A Rewriting Framework for Activities Subject to Regulations

Max Kanovich[1], Tajana Ban Kirigin[2], Vivek Nigam[3], Andre Scedrov[4], Carolyn Talcott[5], and Ranko Perovic[6]

1 **Queen Mary, University of London, UK, e-mail:** `mik@eecs.qmul.ac.uk`
2 **University of Rijeka, HR, e-mail:** `bank@math.uniri.hr`
3 **Ludwig-Maximilians-Universität, Germany, e-mail:** `vivek.nigam@ifi.lmu.de`
4 **University of Pennsylvania, USA, e-mail:** `scedrov@math.upenn.edu`
5 **SRI International, USA, e-mail:** `clt@csl.sri.com`
6 **Senior Clinical Trial Specialist, e-mail:** `perovicrankomd@gmail.com`

Abstract

Activities such as clinical investigations or financial processes are subject to regulations to ensure quality of results and avoid negative consequences. Regulations may be imposed by multiple governmental agencies as well as by institutional policies and protocols. Due to the complexity of both regulations and activities there is great potential for violation due to human error, misunderstanding, or even intent. Executable formal models of regulations, protocols, and activities can form the foundation for automated assistants to aid planning, monitoring, and compliance checking. We propose a model based on multiset rewriting where time is discrete and is specified by timestamps attached to facts. Actions, goal and critical states may be constrained by means of relative time constraints. Moreover, actions may have non-deterministic effects, *i.e.*, they may have different outcomes whenever applied. We demonstrate how specifications in our model can be straightforwardly mapped to the rewriting logic language Maude, and how one can use existing techniques to improve performance. Finally, we also determine the complexity of the plan compliance problem, that is, finding a plan that leads from an initial state to a desired goal state without reaching any undesired critical state. We consider all actions to be balanced, *i.e.*, their pre and post-conditions have the same number of facts. Under this assumption on actions, we show that the plan compliance problem is PSPACE-complete when all actions have only deterministic effects and is EXPTIME-complete when actions may have non-deterministic effects.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity; F.4 Mathematical Logic and Formal Languages; J3. Life and Medical Sciences

Keywords and phrases Multiset Rewrite Systems, Collaborative Systems, Applications of Rewrite Systems, Clinical Investigations, Maude

1 Introduction

While carrying out a clinical investigation (CI)—that is, a set of procedures in medical research and drug development, to test a new drug or other intervention on human subjects, it is important that conclusive data is collected and that the health of the subjects participating in the CI is not compromised. In order to collect the most conclusive data, for instance, drug samples have to be taken and all the necessary tests have to be carried out in well defined periods of time. Moreover, since these experiments might compromise the health of subjects, CIs are rigorously regulated by policies elaborated by governmental agencies such as the Food and Drug Administration (FDA) [12]. These regulations require prompt action whenever a serious and unexpected problem with any subject is reported. In the current state of affairs, there is little to almost no automation in the management of CIs and therefore the process is prone to human error. As described in [22], there is plenty of room for the use of automated assistants to help reduce human mistakes from happening. For instance, a computer assistant can automatically generate plans that guide the clinical staff on how a CI has to

be carried out. An assistant can also monitor the execution of a CI and signal alarms whenever a deviation to the specification is detected.

This paper proposes a rewriting framework that can be used to specify collaborative systems, such as CIs, and can be used as the foundations for building automated assistants. Our model is an extension with explicit time of the systems used for modeling collaborative systems proposed in [17]. An important feature of our model is that its specifications can be directly written and executed in Maude [6], a powerful tool based on rewrite logic [20]. By using its search mechanisms, Maude can be used to automatically generate plans from a an specification. Moreover, one can rely on all the existing machinery and optimizations implemented in Maude. For instance, since Maude implements rewriting modulo axioms , the execution of systems with commutative and associative constructors, such as those building multisets, is greatly improved when using Maude.

A second feature of our framework is that its specifications can mention time explicitly. Time is often a key component used in policies specifying the rules and the requirements of a collaboration. For a correct collaboration and to achieve a common goal, participants should usually follow strict deadlines and should have quick reactions to some (unexpected) event. For instance, the paragraph 312.32 on Investigational New Drug Application (IND) safety [12] includes explicit time intervals that must be followed in case of any unexpected, serious or life-threatening adverse drug experience: (The emphasis in the text is ours.)

> " (c) IND safety reports
> (1) Written reports –(i) The sponsor shall notify FDA and all participating investigators in a written IND safety report of: (A) Any adverse experience associated with the use of the drug that is both serious and unexpected; [$\cdots$] Each notification shall be made as soon as possible and *in no event later than 15 calendar days* after the sponsor's initial receipt of the information [$\cdots$]
> (2) Telephone and facsimile transmission safety reports. The sponsor shall also notify FDA by telephone or by facsimile transmission of any unexpected fatal or life-threatening experience associated with the use of the drug as soon as possible but *in no event later than 7 calendar days* after the sponsor's initial receipt of the information."

The clause above mentions explicitly two different time intervals. The first one specifies that a detailed safety report must be sent to the FDA within 15 days after a serious and unexpected event is detected, while the second specifies the obligation of notifying FDA of such an event within 7 days.

In order to accommodate explicit time, we attach to facts a *natural number* called *timestamp*. Timestamps can be used in different ways depending on the system being modeled. In the example above, the timestamp t of the fact $Dose(id)@t$ could denote that the subject with anonymous identification number id received a dose at time t. Moreover, we keep track of time by assuming a discrete global time, using the special fact $Time@t$, which denotes that the current time is t. The global time advances by replacing $Time@t$ by $Time@(t+1)$.

Agents change the state of the system by performing actions. In order to specify the type of time requirements illustrated above, a set of *time constraints* may be attached to actions This set acts as a guard of the action. A time constraint is a comparison involving exactly two timestamps, *e.g.*, $T_1 \leq T_2 + 7$ (see Eq. 1).

Besides allowing guards with time constraints, we also allow actions to have non-deterministic effects. In particular, actions are allowed to have a finite number of post-conditions specifying a finite number of possible resulting states. These actions are useful when specifying systems, such as CIs, whose actions may lead to different outcomes, but it is not certain beforehand which one of the outcomes will actually occur. For instance, when carrying out a blood test for the presence of some substance, it is not clear a priori what the test result will be. However, one can classify any result as

either *positive* or *negative*. Depending on this result, one would need to take a different set of future actions. For example, if a the blood test is positive, then one might not be suitable for a participating as a subject in a particular CI, but may be suitable for other CIs. We classify actions that have more than one outcome as *branching actions*.

Finally, in collaborative systems agents collaborate in order to achieve a common goal, but they also should avoid *critical states* that, for example, violate policies. An example of a goal state for CIs would be to collect conclusive data without compromising the health of subjects, while a critical state would be a state that violates the FDA policies that could lead to heavy penalties. In our model, critical, goal and initial states can also mention time explicitly by using time constraints.

This paper's contribution is twofold.

1. We determine the complexity of the *plan compliance* problem [17], that is, the problem of determining whether there is a plan where the collaboration achieves the common goal and in the process no critical state is reached. It has been shown that the plan compliance problem is undecidable in general [15]. However, we get decidability in the important case when all actions are *balanced*, *i.e.*, pre and post-conditions of actions have the same number of facts. Intuitively, this restriction bounds the memory of agents, as they can remember at any point only a bounded number of facts. Additionally, we assume that the facts created by an action, that is, the new facts that appear in its postcondition, can only have timestamps of the form $T + d$, where T is the current global time and d a natural number. Under these two assumptions on actions, we show that (1) the plan compliance problem is PSPACE-complete if no branching actions are allowed and (2) is EXPTIME-complete if branching actions are allowed.
2. We describe a Maude implementation of a small scenario of a clinical investigation visit specified in our rewriting model. We show how the search capabilities of Maude can be used for planning and compliance checking (run-time monitoring). Furthermore, we show how to improve execution performance by using two existing optimizations. The first optimization was proposed in [24], where it is shown how one can reduce search space due to interleavings by merging a collection of (small-step) rules that do not interfere with each other into a single (big-step) rule. The second optimization is to reduce the state space by declaring plan branch formation to be commutative as Maude works on equivalence classes modulo axioms. Our experiments demonstrate that it is possible to reduce in average the number of states to be traversed by a factor of 58 and search time by a factor of 118 when using such optimizations.

Regarding contribution **1** described above, even in the case of balanced actions, we have to deal with the problem that a plan can generate unbounded timestamps T. In particular, the state space is internally infinite since an *arbitrary* number of time advances can occur (as illustrated at the beginning of Section 4). In our previous work [14] we were able to solve a similar unboundedness problem caused by the presence of freshly created objects that are called *nonces* in protocol security literature. However, the solution proposed in [14] is not applicable to the problem of unboundedness of time. As a result, in this paper we have made special precautions in our choice of a novel equivalence relation among states based on the time differences of the timestamps of facts. This allows us to cover all plans of unbounded length caused by uncontrolled time advances, with providing our upper bounds for the timed collaborative systems (Theorem 6). We also show that our new technique introduced in this paper can be combined with the technique introduced in [14] to solve the unboundedness for both time and nonces in timed systems. In our experiments, we used this novel equivalence relation among states.

The paper is organized as follows. Section 2 introduces the formal model for timed collaborative systems called Timed Local State Transition Systems ($TLSTS$) as well as the plan compliance problem described above. (In [22], $TLSTS$es were only mentioned, but not formally introduced.) Section 3 expands the discussion in [22] on how to implement $TLSTS$ specifications in Maude.

Section 4 introduces an equivalence relation between states of the system that allows us to handle the unboundedness of time with finite space. The machinery introduced in this section is used in Section 5 to demonstrate the decidability of the plan compliance problem, and in our experiments in Section 6. Section 5 contains the complexity results mentioned above. Section 6 explains the scenario and the experimental results obtained. Finally in Sections 7 and 8 we discuss related and future work.

2 Basic Definitions

At the lowest level, we have a first-order alphabet Σ that consists of a set of predicate symbols $P_1, P_2, \ldots$, function symbols $f_1, f_2, \ldots$, constant symbols $c_1, c_2, \ldots$, and variable symbols $x_1, x_2, \ldots$ all with specific sorts (or types). The multi-sorted terms over the alphabet are expressions formed by applying functions to arguments of the correct sort. Since terms may contain variables, all variables must have associated sorts. A fact is an atomic predicate over multi-sorted terms.

In order to accommodate time in our model, we associate to each fact a timestamp. *Timestamped facts* are of the form $P(t_1, \ldots, t_n)@t$, where t is the timestamp of the fact $P(t_1, \ldots, t_n)$. Among the set of predicates, we distinguish the zero arity predicate $Time$, which intuitively denotes the current global time of the system. For instance, the fact $Time@2$ denotes that the global time is 2. Here, we assume that timestamps are natural numbers. The intuitive meaning of a timestamp may depend on the system one is modeling. For instance, in our clinical investigations example, the timestamp associated to a fact could denote the time when a problem with a subject has been detected.

The *size of a fact*, P, denoted by $|P|$, is the total number of symbols it contains. We count one for each constant, variable, predicate, and function symbols, *e.g.*, $|P(x, c, x)| = 4$, and $|P(f(x))| = 3$. For our complexity results, we assume an upper bound on the size of facts, as in [11, 17, 14]. This means that for all facts, $P(t_1, \ldots, t_n)@t$, the arity of symbols, n, and the depth of terms, $t_1, \ldots, t_n$, are bounded. However, we make no assumptions on the depth of timestamps, t, that is, the size of timestamps may be unbounded.

A *state*, or *configuration* of the system is a finite multiset, $Q_1@t_1, \ldots, Q_n@t_n$, of *grounded* timestamped facts, *i.e.*, timestamped facts not containing variables. Configurations are assumed to contain exactly one occurrence of the predicate $Time$. We use W, X to denote the multiset resulting from the multiset union of W and X. For instance, the configuration

$\{Time@5, Blood(id_1, scheduled)@7, Dose(id_1)@5, Status(id_1, normal)@5\}$

denotes that that current time is 5, that the blood test for subject identified by id_1 should be taken on time 7, that the same subject took a dose of the drug at time 5, and his status is $normal$, *i.e.*, no problem has been detected.

Following [17], we assume that the global configuration is partitioned into different local configurations each of which is accessible only to one agent. There is also a public configuration, which is accessible to all agents. This separation of the global configuration is done by partitioning the set of predicate symbols in the alphabet and it will be usually clear from the context. The time predicate $Time$ is assumed to be public. For instance, in the configuration above all facts, except $Time$, belong to the health institution monitoring the subject id_1.

Time constraints The time requirements of a system are specified by using *time constraints*. Time constraints are arithmetic comparisons involving exactly two timestamps:

$$T_1 = T_2 \pm d,\ T_1 > T_2 \pm d,\ \text{or}\ T_1 \geq T_2 \pm d, \tag{1}$$

where d is a natural number and T_1 and T_2 are time variables, which may be instantiated by the timestamps of any fact including the global time.

A concrete motivation for time constraints to be relative is that, as in physics, the rules of a collaboration are also not affected by time shifts. If we shift the timestamps of all facts by the same value, the same rules and conditions valid with respect to the original state are also valid with respect

to the resulting state. If time constraints were not relative, however, then one would not be able to establish this important invariant. Indeed, as we show in the companion technical report [21], the reachability problem is undecidable for systems with non-relative time constraints.

Branching Actions and Branching Plans Actions work as multiset rewrite rules. As in [17, 14] we assume that each agent has a finite set of actions. However, we extend actions in two different ways by adding guards to actions and by allowing actions to have non-deterministic effects.

In their most general form, actions have the following shape:

$$W \mid \Upsilon \longrightarrow_A [\exists \vec{x_1}.W_1] \oplus \cdots \oplus [\exists \vec{x_n}.W_n] \tag{2}$$

The subscript A is the name of the agent that owns this action. W is the pre-condition of this rule, while $W_1, \ldots, W_n$ are its post-conditions. All facts in $W, W_1, \ldots, W_n$ are public and/or belong to the agent A. Υ is the guard of the action consisting of a finitely many *time constraints*. The existentially quantified variables specify the creation of fresh values, also known as nonces in protocol security literature. If $n > 1$, then we classify this action as *branching*, otherwise when $n = 1$ we classify this action as *non-branching*.

We will assume that the only action that can change the global time is the following action belonging to the special agent $clock$:

$$Time@T \mid \{\} \rightarrow_{clock} Time@(T+1). \tag{3}$$

The action above does not have any constraints, which is specified by the empty set $\{\}$. It is the only action of the agent $clock$.

For the actions belonging to the remaining agents, we will further impose the following two conditions on actions depicted in Eq. 2. Firstly, the global time $Time@T$ appears in the pre-condition, W, and in each of the post-conditions $W_1, \ldots, W_n$ exactly once. Secondly, if $Time@T$ is in the pre-condition W, then all facts created in the post-conditions $W_1, \ldots, W_n$ are of the form $P@(T+d)$, where d is a natural number, possibly zero. That is, all the created facts have timestamps greater or equal to the global time. Notice that in this type of actions the timestamp of $Time$ does not change, that is, actions are *instantaneous*. For instance, the following action is not allowed:

$Time@T, R@T_1, P@T_2 \mid T_1 < T \longrightarrow_A Time@T, R@T_1, S@T_1$

because the timestamp of the created fact $S@T_1$ is not of the form $(T + d)$. That is, actions are only allowed to create facts whose timestamps are in the present or in the future.

A *branching plan* is a tree whose nodes are configurations and whose edges are labeled with a pair consisting of an action and a number, $\langle \alpha, i \rangle$. A plan is constructed by applying an action to one of its leaves. Formally, when a branching action α of the form shown in Eq. 2 is applied to a leaf of a plan labeled with W_I, the corresponding branch of the plan is extended by adding n leaves. The configuration labeling the i^{th} leaf is obtained by replacing α's pre-condition W in W_I by the post-condition W_i of α. The edge connecting W_I with i^{th} new leaf is labeled with $\langle \alpha, i \rangle$. In the process fresh values are created, which replace the existentially quantified variables, $\vec{x_i}$.

For example, let $\{Time@6, P(t_1)@1, Q(t_2)@4\}$ be a configuration appearing in the leaf of a plan $\mathcal{P}$. Then the following branching action is applicable:

$Time@T, Q(Y)@T_1 \mid \{T > T_1 + 1\} \longrightarrow_A [\exists x. Time@T, R(Y, x)@T] \oplus [Time@T, S(Y)@T]$

and it extends the plan $\mathcal{P}$ creating the following two leaves $\{Time@6, P(t_1)@1, R(t_2, n)@6\}$ and $\{Time@6, P(t_1)@1, S(t_2)@6\}$, where n is a fresh value.

▶ **Definition 1.** A *timed local state transition system* ($TLSTS$) $\mathcal{T}$ is a tuple $\langle \Sigma, I, R_{\mathcal{T}} \rangle$, where Σ is the alphabet of the language, I is a set of agents, such that $clock \in I$, and $R_{\mathcal{T}}$ is a finite set of actions owned by the agents in I of the two forms described above.

We classify an action as *balanced* if its post-conditions, W_i, and the pre-condition, W, have the same number of facts (see Eq. 2). As discussed in [17], if all actions in a system are balanced, then the size of all configurations in a plan remains the same as in the initial configuration. Since we assume facts to have a bounded size, the use of balanced actions imposes a bound on the storage capacity of the agents in the system.

Timed Initial, Goal and Critical Configurations In a collaboration, agents interact in order to achieve some common goal. However, since they do not trust each other completely, they also want to avoid some critical situations. Often these goals and critical situations mention time explicitly. For instance, in clinical investigations example discussed in the Introduction, the participants want to collect conclusive data without violating regulations. The sponsor should send a safety report to the FDA whenever a serious and unexpected problem is detected within 15 days. Otherwise, the sponsor can be severely penalized.

In order to formalize such aspects of a collaboration, we extend the notion of initial, goal and critical configurations proposed in [17] by attaching a set of time constraints. In particular, timed initial, goal and critical configurations have the following form:

$\{Q_1@T_1, Q_2@T_2, \ldots, Q_n@T_n\} \mid \Upsilon$

where Υ is a finite set of time constraints as shown in Eq. 1 and whose variables are in $T_1, T_2, \ldots, T_n$.

For instance, in the clinical investigations example, a possible goal configuration is when the data of a subject is collected in specified intervals for some number of times. The following goal configuration specifies that the goal is to collect the data of a subject 25 times in intervals of 28 days, but with a tolerance of 5 days: $\{Time@T, Data(Id, 1)@T_1, \ldots, Data(Id, 25)@T_{25}\}$ with the time constraints $T_i + 23 \leq T_{i+1} \leq T_i + 33$ and that $T > T_i$, for $1 \leq i \leq 25$. Formally, any instantiation of the variables $T_1, \ldots, T_{25}$ that satisfy the set of constraints above is considered a goal configuration.

Similarly, a configuration is critical for the participants of a clinical investigation when a problem is detected at time T_1, but the written report is not sent to the FDA on time, *i.e.*, within 15 days after the problem is detected: $\{Detect(Id)@T_1, Report(Id)@T_2\} \mid \{T_2 > T_1 + 15\}$.

Adding time constraints to configurations is not a restriction of the model. Quite the contrary, we provide much more flexibility within our formalism by either not constraining configurations at all or by providing a kind of temporal interference to specify initial/goal/critical configurations, in addition to the timestamps given separately.

For simplicity, we often omit the word "timed" in initial/goal/critical configurations regardless of time constraints being attached or not.

Planning Problem In [15], three compliance problems were introduced in the setting without explicit time or branching (actions with non-deterministic effects). We now restate one of these problems, called plan compliance problem in our setting with explicit time and branching. The remaining problems are dealt in detail in the technical report [21].

Given an initial configuration W and a finite set of goal and critical configurations, we call a *branching plan* $\mathcal{P}$ *compliant* if it does not contain any critical configurations and moreover if all branches of $\mathcal{P}$ lead from configuration W to any goal configuration.

(*Plan compliance problem*) Given a timed local state transition system $\mathcal{T}$, an initial configuration W consisting of grounded timestamped facts and a finite, possibly empty, set of time constraints, a finite set of goal and a finite set of critical configurations, is there a compliant plan?

3 Implementing a $TLSTS$ in Maude

The general-purpose computational tool Maude [6] provides all the machinery necessary to implement directly $TLSTS$ specifications. As Maude is based on rewriting, the Maude code looks similar to the specification itself. We now illustrate by using examples of how the encoding works.

Configurations We start by specifying the signature of a TLSTS, *i.e.*, the set of constants and predicate symbols. For instance, the code below specifies that the zero arity fact `time` is of sort (or

type) `Fact` and that `blood` is a binary fact whose argument is of sort `Id` and `Result`.

```
op time :  -> Fact .    op blood :  Id Result -> Fact .
```

Other predicates of the sort `Fact` can be specified in a similar fashion.

We specify as follows the operator `@` which attaches a natural number to facts and are used to specify timestamped facts which are of sort `TFact`.

```
op _@_ :  Fact Nat -> TFact .
```

To encode configurations, we first specify that timestamped facts is a subsort of configuration, denoted by the symbol <, that the empty set is a configuration, specified by the operator `none`, and that the juxtaposition of two configurations is also a configuration.

```
subsort TFact < Conf .
op none :  -> Conf .    op __ :Conf Conf->Conf [assoc comm id:none] .
```

The last statement also specifies that configurations are multisets by attaching the keywords `assoc` and `comm`, which specify that the operator constructing configurations is both associative and commutative. Hence, when Maude checks whether an action (specified below) is applicable, Maude will consider all possible permutations of elements until it finds a match which satisfies the action's pre-condition as well as its guard. Finally, the keyword `id: none` specifies that the constructor `none`, specifying the empty set, is the identity of an operator. For instance, it is used to identify the configurations `none (time@2) none (blood(id1,positive)@3)` and `(time@2) (blood(id1,positive)@3))`.

Timed Critical and Timed Goal Configurations Timed critical and timed goal configurations are specified as *equational theories*. For instance, the following equational theory specifies in Maude the critical configuration when the FDA is not notified 7 days after a serious and unexpected problem is detected. Here `Num` is the fresh value, *e.g.*, a number, uniquely identifying a serious and unexpected event with subject identified by `Id`.

```
ceq critical((C:Conf)(time@T)(detected(Id,Num)@T1)(fda(Id,no,Num)@T2))
      = true  if T > T1 + 7
```

Maude automatically replaces `critical(C)` by the boolean `true` if the configuration `C` satisfies the condition specified by the equation above. Timed goal configurations are also specified as equational theories in a similar way, only that we use the predicate `goal`, instead of `critical` to specify goal configurations.

Branching Actions and Searching for Compliant Plans Whereas critical and goal configurations are specified by using equational theories, actions are specified as rewrite rules in Maude. To accommodate branching actions, we use three new operators `noPlan`, denoting when a branching plan has no leaves, brackets used to mark a leaf of a plan, and + used to construct the list of leaves of a branching plan. The leaves of a branching plan belong to the sort `Plan`.

```
op noPlan:  -> Plan.
op {_}:  Conf -> Plan.  op _+_ :  Plan Plan-> Plan [assoc id:noPlan] .
```

The operator + is also used to specify the different outcomes of an action. For instance, the following conditional rule specifies that there are two possible outcomes when a blood test `scheduled` at time `T1` is carried out, namely, the blood test is `positive` or `negative`. Moreover, the boolean conditions specifies that test can only be carried out at the same day when it was scheduled and if none of its outcomes is a critical configuration.

```
crl[blood]: {(C:Conf)(time@T)(blood(Id,scheduled)@T1)} =>
              {(C:Conf)(time@T)(blood(Id,positive)@T)} +
              {(C:Conf) (time@T) (blood(Id,negative)@T)}
 if T1 = T ∧ not (critical((C:Conf)(time@T)(blood(Id,positive)@T))) ∧
    not (critical((C:Conf)(time@T)(blood(Id,negative)@T)))
```

Formally, when this rule is applied then two different leaves are created, one for each possible result. The remaining facts appearing in the configuration `C` are left untouched.

As in the rule above, we allow a rule to be applied only if *all* its outcomes are *not* critical configurations. For instance, the action that advances time (Eq. 3) is specified in Maude with an extra condition allowing the time to be incremented only if the resulting configuration is not critical:

```
crl[time]: {(C:Conf)(time@T)} => {(C:Conf)(time@(T+1))}
 if not (critical((C:Conf)(time@(T+1))))
```

This means that it is not possible to reach a critical configuration when using the rules as encoded above. Therefore, in order to search for a compliant plan, one does not need to care whether a critical configuration is reached, as this is not possible, but only check whether there is a plan from an initial configuration to a goal configuration obtained by using the actions as mentioned above. Maude can automatically perform this search by using a command of the following form:

```
search in MODULE_NAME : I =>+ P:Plan such that goals(P:Plan) = true .
```

where `I` is the initial configuration, `MODULE_NAME` is the name of the Maude module containing all the rules of the $TLSTS$, and finally `goals` is an equational theory that returns true when given $\{C_1\}$ + ⋯ + $\{C_n\}$ of type `Plan` only if `goal(`C_i`)` evaluates to true for all $1 \leq i \leq n$.

Besides searching for plans, the same theory can also be used for monitoring CI executions. For instance, by using the equational theory specifying critical configurations, one can detect when a deviation has occurred and send alarms to the responsible agents. After a CI has been carried out, one could also use the actual plan carried out to study how CIs have been executed.

4 Dealing with the Unboundedness of Time

Comparing our *timed* collaborative models here with the results on the *untimed* collaborative systems in our previous work, we meet here with a number of the crucial difficulties. In the case of planning problems for the untimed systems with balanced actions, we are dealing with a *finite* (though huge) state space. Here the state space is *internally infinite*, since an arbitrary number of time advances is allowed in principle. For a straightforward example, consider a plan where time is eagerly advanced. That is, consider a plan with a single branch where time advances constantly:

$Time@0, W \longrightarrow_{clock} Time@1, W \longrightarrow_{clock} Time@2, W \longrightarrow_{clock} \cdots$

Since there are no bounds on the length nor depth of plans, the final value of the global time cannot be bounded in advance.

This section describes how to overcome the problem above by proposing an equivalence relation between configurations. The key idea is that since time constraints are relative, that is, they involve the difference of two timestamps, we do not need to keep track of the values for timestamps separately, in order to determine whether our time constraints are satisfied or not.

Truncated time differences In particular, we will store the time differences among the facts, but truncated by an upper bound. Formally, assume D_{max} be an upper bound on the numbers appearing explicitly in a given planning problem with the model $\mathcal{T}$ - that is, the numbers in the actions and time constraints in $\mathcal{T}$, and in the initial, goal and critical configurations, for instance, the a in Eq. 1. Then the *truncated time difference* of two timed facts $P@T_1$ and $Q@T_2$ with $T_1 \leq T_2$, denoted by $\delta_{P,Q}$, is defined as follows:

$$\delta_{P,Q} = \begin{cases} T_2 - T_1, \text{ provided } T_2 - T_1 \leq D_{max} \\ \infty, \text{ otherwise} \end{cases}$$

Intuitively, we can truncate time differences without sacrificing soundness nor completeness because time constraints are relative as shown in Eq. 1. Hence, if the time difference of two facts is greater than the upper bound D_{max}, then it does not really matter how much greater it is, but just that it is greater. For instance, consider the time constraint $t_1 \geq t_2 + d$ involving the timestamps of the facts $P@t_1$ and $Q@t_2$. If $\delta_{Q,P} = \infty$, this time constraint is necessarily satisfied.

Equivalence between configurations We use the notion of truncated time differences introduced above to formalize the following equivalence relation among configurations.

▶ **Definition 2.** Given a planning problem with the model $\mathcal{T}$, let D_{max} be an upper bound on the the numeric values appearing in $\mathcal{T}$ and in the initial, goal and critical configurations. Let

$\mathcal{S} = Q_1@T_1, Q_2@T_2, \ldots, Q_n@T_n$ and $\widetilde{\mathcal{S}} = Q_1@\widetilde{T}_1, Q_2@\widetilde{T}_2, \ldots, Q_n@\widetilde{T}_n$

be two configurations written in canonical way where the two sequences of timestamps $T_1, \ldots, T_n$ and $\widetilde{T}_1, \ldots, \widetilde{T}_n$ are non-decreasing. (For the case of equal timestamps, we sort the facts in alphabetical order, if necessary.) Then $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ are equivalent if for any $1 \leq i < n$ either of the following holds: $T_{i+1} - T_i = \widetilde{T}_{i+1} - \widetilde{T}_i \leq D_{max}$ or both $T_{i+1} - T_i > D_{max}$ and $\widetilde{T}_{i+1} - \widetilde{T}_i > D_{max}$.

In order to illustrate the equivalence above, assume that $D_{max} = 3$ and consider the following two configurations: $\{R@3, P@4, Time@11, Q@12, S@14\}$ and $\{R@0, P@1, Time@6, Q@7, S@9\}$. According to the definition above, these configurations are equivalent since their truncated time differences are the same. This can be observed by the following *canonical* representation, called δ*-representation*. A δ-representation is constructed from a given configuration by sorting its facts according to their timestamps and sorting facts in alphabetical order as tie-breaker. Then we compute the time difference among two consequent facts, $\delta_{Q_i,Q_{i+1}}$. For instance, both configurations above have the following δ-representation: $\langle R, 1, P, \infty, Time, 1, Q, 2, S\rangle$

Here a value appearing between two facts, Q_i and Q_{i+1}, is the truncated time difference of the corresponding facts, $\delta_{Q_i,Q_{i+1}}$, *e.g.*, $\delta_{R,P} = 1$ and $\delta_{P,\text{Time}} = \infty$. It is also easy to see that from the tuple above, one can compute the remaining truncated time differences. For instance, $\delta_{\text{Time},S} = 3$, since $1 + 2 = 3$, while $\delta_{R,Q} = \infty$, since $1 + \infty + 1 = \infty$.

We now formalize the intuition described above that using time differences that are truncated by an upper bound is enough to determine whether a time constraint is satisfied or not.

▶ **Lemma 3.** *Let S and $\widetilde{S}$ be two equivalent configurations from Definition 2.*

$\mathcal{S} = Q_1@T_1, Q_2@T_2, \ldots, Q_n@T_n$ *and* $\widetilde{\mathcal{S}} = Q_1@\widetilde{T}_1, Q_2@\widetilde{T}_2, \ldots, Q_n@\widetilde{T}_n$.

Then the following holds for all i and j such that $i > j$, and for all $a \leq D_{max}$:

$T_i - T_j = a$ *if and only if* $\widetilde{T}_i - \widetilde{T}_j = a$

$T_i - T_j < a$ *if and only if* $\widetilde{T}_i - \widetilde{T}_j < a$

$T_i - T_j > a$ *if and only if* $\widetilde{T}_i - \widetilde{T}_j > a$

Proof The only interesting case is the last one, which can be proved by using the fact that $a \leq D_{max}$ and that S and $\widetilde{S}$ are equivalent. Hence, $T_i - T_j > D_{max} > a$ is true if and only if $\widetilde{T}_i - \widetilde{T}_j > D_{max} > a$ is true, and $D_{max} \geq T_i - T_j > a$ is true if and only if $D_{max} \geq \widetilde{T}_i - \widetilde{T}_j > a$, since $T_i - T_j = \widetilde{T}_i - \widetilde{T}_j$. ◀

Handling time advances and action applications Our next task is to show that our equivalence relation using truncated time differences is well-defined with respect to actions. That is, we show that actions preserve the equivalence among configurations. This will allow us to represent plans using δ-representations only.

However, in order to prove such a result, we need yet another assumption on configurations in order to faithfully handle time advances. The problem lies with the *future facts*, that is, those facts whose timestamps are greater than the global time. If there is a future fact P such that $\delta_{\text{Time},P} = \infty$, then it is not the case that equivalence is preserved when we advance time. For example, consider the following two configurations equivalent under the upper bound $D_{max} = 3$:

$\mathcal{S}_1 = \{Time@0, P@5\}$ and $\mathcal{S}_2 = \{Time@0, P@4\}$.

If we advance time on both configurations, then the resulting configurations, $\mathcal{S}'_1$ and $\mathcal{S}'_2$, are not equivalent. This is because the truncated time difference $\delta_{\text{Time},P}$ is still ∞ in $\mathcal{S}'_1$, while it changes to 3 in $\mathcal{S}'_2$. Notice that the same problem does not occur neither with present nor past facts, *i.e.*, those facts whose timestamps are less or equal to the global time.

▶ **Definition 4.** Given an upper bound D_{max} in a planning problem (as per Definition 2), a configuration $\mathcal{S}$ is called *future bounded* if for any future fact P in $\mathcal{S}$, the time difference $\delta_{\text{Time},P} \leq D_{max}$.

Recall from Section 2 that there are two types of actions, namely, the action that advances time and instantaneous actions belonging to agents. Moreover, recall that the latter actions are restricted in such a way that all created facts have timestamps of the form $T + d$, where T is the global time. This restriction allows us to show that actions preserve the future boundedness of configurations as states the following result.

▶ **Lemma 5.** *Let $\mathcal{T}$ be a TLSTS and $\mathcal{S}$ be a future bounded configuration. Let $\mathcal{S}'$ be the configuration obtained from $\mathcal{S}$ by applying an arbitrary action in $\mathcal{T}$. Then $\mathcal{S}'$ is also future bounded.*

As per Definition 2 the initial configuration in a planning problem is future bounded, which from the lemma above implies that all configurations in a plan are also future bounded.

We are now ready to show the main result of this section.

▶ **Theorem 6.** *For any given planning problem the equivalence relation between configurations given by Definition 2 is well-defined with respect to the actions of the system (including time advances) and goal and critical configurations. Any plan starting from the given initial configuration can be conceived as a plan over δ-representations.*

Proof (Sketch) Let $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ be two equivalent configurations. Assume that $\mathcal{S}$ is transformed in $\mathcal{S}'$ by means of an action α. By Lemma 3 the configuration $\widetilde{\mathcal{S}}$ also complies with the time constraints required in α, and hence the action α will transform $\widetilde{\mathcal{S}}$ into some $\widetilde{\mathcal{S}}'$. It remains to show that $\widetilde{\mathcal{S}}'$ is equivalent to $\mathcal{S}'$.

We consider our two types of actions. Let the time advance transform $\mathcal{S}$ into $\mathcal{S}'$, and $\widetilde{\mathcal{S}}$ into $\widetilde{\mathcal{S}}'$. From Lemma 5, we have that both $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ are future bounded and therefore $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ are trivially equivalent. For the second type of actions, namely the instantaneous actions belonging to agents, the reasoning is similar. Each created fact in the configuration $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ will be of the form $P@(T + d)$ and $P@(\widetilde{T} + d)$, where T and $\widetilde{T}$ are the global time in $\mathcal{S}$ and $\widetilde{\mathcal{S}}$, respectively. Therefore each created fact has the same difference d to the global time, which implies that these created facts have the same truncated time differences to the remaining facts. Hence $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ are equivalent.

Finally, also from Lemma 3, $\mathcal{S}$ is a goal (respectively, critical) configuration if and only if $\widetilde{\mathcal{S}}$ is a goal (respectively, critical) configuration. ◀

The theorem above establishes that using δ-representations for writing plans is well defined, but it does not establish a bound on the number of δ-representations. To achieve this, we need the further assumption that all actions are balanced. Recall that balanced actions are actions that have the same number of facts in their pre and post-conditions. By using balanced actions, the number of facts in any configuration of a plan is the same as the number of facts in the plan's initial configuration. Hence, we can also establish that there is a finite number of δ-representations. Later in the Section 5 we provide more precise bounds.

▶ **Corollary 7.** *In the case of a model $\mathcal{T}$ with balanced actions, we can deal with the finite space of representatives of the form $(Q_1, \delta_{12}, Q_2, \delta_{23}, Q_3, \ldots, Q_i, \delta_{i,i+1}, Q_{i+1}, \ldots, Q_m, \delta_{m,m+1}, Q_{m+1})$, the size of each of the representatives is polynomial with respect to m, k, and $\log_2 D_{max}$, where m is the number of facts in the initial configuration, k is the upper bound on the size of facts, and D_{max} is the upper bound on the numeric values appearing in $\mathcal{T}$ and in the initial, goal and critical configurations.*

5 Complexity Results

This section enters into the details of the complexity of the plan compliance problem described at the end of Section 2. Throughout this section, we assume that all actions are balanced, *i.e.*, actions have the same number of facts in their pre and post-conditions. Our main results are summarized in Table 1.

Table 1 Summary of the complexity results for the plan compliance problem. We mark the new results appearing here with a $\star$.

Balanced Actions	Non-Branching	PSPACE-complete*
	Possibly Branching	EXPTIME-complete*
Not Necessarily Balanced Actions		Undecidable [15]

Plan Compliance with Non-Branching Actions only We consider the plan compliance problem when actions are non-branching and balanced and when the size of facts is bounded. We show that this problem is PSPACE-complete.

PSPACE-hardness: It was shown in [14] that one can faithfully encode a Turing machine with tape of size n using systems with balanced actions. The same idea works in our setting with time. It is easy to modify the encoding in [14]. Timestamps do not play any important role in such encoding.

PSPACE upper bound: It is more interesting to show that the plan compliance problem is in PSPACE when the size of facts is bounded and actions are non-branching and balanced. In particular, we will now use all the machinery introduced in Section 4 by using δ-representations of configurations to search for compliant plans.

In order to determine the existence of a compliant plan, it is enough to consider plans that never reach configurations with the same δ-configuration twice. If a plan reaches to a configuration whose δ-representation is the same as a previously reached configuration, there is a cycle of actions which could have been avoided. The following lemma imposes an upper bound on the number of different δ-representations in a plan given an initial finite alphabet. Such an upper bound provides us with the maximal length of a plan one needs to consider.

▶ **Lemma 8.** *Given a TLSTS $\mathcal{T}$ under a finite alphabet Σ, an upper bound on the size of facts, k, and an upper bound, D_{max}, on the numeric values appearing in the planning problem, namely, in $\mathcal{T}$ and in the initial, goal and critical configurations, then the number of different δ-representations, denoted by $L_T(m, k, D_{max})$, with m facts (counting repetitions) is such that $L_T(m, k, D_{max}) \leq (D_{max} + 2)^{(m-1)} J^m (D + 2mk)^{mk}$, where J and D are, respectively, the number of predicate and the number of constant and function symbols in the initial alphabet Σ.*

Proof Let $\langle Q_1, \delta_{Q_1,Q_2}, Q_2, \ldots, Q_{m-1}, \delta_{Q_{m-1},Q_m}, Q_m \rangle$ be a δ-representation with m facts. There are m slots for predicate names and at most mk slots for constants and function symbols. Constants can be either constants in the initial alphabet Σ or names for fresh values (nonces). Following [14], we need to consider only $2mk$ names for fresh values (nonces). Finally, only time differences up to D_{max} have to be considered together with the symbol ∞ and there are $m - 1$ slots for time differences in a δ-representation. ◀

Intuitively, our upper bound algorithm keeps track of the length of the plan it is constructing and if its length exceeds $L_T(m, k, D_{max})$, then it knows that it has reached the same δ-representation twice. This is possible in PSPACE since the number above, when stored in binary, occupies only polynomial space with respect to its parameters. For the result below, we assume that it is possible to check in polynomial space when a configuration is critical, when it is a goal configuration, and when an action is applicable in a configuration. The proof of the theorem below is in the Appendix.

▶ **Theorem 9.** *Let $\mathcal{T}$ be a model with balanced non-branching actions. Then the plan compliance problem is in PSPACE with respect to m, k, and $\log_2 D_{max}$, where m is the number of facts in the initial configuration, k is the upper bound on the size of facts, and D_{max} is the upper bound on the numeric values appearing in the model $\mathcal{T}$, and in the initial, goal and critical configurations.*

Plan Compliance with possibly Branching Actions We now consider the plan compliance problem when actions may also be branching. In particular, we show that when actions are balanced then the plan compliance problem is EXPTIME-complete with respect to the number of facts, m, in the initial configuration, the upper bound, k, on the size of facts, the upper bound, D_{max}, on the numbers explicitly appearing in the planning problem, and the upper bound, p, on the number of post-conditions of an action.

EXPTIME-hardness: The lower bound for the plan compliance problem can be inferred from a similar lower bound described in [18]. It was shown that one can encode alternating Turing machines [5] by using propositional actions that are balanced and branching. Time does not play an important role for that encoding.

EXPTIME upper bound: Our upper bound algorithm uses an alternating Turing machine. In particular, we show that the plan compliance problem is in alternating-PSPACE (APSPACE) with respect to the number of facts, m, in the initial configuration, the upper bound, D_{max}, on the numbers appearing explicitly in the planning problem, and the upper bound, p, on the number of post-conditions of any action. That is, an alternating Turing machine can solve the plan compliance problem using polynomial space. From the equivalence between APSPACE and EXPTIME shown in [5], we can infer that the plan compliance problem is in EXPTIME with respect the same parameters. The proof can be found in the Appendix.

▶ **Theorem 10.** *Let $\mathcal{T}$ be a model with balanced actions. Then the plan compliance problem is in EXPTIME with respect to m, k, and $\log_2 D_{max}$, and p, where m is the number of facts in the initial configuration, k is the upper bound on the size of facts, and D_{max} is the upper bound on the numeric values appearing in the model $\mathcal{T}$, and in the initial, goal and critical configurations, and p is the upper bound on the number of post-conditions of actions in $\mathcal{T}$.*

6 Scenario Implemented and Experimental Results Summary

We implemented a small scenario simulating a visit of a subject in a clinical investigation. In this scenario, a subject has to undergo three tests, namely, *vital signs*, *hematology*, and *urine tests* and in some cases a further *nephrology test*. The first three tests have to be performed at the same day of the subject's visit. While the vital signs and hematology tests have a single outcome, where the data is collected, the results of the urine test may be classified in three levels: normal, high, or very high (typically over three times the normal upper bound). That is, the urine test has three outcomes according to the urine test result. If the result is very high, the urine test must be repeated *within five days*, in order to be sure that the first result is not an isolated result. Moreover, if the result of the second urine test is either high or very high, then an extra nephrology test must be performed on the same day as the second urine test. The visit is over when all necessary tests have been carried out.

As described in Section 3, tests are specified as a rewrite theory specifying an action, while the time conditions in the scenario are specified using the equational theory for critical configurations. In particular, the action for urine test has three outcomes, one for each possible result of the test. We have also implemented the machinery described in Section 4. For this example, it is enough to compute the canonical form whenever time advances.

For our experiments using Maude, we considered the following two optimizations. Since the order in which the leaves of a plan appear do not really matter, we can specify the + to be also commutative by adding the attribute `comm` to its definition. Since Maude implements rewriting modulo axioms, this reduces both the state space and the number of solutions. The second optimization, on the other hand, follows the lines described in [24] and involves avoiding interleavings of actions by merging (small-step) actions into larger (big-step) actions. However, in order to be sound and complete, such a merging of actions can only involve actions that are mutually independent. For instance, the order in which one performs the vital signs, the hematology and the first urine test is not important. Hence,

Table 2 Summary of our experimental results with different optimizations, *e.g.*, big-step rules and commutative +. An entry of the form n / t / s denotes that the search space had a total of n states and it took Maude t seconds to traverse all states finding s solutions. DNF denotes that Maude did not terminate after 40 minutes.

D_{max}		0	2	4	6	8
Small Step	Non-Comm.	63k / 91 / 6	166k / 263 / 364	373k / 603 / 4k	755k / 1651 / 19k	DNF
	Commutative	43k / 71 / 6	83k / 119 / 56	141k / 188 / 252	222k / 340 / 792	332k / 640 / 2k
Big Step	Non-Comm.	3k / 3 / 6	14k / 14 / 364	51k / 67 / 4k	140k / 220 / 19k	329k / 508 / 66k
	Commutative	1k / 1 / 6	3k / 2 / 56	6k / 5 / 252	13k / 13 / 792	23k / 26 / 2k

instead of specifying each test as a different action, we can execute all three tests as a single action. Moreover, since the urine test has three possible outcomes, while the other test have only one outcome, the resulting (big-step) action will also have three possible outcomes.

Table 2 summarizes our main experimental results for the scenario described above when using different parameters D_{max} with the upper-bound of numbers appearing anywhere in the theory (see Section 4) as well as the two optimizations described above. We performed these experiments on an Ubuntu machine (Kernel 2.6.32-37) with 3.7 Gb memory and 4 processors of 2.67 GHz (Intel Core i5). We observed that using a commutative + reduced in average the number of states by a factor of 8, search time by a factor of 11, and the number of solutions by a factor of 16. The use of big-step rules, on the other hand, did not affect the number of solutions found, but reduced considerably the number of states, by a factor of 23, and search time, by a factor 40. The accumulated reduction when using both optimizations was of a factor 58 on the number of states, 118 on search time, and 16 on the number of solutions.

The Maude code for this scenario using all combinations of the two optimizations described above as well as their experimental results can be found in [21].

7 Related Work

The specification of regulations has been topic of many recent works. In [3, 4, 19], a temporal logic formalism for modeling collaborative systems is introduced. In this framework, one relates the scope of privacy to the specific roles of agents in the system. For instance, a patient's test results, which normally should not be accessible to any agent, are accessible to the agent that has the role of the patient's doctor. We believe that our system can be adapted or extended to accommodate such roles depending on the scenario considered. In particular, it also seems possible to specify in our framework the health insurance scenario discussed in [19]. De Young *et al.* describe in [7] the challenges of formally specifying the temporal properties of regulations, such as HIPAA and GLPA. They extend the temporal logic introduced in [3] with fixed point operators, which seem to be required in order to specify these regulations. A temporal logic to specify regulations, such as the FDA Code of Federal Regulations (CFR), as properties of traces abstractly representing the operations of an organization is given in [9]. Notions of permissions and obligations are introduced to deal with regulatory sentences as conditions or exceptions to others. An algorithm to check conformance of audit logs to security and privacy policies expressed in a first-order logic with restricted quantification is presented in [13]. In the case of incomplete logs a residual formual/policy is returned.

Temporal logics are suitable for specifying the temporal properties that need to be satisfied by the traces of a system's operation. Our approach starts with an executable specification of a system using rewriting logic, combined with a mechanism to specify and check properties of executions. Specifically, critical and goal configurations defined in the equational sublogic allow us to express properties needed for generating plans for patient visits, and for monitoring clinical investigations including FDA reporting regulations. Timestamps allow us to express both temporal properties

and timing constraints. Moreover, this approach allows us to use existing rewriting tools, such as Maude [6], to implement our specifications and analyses.

Real time systems differ from our setting in that dense time domains, such as the real numbers, are required, while in our intended applications, such as clinical investigations, discrete numbers suffice. The models introduced in [1, 16, 23] deal with the specification of real time systems and also explore the complexity of some problems.

Kanovich *et al.* in [16] propose a linear logic based framework for specifying and model-checking real time systems. In particular, they demonstrate fragments of linear logic for which safety problems are PSPACE-complete. Interestingly, their examples are all balanced which is in accordance to some of our conditions. However, as discussed in [8], their model is limited since one is not allowed to specify properties which involve different timestamps. In our formalism, such properties can be specified using time constraints. In [23] conditions are identified for which the problem of checking whether a system satisfies a property, specified in linear temporal logic, is decidable. As their main application is for real time systems, they also assume dense time domains, although discrete time domains can also be accommodated. They identify non-trivial conditions on actions which allow one to abstract time and recover completeness. We are currently investigating whether a simpler definition of balanced actions and relative time constraints can provide more intuitive abstractions for systems with dense times.

Finally, there is a large body of work on Timed Automata. (See [1] for a survey.) While we extend multiset rewriting systems with time, Timed Automaton extend automaton with real-time clocks. Although timed automaton seem suitable for modeling real-time systems, such as circuits, it is not yet clear whether it is also suitable for modeling collaborative systems with explicit time. We are currently investigating connections between our formalism and Timed Automata.

8 Conclusions and Future Work

This paper introduced a model based on multiset rewriting that can be used for specifying policies and systems which mention time explicitly. We have shown that the plan compliance problem for balanced systems not containing branching actions is PSPACE-complete and the same problem for balanced systems possibly containing branching actions is EXPTIME-complete.

There are many directions which we intend to follow. In [22], we describe how an assistant can help the participants of clinical investigations to reduce mistakes and comply with policies. We are currently extending our current implementation into a small scale prototype in Maude in order to collect more feedback from the health care community. One main challenge, however, is to specify procedures in a modular fashion. One might need to specify intermediate languages that are closer to the terminology and format used in the specification of CIs, but that are still precise enough to translate them to a TLSTS. We hope that the work described in [10] may help us achieve this goal.

We would also like to extend our model to include dense times. This would allow us to specify policies for which real-times are important. For instance, [2] describes how one can reduce human errors by connecting medical devices and configuring them according to some hospital policies.

Another interesting problem to explore is checking whether a given plan, for example, a plan embedded in a protocol, complies with regulations no matter how it is executed. Such checks would help protocol design and review, and FDA audits as well as sponsors to monitor CIs and detect mistakes as early as possible.

Acknowledgments: We thank Anupam Datta, Nikhil Dinesh, Deepak Garg, Insup Lee, John Mitchell, Grigori Mints, Oleg Sokolsky, and Martin Wirsing for helpful discussions.

References

1 R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *SFM*, pages 1–24, 2004.

2 D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky. Toward patient safety in closed-loop medical device systems. In *ICCPS*, pages 139–148. ACM, 2010.

3 A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*, pages 184–198, 2006.

4 A. Barth, J. C. Mitchell, A. Datta, and S. Sundaram. Privacy and utility in business processes. In *CSF*, pages 279–294, 2007.

5 A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28:114–133, 1981.

6 M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*. Springer, 2007.

7 H. DeYoung, D. Garg, L. Jia, D. K. Kaynar, and A. Datta. Experiences in the logical specification of the HIPAA and GLBA privacy laws. In *WPES*, pages 73–82, 2010.

8 H. DeYoung, D. Garg, and F. Pfenning. An authorization logic with explicit time. In *CSF*, pages 133–145, 2008.

9 N. Dinesh, A. K. Joshi, I. Lee, and O. Sokolsky. Reasoning about conditions and exceptions to laws in regulatory conformance checking. In *DEON*, pages 110–124, 2008.

10 N. Dinesh, A. K. Joshi, I. Lee, and O. Sokolsky. Permission to speak: A logic for access control and conformance. *J. Log. Algebr. Program.*, pages 50–74, 2011.

11 N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

12 FDA. Code of federal regulations, Title 21, Chapter 1, Subchapter D, Part 312: Investigational new drug application. Available at `http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?CFRPart=312`.

13 D. Garg, L. Jia, and A. Datta. Policy auditing over incomplete logs: Theory, implementation and applications. In *CCS'11*, 2011.

14 M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. In *FAST*, 2010.

15 M. Kanovich, P. Rowe, and A. Scedrov. Policy compliance in collaborative systems. In *CSF 2009*, pages 218–233. IEEE Computer Society, 2009.

16 M. I. Kanovich, M. Okada, and A. Scedrov. Specifying real-time finite-state systems in linear logic. *Electr. Notes Theor. Comput. Sci.*, 16(1), 1998.

17 M. I. Kanovich, P. Rowe, and A. Scedrov. Collaborative planning with confidentiality. *J. Autom. Reasoning*, 46(3-4):389–421, 2011.

18 M. I. Kanovich and J. Vauzeilles. The classical ai planning problems in the mirror of horn linear logic: semantics, expressibility, complexity. *Mathematical Structures in Computer Science*, 11(6):689–716, 2001.

19 P. E. Lam, J. C. Mitchell, and S. Sundaram. A formalization of HIPAA for a medical messaging system. In *TrustBus*, pages 73–85, 2009.

20 J. Meseguer. Conditional Rewriting Logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

21 V. Nigam, T. B. Kirigin, A. Scedrov, C. Talcott, M. Kanovich, and R. Perovic. Timed collaborative systems. `http://www2.tcs.ifi.lmu.de/~vnigam/docs/TR-TLSTS/`, June 2011.

22 V. Nigam, T. B. Kirigin, A. Scedrov, C. Talcott, M. Kanovich, and R. Perovic. Towards an automated assistant for clinical investigations. In *IHI*, 2012.

23 P. C. Ölveczky and J. Meseguer. Abstraction and completeness for Real-Time Maude. *Electr. Notes Theor. Comput. Sci.*, 176(4):5–27, 2007.

24 S. F. Smith and C. L. Talcott. Specification diagrams for actor systems. *Higher-Order and Symbolic Computation*, 15(4):301–348, 2002.

A Proof of Theorem 9

Assume given three programs, $\mathcal{C}, \mathcal{G}$, and $\mathcal{A}$, such that they return the value 1 in polynomial space when given as input, respectively, a configuration that is critical, a configuration that contains the goal configuration, and a transition that is valid, that is, an instance of an action in the $TLSTS$, and return 0 otherwise.

Let m be the number of facts in the initial configuration W. Moreover, assume as inputs an upper bound, k, on the size of facts, an upper bound, D_{max}, on the numeric values appearing in the planning problem, that is in the given $TLSTS$ $\mathcal{T}$, in the initial, goal and critical configurtions, programs $\mathcal{G}, \mathcal{C}$, and $\mathcal{A}$, as described above, and a natural number $0 \leq i \leq L_T(m, k, D_{max})$.

We modify the algorithm proposed in [14] in order to accommodate explicit time. The algorithm must return "yes" whenever there is compliant plan from the initial configuration W to a goal configuration, *i.e.*, a configuration S such that $\mathcal{G}(S) = 1$. In order to do so, we construct an algorithm that searches non-deterministically whether such configuration *is reachable*. Then we apply Savitch's Theorem to determinize this algorithm. However, instead of searching for a plan using concrete values, we rely on the equivalence described in Section 4 and use δ-representations only. Theorem 6 guarantees us that this abstraction is sound and faithful. From $\mathcal{G}, \mathcal{C}$, and $\mathcal{A}$, it is easy to construct new functions $\mathcal{G}', \mathcal{C}'$, and $\mathcal{A}'$ that use δ-representations instead of configurations. In particular, since time constraints associated to goal and critical configurations are also relative, these can be checked by using the truncated time differences in δ-representations.

The algorithm begins with W_0 set to be the δ-representation of W and iterates the following sequence of operations:

1. If W_i is representing a critical configuration, *i.e.*, if $\mathcal{C}'(W_i) = 1$, then return FAIL, otherwise continue;
2. If W_i is representing a goal configuration, *i.e.*, if $\mathcal{G}'(W_i) = 1$, then return ACCEPT; otherwise continue;
3. If $i > L_T(m, k, D_{max})$, then FAIL; else continue;
4. Guess non-deterministically an action, r, applicable to W_i, *i.e.*, $\mathcal{A}'(W_i, r) = 1$. If no such action exists, then return FAIL. Otherwise replace W_i by the δ-representation W_{i+1} resulting from applying the action r to the δ-representation W_i. This is done as expected, by updating the positions of facts and the corresponding truncated time differences and continue;
5. Set i = i + 1.

We now show that this algorithm runs in polynomial space. We start with the step-counter i: The greatest number reached by this counter is $L_T(m, k, D_{max})$. When stored in binary encoding, this number takes only space polynomial to the given inputs:

$$\log(L_T(m, k, D_{max})) \quad \leq \quad (m-1)\log(D_{max}+2) + m\log(J) + mk\log(D+2mk).$$

Therefore, one only needs polynomial space to store the values in the step-counter.

We must also be careful to check that any δ-representation, W_i, can also be stored in polynomial space to the given inputs. Since our system is balanced, the size of facts is bounded, and the values of the truncated time differences are bounded, the size of any δ-representation, $\langle Q_1, \delta_{Q_1,Q_2}, Q_2, \ldots, Q_{m-1}, \delta_{Q_{m-1},Q_M}, Q_m \rangle$, in a plan is bounded.

Finally, the algorithm needs to keep track of the action r guessed when moving from one configuration to another and for the scheduling of a plan. It has to store the action that has been used at the i^{th} step. Since any action can be stored by remembering two δ-representations, one can also store these actions in space polynomial to the inputs.

B Proof of Theorem 10

We exploit the fact that the complexity classes APSPACE and EXPTIME are equivalent [5] and show that the plan compliance problem can be solved by an alternating Turing machine in PSPACE.

As with the proof of Theorem 9, we will use the equivalence relation described in Section 4 by using the δ-representations of configurations. Theorem 6 ensures that such an abstraction is sound and complete. We also assume that it is possible to check in APSPACE whether a δ-representation is a goal δ-representation or a critical δ-representation and whether an action is valid.

We define the following function FIND(i, W_i), which takes a natural number, i, specifying the depth of a plan and a δ-representation, W_i, of a configuration. Recall from Lemma 8 that the depth of plan is bounded by $L_T(m, k, D_{max})$. Our upper bound algorithm is as follows: Initialize $i = 0$ and W_0 as the δ-representation of the initial configuration W. Then proceed as follows:

1. If W_i is representing a critical configuration, then FAIL, else continue;
2. If W_i is representing a goal configuration, then ACCEPT, else continue;
3. If $i > L_T(m, k, D_{max})$ then FAIL, else continue;
4. Guess non-deterministically an action $W \mid \Upsilon \longrightarrow_A \exists x_1.W_1 \oplus \cdots \exists x_n.W_n$, that is applicable to a configuration represented by W_i, yielding configurations represented by $W^1_{i+1}, \ldots, W^n_{i+1}$;
5. If all executions of FIND($i + 1$, W^1_{i+1}), …, FIND($i + 1$, W^n_{i+1}) return ACCEPT, then return ACCEPT, otherwise return FAIL;

The fifth step is where we need the extra capabilities of an alternating Turing machine as we require that *all* executions of FIND return ACCEPT. Given the proof of Theorem 9 and the bound, p, on the number of post-conditions of actions, it is easy to check that the alternating Turing machine runs in polynomial space.