_{new}}$;
$\quad\quad\quad$ *(C)* ***if*** $H^\star \notin Loc\_Seen \wedge H^\star$ *is solution closed*
$\quad\quad\quad\quad$ ***then*** $found$ = **true**

else changed $= (oldH^\star \neq H^\star)$;

(iii) $Loc\_Seen := Loc\_Seen \cup \{H^*\}$; $\mathsf{ic}\text{-}\mathcal{T}S_{new} := H^*$

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

The following lemma states that the above implementation of Algorithm 8.3 satisfies the output conditions of Definition 8.16.

Lemma 8.21 *Suppose the find_member_chs*($\mathsf{ic}\text{-}\mathcal{T}S_{new}, Loc\_Seen$) *algorithm is correctly implemented. Then:*

(1) If algorithm ***ComputeTSS*** *returns a temporal status set* H^*, *then* H^* *satisfies the output conditions of Definition 8.16.*
(2) If algorithm ***ComputeTSS*** *returns "No", then there is no temporal status set satisfying the output conditions of Definition 8.16.*

Proof: We start by observing that successive executions of the inner while loop cause $\mathcal{TP}_{new}$ to increase monotonically (i.e. w.r.t. subset inclusion) and hence, successive executions of the inner while loop cause $H^\star$ to expand monotonically (w.r.t. subset inclusion) as well. Hence, successive iterations of step (*iii*) cause $\mathsf{ic}\text{-}\mathcal{T}S_{new}$ to grow monotonically as well (as long as done remains false).

Concerning the outer while loop, we have to distinguish whether the **if** part (step *(a)*) or the **else** part (step *(b)*) has been applied. In the beginning, the else part is active. Then a hitting set H is computed (based on $\mathsf{ic}\text{-}\mathcal{T}S$) which is then refined (to guarantee that it is closed under the program rules). The inner loop computes the least fixpoint of H (using the operator $\mathsf{D}_{\mathcal{TP}_{new}}$). Once this least fixpoint H^* is computed, the outer loop has to be called again[8] with H^* as a new $\mathsf{ic}\text{-}\mathcal{T}S_{new}$. The iteration ends only, if

- H^* is a solution as determined in Step *(C)*). In this case the boolean "found" is set to **true**. Or,
- we enter the **if** part in step *(a)*. This means that all attempts to extend H^* to a solution closed set (i.e. a temporal status set closed under the rules of $\mathcal{TP}$) failed and we have to backtrack. Thus we have to add H^* to *Loc_Seen* and to find a new hitting set.

(1) Suppose **ComputeTSS** returns H^*. Clearly H^* is closed under the rules of $\mathcal{TP}$ because $H^\star = \mathsf{D}_{\mathcal{TP}_{new}}$ which is closed under all the rules of $\mathcal{TP}_{new}$ and $\mathcal{TP} \subseteq \mathcal{TP}_{new}$. H^* is compatible with $\mathsf{ic}\text{-}\mathcal{T}S$ because the function *find_member_chs*($\mathsf{ic}\text{-}\mathcal{T}S_{new}, Loc\_Seen$) correctly finds a constraint hitting set $H \subseteq H^*$ of $\mathsf{ic}\text{-}\mathcal{T}S_{new}$—however, it is easy to see that $\mathsf{ic}\text{-}\mathcal{T}S \subseteq \mathsf{ic}\text{-}\mathcal{T}S_{new}$ and hence H is a constraint hitting set of $\mathsf{ic}\text{-}\mathcal{T}S$ (though it may not satisfy

[8] See the explanation just above Definition 8.20.

the minimality requirement of a hitting set)—hence H is compatible with $\mathsf{ic}\text{-}\mathcal{TS}$. As $H \subseteq H^*$, H^* is also compatible with $\mathsf{ic}\text{-}\mathcal{TS}$. Lastly, Step (C) ensures that H^* is not in *Seen*.

(2) Suppose **ComputeTSS** returns "No." This can only happen in step (3), which itself happens only when the **if** part in step *(C)* applied. This means that there are no more unseen hitting sets. As all compatible status sets of $\mathsf{ic}\text{-}\mathcal{TS}$ must be supersets of some hitting set of $\mathsf{ic}\text{-}\mathcal{TS}$, there are no compatible temporal status sets of $\mathsf{ic}\text{-}\mathcal{TS}$ which are not in *Seen* and closed under the rules of $\mathcal{TP}$.

    Note that once a hitting set H is determined in the outer loop, there is no freedom in extending H to a potential solution. The algorithm extends H only by those elements that are strictly needed (iterating the $\mathsf{D}_{\mathcal{TP}}$ operator): the construction is deterministic. The only indeterminism is in step *(a)* where we backtrack over all possible hitting sets.

We note, in view of Remark 8.4, that the set generated by **ComputeTSS** is not a minimal one. i.e. there might be smaller sets closed under the rules which are compatible with $\mathsf{ic}\text{-}\mathcal{TS}$. The reason is that the generated hitting set might not have been optimal. As an illustration, we consider the $\mathsf{ic}\text{-}\mathcal{TS}$ 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 $\mathsf{ic}\text{-}\mathcal{TS}$. It is easy to modify Algorithm 8.3 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.

8.2.3 Algorithm for Modified_Set Computation

The function **Modified_Set** used as a subroutine in Algorithm 8.2 takes a set $\mathcal{TS}_{\mathsf{t_{now}}}$ known to be closed under the rules of $\mathcal{TP}$ and a culprit $\mathsf{Op}\alpha : [i,i]$ and computes a new set $\mathcal{TS}^{new}_{\mathsf{t_{now}}}$ such that $\mathsf{Op}\alpha \in \mathcal{TS}^{new}_{\mathsf{t_{now}}}(i)$ (note that, by the very definition of a culprit, $\mathsf{Op}\alpha \notin \mathcal{TS}_{\mathsf{t_{now}}}(i)$). The idea is to modify $\mathcal{TS}_{\mathsf{t_{now}}}$ as little as possible, and to try to restore closure under the rules (simply adding $\mathsf{Op}\alpha$ to $\mathcal{TS}_{\mathsf{t_{now}}}(i)$ might result in rules which did not fire before to fire, and thus this simple extension is not closed under the program rules). Here is a simple way to realize **Modified_Set** by using the function **ComputeTSS**.

(1) First, set *Seen* $:= \emptyset$.
(2) Then, define $\mathcal{TS}^{new}_{\mathsf{t_{now}}}$ to be exactly like $\mathcal{TS}_{\mathsf{t_{now}}}$, except that $\mathsf{Op}\alpha \in \mathcal{TS}^{new}_{\mathsf{t_{now}}}(i)$. Then if $\mathsf{Op}\alpha \in \mathcal{TS}_{\mathsf{t_{now}}}(j)$ and there is a program rule in $\mathcal{TP}$ with head

$\mathsf{Op}\alpha$: tic such that both j and i are solutions of tic, then make sure that $\mathsf{Op}\alpha \notin \mathcal{TS}^{new}_{\mathsf{t_{now}}}(j)$. This condition ensures minimality of the computed temporal status set: without it, we can generate larger and larger temporal status sets just by adding unnecessary atoms. We are interested in generating rational temporal status sets (rather than all feasible status sets) because the latter have various desired epistemic properties (see (Eiter, Subrahmanian, and Pick 1999) for details)—for instance, they do not perform more actions than needed.

(3) We can then apply **ComputeTSS**($\mathcal{TS}^{new}_{\mathsf{t_{now}}} \cup \mathsf{ic}\text{-}\mathcal{TS}$,$\mathcal{TP}$,*Seen*) (here we slightly abuse notation and identify $\mathcal{TS}^{new}_{\mathsf{t_{now}}}$ with the obviously induced interval constraint status set) and get a new temporal status set closed under the rules of $\mathcal{TP}$.

(4) Finally, we have to check that the computed status set does not contain $\mathcal{TS}_{\mathsf{t_{now}}}$ (in that case, it would not be rational). If it does contain $\mathcal{TS}_{\mathsf{t_{now}}}$, then we set $Seen := Seen \cup \{\mathcal{TS}_{\mathsf{t_{now}}}\}$ and go back to (2).

Note that this algorithm may also produce "No", namely via the function **ComputeTSS**. It is easy to see that the whole procedure correctly satisfies the requirements laid out in Definition 8.17.

Theorem 8.22 (Algorithm 8.1 is correct and complete)
Algorithm 8.1 generates a feasible temporal status set (if one exists).

Proof:

Suppose Algorithm 8.1 returns $\mathcal{TS}_{\mathsf{t_{now}}}$. In this case, $\mathcal{TS}_{\mathsf{t_{now}}}$ is compatible via step (5)(a), and feasible via step 5(b).

Conversely, suppose Algorithm 8.1 returns "No." In this case, we know that **FindCompTSS** returned "No" which means that it was unable to find a minimal temporal status set compatible with *pre*-ic-$\mathcal{TS}$. But this means that all temporal status set compatible with *pre*-ic-$\mathcal{TS}$ are in *Seen* which means none of them is feasible. ∎

Remark 8.4 A slight modification of the **ComputeTSS** subroutine allows for the computation of *rational* status sets. Namely if we require the computed set of Definition 8.16 to be *minimal while being closed under $\mathcal{TP}$, and compatible with ic-$\mathcal{TS}$*, and change Algorithm 8.3 accordingly, then Algorithm 8.1 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 **Modified_Set** and **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 8.5 Note that we required a temporal status set to satisfy $\mathcal{TS}_{\mathsf{t_{now}}}(i) \neq \emptyset$ 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: $\mathbf{P}\alpha \in \mathcal{TS}_{\mathsf{t_{now}}}(i)$.

It is obvious that in general, the complexity introduced by the two subroutines **ComputeTSS** and **Modified_Set** 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 functions **ComputeTSS** and **Modified_Set** can be realized with low overall complexity. To define such classes syntactically, we have to make sure that situations as described in Example 8.13 are excluded.

9 Applications of taps

In this section, we present three multiagent applications of temporal agent programs involving time. The first describes how different agents The first describes how different agents reconcile existing commitments to a group with new requests/opportunities. The second studies how different agents may negotiate with one another to reduce pollution. The third deals with the well known contract net framework.

9.1 Intention Reconciliation by Collaborative Agents

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; Glass and Grosz 1999) 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 1999) and agent developers can choose to use it if their agent needs to perform intention reconciliation of the sort described in SPIRE.

- utility : $\mathit{current\_income}(\mathtt{A}, \mathtt{Rep}) \rightarrow \mathtt{Real}$

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 calls listed below 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 appendix B.1 and Appendix B.1. 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: $\mathbf{P}\mathit{addAction}(\mathtt{Y}, \mathtt{T}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow$
$(\,\mathbf{in}(\langle Y, T\rangle, \mathsf{msgbox} : \mathit{gather\_outside\_req}())\ \ \&$
$\mathbf{in}(\mathtt{null}, \mathsf{schedule} : \mathit{check}(\mathtt{T}))$
$) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$

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

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.

There are situations in which there is a need to reduce air pollution for a short time period in a specific geographical area because of external factors (such as weather). Plants have to reduce emission of several polluting substances by some percentage. The solution that is currently implemented is simply the reduction of emission by each plant by the appropriate percentage. But, it may be less costly for one plant to reduce a specific substance emission than it is for another plant. Therefore, plants in the given geographical area can reach a beneficial agreement with respect to emission of what substances and by which percentage each of them must reduce emission.

Several countries have been using human bargaining and auctions for distribution of long term emission allowances for efficient control of pollution (e.g.,(EPA 1999)). For short term reductions which should be agreed upon in a short time period, we propose that the plants be represented by automated agents that will negotiate to reach an agreement on the pollution reduction. Each agent would like to maximize the profit of its plant, but the total emission of each substance must not exceed the maximal permissible emission of this substance according to the authorities' instructions.

We propose to use the model of alternating offers, in which all agents negotiate to reach an agreement which specifies the reduction in each substance for each of the plants. An agent may opt out of the negotiations and choose to use the equal percentage reduction instructed by the authorities.

The protocol of alternative offers involves several iterations until an agreement is reached. During even time periods, one agent makes an offer, and in the next time period each of the other agents may either accept the offer, reject it, or opt out of the negotiation. If an offer is accepted by all agents, then the negotiation ends, and this offer is implemented. If at least one of the agents opts out of the negotiation, then the plants will reduce their pollution according to the equal percentage reduction instructed by the authorities. If no agent has chosen "Opt", but at least one of the agents has rejected the offer, the negotiation proceeds to the next time period, another agent makes a counter-offer, the other agents respond, and so on.

The problem described here is similar to the data allocation in the multi-servers environment problem discussed in (Schwartz and Kraus 1997). The details of the strategies that are in perfect-equilibrium which are specified in (Schwartz and Kraus 1997) are also in perfect equilibrium in our domain. As in the data allocation domain, the perfect equilibrium leads to an agreement with no delay. We demonstrate here how a designer of an agent can program these strategies in `tap`.

We assume that there are $\#agents$ agents (where $\#agents > 2$) who are numbered $0, 1, 2, ..., \#agents - 1$, and that the temporal agent program specified below is for agent i. Furthermore, we assume that each agent has an evaluation function V^i representing its preferences. It is a function from the set of all possible allocations of the emission reductions to the real numbers. We denote by $\mathcal{V}$, the set of evaluation functions of all the agents.

We associate with each agent, a specialized package called $\mathfrak{nego}$ that supports a function, *findAlloc* which takes as input, a set of *substances* to be reduced, the equal percentage reduction instructed by the authorities, *equal_reduction*, the valuation functions of the agents and a maximization criteria (e.g., *max_sum*) which returns an allocation that is better for all agents than opting out and maximizes the given criteria. This can be done, for example, by a package supporting simplex if the evaluation functions of the agents are linear functions.

$\mathfrak{nego}$ also supports the code call $\mathfrak{nego}: endNego(\texttt{Messages}) \rightarrow \texttt{Boolean}$. *endNego* gets as an input a set of answers or an offer or an empty set and returns **true** if the negotiations ends and **false** otherwise.

The *msgbox* package is discussed in Appendix B.1. We now extend it with the following code calls:

- $\mathfrak{msgbox}: gatherAnswers(\texttt{NumAgents}, \texttt{Time}) \rightarrow \texttt{SetofAnswers}$
  gatherAnswers gets as input a number of agents and a time point and returns the answers sent by these agents at the specified time point.
- $\mathfrak{msgbox}: doneBroadcast(\texttt{Time}) \rightarrow \texttt{Boolean}$
  doneBroadcast gets as input a time point, and returns **true** if the agent has broadcast the required answer at this time period and false otherwise.
- $\mathfrak{msgbox}: gatherProposal(\texttt{Time}) \rightarrow \texttt{SetofProposals}$
  gatherProposal gets as input a time point and returns the proposal made by the relevant agent at the specified time point.

The following rules can be used to program the agents in the strategic negotiation.

r1: $\mathbf{O}\,broadcast(\texttt{Offer}):[\texttt{X}_{\textsf{now}}, \texttt{X}_{\textsf{now}}] \leftarrow$
$\quad\mathbf{Do}\,processAnswers():[\texttt{X}_{\textsf{now}} - 1, \texttt{X}_{\textsf{now}} - 1]$ &
$\quad(\ \mathbf{in}(\texttt{X1}, \mathfrak{msgbox}: gatherAnswers(\#\texttt{agents} - 1, \texttt{X}_{\textsf{now}} - 1))$ &
$\quad\ \mathbf{in}(\mathbf{false}, \mathfrak{nego}: endNego(\texttt{X1}))$ &
$\quad\ \mathbf{in}(\texttt{Offer}, \mathfrak{nego}: findAlloc(\texttt{subst}, \mathcal{V}, \texttt{equal\_reduction}, \texttt{max\_sum}))$ &
$\quad\ \mathbf{in}(0, \mathfrak{math}: remainder(\texttt{X}_{\textsf{now}}, 2))$ &
$\quad\ \mathbf{in}(\texttt{i}, \mathfrak{math}: remainder(\texttt{X}_{\textsf{now}}/2, \#\texttt{agents}))$
$\quad):[\texttt{X}_{\textsf{now}}, \texttt{X}_{\textsf{now}}]$

If the negotiations have not ended in the preceding time period, then the agent whose turn it is to make an offer is obliged to make an offer. The offer is the one that maximizes the sum of the utilities among the allocations that yield each agent at least its opting out utility (i.e., the utility from following the authorities requirements.) The desired proposal is determined using the *findAlloc* function.

In the protocol described above, offers are made only in even time periods. Therefore, the agent is obliged to broadcast a proposal only when the time period is even. In particular, agent i should make offers in time periods, $2i, 2(i + \#agents, 2(i + 2 \times \#agents$, etc.

The agent doesn't need to send a proposal if the negotiation ends. This is determined by processing the answers of the previous time period which is gathered using the *gatherAnswers* code call.

r2: $\mathbf{O}\,broadcast(\texttt{"yes"}):[\mathtt{X_{now}}, \mathtt{X_{now}}] \leftarrow$
$\quad\mathbf{Do}\,processAnswers():[\mathtt{X_{now}} - 1, \mathtt{X_{now}} - 1]$ &
$\quad\mathbf{in}(\texttt{Offer}, \mathtt{msgbox}: gatherProposal()):[1, \mathtt{X_{now}} - 1]$ &
$\quad(\,\mathbf{in}(\mathbf{false}, \mathtt{nego}: endNego(\texttt{Offer}))$ &
$\quad\;\mathbf{in}(\texttt{Offer}, \mathtt{nego}: findAlloc(\texttt{subst}, \mathcal{V}, \texttt{equal\_reduction}, \texttt{max\_sum}))$ &
$\quad\;\mathbf{in}(1, \mathtt{math}: remainder())$
$\quad):[\mathtt{X_{now}}, 2]$ &
$\quad\mathbf{not\_in}(\mathtt{i}, \mathtt{math}: remainder((\mathtt{X_{now}} - 1)/2, \#\mathtt{agents})):[\mathtt{X_{now}}, \mathtt{X_{now}}]$

The agent answers "yes" if it received a proposal that maximizes the sum of the utilities among the allocations that yield each agent at least its opting out utility. Answers are given only in odd time points and an agent i should answer only when it wasn't its turn to make an offer, i.e. if the remainder of the the division of $(\mathtt{X_{now}} - 1)/2$ by $\#agents$ is not i. This is relevant only if the negotiations haven't ended yet, i.e., the code call *endNego* returns **false**, given the offer made in the previous time point (i.e., `Offer`).

r3: $\mathbf{O}\,opt:[\mathtt{X_{now}}, \mathtt{X_{now}}] \leftarrow$
$\quad\mathbf{Do}\,processAnswers():[\mathtt{X_{now}} - 1, \mathtt{X_{now}} - 1]$ &
$\quad\mathbf{in}(\texttt{Offer}, \mathtt{msgbox}: gatherProposal()):[1, \mathtt{X_{now}} - 1]$ &
$\quad(\;\mathbf{in}(\mathbf{false}, \mathtt{nego}: endNego(\texttt{Offer}))$ &
$\quad\;\;\mathbf{in}(\texttt{Offer1}, \mathtt{nego}: findAlloc(\texttt{subst}, \mathcal{V}, \texttt{equal\_reduct}, \texttt{max\_sum}))$ &
$\quad\;\;\neq (\texttt{Offer}, \texttt{Offer1})$ &
$\quad\;\;\mathbf{in}(1, \mathtt{math}: remainder())$
$\quad):[\mathtt{X_{now}}, 2]$ &
$\quad\mathbf{not\_in}(\mathtt{i}, \mathtt{math}: remainder((\mathtt{X_{now}} - 1)/2, \#\mathtt{agents})):[\mathtt{X_{now}}, \mathtt{X_{now}}]$

The agent opts out of the negotiations if it received a proposal which does not maximize the sum among the allocations that yield each of at least its opting out utilities. The only difference between r3 and r2 is that in r3 `Offer1` is not equal to `Offer` and according to the specifications of the strategies that are in equilibrium, it should opt out of the negotiation.

r4: $\mathbf{F}\,processAnswers():[\mathtt{X_{now}}, \mathtt{X_{now}}] \leftarrow$
$\quad\mathbf{in}(\mathbf{false}, \mathtt{msgbox}: doneBroadcast(\texttt{T1})):[\mathtt{X_{now}}, \mathtt{X_{now}}]$

An agent is forbidden to read the answer of other agents before it broadcasts its answers.

r5: $\mathbf{O} \mathit{processAnswers}() : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow$
$\quad \mathbf{in}(1, \mathtt{math} : \mathit{remainder}(\mathtt{X}_{\mathsf{now}}, 2)) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]$

An agent is obliged to process the answers it obtained from other agents.

9.3 Delivery Agents in Contract Net Environments

In this section, we show how agent programs may be used to simulate a contract net problem. We consider the problem described in (Sandholm 1993) where there are several companies each having a set of geographically dispersed *dispatch centers* which ship the companies' products to other locations.

Each dispatch center is responsible for the deliveries initiated by certain factories and has a certain number of vehicles to take care of deliveries. The geographical areas of operation of the dispatch centers overlap considerably. This enables several centers to handle a given delivery. Every delivery has to be included in the route of some vehicle. The problem of each dispatch center is to find routes for its vehicles minimizing its transportation costs and maximizing its benefits.

Sandholm (1993) suggests that in solving this problem, each dispatch center—represented by one agent—first solves its local routing problem. After that, an agent can potentially negotiate with other dispatch agents to take on some of their deliveries or to let them take on some of its deliveries for a dynamically negotiated fee. Sandholm presents a formal model of the bidding and award process which is based on marginal cost calculations. Here, we demonstrate how to program such agents in *IMPACT*. We use the terms "center" and its associated agent interchangeably.

We will use several specialized packages and functions in building such agents. The optimizer package provides functions to compute prices, costs, etc. and for identifying deliveries that can be sub-contracted to other centers. It includes the following code calls:

- $\mathtt{optimizer} : \mathit{findDelivery}() \rightarrow \mathtt{Delivery}$
  findDelivery chooses deliveries (from the deliveries handled by the center) which will be announced to other centers in order to get bids from them. In Sandholm's implementation, deliveries were chosen randomly from those deliveries whose destination lies in another center's main operation area.
- $\mathtt{optimizer} : \mathit{maxPrice}(\mathtt{Del}) \rightarrow \mathtt{Real}$
  maxPrice takes a delivery as input and returns the maximum price of the announcement (i.e., the maximum price the agent is willing to pay). It uses a heuristic approximation of the marginal cost saved if the delivery is removed

from the routing solution of the agent. An example of such a heuristic is described in (Sandholm 1993).

- optimizer : *price*(Del) → Real
  price takes a delivery as input and returns the price that the delivery would cost if done by this server.
- optimizer : *feasible*(Del) → Boolean
  feasible takes a delivery as input and returns **true** if it is feasible for the agent to perform the delivery, given its current and pending commitments.
- optimizer : *getTime*(Del) → Time
  getTime takes a delivery as input and returns the time until which the agent is willing to accept bids for an announcement for this delivery.
- optimizer : *gatherAnnounceExpired*(Time) → ListofAnnouncements
  gatherAnnounceExpired takes a time point as input and finds the announcements whose waiting time has expired.
- optimizer : *sentBid*(Id) → Boolean
  sentBid takes an announcement Id as input and returns **true** if the agent has already sent a bid for this announcement and **false** otherwise.
- optimizer : *verifyAward*(A) → Boolean
  verifyAward takes an award as input and returns **true** if the details of the award are the same as in the bid sent by the agent.
- optimizer : *annTime*(Time) → Boolean
  annTime takes a time point as input and returns **true** if the agent should consider sending announcements at the specified time point.
- optimizer : *bidTime*(Time) → Boolean
  annTime takes a time point as input and returns **true** if the agent should consider sending bids at the specified time point.

The gis package includes functions which provide information on the geographical locations of the dispatch centers. Such a package could well be any commercial geographic information system. The gis package supports the following code call:

- gis : *centers_located*(Del) → ListofCenters
  centers_located takes a delivery as input and returns centers (other than the agent's center) whose associated operations area cover the destination of the delivery.

The msgbox package is extended with the following code calls:

- msgbox : *getId*() → String
  getId provides a unique name for an announcement. It could be implemented as a combination of the agent's identification number and a counter which is increased each time *getId* is called.
- msgbox : *gatherAnnounce*(Time) → ListofAnnouncements
  gatherAnnounce takes a time point as input and returns announcements

which were received from other agents and their expiration time is before the specified time. The fields of an announcement are: its identification number (Id), the delivery (Del), the sender (Center), the expiration time (Time) and the maximal price (C_max).

- msgbox : *gatherBids*(Id) → ListofBids
  gatherBids takes an identification number of an announcement as input and returns all bids that were sent as a response to the specified announcement.
- msgbox : *gatherAwards*() → ListofAnnouncements
  gatherAwards returns all the awarding messages the agent received.

The following is the agent program that solves the contract net problem.

$$
\begin{array}{l}
\textbf{r1: } \mathbf{O}\mathit{send\_ann}(\mathtt{Center}, \mathtt{Id}, \mathtt{Del}, \mathtt{C\_max}, \mathtt{Time}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow \\
\quad (\,\mathbf{in}(\mathbf{true}, \mathsf{optimizer} : \mathit{annTime}(\mathtt{X}_{\mathsf{now}}))\ \& \\
\quad\ \ \mathbf{in}(\mathtt{Del}, \mathsf{optimizer} : \mathit{findDelivery}())\ \& \\
\quad\ \ \mathbf{in}(\mathtt{Center}, \mathsf{gis} : \mathit{centers\_located}(\mathtt{Del}))\ \& \\
\quad\ \ \mathbf{in}(\mathtt{C\_max}, \mathsf{optimizer} : \mathit{maxPrice}(\mathtt{Del}))\ \& \\
\quad\ \ \mathbf{in}(\mathtt{Time}, \mathsf{optimizer} : \mathit{getTime}(\mathtt{Del}))\ \& \\
\quad\ \ \mathbf{in}(\mathtt{Id}, \mathsf{msgbox} : \mathit{getId}()) \\
\quad) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]
\end{array}
$$

An agent should consider sending announcements if the optimizer indicates (using *annTime*) that $\mathtt{X}_{\mathsf{now}}$ is an appropriate time to do so. At these time points, an agent is obliged to send an announcement for the deliveries chosen by the optimizer to all the centers whose operation areas include the destination of the delivery. The delivery is chosen by the optimizer, who also computes the maximal price the agent is willing to pay.

$$
\begin{array}{l}
\textbf{r2: } \mathbf{O}\mathit{send\_bid}(\mathtt{A.Center}, \mathtt{A.Id}, \mathtt{A.Del}, \mathtt{C\_bid}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow \\
\quad (\mathbf{in}(\mathbf{true}, \mathsf{optimizer} : \mathit{bidTime}(\mathtt{X}_{\mathsf{now}}))\ \& \\
\quad\ \ \mathbf{in}(\mathtt{A}, \mathsf{msgbox} : \mathit{gatherAnnounce}(\mathtt{X}_{\mathsf{now}}))\ \& \\
\quad\ \ \mathbf{in}(\mathbf{false}, \mathsf{optimizer} : \mathit{sentBid}(\mathtt{A.Id}))\ \& \\
\quad\ \ \mathbf{in}(\mathbf{true}, \mathsf{optimizer} : \mathit{feasible}(\mathtt{A.delivery}))\ \& \\
\quad\ \ \mathbf{in}(\mathtt{C\_bid}, \mathsf{optimizer} : \mathit{price}(\mathtt{A.Del}))\ \& \\
\quad\ \ <(\mathtt{C\_bid}, \mathtt{A.C\_max}) \\
\quad) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}]
\end{array}
$$

The agent considers announcements received from other agents. It first verifies that the time of the announcement hasn't passed. Then it verifies that performing the delivery of the announcement is feasible given its other commitments. Finally, it determines the bidding price using the *price* function. If this price it lower than the maximal price the announcer is willing to pay, then it is obliged to send the bid.

$$
\begin{array}{l}
\textbf{r3: } \mathbf{O}\mathit{send\_award}(\mathtt{B2.Center}, \mathtt{B2.Id}) : [\mathtt{X}_{\mathsf{now}}, \mathtt{X}_{\mathsf{now}}] \leftarrow \\
\quad (\mathbf{in}(\mathtt{A}, \mathsf{optimizer} : \mathit{gatherAnnounceExpired}(\mathtt{X}_{\mathsf{now}}))\ \& \\
\quad\ \ \mathbf{is}(\mathtt{B1}, \mathsf{msgbox} : \mathit{gatherBids}(\mathtt{A.Id}))\ \&
\end{array}
$$

$\mathbf{in}(\texttt{B2}, \mathtt{math}: \mathit{min}(\texttt{B1}))$ &
$<$(B2.price,A.C_max) &
Do $\mathit{delete}(\texttt{A.Del})$
$):[\texttt{X}_{\mathsf{now}}, \texttt{X}_{\mathsf{now}}]$

When the time of an announcement has expired, the agent is determining the identity of the successful bidder. These announcements are identified by *gatherAnnounceExpired*. The agent awards the delivery to the center which offers the lowest bid, given that this bid is lower than the maximal price the agent is willing to pay. All the bids are gathered by the *gatherBids* function and *min* returns the one with the minimal price. In addition, the delivery of the announcement is removed from the current deliveries of the center.

r4: $\mathbf{O}\ \mathit{send\_reject}(\texttt{B2.Center}, \texttt{B2.Id}) \leftarrow$
$(\ \mathbf{in}(\texttt{A}, \mathtt{optimizer}: \mathit{gatherAnnounceExpired}(\texttt{X}_{\mathsf{now}}))$ &
$\mathbf{is}(\texttt{B1}, \mathtt{msgbox}: \mathit{gatherBids}(\texttt{A.Id}))$ &
$\mathbf{in}(\texttt{B2}, \mathtt{math}: \mathit{max}(\texttt{B1}))$ &
$>$(B2.price,B.C_max)
$):[\texttt{X}_{\mathsf{now}}, \texttt{X}_{\mathsf{now}}]$

This rule indicates a situation when no award is made. It occurs when the minimal bid is higher than the maximal price the agent is willing to pay. Most of the conditions are the same as in rule r3, except for $\texttt{B.price} >$ $\texttt{C\_max}$.

r5: $\mathbf{O}\mathit{send\_reject}(\texttt{B3.Center}, \texttt{B3.Id}):[\texttt{X}_{\mathsf{now}}, \texttt{X}_{\mathsf{now}}] \leftarrow$
$(\ \mathbf{in}(\texttt{A}, \mathtt{optimizer}: \mathit{gatherAnnounceExpired}(\texttt{X}_{\mathsf{now}}))$ &
$\mathbf{is}(\texttt{B1}, \mathtt{msgbox}: \mathit{gatherBids}(\texttt{A.Id}))$ &
$\mathbf{in}(\texttt{B2}, \mathtt{math}: \mathit{min}(\texttt{B1}))$ &
$\mathbf{in}(\texttt{B3}, \mathtt{msgbox}: \mathit{gatherBids}(\texttt{A.Id}))$ &
$\neq(\texttt{B2},\texttt{B3})$
$):\ [\texttt{X}_{\mathsf{now}}, \texttt{X}_{\mathsf{now}}]$

The agent is obliged to send rejection messages to all unsuccessful bidders. The conditions involved are similar to these of r3. The difference is that the bid (B3) is different from the minimal one (B2).

r6: $\mathbf{O}\mathit{indicate}(\texttt{A.Center}, \texttt{A.Id}, \texttt{A.Del}, \texttt{C\_bid}):[\texttt{X}_{\mathsf{now}}, \texttt{X}_{\mathsf{now}}] \leftarrow$

ctasc**Do** $\mathit{send\_bid}(\texttt{A.Center}, \texttt{A.Id}, \texttt{A.Del}, \texttt{C\_bid})[\texttt{X}_{\mathsf{now}}, \texttt{X}_{\mathsf{now}}]$

The agent must keep the details of its bids. It will be used to verify the correctness of the details of awards.

r7: $\mathbf{O}\mathit{add}(\texttt{A.del}):[\texttt{X}_{\mathsf{now}}, \texttt{X}_{\mathsf{now}}] \leftarrow$
$(\ \mathbf{in}(\texttt{A}, \mathtt{msgbox}: \mathit{gatherAwards}(\texttt{X}_{\mathsf{now}}))$ &
$\mathbf{in}(\mathbf{false}, \mathtt{optimizer}: \mathit{verifyAward}(\texttt{A}))$
$):[\texttt{X}_{\mathsf{now}}, \texttt{X}_{\mathsf{now}}]$

Whenever an agent receives an award based on one of the bids it submitted, it must add delivery of the supply item involved to its schedule, assuming that the details specified in the award are the same as in the bid sent by the agent. Note that in this system, submitted bids are binding.

10 Related Work

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 logics 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 in this book.
- 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) that use a translation of their interval-based temporal logic (LLP) into constraint logic (CPL) to obtain an efficient deduction system.
- Most traditional AI planning frameworks assume that an action's effects are realized only after the entire action is successfully executed (Hendler,

Tate, and Drummond 1990; Hendler and McDermott 1995). We proposed a mechanism that allows an agent designer to specify intermediate effects.

- 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.

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.

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 his colleagues (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.

Kraus, Sycara, and Evenchik (1998) presented a logical model of the mental states of agents based on a representation of beliefs, desires, intentions, and goals and use it as the basis for the development of automated negotiators. As in our model, they also use explicit time structures and their modal operators G (for goal), Int (for intention), and Do have some similarities to our obligation, permission and do operators. However, their semantics is very different from ours. They use a *minimal structures* Chellas (1980) style semantics for each of their modal operators which leads to a set of axioms that are not appropriate for our agents. In addition, they require a fully specified history and use a discrete "point-based" representation of time, while we use an interval-based representation of discrete time.

An interesting line of research begun in (Schroeder, de Almeida Mora, and Pereira 1997) showed how *extended logic programming* may be used to specify the behavior of a diagnostic agent. Their architecture supports cooperation between multiple diagnostic agents. Issues of interest arise when conflicting diagnoses are hypothesized by different agents. This problem is tackled by using previous work on belief revision (Alferes and Pereira 1996) and and inference mechanism based on the REVISE algorithm (d'Inverno, Kinny, Luck, and Wooldridge 1997) for eliminating contradictions. Another logic programming based framework called CaseLP has been proposed in (Martelli, Mascardi, and Zini 1998; Martelli, Mascardi, and Zini 1997). It is intended to be used for multiagent applications by building on top of existing software. In a similar vein, (Kowalski and Sadri 1999) have developed an abductive framework within which logic programs can be used to support both rational and reactive agent reasoning. They present an agent cycle, including both these facets, and develop an abductive proof procedure. As in our work, agents have states, and states are changed by the agents' actions, and the behavior of an agent is encoded through rules.

Singh (1998) studies closely the problem of actions with temporal duration. He presents a general formalization of actions with several important properties which include: (1) actions have different durations; (2) the actions may be performed concurrently by different agents; (3) the time can be either continuous or discrete; (4) the model allow branching into the future. We allow property (1), but do not handle explicitly (2-4) because our goal has been different than Singh's. While Singh developed a model that can serve as a basis for a system that can be used for reasoning about multiple agents activities, we focus on the development of a framework to program temporal agents and to provide the agents with algorithms to decide what to do in a given time point. It will be interesting to study how can Singh's framework can be used to reason on our agents.

11 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, the in 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. We further show that certain important applications, viz. reconciliation of intentions and conflicts between collaborating agents, strategic multiagent negotiations, and contract net computations, may be neatly expressed via the notion of temporal agent programs proposed in this paper.

References

Alferes, J. J. and L. M. Pereira (1996). Reasoning with Logic Programming. In *Springer-Verlag Lecture Notes in AI*, Volume 1111.

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.

Arens, Y., C. Y. Chee, C.-N. Hsu, and C. Knoblock (1993). Retrieving and Integrating Data From Multiple Information Sources. *International Journal of Intelligent Cooperative Information Systems 2*(2), 127–158.

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.

Baldoni, M., L. Giordano, A. Martelli, and V. Patti (1998). An Abductive Proof Procedure for Reasoning about Actions in Modal Logic Programming. In *Workshop on Non Monotonic Extensions of Logic Programming at ICLP '96*, Volume 1216 of *Lecture Notes in AI*, pp. 132–150. Springer-Verlag.

Baral, C. and M. Gelfond (1993, August). Representing Concurrent Actions in Extended Logic Programming. In R. Bajcsy (Ed.), *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, pp. 866–871. Morgan Kaufman.

Baral, C. and M. Gelfond (1994). Logic Programming and Knowledge Representation. *Journal of Logic Programming 19/20*, 73–148.

Baral, C., M. Gelfond, and A. Provetti (1995). Representing Actions I: Laws, Observations, and Hypothesis. In *AAAI '95 Spring Symposium on Extending Theories of Action*.

Baral, C. and J. Lobo (1996). Formal Characterization of Active Databases. In D. Pedreschi and C. Zaniolo (Eds.), *Workshop of Logic on Databases (LID '96)*, Volume 1154 of *Lecture Notes in Computer Science*, San Miniato, Italy, pp. 175–195.

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.

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

Chellas, B. (1980). *Modal Logic*. Cambridge University Press.

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.

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–.

d'Inverno, M., D. Kinny, M. Luck, and M. Wooldridge (1997). A Formal Specification of dMARS. In *International Workshop on Agent Theories, Architectures, and Languages*, Providence, RI, pp. 146–166.

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. Rogers (1999, May). Heterogeneous Active Agents, III: Polynomially Implementable Agents. Technical Report INFSYS RR-1843-99-07, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria. to appear in AI.

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

EPA, U. S. E. P. A. (1999). Acid rain program. `http://www.epa.gov/acidrain/overview.html`.

Etzioni, O. and D. Weld (1994). A Softbot-Based Interface to the Internet. *Communications of the ACM 37*(7), 72–76.

Fiadeiro, J. and T. Maibaum (1991). Temporal Reasoning over Deontic Specifications. *Journal of Logic and Computation 1*(3), 357–395.

Gelfond, M. and V. Lifschitz (1993). Representing Actions and Change by Logic Programs. *Journal of Logic Programming 17*, 301–322.

Gelfond, M. and V. Lifschitz (1998). Action languages. *Electronic Transactions on AI 6*(16).

Glass, A. and B. Grosz (1999). Socially conscious decision-making. Technical Report 11-99, Harvard University, Cambridge, MA.

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.

Hendler, J. and D. McDermott (1995). Planning: What it could be, An introduction to the special issue on Planning and Scheduling. *Artificial Intelligence 76*(1–2), 1–16.

Hendler, J., A. Tate, and M. Drummond (1990). Systems and techniques: AI planning. *AI Magazine 11*(2), 61–77.

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 (Eds.) (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.

Kowalski, B. and F. Sadri (1999). From LP towards multi-agent programs. *Annals of Mathematics and AI 25*(3-4), 391–419.

Kraus, S. (1997). Negotiation and cooperation in multi-agent environments. *Artificial Intelligence journal 94*(1-2), 79–98.

Kraus, S., K. Sycara, and A. Evenchik (1998). Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence 104*(1-2), 1–69.

Labrou, Y. and T. Finin (1994). A Semantics Approach for KQML – A General Purpose Communications Language for Software Agents. In *Proceedings of the International Conference on Information and Knowledge Management*, pp. 447–455.

Labrou, Y. and T. Finin (1997). Semantics for an Agent Communication Language. In *International Workshop on Agent Theories, Architectures, and Languages*, Providence, RI, pp. 199–203.

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.

Lifschitz, V. (1997). Two components of an action language. *Annals of Math and AI 21*, 305–320.

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

Maes, P. (1994). Agents that Reduce Work and Information Overload. *Communications of the ACM 37*(7), 31–40.

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

Martelli, M., V. Mascardi, and F. Zini (1997). CaseLP: a Complex Application Specification Environment based on Logic Programming. In *Proceedings of ICLP'97 Post Conference Workshop on Logic Programming* *and Multi-Agents*, Leuven, Belgium, pp. 35–50.

Martelli, M., V. Mascardi, and F. Zini (1998). Towards Multi-Agent Software Prototyping. In *Proceedings of The Third International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM98)*, London, UK, pp. 331–354.

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.

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

Nilsson, N. (1986). Probabilistic Logic. *Artificial Intelligence 28*, 71–87.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann.

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.

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

Sandholm, T. (1993, July). An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations. In *Proceedings of the 11th National Conference on Artificial Intelligence*, Washington, DC, pp. 256–262. AAAI Press/MIT Press.

Schroeder, M., I. de Almeida Mora, and L. M. Pereira (1997). A Deliberative and Reactive Diagnosis Agent based on Logic Programming. In M. W. J.P. Muller and N. Jennings (Eds.), *Intelligent Agents III: Lecture Notes in Artificial Intelligence Vol. 1193*, pp. 293–307. Springer-Verlag.

Schwartz, R. and S. Kraus (1997). Bidding Mechanisms for Data Allocation in Multi-Agent Environments. In *International Workshop on Agent Theories, Architectures, and Languages*, Providence, RI, pp. 56–70.

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.

Smith, R. and R. Davis (1983). Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence 20*, 63–109.

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. S. (1994). Amalgamating Knowledge Bases. *ACM Transactions on Database Systems 19*(2), 291–331.

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.

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.

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

Wiederhold, G. (1993). Intelligent Integration of Information. In *Proceedings of ACM SIGMOD Conference on Management of Data*, Washington, DC, pp. 434–437.

Wooldridge, M. and N. R. Jennings (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Reviews 10*(2), 115–152.

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

A Agent Programs without Time

The following definitions are taken from (Eiter, Subrahmanian, and Pick 1999).

A.1 Feasible, Rational and Reasonable Semantics

Definition A.1 (Status Set) *A* status set *is any set S of ground action status atoms over $\mathcal{S}$. For any operator $\mathsf{Op} \in \{\mathbf{P}, \mathbf{Do}, \mathbf{F}, \mathbf{O}, \mathbf{W}\}$, we denote by $\mathsf{Op}(S)$ the set $\mathsf{Op}(S) = \{\alpha \mid \mathsf{Op}(\alpha) \in S\}$.*

Definition A.2 (Deontic and Action Consistency) *A status set S is called* deontically consistent*, if it satisfies the following rules for any ground action*

α:

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

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

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

Definition A.3 (Deontic and Action Closure) *The* deontic closure *of a status S, denoted $\mathbf{D\text{-}Cl}(S)$, is the closure of S under the rule*

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

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

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

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

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

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

Definition A.4 (Operator $\mathbf{App}_{\mathcal{P},\mathcal{O}_{\mathcal{S}}}(S)$) *Suppose $\mathcal{P}$ is an agent program, and $\mathcal{O}_{\mathcal{S}}$ is an agent state. Then, $\boldsymbol{App}_{\mathcal{P},\mathcal{O}_{\mathcal{S}}}(S)$ is defined to be the set of all ground action status atoms A such that there exists a rule in P having a ground instance of the form $r : A \leftarrow L_1, \ldots, L_n$ such that*

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

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

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

Definition A.5 (Feasible Status Set) *Let $\mathcal{P}$ be an agent program and let $\mathcal{O}_\mathcal{S}$ be an agent state. Then, a status set S is a* feasible status set *for $\mathcal{P}$ on $\mathcal{O}_\mathcal{S}$, if the following conditions hold:*

(S1): *(closure under the program rules)* $\boldsymbol{App}_{\mathcal{P},\mathcal{O}_\mathcal{S}}(S) \subseteq S$;
(S2) *(deontic and action consistency)* *S is deontically and action consistent;*
(S3) *(deontic and action closure)* *S is action closed and deontically closed;*
(S4) *(state consistency)* *$\mathcal{O}'_\mathcal{S} \models \mathcal{IC}$, where $\mathcal{O}'_\mathcal{S} = apply(\mathbf{Do}(S), \mathcal{O}_\mathcal{S})$ is the state which results after taking all actions in $\mathbf{Do}(S)$ on the state $\mathcal{O}_\mathcal{S}$.*

Definition A.6 (Groundedness; Rational Status Set) *A status set S is* grounded*, if there exists no status set $S' \neq S$ such that $S' \subseteq S$ and S' satisfies conditions $(S1)$–$(S3)$ of a feasible status set.*

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

Definition A.7 (Reasonable Status Set) *Let $\mathcal{P}$ be an agent program, let $\mathcal{O}_\mathcal{S}$ be an agent state, and let S be a status set.*

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

B Code Calls and Actions in the Rescue Example

B.1 The Message Box Domain

We assume that each agent's associated software code includes a special type called `Msgbox` (short for message box). The message box is a buffer that may be filled (when it sends a message) or flushed (when it reads the message) by the agent.

The msgbox operates on objects of the form

$$(\imath/\mathtt{o}, \texttt{"}\mathrm{src}\texttt{"}, \texttt{"}\mathrm{dest}\texttt{"}, \texttt{"}\mathrm{message}\texttt{"}, \texttt{"}\mathrm{time}\texttt{"}).$$

The parameter $\imath/\mathtt{o}$ signifies an incoming or outgoing message respectively. The variable "src" specifies the originator of the message whereas "dest" specifies the destination. The "message" is a table consisting of triples of the form ("varName", "varType", "value") where "varName" is the name of the variable, "varType" is the type of the variable and the "value" is the value of the variable in string format. Finally, "time" denotes the time at which the message was sent.

We will assume that the agent has the following functions that are integral in managing this message box.

- *sendMessage*(<*source_agent*>, <*dest_gent*>, <*message*>): This causes a quintuple ($\mathtt{o}$, "src", "dest", "message", "time") to be placed in `Msgbox`. The parameter `o` signifies an outgoing message. When a call of

$$\mathit{sendMessage}(\texttt{"}\mathrm{src}\texttt{"}, \texttt{"}\mathrm{dest}\texttt{"}, \texttt{"}\mathrm{message}\texttt{"})$$

  is executed, the state of `Msgbox` changes by the insertion of the above quintuple denoting the sending of a message from the source agent `src` to a given Destination agent `dest` involving the message body "message".
- *getMessage*(<*src*>): This causes a collection of quintuples

$$(\mathtt{i}, \texttt{"}\mathrm{src}\texttt{"}, \texttt{"}\mathrm{agent}\texttt{"}, \texttt{"}\mathrm{msg}\texttt{"}, \texttt{"}\mathrm{time}\texttt{"})$$

  to be read from `Msgbox`. The i signifies an incoming message. Note that all messages from the given source to the agent `agent` whose message box is being examined, are returned by this operation. "time" denotes the time at which the message was received.
- *timedGetMessage*(<*op*>, <*valid*>): This causes the collection of all quintuples *tup* of the form $tup =_{def} (\mathtt{i}, <\!src\!>, <\!agent\!>, <\!message\!>, \mathtt{time})$ to be read from `Msgbox`, such that the comparison `tup.time` *op valid* is true,

where *op* is required to be any of the standard comparison operators $<$, $>$, $\leq$, $\geq$, or $=$.

- $getVar(<mssgId>, <varName>)$: This functions searches through all the triples in the "message" to find the requested variable. First, it converts the variable from the string format given by the "value" into its corresponding data type which is given by "varType". If the requested variable is not in the message determined by the "MssgId", then an error string is returned.

B.2 The `comc` *agent*

In addition to the actions and functions mentioned above, the `comc` agent apply the following functions:

- Returns the status of `Vehicle`:
  `comc` : *checkStatus*(`Vehicle`) $\rightarrow$ `Status`
- Returns triplets specifying a vehicle, a status and a time point, indicating that the specified vehicle should be notified of the specified status in the given time:
  `comc` : *vehicle_to_be_notified*() $\rightarrow \langle String, String, Time \rangle$

B.3 Truck Agents

This agent provides and manages truck schedules using routing algorithms.

B.3.1 Functions

(1) Return the current location of the truck
 `truck` : *location*() $\rightarrow$ `2DPoint`
(2) Returns the current fuel level of the truck in gallons.
 `truck` : *fuelLeval*() $\rightarrow$ `Galons`
(3) Returns **true** if the `truck`'s tank is empty, and **false** otherwise.
 `truck` : *tank_empty*() $\rightarrow$ `Boolean`
(4) Returns roads and areas that are restricted.
 `msgbox` : *gatherWarning*() $\rightarrow$ `String`
(5) Returns the truck's load at the specified time.
 `truck` : *load*(`T`) $\rightarrow$ `String`

B.3.2 Actions

The action *order_item*() of the `truck` agent may be described with the following components;

- **Name:** *order_item*(`Itm`)
- **Schema:** (`String`)
- **Pre:**
- **Dur:** $\{1\}$
- **Tet:**

$$\begin{aligned}
\text{1st arg}: &\quad \mathsf{rel}:\{1\}\\
\text{2nd arg}: &\ \{\ \mathbf{in}(\langle \mathtt{Itm}, \mathtt{X}_{\mathsf{now}}\rangle, \mathtt{msgbox}: \mathit{supplier_to_be_notified}())\}\\
\text{3rd arg}: &\quad \{\}
\end{aligned}$$

The **Tet** part says that the `truck` agent requests from its supplier to deliver the needed item.

B.4 Helicopter Agents

In addition to the functions mentioned above, the `heli` agent apply the following functions:

(1) Returns the items the `heli` agent needs in case of emergency
`heli`: *emergency_items*() $\rightarrow$ `String`

(2) Return the minimal amount of the specified item that the `heli` agent needs in its inventory. `heli`: *minimal_inventory*(`Item`) $\rightarrow$ `Integer`