A Rewriting Framework and Logic for Activities Subject to Regulations

Max Kanovich, *Queen Mary, University of London, UK.*
E-mail: mik@eecs.qmul.ac.uk

Tajana Ban Kirigin, *University of Rijeka, Croatia.*
E-mail: bank@math.uniri.hr

Vivek Nigam, *Federal University of Paraíba, Brazil.*
E-mail: vivek.nigam@gmail.com

Andre Scedrov, *University of Pennsylvania, USA.*
E-mail: scedrov@math.upenn.edu

Carolyn Talcott, *SRI International, USA. E-mail: clt@csl.sri.com*

Ranko Perovic, *Senior Clinical Trial Specialist, USA.*
Email: 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, as well as initial, 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 present a formal semantics of our model based on focused proofs of linear logic with definitions. Furthermore, 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. We also determine the computational complexity of various planning problems. Plan compliance problem, for example, is the problem of 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. Finally, we show that the restrictions on the form of actions and time constraints taken in the specification of our model are neccessary for decidability of the planning problems.

1 Introduction

Regulations are commonly used to set the rules of conduct of numerous activities in order to ensure quality of results and avoid negative consequences. For example, 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) [16]. 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 [32], 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 foundation for building automated assistants. Our model is an extension of the systems used for modeling collaborative systems proposed in [25] with explicit time. An important feature of our model is that its specifications can be directly written and executed in Maude [9], a powerful tool based on rewrite logic [29]. For more details see [32] where we address implementation.

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) events. For instance, the paragraph 312.32 on Investigational New Drug Application (IND) safety [16] 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 below 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 above clause explicitly mentions 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. Alternatively, the timestamp, t_2, of the fact $Deadline@t_2$ could denote the time of when some activity should end. Moreover, we keep track of time by assuming a discrete global time, using the special fact $Time@t$ that 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, *i.e.* the action can only be applied if its time constraints are satisfied. Formally, a time constraint is a comparison involving exactly two timestamps, *e.g.*, $T_1 \leq T_2 + 7$ (see Eq. 2.1).

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, containing actions that 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 a priori clear what the test result will be. Nevertheless, 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 the blood test is positive, then one might not be suitable for 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 should also 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. In our model, critical, goal and initial states can also mention time explicitly by using time constraints.

This paper's contributions are the following:

1. Timed local state transition systems are specified in order to formalize systems with explicit time. This specification takes necessary restrictions on the type of actions and time constraints so that explicit time requirements are expressible in the system, but at the same time it is precise with respect to complexity results of the associated planning problems so that we provide decidability.
2. We determine the complexity of the *plan compliance* problem [25], 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 [21]. 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 post-condition, 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. We also investigate the complexity of the reachability problem and the timed system compliance problems.
3. We present a formal semantics of our model based on logic, namely, on linear logic with definitions [36, 5, 4]. In particular, we provide an encoding and prove that there is a one-to-one correspondence between the plans and the (cut-free) focused proofs [2] of its

encoding.

Regarding contribution **1** described above, even in the case of balanced actions, we have to deal with the problem that a plan can generate timestamps T of unbounded numeric values. In particular, the state space is internally infinite since an *arbitrary* number of time advances can occur (as illustrated at the beginning of Section 5). In our previous work [20] 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 [20] 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 5.6). We also show that our new technique introduced in this paper can be combined with the technique introduced in [20] 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 [32], $TLSTS$es were only mentioned, but not formally introduced.) In Section 4 we give a formal semantics of our model based on focused proofs of linear logic with definitions. Section 5 introduces an equivalence relation between states of the system that allows us to handle the unboundedness of time with a finite space. The machinery introduced in this section is used in Section 6 to demonstrate the decidability of the plan compliance problem. Section 6 contains the complexity results mentioned above. Section 7 we show that relaxing any of the main conditions on rules described above leads to the undecidability of the reachability problem and thus the undecidability of the other compliance problem described above. Finally in Sections 9 and 10 we discuss related and future work.

This paper extends the conference paper [23] by providing in Section 4 a linear logic semantics to our model based on multiset rewriting and time constraints and also by providing full details for our complexity results for the plan compliance problem. In addition we investigate the reachability problem and the system compliance problem, which were not dealt in our conference paper [23]. We also show in Section 7 that the restrictions we impose on the form of actions and time constraints in our systems are necessary for obtaining the decidability of these problems. Relaxing any of those restrictions leads to undecidability. These results are also novel with respect to our previous work [23].

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 the dimension of 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 the number 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 symbol, *e.g.*, $|P(f(x))| = 3$, and $|P(x, c, x)| = 4$. For our complexity results, we assume an upper bound on the size of facts, as in [15, 25, 20]. This means that for all facts, $P(t_1, \ldots, t_n)@t$, the arity of predicate 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

$$\{Time@5, Blood(id_1, scheduled)@7, Dose(id_1)@4, Status(id_1, normal)@5\}$$

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 4, and his status is $normal$, *i.e.*, no problem has been detected.

For simplicity we often omit the word "timestamped" and just use the wording fact.

Following [25], 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. As argued in [25], this separation allows one to specify systems for which it is important to know which facts are owned and can be manipulated by an agent of the system. Formally, 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 above configuration 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:

$$T_1 = T_2 \pm d,\ T_1 > T_2 \pm d,\ \text{or}\ T_1 \geq T_2 \pm d, \tag{2.1}$$

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 Section 7, the reachability problem is undecidable for systems with non-relative time constraints.

2.1 Branching Actions and Plans

Branching Actions Actions work as multiset rewrite rules. As in [25, 20] we assume that each agent has a finite set of actions. However, we extend actions in two different ways: First, we add *guards to actions*; and second we allow actions to have a finite number of *non-deterministic effects*.

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

$$W \mid \Upsilon \;\longrightarrow_A\; [\exists \vec{x_1}.W_1] \oplus \cdots \oplus [\exists \vec{x_n}.W_n] \tag{2.2}$$

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 finitely many *time constraints* of the form shown in Equation 2.1. The existentially quantified variables specify the creation of fresh values, also known as nonces in protocol security literature.[1] Finally, if $n > 1$, then we classify the action as *branching*, otherwise, when $n = 1$, we classify the action as *non-branching*.

We say that a rule r of the form shown in Equation 2.2 creates a fact $F@T$, if $F@T$ does not appear in its pre-condition W, but appears in at least one of its post-conditions W_i.

With the exception of Section 7, we only consider in this paper systems with actions of the form shown in Equation 2.2 that may be of the following two types:

(*Time Tick Action*) The first one is the following action belonging to the special agent $clock$:

$$Time@T \mid \{\} \rightarrow_{clock} Time@(T+1). \tag{2.3}$$

The above action does not have any constraints, which is specified by the empty set $\{\}$. It is the only action of the agent $clock$ and is the *only action* that can change the global time.

(*Atomic Actions*) The second type of actions are those belonging to the remaining agents. We impose the following two conditions on actions depicted in Equation 2.2. 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 by the rule are of the form $P@(T+d)$, where d is a natural number, possibly zero. That is, all the facts created by this action have timestamps greater or equal to the global time. Notice that in this type of action the timestamp of $Time$ does not change, that is, these actions are *instantaneous*. Also notice that, for example, the following action is not allowed

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

This is because the timestamp of the created fact $S@T_1$ is not of the form $(T+d)$. That is, actions cannot create facts with arbitrary timestamps, instead they are only allowed to create facts whose timestamps are in the present or in the future, that is equal to or greater than the current time.

As we discuss in Sections 5 and 7, the two conditions on the actions belonging the agents different from the $clock$ agent, discussed above, play an important role for the decidability of the system.

[1] Fresh values are also often used in administrative processes, such as when a transaction number is issued. In particular, the transaction number has to be fresh. For a more detailed account for fresh values in administrative processes, see [19].

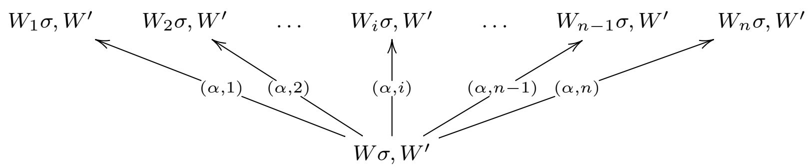

FIG. 1: A branching plan obtained by applying an action α of the form shown in Equation 2.2. Here σ is a ground substitution for α's pre-condition W, while $W_1'\sigma, \dots, W_n'\sigma$ are ground instantiations of α's post-conditions.

Branching Plans A *branching plan*, or simply *plan* is a tree whose nodes are configurations and whose edges are labeled with a pair consisting of an action and a number, $\langle \alpha, i \rangle$. As depicted in Figure 1, a plan is constructed by applying an action to one of its leaves. Formally, consider a branching action α of the form shown in Equation 2.2, that is, with pre-condition W and post-condition $W_1 \oplus \cdots \oplus W_n$. We enumerate the post-conditions as $W_1, \dots, W_n$. When such an action 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, $W\sigma$, instantiated by a ground substitution σ in W_I by the corresponding post-condition of α, $W_i\sigma$, instantiated by the same substitution σ. The edge connecting W_I with i^{th} new leaf is labeled with $\langle \alpha, i \rangle$. In the process fresh values are created, replacing 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 a 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 $\{Time@6, P(t_1)@1, R(t_2, z)@6\}$ and $\{Time@6, P(t_1)@1, S(t_2)@6\}$, where z is a fresh value.

If only non-branching actions are used, the plan has a single branch, *i.e.* the plan is simply a sequence of actions.

DEFINITION 2.1
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.

Balanced Actions We classify an action as *balanced* if its post-conditions, W_i, and the pre-condition, W, have the same number of facts. In Equation 2.2, this means that the number of facts in W and W_i are the same for all $1 \leq i \leq n$. We classify a $TLSTS$ as *balanced* if all its actions are balanced.

For any plan P obtained from a balanced system, one can easily prove that *all configurations in P have the same number of facts*, namely the number of facts in P's initial configuration. Intuitively, this means the number of facts that can be owned by an agent in the system is bounded by the number of facts in the initial configuration. In the remainder of this paper, we use the letter m to denote this number. Moreover, since we assume facts to have a bounded size, denoted using the letter k, the use of balanced actions imposes roughly a bound on the storage capacity of the agents in the system. In particular, any configuration in a plan obtained from a balanced system, may have at most mk symbols. For more about balanced systems, we point the reader to [25, 19].

As we further discuss in Section 6, the assumption that all actions in the systems are balanced is crucial for showing that the reachability problem is in PSPACE. In fact, it was shown in previous work [21] that this problem is undecidable if we allow actions to be un-balanced.

2.2 Planning Problems

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 the clinical investigations example discussed in the Introduction, the participants want to collect conclusive data without violating regulations. Moreover, 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 [25] by attaching a set of time constraints to configurations. 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. 2.1 such that its variables are in $T_1, T_2, \ldots, T_n$.

For instance, in the clinical investigations example, a possible goal configuration is the one representing a situation 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 satisfies 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 no written report is 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, time constraints provide a general mechanism to specify in a succinct fashion the set of goal and critical configurations expressing time requirements.

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

As in [19], we assume that the goal and critical configurations are closed with respect to fresh values. That is, if a configuration C containing some nonces is a goal (respectively, critical) state, then $C\sigma$ is also a goal (respectively, critical), where σ is a renaming of nonce names. This assumption is sensible, as when defining critical and goal configurations, one cannot specify the nonce names in advance, since these are freshly generated during the execution of the process being modeled. The particular nonce name should not matter for classifying a configuration as critical or a goal configuration.

Planning Problems In [25, 21] three compliance problems were introduced in the setting without explicit time or branching (actions with non-deterministic effects). We now restate two if these problems in our setting with explicit time and branching.[2]

Given an initial configuration W_I 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 the initial configuration W_I to a goal configuration.

- (Timed plan compliance problem) Given a timed local state transition system $\mathcal{T}$, an initial configuration W consisting of timestamped facts and a finite, possibly empty, set of time constraints, a timed goal configuration Z, and a finite set of timed critical configurations, is there a compliant plan which leads from W to Z?
- (Timed system compliance problem) Given a timed local state transition system $\mathcal{T}$, an initial configuration W consisting of timestamped facts and a finite, possibly empty, set of time constraints, a timed goal configuration Z, and a finite set of timed critical configurations, is no timed critical configuration reachable from W, and does there exist a plan leading from W to Z?

In [21], the plan compliance problem without explicit time was called weak plan compliance.

Although the above problems are stated as decision problems, we prove more than just existence of a plan. Ideally, we are also able to generate a plan when there is a solution. Unfortunately, the number of actions in the plan may be very large, potentially increasing the complexity of the plan generation. For this reason we follow [25] and use the notion of "scheduling" a plan. However, since we are dealing with branching plans, whereas [25] considered non-branching plans, we need to agree how the nodes of a branching tree are enumerated. Therefore, we assume fixed a *tree traversal procedure*. It can be any traversal procedure, for instance, depth-first traversal procedures (pre, in-order, or post-order) or a breadth-first traversal procedure. Assuming such a tree traversal procedure, a scheduling algorithm takes an input i representing the node in the agreed traversal and outputs the i^{th} action of the plan, which extends this node.

DEFINITION 2.2
Assume pre-defined any tree traversal procedure. An algorithm is said to *schedule* a plan if it (1) finds a plan if one exists, and (2) on input i, if the plan contains at least i nodes, then it outputs the i^{th} action of the plan, otherwise it outputs *no*.

3 Implementing a $TLSTS$ in Maude

The general-purpose computational tool Maude [9] provides all the machinery necessary to implement $TLSTS$ specifications directly. As Maude is based on rewriting, the Maude code looks similar to the specification itself. We now illustrate this by using examples of how the encoding works.

[2]The third compliance problem, introduced as the *plan compliance problem* in [21], was called *semi-critical plan compliance problem* in [19] where it was observed that, for systems without explicit time, this problem is reducible to an instance of the plan compliance problem with a larger set of critical configurations. This set includes the set of *semi-critical configurations* from which it is possible to reach a critical state of a particular agent without the participation of this agent. The same reduction can be obtained for $TLSTS$es.

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

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

To encode configurations, we first specify that the sort of timestamped facts is a subsort of the sort 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. It is used to identify configurations, for example, the configurations below are identified

`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 by *equational theories*. For instance, the following equational theory in Maude specifies the critical configuration when the FDA is not notified within 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)` with 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 are of 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 the 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.

Notice that the definition of the _+_ operator does not specify it to be commutative. However, regarding the compliance problem that we are interested in (described in Section 2), changing the order of the branches of a plan preserves its compliance as the resulting plan does not reach any critical configuration and each of its leaves are goal configurations. Thus, we can safely change the definition of _+_ to also be commutative. As we demonstrate in Section 8, this change reduces the number of possible states in average by a factor of 8.

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. 2.3) 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 a boolean function (predicate) specified by an equational theory that returns true when given $\{\mathtt{C}_1\}\ +\ \cdots\ +\ \{\mathtt{C}_n\}$ of type `Plan` only if $\mathtt{goal(C}_i\mathtt{)}$ evaluates to true for all $1 \leq i \leq n$.

It is often possible to demonstrate the non-interference of two actions, α and β, syntactically. For instance, if there is no intersection between the facts modified by α and β, these actions do not interfere between each other as they mention different parts of a configuration. The following action specifying a vital sign test does not interfere with the action above specifying a blood test:

```
crl[vital]: {(C:Conf)(time @ T)(vital(I,ID,false)@T1))} =>
                 {(C:Conf)(time @ T)(vital(I,ID,true)@T)}
 if T1 = T ∧
  not (critical((C:Conf)(time@T)(vital(I,ID,true)@T))
```

This means that a compliant plan containing a sequence of actions $\alpha;\beta$ can be replaced with another compliant plan where the order is inverted $\beta;\alpha$. In our example scenario, such interleavings increase the number of states Maude must explore by a factor of 23. A better approach would be to merge these actions into a (big-step) action. For example, the big-step action obtained from the two actions above would specify the actions of performing the vital signs and blood test at the same time. For instance, one of its post-conditions specifies when the blood test is positive:

$$\{\texttt{(C:Conf)(time@T)(vital(I,ID,true)@T)(blood(Id,positive)@T)}\}.$$

Finally, 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 Formal Semantics using Linear Logic with Definitions

This Section provides a formal semantics for $TLSTS$es based on linear logic with definitions [36, 4]. In particular, we provide an encoding for $TLSTS$es such that given an initial configuration W and a $TLSTS$, then there is a one-to-one correspondence between the set of plans from W to a goal state Z and the set of (cut-free) focused proofs [2] of its encoding.

4.1 Focused Proof System for Linear Logic with Definitions

The focused proof system, LLF, for linear logic is depicted in Figure 2 and was introduced by Andreoli [2]. Focused proofs can be regarded as the normal form proofs for proof search. In order to formally introduce LLF, we first classify the connectives $1, \otimes, \oplus$, and $\exists$ as positive and the remaining as negative. This distinction is natural as the introduction rules for the positive connectives are not-necessarily invertible, while the rules for the negative connectives are invertible. The same distinction, however, does not apply so naturally to literals and hence these are *arbitrarily* classified as positive or negative. Positive polarity literals and formulas whose main connective is positive are classified as positive formulas and the remaining as negative formulas.

As one can see from an inspection of LLF in Figure 2, there are two different sequents in LLF: those containing $\Uparrow$ which belong to the negative phase where only negative formulas are introduced, and those containing $\Downarrow$ which belong to the positive phase and only positive formulas are introduced. The decide rules D_1, D_2, reaction rules $R \Uparrow, R \Downarrow$ and the bang introduction rule ! mark transition between positive and negative phases.

A key property of LLF is that it allows one to construct macro-rules that introduce synthetic connectives. For example, assume that the N_1, N_2, N_3 are all negative formulas. Then from the focusing discipline, there are only two possible ways to introduce the sequent $\vdash \Theta : \Gamma_1, \Gamma_2 \Downarrow (N_1 \oplus N_2) \otimes N_3$:

$$\frac{\vdash \Theta : \Gamma_1 \Uparrow N_1 \qquad \vdash \Theta : \Gamma_2 \Uparrow N_3}{\vdash \Theta : \Gamma_1, \Gamma_2 \Downarrow (N_1 \oplus N_2) \otimes N_3} \qquad \text{and} \qquad \frac{\vdash \Theta : \Gamma_1 \Uparrow N_2 \qquad \vdash \Theta : \Gamma_2 \Uparrow N_3}{\vdash \Theta : \Gamma_1, \Gamma_2 \Downarrow (N_1 \oplus N_2) \otimes N_3}$$

Hence, the formula $(N_1 \oplus N_2) \otimes N_3$, under the focusing discipline, specifies such macro-rules which are obtained by applying the corresponding positive and a negative phase rules.

One can specify in a similar way a multiset rewrite rule r as a linear logical formula $F(r)$ in such a way that the macro-rule obtained by focusing on $F(r)$ corresponds *exactly* to the

Introduction Rules

$$\frac{\vdash \Theta : \Gamma \Uparrow L}{\vdash \Theta : \Gamma \Uparrow L, \bot}\ [\bot] \qquad \frac{\vdash \Theta : \Gamma \Uparrow L, F, G}{\vdash \Theta : \Gamma \Uparrow L, F \parr G}\ [\parr] \qquad \frac{\vdash \Theta, F : \Gamma \Uparrow L}{\vdash \Theta : \Gamma \Uparrow L, ?F}\ [?]$$

$$\frac{}{\vdash \Theta : \Gamma \Uparrow L, \top}\ [\top] \qquad \frac{\vdash \Theta : \Gamma \Uparrow L, F \quad \vdash \Theta : \Gamma \Uparrow L, G}{\vdash \Theta : \Gamma \Uparrow L, F \,\&\, G}\ [\&] \qquad \frac{\vdash \Theta : \Gamma \Uparrow L, F[c/x]}{\vdash \Theta : \Gamma \Uparrow L, \forall x\, F}\ [\forall]$$

$$\frac{}{\vdash \Theta : \Downarrow 1}\ [1] \qquad \frac{\vdash \Theta : \Gamma \Downarrow F \quad \vdash \Theta : \Gamma' \Downarrow G}{\vdash \Theta : \Gamma, \Gamma' \Downarrow F \otimes G}\ [\otimes] \qquad \frac{\vdash \Theta : \Uparrow F}{\vdash \Theta : \Downarrow !\, F}\ [!]$$

$$\frac{\vdash \Theta : \Gamma \Downarrow F}{\vdash \Theta : \Gamma \Downarrow F \oplus G}\ [\oplus_l] \qquad \frac{\vdash \Theta : \Gamma \Downarrow G}{\vdash \Theta : \Gamma \Downarrow F \oplus G}\ [\oplus_r] \qquad \frac{\vdash \Theta : \Gamma \Downarrow F[t/x]}{\vdash \Theta : \Gamma \Downarrow \exists x\, F}\ [\exists]$$

Identity, Reaction, and Decide rules

$$\frac{}{\vdash \Theta : A_p^{\perp} \Downarrow A_p}\ [I_1] \qquad \frac{}{\vdash \Theta, A_p^{\perp} : \Downarrow A_p}\ [I_2] \qquad \frac{\vdash \Theta : \Gamma, S \Uparrow L}{\vdash \Theta : \Gamma \Uparrow L, S}\ [R \Uparrow]$$

$$\frac{\vdash \Theta : \Gamma \Downarrow P}{\vdash \Theta : \Gamma, P \Uparrow}\ [D_1] \qquad \frac{\vdash \Theta, P : \Gamma \Downarrow P}{\vdash \Theta, P : \Gamma \Uparrow}\ [D_2] \qquad \frac{\vdash \Theta : \Gamma \Uparrow N}{\vdash \Theta : \Gamma \Downarrow N}\ [R \Downarrow]$$

FIG. 2: The focused proof system, LLF, for linear logic [2]. Here, L is a list of formulas, Θ is a multiset of formulas, Γ is a multiset of literals and positive formulas, A_p is a positive literal, N is a negative formula, P is not a negative literal, and S is a positive formula or a negated atom.

operational semantics of the rewrite rule r. But in order to specify in the same way the semantics of time constraints of $TLSTS$es, we need more machinery, namely definitions [36, 4]. A *definition* is a finite set of *clauses* which are written as $\forall \vec{x}[P(\vec{x}) \triangleq B]$: here P is a predicate and every free variable of B (the *body* of the clause) is contained in the list $\vec{x}$. The symbol $\triangleq$ is not a logical connective but is used to indicate a definitional clause. We consider that every defined predicate occurs at the head of exactly one clause. Introduction rules for definitions are shown below, where a definition can be unfolded on both the positive phase and the negative phase:

$$\frac{\vdash \Theta : \Gamma \Downarrow B\theta}{\vdash \Theta : \Gamma \Downarrow P(\vec{c})}\ [def\Downarrow] \qquad \frac{\vdash \Theta : \Gamma \Uparrow L, B\theta}{\vdash \Theta : \Gamma \Uparrow L, P(\vec{c})}\ [def\Uparrow]$$

The proviso for both of these rules is: $\forall \vec{x}[P(\vec{x}) \triangleq B]$ is a definition clause and θ is the substitution that maps the variables $\vec{x}$ to the terms $\vec{c}$, respectively. Thus, in either phase of focusing, if a defined atom is encountered, it is simply replaced by its definition and the proof search phase does not change.

We also include the rules for equality shown below:

$$\frac{\{(\vdash \Theta : \Gamma \Uparrow L)\theta \mid \theta \in CSU(r, s)\}}{\vdash \Theta : \Gamma \Uparrow L, r \neq s}\ [\neq_r] \qquad \frac{}{\vdash \Theta : \cdot \Downarrow r = r}\ [=_r]$$

where $CSU(s, r)$ denotes the complete set of unifiers of two terms. Since we are dealing with first-order logic terms, this set either contains one unifier, the most general unifier, or it is empty when the terms r and s are not unifiable. Notice that right equality introduction rule behaves exactly as the rule [1]. The proof theory of inference rules such as these is well studied (see, for example, [5, 28, 4]).Linear logic with definitions admits cut-elimination of LLF with definitions [28][3] and the focusing discipline used above was shown to be complete [4]. This paper will only need the $=_r$ rule.

4.2 Encoding $TLSTS$es in Linear Logic with Definitions

Encoding Arithmetic Conditions We show how to express the semantics of $TLSTS$es as search for cut-free focused linear logic proof with definitions. In particular, we use the following definitions to specify, for example, the arithmetic operations of $\leq, <$ and $+$ that appear in constraints:

$$
\begin{array}{lcl}
x \leq y & \triangleq & [x = zr] \oplus \\
 & & [\exists x'y'.(x = s(x')) \otimes (y = s(y')) \otimes (x' \leq y')]. \\
x < y & \triangleq & [\exists y'.(x = zr) \otimes (y = s(y'))] \oplus \\
 & & [\exists x'y'.(x = s(x')) \otimes (y = s(y')) \otimes (x' \leq y')]. \\
Plus(x, y, z) & \triangleq & [(x = zr \otimes y = z)] \oplus \\
 & & [\exists x'z'.((x = s(x')) \otimes (z = s(z')) \otimes Plus(x', y, z')].
\end{array}
$$

where natural numbers are expressed by using the successor function s and the constant zr denoting the natural number zero. For instance, the definition for $\leq$ contains two disjuncts: the left disjunct specifies the base case when the value of x is zero, and the right disjunct the inductive case, where both x and y are the successors of two numbers x' and y' such that $x' \leq y'$. The other arithmetic operations can be specified in a symmetric way.

As observed in [33], the definitions above can be used to compute an arithmetic operation in a single focused step. This is because the body of all the definitions above is positive. Therefore, once one focuses on one of the atoms defined above, one does not lose focus anymore and hence a proof consists necessarily of a single positive phase. For example, if we focus on the atom $s(zr) \leq s(s(zr))$ one obtains the following derivation:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \Theta : \cdot \Downarrow s(zr) = s(zr)}\ [=_r] \quad \dfrac{}{\vdash \Theta : \cdot \Downarrow s(s(zr)) = s(s(zr))}\ [=_r] \quad \vdash \Theta : \cdot \Downarrow zr \leq s(zr)}{\vdash \Theta : \cdot \Downarrow s(zr) = s(zr) \otimes s(s(zr)) = s(s(zr)) \otimes zr \leq s(zr)}\ [2 \times \otimes]}{\vdash \Theta : \cdot \Downarrow \exists x'y's(zr) = s(x') \otimes s(s(zr)) = s(y') \otimes x' \leq y'}\ [2 \times \exists]}{\vdash \Theta : \cdot \Downarrow [s(zr) = zr] \oplus [\exists x'y's(zr) = s(x') \otimes s(s(zr)) = s(y') \otimes x' \leq y']}\ [\oplus_2]}{\vdash \Theta : \cdot \Downarrow s(zr) \leq s(s(zr))}\ [def\Downarrow]
$$

At the open branch, the definition for the atom $zr \leq s(zr)$ is necessarily unfolded and the left disjunct of its definition is used to finish the proof. Notice that under the focusing discipline there is no other way to introduce a sequent focused on the atom $s(zr) \leq s(s(zr))$. If, for example, one attempts to prove the sequent by choosing instead the left disjunct of its body definition, one would fail since it is not possible to introduce a sequent focused on the equality $s(zr) = zr$. For a similar reason, to obtain a proof, one has to instantiate the variables x'

[3] Technically it was shown that cut-elimination works when definitions satisfy certain conditions which are out of scope of this paper. The definitions that we need here fall under this fragment.

and y' with zr and $s(zr)$, respectively. Otherwise, it is not possible to introduce the resulting equalities.

Encoding of Timestamped Facts and Constraints Using above definitions, we encode a constraint of the form $T_1 \circ T_2 + d$ as a logical formula

$$[Plus(T_2, \ulcorner d \urcorner, T2')] \otimes [T_1 \circ T_2'],$$

where $\circ \in \{>, \geq, =, \leq, <\}$ and $\ulcorner d \urcorner$ is the term corresponding to the natural number d. The encoding contains the constants s and zr. For instance, the natural number 2 is translated into the term $s(s(zr))$. Notice as well that this formula has only positive connectives and as illustrated above, once it is focused on, focusing is never lost. Therefore, we can use them to check in one positive phase whether a constraint is satisfied. If C is a constraint then we denote $\ulcorner C \urcorner$ as the logical formula obtained from C.

To encode a timestamped fact with predicate name P in linear logic, we use a new predicate name P' with arity increased by one. The encoding $\ulcorner P(\vec{c})@t \urcorner$ is the formula $P'(\vec{c}, t)$. We extend the definition of $\ulcorner \cdot \urcorner$ for constraints, natural numbers, and timestamped facts to multiset as usual.

The encoding of a configuration will be placed in the linear context of sequents, namely in the context Γ of the sequents $\vdash \Theta : \Gamma \Uparrow$. As we show below, the encoding of rewrite rules will be placed in the classical context, that is, the context Θ. This is because rewrite rules can be used any number of times.

Encoding of Actions To encode an action of the form

$$W \mid \Upsilon \to_A \exists \vec{t_1}.W_1 \oplus \cdots \oplus \vec{t_n}.W_n$$

in linear logic, we first need to specify the timestamps of the form $T + d_i$ appearing in the post-conditions W_js. For this, we construct two sets W_j^f and W_j^c from W': for each fact of the form $Q_i@(T + d_i)$ in W' we add $Q_i@T_i$ in W_j^f, where T_i is a new variable, and the formula $Plus(T, \ulcorner d_i \urcorner, T_i)$ to W_j^c; and for each fact of the form $Q_i@(T)$ in W' we add the same fact to W_j^f and no formula in W_j^c. Intuitively, the set W_j^c specifies the values for the new time variables used in W_j^f to be the same as specified in the original timestamps in W. One could regard the set W_j^c as a set of constraints to the new time variables introduced. For instance, the post-condition of the following action,

$$Time@T, P(x)@T_1, Q(y)@T_2 \mid \{T_2 > T_1{+}1\} \to_A \exists u.Time@T, P(u)@(T{+}2), R(x)@T,$$

returns the sets $\{Time@T, P(u)@(T_2'), R(x)@T\}$ and $\{Plus(T, s(s(zr)), T_2')\}$.

Now, we are ready to encode actions in linear logic: an action of the form $W \mid \Upsilon \to_A \exists \vec{t_1}.W_1 \oplus \cdots \oplus \vec{t_n}.W_n$ is encoded as the linear logic formula

$$F = \forall \vec{x} \left[\bigotimes \ulcorner W \urcorner \otimes q_A \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes_{j=1}^{j=n} W_j^c \multimap \bigoplus_{j=1}^{j=n} \left[\exists \vec{t}. \bigotimes \ulcorner W_j^f \urcorner \otimes q_A \right] \right],$$

where $\vec{x}$ are the free variables appearing in the rule together with all the new variables introduced by the translation $\ulcorner \cdot \urcorner$ and in the set W_j^c. Also, the atomic formula q_A is used only to mark that this action belongs to agent A. Moreover, the encoding of a set of transition rules $\ulcorner R_{\mathcal{T}} \urcorner$ is the set with the encoding of all the transition rules in $R_{\mathcal{T}}$, and the set of propositions used to mark a rule to an agent is defined as $Q_I = \{q_A : A \in I\}$. Intuitively, the encodings of actions are placed in the unbounded context in the left-hand-side of a sequent. However, since we are using a one-sided proof system, we use its negation in the one-sided LLF system with definitions:

$$F^{\perp} \equiv \exists \vec{x} \left[\left(\bigotimes \ulcorner W \urcorner \otimes q_A \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes_{j=1}^{j=n} W_j^c \right) \otimes \mathop{\Large\&}_{j=1}^{j=n} \left[(\forall \vec{t} \bigparr{}^{4} \ulcorner W_j^f \urcorner^{\perp} \parr q_A^{\perp}) \right] \right].$$

Assume now that all atomic formulas have positive polarity, and consequently their negation negative polarity. The focused derivation introducing $F^{\perp}$ necessarily has to be of the form below. Recall that the encodings of rewrite rules are in the classical context Θ, thus $F^{\perp} \in \Theta$.

$$\cfrac{\cfrac{\vdash \Theta : \Delta \Downarrow \bigotimes \ulcorner W \urcorner \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes_{j=1}^{j=n} W_j^c \otimes q_A \qquad \vdash \Theta : \Gamma \Downarrow \mathop{\&}_{j=1}^{j=n} \left[(\forall \vec{t} \bigparr \ulcorner W_j^f \urcorner^{\perp} \parr q_A^{\perp}) \right]}{\vdash \Theta : \Gamma, \Delta \Downarrow \left(\bigotimes \ulcorner W \urcorner \otimes q_A \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes_{j=1}^{j=n} W_j^c \right) \otimes \mathop{\&}_{j=1}^{j=n} \left[(\forall \vec{t} \bigparr \ulcorner W_j^f \urcorner^{\perp} \parr q_A^{\perp}) \right]}\ [\otimes]}{\cfrac{\vdash \Theta : \Gamma, \Delta \Downarrow F^{\perp}}{\vdash \Theta : \Gamma, \Delta \Uparrow \cdot}\ [D_2]}\ [n \times \exists]$$

Since $\otimes$ is a positive connective, the left-premise is necessarily introduced by a completely positive phase introducing all tensors in $\bigotimes \ulcorner W \urcorner \otimes q_A \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes_{j=1}^{j=n} W_j^c$ until one only focuses on atomic formulas. There are then two types of atomic formulas: the first type are atoms that have a definition, such as $Plus$, and those that do not have a definition, such as q_A. When one of the former is focused on, the focusing discipline forces its definition to be opened, thus computing the values of the timestamps of the facts in the post-conditions, specified in W_j^c. Moreover, as discussed above, these are proved without using any formulas from Δ. That is, the sequent below is proved in a single positive phase:

$$\vdash \Theta : \cdot \Downarrow \bigotimes_{j=1}^{j=n} W_j^c$$

The same happens when checking whether the rule guard is satisfied or not. In particular, the following sequent should be proved in a single positive phase

$$\vdash \Theta : \cdot \Downarrow \bigotimes \ulcorner \Upsilon \urcorner$$

and is provable if and only if the guard Υ is satisfied.

The second type of atoms are those that do not have definitions and appear in $\ulcorner W \urcorner$ and the fact q_A. Since these are assumed to have positive polarity, the only applicable rule when these are focused on is an initial rule. This forces Δ to be exactly the negation of the facts in $\ulcorner W \urcorner$ union the fact $q_A^{\perp}$. In contrast, for the right-premise of the derivation above, since $\forall$ and $\parr$ are negative connectives, the right-premise is necessarily introduced by a negative phase introducing these connectives. Hence, the macro-rule introducing an encoding of the transition rule is necessarily of the form, which corresponds to the one in Figure 1:

$$\cfrac{\vdash \Theta : \Gamma, q_A^{\perp}, \ulcorner W_1^f \urcorner^{\perp} \sigma \Uparrow \cdot \qquad \cdots \qquad \vdash \Theta : \Gamma, q_A^{\perp}, \ulcorner W_n^f \urcorner^{\perp} \sigma \Uparrow \cdot}{\vdash \Theta : \Gamma, q_A^{\perp}, \ulcorner W \urcorner^{\perp} \sigma \Uparrow \cdot}$$

Notice that if the pre-condition or the constraints in Υ of the action are not satisfied, then there is no focused proof which focuses on the encoding of this transition. This handles the inductive case of the adequacy result of our encoding of $TLSTS$ in Linear Logic with Definitions (Theorem 4.1).

Encoding of Partial Goal The base case of our adequacy result consists in checking if a partial goal is reached. This is specified in a similar way as before. Let the set of facts Z and the set of time constraints Υ constitute a goal configuration G. To check whether this goal configuration is reached, we encode G, written $\ulcorner G \urcorner$ as follows:

$$\exists \vec{x}. \bigotimes \ulcorner Z \urcorner \otimes \ulcorner \Upsilon \urcorner \otimes \top,$$

where $\vec{x}$ is the set of time variables appearing in the goal configuration. This formula is necessarily introduced by the following focused derivation:

$$\dfrac{\dfrac{\dfrac{\overline{\overline{\vdash \Theta : \Delta \Downarrow \bigotimes \ulcorner Z \urcorner}} \quad \overline{\overline{\vdash \Theta : \cdot \Downarrow \bigotimes \ulcorner \Upsilon \urcorner}} \quad \overline{\overline{\vdash \Theta : \Gamma \Downarrow \top}}\ [R\Downarrow, \top]}{\vdash \Theta : \Gamma, \Delta \Downarrow \bigotimes \ulcorner Z \urcorner \otimes \ulcorner \Upsilon \urcorner \otimes \top}\ [2 \times \otimes]}{\vdash \Theta : \Gamma, \Delta \Downarrow \exists \vec{x}. \bigotimes \ulcorner Z \urcorner \otimes \ulcorner \Upsilon \urcorner \otimes \top}\ [n \times \exists]}{\vdash \Theta : \Gamma, \Delta \Uparrow \cdot}\ [D_2]$$

As before, since all atoms are assigned with positive polarity, the focusing discipline forces that Δ contains exactly the negation of the facts appearing in $\ulcorner Z \urcorner$, that all constraints in Υ are satisfied, and that Γ contains the remaining facts in the sequent. That is, the current configuration is a goal configuration.

Given the discussion above, we prove the following connection between linear logic with definitions and reachability using $TLSTS$ by induction on the height of derivation trees and on the height/length of branching plans.

THEOREM 4.1
Let $\mathcal{T} = \langle \Sigma, I, R_{\mathcal{T}} \rangle$ be a timed local transition system. Let W be an initial configuration and G be a goal configuration under the signature Σ. Then the sequent

$$\vdash \ulcorner R_{\mathcal{T}} \urcorner^{\perp} : Q_I^{\perp}, \ulcorner W \urcorner, \ulcorner G \urcorner \Uparrow \cdot$$

is provable in linear logic with definitions where $\ulcorner X \urcorner$ is the encoding as defined above iff there is a branching plan whose root is W and whose leaves contain G.

In fact, the adequacy we get is stronger than what is stated by the result above. The adequacy is on the level of derivations [34]. That is, proof search in the linear logic encoding corresponds exactly to search using the encoded $TLSTS$. However, we must also notice that our encoding only deals with reachability and not with the Planning Problem as we do not check whether a state is critical. But, one can check whether a critical state C is reachable from an initial state by specifying the reachability goal G to be the critical state C.

Remark The representation of $TLSTS$ in Maude described in [32] serves as an executable specification and comes with tools for simulation, reachability analysis and model checking. The Linear Logic semantics of $TLSTS$ provides a different and complimentary tool for reasoning both about inference strategies and about systems specified in the formalism. Interestingly there is a close correspondence between LL derivations and rewriting logic derivations for $TLSTS$ specifications (and more generally for multiset rewrite systems) [18].

5 Dealing with the Unboundedness of Time

Comparing our *timed* collaborative models introduced here with the results on the *untimed* collaborative systems in our previous work [19], we meet 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 above problem 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 actual values of timestamps, 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 number d in Eq. 2.1. 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 defined in Eq. 2.1. 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 5.1

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

$$\mathcal{S} = Q_1@T_1, Q_2@T_2, \ldots, Q_m@T_m \text{ and } \widetilde{\mathcal{S}} = Q_1@\widetilde{T}_1, Q_2@\widetilde{T}_2, \ldots, Q_m@\widetilde{T}_m$$

be two configurations written in canonical way where the two sequences of timestamps $T_1, \ldots, T_m$ and $\widetilde{T}_1, \ldots, \widetilde{T}_m$ 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 $1 \leq i < m$ either of the following holds:

$$T_{i+1} - T_i = \widetilde{T}_{i+1} - \widetilde{T}_i \leq D_{max} \text{ or both } T_{i+1} - T_i > D_{max} \text{ and } \widetilde{T}_{i+1} - \widetilde{T}_i > D_{max}.$$

In order to illustrate the above equivalence, assume that $D_{max} = 3$ and consider the following two configurations:

$$\{R@3, P@4, Time@11, Q@12, S@14\} \text{ and } \{R@0, P@1, Time@6, Q@7, S@9\}\,.$$

According to the above definition, these configurations are equivalent since their truncated time differences are the same. This can be observed by checking their *canonical* representation, called δ*-representation* defined below.

DEFINITION 5.2
Let $\mathcal{S} = Q_1@T_1, Q_2@T_2, \ldots, Q_m@T_m$ be a configuration written in canonical way where the sequence of timestamps $T_1, \ldots, T_m$ is non-decreasing(for the case of equal timestamps, we sort the facts in alphabetical order, if necessary) and let D_{max} be an upper bound in a planning problem (as per Definition 5.1). The δ*-representation* of configuration of $\mathcal{S}$, denoted by $\delta_{\mathcal{S}}$, is the tuple

$$\langle Q_1, \delta_{Q_1,Q_2}, Q_2, \delta_{Q_2,Q_3}, Q_3, \ldots, Q_i, \delta_{Q_i,Q_{i+1}}, Q_{i+1}, \ldots, Q_{m-1}, \delta_{Q_{m-1},Q_m}, Q_m\rangle\,.$$

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 given 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 instead of actual timestamps, we are able to determine whether a time constraint is satisfied or not.

LEMMA 5.3
Let S and $\widetilde{S}$ be two equivalent configurations from Definition 5.1.

$\mathcal{S} = Q_1@T_1, Q_2@T_2, \ldots, Q_n@T_n$ 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 $a \leq D_{max}$:

$$\begin{array}{lll} T_i - T_j = a & \text{if and only if} & \widetilde{T}_i - \widetilde{T}_j = a \\ T_i - T_j < a & \text{if and only if} & \widetilde{T}_i - \widetilde{T}_j < a \\ T_i - T_j > a & \text{if and only if} & \widetilde{T}_i - \widetilde{T}_j > a \end{array}$$

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 $T_i - T_j = \widetilde{T}_i - \widetilde{T}_j$. ∎

Following Lemma 5.3, we say that a δ*-representation* Δ *satisfies a constraint* if a configuration W, such that $\delta_W = \Delta$, satisfies that constraint.

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.

We extend action application to δ-representations. It follows from the Lemma 5.3 that the same action is applicable in configurations with the same δ-representation. We, therefore, say that an *action is applicable in a δ-representation* Δ if the same action is applicable in a configuration W, such that $\delta_W = \Delta$. That is, any action a that is applicable in some configuration $\mathcal{S}$ is applicable in its δ-representation $\delta_{\mathcal{S}}$, and the resulting δ-representation, $\delta'_{\mathcal{S}}$, is the δ-representation of $\mathcal{S}'$, where $\mathcal{S} \to_a \mathcal{S}'$:

$$\begin{array}{ccc} \delta_{\mathcal{S}} & \to_a & \delta_{\mathcal{S}'} \\ \wr & & \wr \\ \mathcal{S} & \to_a & \mathcal{S}' \end{array} \tag{5.1}$$

This is well defined if it is independent of the choice of configurations. Recall that there are two types of actions, namely time advances and instantaneous actions that belong to agents. Time advances only change the timestamp denoting the global time while the rest of the configuration remains unchanged. Therefore, when we advance time in a δ-representation, the position of $Time$ and the truncated time differences involving $Time$ need to be updated. Depending on concrete values of time differences, the fact $Time$ may move to the right.

For example, for $D_{max} = 5$ and the configuration $\{R@0, P@1, \mathit{Time}@3, Q@5, S@7\}$ with the time advance action $Time@T \longrightarrow_{clock} Time@(T+1)$ we get

$$\{R@0, P@1, \mathit{Time}@3, Q@5, S@7\} \longrightarrow_{clock} \{R@0, P@1, \mathit{Time}@4, Q@5, S@7\}$$
$$\textit{i.e.} \quad \langle R, 1, P, 2, \mathrm{Time}, 2, Q, 2, S\rangle \longrightarrow_{clock} \langle R, 1, P, 3, \mathrm{Time}, 1, Q, 2, S\rangle \; .$$

With another application of time tick action we then get:

$$\langle R, 1, P, 3, \mathrm{Time}, 1, Q, 2, S\rangle \longrightarrow_{clock} \langle R, 1, P, \infty, Q, 0, \mathrm{Time}, 2, S\rangle \; .$$

Generally, the time advance action $Time@T \longrightarrow_{clock} Time@(T+1)$ applied to

$$\Delta = \langle Q_1, \delta_1, \ldots, Q_{i-1}, \delta_{i-1}, Time, \delta_i, Q_{i+1}, \delta_{i+1}, \ldots, , \delta_{m-1}, Q_m\rangle$$

results in the following δ-representation Δ', alphabetically sorted whenever truncated time differences are equal to 0:

$$\begin{array}{l} \langle Q_1, \delta_1, \ldots, Q_{i-1}, [\delta_{i-1}+1], Time, \delta_i - 1, Q_{i+1}, \delta_{i+1}, \ldots, \delta_{m-1}, Q_m\rangle, \quad \text{if } \delta_i \geq 1 \\ \langle Q_1, \delta_1, \ldots, Q_{i-1}, \delta_{i-1}, Q_{i+1}, \ldots, Q_{i+l}, [\delta_{i+l}+1], Time, \delta_{i+l} - 1, \ldots, \delta_{m-1}, Q_m\rangle, \\ \hfill \text{if } \delta_i = \delta_{i+l-1} = 0, \delta_{i+l} > 0 \end{array}$$

where $[d]$ denotes d, for $d < D_{max}$, and denotes ∞ otherwise.
In case $\Delta = \langle Q_1, \delta_1, \ldots, \delta_{m-1}, Time\rangle$, then $\Delta = \langle Q_1, \delta_1, \ldots, [\delta_{m-1}+1], Time\rangle$.

For the application of instantaneous actions recall that the fact $Time@T$ remains unchanged, while some facts from the pre-condition of the action are replaced with other facts whose timestamps are of the form $T + d$. We modify the δ-representation in the following way. We first remove the facts that appear in the pre-condition of the action and not in its post-condition. Then we insert the new facts from the post-condition, positioning them on the basis of their time difference to the fact $Time$, and alphabetically if necessary. Finally, we fill in the new time differences. This is best explained on an example. Consider the following δ-representation

$$\Delta = \langle B(d), 0, F(c), 1, G(a,b), 3, Time, 1, F(a), 2, F(d)\rangle$$

with $D_{max} = 3$ and the action

$$Time@T, G(x,y)@T_1, F(x)@T_2 \rightarrow \exists z.Time@T, G(y,z)@(T+1), F(y)@T$$

which is applicable to Δ with the substitution $\sigma(x) = a, \sigma(y = b)$. We remove those facts from the pre-condition that do not appear in the post-condition, namely $G(a,b)$ and $F(a)$, and get an expression

$$B(d), 0, F(c), 1, \_, 3, Time, 1, \_, 2, F(d)\,.$$

Next we insert the facts that appear in the post-condition and not in the pre-condition. In our case above that is the fact $G(b,n)$, where n is a fresh value. The placement of these facts is determined by the timestamps appearing in the action, which are of the form $T+d$, where T is the global time. In our example the fact $G(b,n)$ comes with the timestamp $(T+1)$ and we get:

$$\langle B(d), 0, F(c), \infty, Time, 1, G(b,n), 2, F(d)\rangle\,.$$

after updating the truncated time differences. Notice that, for example, the relative time difference between facts $F(d)$ and $Time$ is still 3.

However, in order to prove that actions preserve the equivalence among configurations, we need yet another assumption to be able to faithfully handle time advances. The problem lies within the *future facts*, that is, the facts with timestamps 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 with the upper bound $D_{max} = 3$:

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

If we advance time on both configurations, then the resulting configurations, $\mathcal{S}_1'$ and $\mathcal{S}_2'$, are not equivalent. In particular, 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.*, the facts with timestamps that are smaller or equal to the global time.

DEFINITION 5.4

Given an upper bound D_{max} in a planning problem (as per Definition 5.1), 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 5.5

Let $\mathcal{T}$ be a $TLSTS$, D_{max} be the upper bound in a planning problem (as per Definition 5.1), 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.

PROOF. Let $S \stackrel{a}{\to} S'$, and assume S' is not future bounded. Then there is a fact $Q'@T'$ in S' such that $T' - T > D_{max}$, where T is the timestamp of $Time$, *i.e.* the global time in both S and S'. Since S is future bounded, the fact $Q'@T'$ does not appear in S, but is created by the action a. Hence, $T' = T + D$ for some number $D \leq D_{max}$, which contradicts with $T' - T > D_{max}$. ∎

As per Definition 5.1 the initial configuration in a planning problem is future bounded, which as per above lemma implies that all configurations in a plan are also future bounded. Notice that even if we relax the assumption that the initial configuration is future bounded, we can make it future bounded by setting the value of D_{max} to be the greater than all the timestamps in the initial configuration, *i.e.*, D_{max} would still be the upper bound on the values of the given $TLSTS$ and in the initial, goal, and critical configurations. The important result, given by the above lemma, is that future boundedness is preserved with action application.

Following Lemma 5.3 and Lemma 5.5, given a planning problem, we say that a δ-representation is an *initial / goal / critical / future bounded δ-representation* if it is the δ-representation of an initial / goal / critical / future bounded configuration. A *plan over δ-representations* is *compliant* for a given planning problem if it does not contain any critical δ-representations and if all of its branches lead from the initial δ-representation to a goal δ-representation.

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

THEOREM 5.6
For any given planning problem the equivalence relation between configurations given by Definition 5.1 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. We first prove that the equivalence among configurations is well defined with respect to application of actions, *i.e.* that action application on δ-representations is unambiguous. It must be independent of the choice of configurations in (5.1). Consider the diagram below, where $\mathcal{S}_1$ and $\mathcal{S}_2$ are two equivalent configurations. Assume that $\mathcal{S}_1$ is transformed to $\mathcal{S}_1'$ by means of an action α. By Lemma 5.3 the configuration $\mathcal{S}_2$ also complies with the time constraints required in α, and hence the action α is applicable to $\mathcal{S}_2$ and will transform $\mathcal{S}_2$ into some $\mathcal{S}_2'$. It remains to show that $\mathcal{S}_1'$ is equivalent to $\mathcal{S}_2'$.

$$\begin{array}{ccc} \mathcal{S}_1 & \to_\alpha & \mathcal{S}_1' \\ \wr & & \\ \mathcal{S}_2 & \to_\alpha & \mathcal{S}_2' \end{array}$$

We consider our two types of actions, namely, time advances and instantaneous actions (see Section 2). Let the time advance transform $\mathcal{S}_1$ into $\mathcal{S}_1{}'$, and $\mathcal{S}_2$ to $\mathcal{S}_2'$. Since only the timestamp T denoting the global time in $Time@T$ is increased by 1, and the rest of the configuration remains unchanged, only truncated time differences involving $Time$ change in the resulting configurations. Because of the equivalence $S_1 \sim S_2$, for a fact $P@T_1^P$ in $\mathcal{S}_1$ with $T_1^P \leq T$, $Time@T$ and $\delta_{P,Time} = t$, we have $P@T_2^P$ with $T_2^P \leq \hat{T}$, $Time@\hat{T}$ and $\delta_{P,Time} = t$ in $\mathcal{S}_2$ as well. Therefore, we have $\delta_{P,Time} = [t+1]$ both in $\mathcal{S}_1'$ and $\mathcal{S}_2'$. On the other hand for any future fact $Q@T^Q$ with $\delta_{Time,Q} = t$ in $\mathcal{S}_1$ and in $\mathcal{S}_2$, we get $\delta_{Time,Q} = t - 1$ in both $\mathcal{S}_1'$ and $\mathcal{S}_2'$. Therefore, $\mathcal{S}_1{}'$ and $\mathcal{S}_2'$ are equivalent. From Lemma 5.5, we have that both $\mathcal{S}_1{}'$ and $\mathcal{S}_2'$ are future bounded.

For the second type of actions, namely the instantaneous actions belonging to agents, the reasoning is similar. Each created fact in the configuration $\mathcal{S}_1'$ and $\mathcal{S}_2'$ will be of the form $P@(T^1 + d)$ and $P@(T^2 + d)$, where T^1 and T^2 represent global time in $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Therefore each created fact has the same difference d to the global time in the corresponding configuration. This implies that the created facts have the same truncated time differences to the remaining facts. Hence ${\mathcal{S}_1}'$ and $\mathcal{S}_2'$ are equivalent. Therefore, action application on δ-representations shown in (5.1) is well defined.

Finally, as per Lemma 5.3, $\mathcal{S}_1$ is a goal (respectively, critical) configuration if and only if $\mathcal{S}_2$ is a goal (respectively, critical) configuration.

By induction on the length of the plan, it immediately follows that, given a planning problem, any compliant plan over configurations can be represented by a compliant plan over δ-representations. That is, the abstraction of configurations to δ-representations is complete.

It remains to show that the abstraction is also sound, namely that, from a compliant plan over δ-representations for a given planning problem, we can extract a concrete plan over configurations and that such a plan is compliant with respect to that planning problem. Any given δ-representation corresponds to an infinite number of configurations. For example, for the δ-representation $\langle Q_1, \delta_1, Q_2, \ldots, Q_{m-1}, \delta_{m-1}, Q_m \rangle$, one of the corresponding configurations is

$$\{Q_1@0, Q_2@\widetilde{\delta_1}, Q_3@(\widetilde{\delta_1} + \widetilde{\delta_2}), \ldots, Q_m@(\widetilde{\delta_1} + \cdots + \widetilde{\delta}_{m-1})\}$$

where $\widetilde{\delta_i} = \delta_i$ if $\delta_i \leq D_{max}$, and $\widetilde{\delta_i} = D_{max} + 1$ if $\delta_i = \infty$. We are, however, already given the initial configuration W_0 in the planning problem, for which we have $\Delta_0 = \delta_{W_0}$.

We prove the existence of a plan over configurations by induction on the length of the plan over δ-representations. Let $\Delta_0 \to_{a_1} \Delta_1 \to_{a_2} \cdots \to_{a_n} \Delta_n$ be a plan over δ-representations, compliant with the respect to the given planning problem. Then Δ_0 is the δ-representation of the initial configuration, *i.e.* $\Delta_0 = \delta_{W_0}$. For each $\Delta_{i-1} \to_{a_i} \Delta_i$, as per Lemma 5.3, since $\Delta_{i-1} = \delta_{W_{i-1}}$, the same action a_i is applicable to the configuration W_{i-1}, resulting in W_i. As proven above, and shown in (5.1), it follows that $\Delta_i = \delta_{W_i}$:

$$\begin{array}{ccccccccc}
\delta_{W_0} & & \delta_{W_{i-1}} & & \delta_{W_i} & & \delta_{W_n} \\
\| & & \| & & \| & & \| \\
\Delta_0 & \to_{a_1} \cdots \to_{a_{i-1}} & \Delta_{i-1} & \to_{a_i} & \Delta_i & \to_{a_{i+1}} \cdots \to_{a_n} & \Delta_n \\
\wr & & \wr & & \wr & & \wr \\
W_0 & \to_{a_1} \cdots \to_{a_{i-1}} & W_{i-1} & \to_{a_i} & W_i & \to_{a_{i+1}} \cdots \to_{a_n} & W_n
\end{array}$$

Hence, we get a plan over configurations consisting of same sequence of actions as the given plan over δ-representations. Since none of the δ-representations $\Delta_i = \delta_{W_i}$ is critical, it is also the case that none of the configurations W_i is critical. Also, since $\Delta_n = \delta_{W_n}$ is the goal δ-representation, it follows that W_n is a goal configuration. Hence the plan $W_0 \to_{a_1} W_1 \to \cdots \to_{a_n} W_n$ is compliant with respect to the given planning problem. ∎

The above theorem 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, as we describe in Section 6, we can establish that there is a finite number of δ-representations.

TABLE 1: Summary of the complexity results for the planning problems for balanced systems. We mark the new results appearing here with a $\star$.

Planning Problems	$LSTS$es (No time, no branching)		$TLSTS$es (Possible nonces)	
	No fresh values	Possible nonces	No branching	Possible branching
(Weak) Plan	PSPACE-complete [25]	PSPACE-complete[20]	PSPACE-complete*	EXPTIME-complete*
System	PSPACE-complete [25]	PSPACE-complete[20]	PSPACE-complete*	EXPTIME-complete*

6 Complexity Results

This section enters into the details of the complexity of the planning problems for $TLSTS$es. These problems were introduced in [25, 21] in the setting without explicit time or branching. At the end of Section 2 we have restated these problems in our setting with explicit time and branching.

Recall that facts are timestamped and that there is a finite, possibly empty set of time constraints attached to a timed initial, goal and critical configuration. Recall as well that for a given initial configuration W and a finite set of goal and critical configurations, we consider a branching plan $\mathcal{P}$ compliant if it does not contain any critical configuration, and moreover if all branches of $\mathcal{P}$ lead from configuration W to some goal configuration.

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, and that the size of facts is bounded.

Our complexity results for the planning problems for $TLSTS$es are summarized in 1.

6.1 Planning Problems for $TLSTS$es with Non-Branching Actions only

We first investigate the complexity of planning problems for $TLSTS$es when actions are non-branching and balanced and when the size of facts is bounded. We show that these problems are PSPACE-complete with respect to the parameters from the given planning problem.

PSPACE-hardness: It was shown in [20] that one can faithfully encode a Turing machine with a fixed size tape using systems with balanced actions. The same idea works in our setting with time. It is easy to modify the encoding in [20]. Timestamps do not play any important role in such encoding. Also, critical configurations are not necessary used in the encoding, so we can conclude that all three planning problems and the reachability problem for $TLSTS$es with non-branching balanced actions and facts of bounded size are PSPACE-hard.

PSPACE upper bound: It is more interesting to show that the planning problems are 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 5 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 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 6.1

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 symbols 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 [20], 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 the length of such a plan exceeds $L_T(m, k, D_{max})$, then the same δ-representation has been reached twice. This is possible in PSPACE since the number of different δ-representations given above, when stored in binary, occupies only polynomial space with respect to its parameters.

For the below results, we assume that, given a $TLSTS$ $\mathcal{T}$, and a finite set of goal and critical configurations, it is possible to check in polynomial space whether a configuration is critical, whether it is a goal configuration, and whether an action is valid, *i.e.* whether it is an instance of an action from $\mathcal{T}$ that is applicable in a given configuration.

THEOREM 6.2

Let $\mathcal{T}$ be a $TLSTS$ 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.

PROOF. 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 pair of a configuration and a transition that is valid, that is, an instance of an action in the $TLSTS$ $\mathcal{T}$ is applicable to the given configuration, 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 configurations, 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 [20] in order to accommodate explicit time. The algorithm must return "yes" (*i.e.* ACCEPT) whenever there is compliant plan from the initial configuration W to a goal configuration, that is a configuration S such that $\mathcal{G}(S) = 1$. In order to do so, we construct an algorithm that searches non-deterministically whether such a 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 5 and use δ-representations only. Theorem 5.6 guarantees 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, from $\mathcal{T}$ applicable to W_i, *i.e.*, $\mathcal{A}'(W_i, r) = 1$. If no such action exists, then return FAIL. Otherwise replace W_i with 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 facts, 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:

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

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 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, hence 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 polynomially 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. ∎

The reachability problem is an instance of the plan compliance problem with an empty set of critical configurations, hence the reachability problem for $TLSTS$es with balanced non-branching actions is in PSPACE as well.

Next we turn to system compliance problem. Recall that besides the existence of a compliant plan it is additionally requested that no critical configuration is reachable by any sequence of actions in the given system.

THEOREM 6.3

Let $\mathcal{T}$ be a $TLSTS$ with balanced non-branching actions. Then the system 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.

PROOF. In order to show that the system compliance problem is in PSPACE we modify the algorithm proposed in [25] to accommodate timestamps and time constraints. Again we rely on the fact that NPSPACE, PSPACE, and co-PSPACE are all the same complexity class. We use the same notation from the proof of Theorem 6.2 and make the same assumptions. In particular, we use the algorithms $\mathcal{G}', \mathcal{C}'$, and $\mathcal{A}'$ that run in polynomial space and that check whether a timed configuration is a goal configuration, a critical configuration, or if an action is valid in the given $TLSTS$ $\mathcal{T}$. Again we rely on the equivalence between configurations described in Section 5 and use δ-representations only. Theorem 5.6 guarantees us that this abstraction is sound and faithful.

We first need to check that none of the critical configurations is reachable from the initial configuration W. To do this we provide a non-deterministic algorithm which returns "yes" exactly when a critical configuration is reachable. The algorithm starts with W_0 set to be the δ-representation of W. For any $i \geq 0$, we first check if $\mathcal{C}'(W_i) = 1$. If this is the case, then the algorithm outputs "yes". Otherwise, we guess an action r such that $\mathcal{A}'(r) = 1$ and that it is applicable to the δ-representation W_i. If no such action exists, then the algorithm outputs "no". Otherwise, we replace W_i with the δ-representation W_{i+1} resulting from applying the action r to δ-representation W_i. This is done as expected, by updating the positions of facts and the corresponding truncated time differences. Following Lemma 6.1 we know that at most $L_T(m,k)$ guesses are required, and therefore we use a global step-counter to keep track of the number of actions. As shown in the proof of Theorem 6.2, the value of this counter can be stored in PSPACE.

Next we apply Savitch's Theorem to determinize the algorithm. Then we swap the accept and fail conditions to get a deterministic algorithm which accepts exactly when all critical configurations are unreachable.

Finally, we have to check for the existence of a compliant plan. For that we apply the same algorithm as for the timed plan compliance problem from Theorem 6.2, skipping the checking of critical states since we have already checked that no critical configurations is reachable from W. From what has been shown above we conclude that the algorithm runs in polynomial space. Therefore the system compliance problem is in PSPACE. ∎

6.2 Planning Problems for $TLSTSes$ with possibly Branching Actions

We now consider the plan compliance problem when actions may 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. For these complexity results we use alternating Turing machines [8].

An alternating Turing machine (ATM) is a non-deterministic Turing machine with states that are either existential or universal states. An alternating Turing machine in an existential state accepts if *some* transition from that state leads to an accepting state, while an alternating Turing machine in a universal state accepts if *every* transition from that state leads to an accepting state. Configurations of ATMs, as with standard Turing machines, consist of a tape contents, head position and a state. Computations of alternating Turing machines can be represented as trees, which is similar to the representation of branching plans in $TLSTS$es.

EXPTIME-hardness: The lower bound for the plan compliance problem can be inferred from a similar lower bound described in [26]. It was shown that one can encode alternating Turing machines 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 on the size of facts, k, 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 [8], we can infer that the plan compliance problem is in EXPTIME with respect to the same parameters.

We also assume here that, given a $TLSTS \mathcal{T}$, and a finite set of goal and critical configurations, 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, *i.e.* whether it is an instance of an action from $\mathcal{T}$ that is applicable in the given δ-representation.

THEOREM 6.4
Let $\mathcal{T}$ be a $TLSTS$ 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, 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 $\mathcal{T}$.

PROOF. We exploit the fact that the complexity classes APSPACE and EXPTIME are equivalent [8] and show that the plan compliance problem can be solved by an alternating Turing machine in polynomial space.

As with the proof of Theorem 6.2, we rely on the equivalence relation described in Section 5 by using the δ-representations of configurations. Theorem 5.6 ensures that such an abstraction is sound and complete.

We define the following function FIND(i, X), which takes a natural number, i, specifying the depth of a plan and a δ-representation, X, and returns ACCEPT if a compliant plan of depth i starting from X exists, and returns FAIL otherwise. Recall from Lemma 6.1 that it suffices to consider plans of depth bounded by $L_T(m, k, D_{max})$. Our upper bound algorithm is the following: Initialize $i = L_T(m, k, D_{max})$ and W_i as the δ-representation of the initial configuration W. Then proceed as follows:

1. If W_i is a critical δ-representation then FAIL, else continue;
2. If W_i is a goal δ-representation, then ACCEPT, else continue;
3. If $i = 0$ then FAIL, else continue;
4. Guess non-deterministically an action $X \mid \Upsilon \longrightarrow_A \exists x_1.X_1 \oplus \cdots \exists x_n.X_n$, that is applicable to W_i, yielding δ-representations $W_{i-1}^1, \ldots, W_{i-1}^n$;
   If no such action exists return FAIL;
5. If all executions of FIND($i-1$, W_{i-1}^1), …, FIND($i-1$, W_{i-1}^n) 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 6.2 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. ∎

THEOREM 6.5
Let $\mathcal{T}$ be a $TLSTS$ with balanced actions. Then the system 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, 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 $\mathcal{T}$.

PROOF. Similar to proof of Theorem 6.3 we first check that a critical δ-representation is not contained in any tree of actions of the system with the root W. As per Lemma 6.1 it is enough to consider trees of depth bounded by $L_T(m, k, D_{max})$.

For that search we define the function CHECK(i, X), which takes a natural number, i, specifying the depth of a tree and a δ-representation, X and returns ACCEPT if a critical δ-representation cannot be reached from X in a tree of depth i, and returns FAIL otherwise. The function CHECK(i, W_i) is defined as follows: We initialize $i = L_T(m, k, D_{max})$ and set W_i to be the δ-representation of the initial configuration W and proceed as follows:

1. If W_i is a critical δ-representation, then FAIL, else continue;
2. If $i = 0$ then ACCEPT, else continue;
3. Guess non-deterministically an action $X \mid \Upsilon \longrightarrow_A \exists x_1.X_1 \oplus \cdots \exists x_n.X_n$, that is applicable to W_i, yielding δ-representations $W_{i-1}^1, \ldots, W_{i-1}^n$;
   If no such action exists return ACCEPT;
4. If all of the executions of CHECK($i-1$, W_{i-1}^1), . . ., CHECK($i-1$, W_{i-1}^n) return ACCEPT, then return ACCEPT, otherwise FAIL.

The forth step is where we use the extra capabilities of an alternating Turing machine as we require that *all* executions return ACCEPT. Consequently, it will return FAIL if *any* execution of CHECK returns FAIL, *i.e.* if any branch reaches a critical δ-representation.

Then, if the function CHECK returned FAIL our upper bound algorithm stops and returns FAIL. Otherwise the algorithm proceeds by checking for the existence of a compliant plan as per algorithm given in the proof of Theorem 6.4. In case FIND(0,W_0) = ACCEPT the algorithm returns ACCEPT, and returns FAIL otherwise.

Given the proof of Theorem 6.2 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. Since the above algorithm is in APSPACE, it is in EXPTIME. We can conclude that the system compliance problem for systems with possibly branching actions is in EXPTIME . ∎

As mentioned in Section 2, in addition to checking for the existence of a plan in the given planning problem, we are also able to schedule a plan in all of the above cases. We take the additional input j and, in the case a compliant plan exists, we output the j-th action of the plan. For our PSPACE results from Section 6.1, we store the action for which the counter i is equal to j. Since an action can be stored as two δ-configurations, we can remember the j-th action in polynomial space with respect to inputs. For our EXPTIME results from Section 6.2, we assume given the tree traversal procedure and in case the compliant plan exists, following our algorithm we run the fixed traversal strategy and output the j-th action.

7 Relaxing the restrictions on TLSTSes

In the previous section, we demonstrated that several problems, including the reachability problem, are decidable (PSPACE-complete or EXPTIME-complete) when assuming that actions have the following restrictions:

1. All actions are Balanced;
2. The timestamps of all facts created by an action are of the form $T + d$, where T is the current time and d a natural number;
3. Time constraints of an action are of the form show in Eq. 2.1, *i.e.*, $T_1 = T_2 \pm d$, $T_1 > T_2 \pm d$, or $T_1 \geq T_2 \pm d$, involving exactly two timestamps and a natural number d.

Besides the intuitions given in Section 2 for these restrictions, we show in this section that relaxing any one of these restrictions leads to the undecidability of the reachability problem. The undecidability of the reachability problem implies the undecidability of the planning problems we study in Section 6.

Kanovich *et al.* [21] have already shown that the reachability problem is undecidable when unbalanced actions are allowed. Thus, we show that the two remaining conditions are indeed necessarily for the decidability of the reachability problem. In Section 7.1, we demonstrate that if we only relax condition 2 above, then the reachability problem is undecidable in general, while in Section 7.2 we show that if we only relax condition 3 above then the reachability is also undecidable in general. In order to obtain these undecidability results, we show that the reachability problem can be reduced to the termination problem of a two counter Minsky machine, which is known to be undecidable [30]. We briefly review Minsky machines:

A Two-Counter Machine proposed by Minsky [30] is a machine that contains two registers r_1 and r_2, a set of states, $\mathcal{S}$, and a set of instructions, Ψ. A configuration of a Minsky machine is a tuple $\langle k, i, j \rangle$, where k is the state of the machine, i is the value stored in the register r_1 and j the value stored in the register r_2.

There are only four types of instructions each of them leading from one state, k, to another state, j or j_1 or j_2, but with the following side effects on the value of registers:

- **(Add** r_i**)** ins_k: $r_i = r_i + 1$;**goto** ins_j;
- **(Subtract** r_i**)** ins_k: $r_i = r_i - 1$;**goto** ins_j;
- **(0-test** r_i**)** ins_k: **if** $r_i = 0$ **goto** ins_{j_1} **else goto** ins_{j_2};
- **(Jump)** ins_k: **goto** ins_j;

(1) An Add r_i instruction increments the register r_i; (2) A Sub r_i instruction is applicable only when r_i has a positive number and decrements it; (3) A 0-test r_i instruction is a branching instruction leading to one state if r_i contains zero and to another state otherwise; finally (4) a Jump instructions simply moves from one state to another without changing the values stored in the registers. Minsky showed that the problem of determining whether a final state, a_0, is reachable from an initial state is undecidable. We assume, with loss of generality, that in the initial state the registers r_1 and r_2 are set to zero.

7.1 Relaxing Advances of Timestamps

This section shows that by relaxing the restriction that timestamps of facts created by actions should be necessarily of the form $T + d$, where T is the current time of the enabling configuration and d a natural number. We generalize actions to be of the following form:

$$
\begin{array}{l}
Time@T, W, P_1(\vec{c}_1)@T_1, \ldots, P_n(\vec{c}_n)@T_n \mid \Upsilon \rightarrow_A \\
\quad \exists u. Time@T, W, P'_1(\vec{c}'_1)@(T + f_1(T_1, \ldots, T_n)), \ldots, P'_m(\vec{c}'_m)@(T + f_m(T_1, \ldots, T_n)),
\end{array}
$$

where in the timestamps of the created facts a polynomial $f_i(T_1, \ldots, T_n)$ is added to the global time, T. Polynomial $f_i(T_1, \ldots, T_n)$ may contain timestamps $T_1, \ldots, T_n$ that appear as timestamps of facts in the precondition of the action. We say that such an action is *linearly-time-advancing* if all polynomials $f_i(T_1, \ldots, T_n)$ are linear.

Given the actions of the above form, we show that the reachability problem for these systems is undecidable already for systems with balanced actions that are linearly-time-advancing. This means that all compliance problems discussed in Section 2 are also undecidable for such systems.

THEOREM 7.1
Given a $TLSTS$ with balanced and linearly-time-advancing actions, the reachability problem is undecidable.

PROOF. The proof is obtained by reducing the reachability problem of TLSTSes with actions that are balanced and linearly-time-advancing to termination of Minsky machines. We encode an arbitrary Minksy machine M as follows:

For each state label k, we associate a zero arity predicate St_k, called state fact, denoting the current state of the machine. Moreover, we use two zero arity predicates R_1 and R_2 to keep track of the value stored in the registers r_1 and r_2, respectively. Our actions will enforce that at any given configuration there is exactly one state fact and exactly one occurrence of a R_1 and a R_2 fact. We encode the values stored in the registers r_1 and r_2, by using the timestamps of the state facts, and by using the facts R_1 and R_2 appearing in a configuration as follows:

If $St_k@T$, $R_1@T_1$ and $R_2@T_2$ are the occurrences of the state fact and R_1, R_2 in a configuration, then such a configuration specifies that the machine is in state k, the value stored in the register r_1 is $(T_1 - T)$ and the value in r_2 is $T - T_2$. For instance, the following configuration $\{Time@7, St_a@1, R_1@3, R_2@5\}$ specifies M's configuration $\langle a, 3, 5\rangle$.

Each of M's instructions is encoded by the corresponding balanced linearly-time-advancing actions. The actions of the encoding have the following shape:

$$
\begin{array}{l}
\textbf{(Add } r_1\textbf{)}\ Time@T, St_k@T_1, R_1@T_2, R_2@T_3 \rightarrow_A \\
\qquad Time@T, St_j@T, R_1@(T + T_2 - T_1 + 1), R_2@(T + T_3 - T_1) \\
\textbf{(Add } r_2\textbf{)}\ Time@T, St_k@T_1, R_1@T_2, R_2@T_3 \rightarrow_A \\
\qquad Time@T, St_j@T, R_1@(T + T_2 - T_1), R_2@(T + T_3 - T_1 + 1) \\
\textbf{(0-test } r_1 \textbf{ if)}\ Time@T, St_k@T_1, R_1@T_2, R_2@T_3 \mid \{T_1 = T_2\} \rightarrow_A \\
\qquad Time@T, St_{j_1}@T, R_1@T, R_2@(T + T_3 - T_1) \\
\textbf{(0-test } r_1 \textbf{ else)}\ Time@T, St_k@T_1, R_1@T_2, R_2@T_3 \mid \{T_1 < T_2\} \rightarrow_A \\
\qquad Time@T, St_{j_2}@T, R_1@(T + T_2 - T_1), R_2@(T + T_3 - T_1) \\
\textbf{(0-test } r_2 \textbf{ if)}\ Time@T, St_k@T_1, R_1@T_2, R_2@T_3 \mid \{T_1 = T_3\} \rightarrow_A \\
\qquad Time@T, St_{j_1}@T, R_1@(T + T_2 - T_1), R_2@T \\
\textbf{(0-test } r_2 \textbf{ else)}\ Time@T, St_k@T_1, R_1@T_2, R_2@T_3 \mid \{T_1 < T_3\} \rightarrow_A \\
\qquad Time@T, St_{j_2}@T, R_1@(T + T_2 - T_1), R_2@(T + T_3 - T_1) \\
\textbf{(Jump)}\ Time@T, St_k@T_1, R_1@T_2, R_2@T_3 \rightarrow_A \\
\qquad Time@T, St_j@T, R_1@(T + T_2 - T_1), R_2@@(T + T_3 - T_1)
\end{array}
$$

It is easy to show that each action faithfully encodes the corresponding instruction in M. For instance, consider the first action above encoding an **(Add r_1)** instruction. At the precondition the values stored in the registers r_1 and r_2 are, respectively, $T_2 - T_1$ and $T_3 - T_1$. In the post-condition, however, since the facts have to advance in time, the timestamp of the fact St_j, denoting the next instruction, is changed to the current global time T. Therefore, the timestamps of the facts R_1 and R_2 have to be updated to $T + T_2 - T_1 + 1$ and $T + T_2 - T_1$, where the value in the register r_1 is increased by one. Also notice that **(0-test** r_i**)** instructions are split into two actions: one for the case when the test is satisfied (**(0-test** r_i **if)** and the other for the case when the test is not satisfied **(0-test** r_i **else)**. The goal is to reach a configuration that reaches the final state a_0, which is encoded by the fact St_{a_0}.

For soundness, the only problem could be with the action that advances the global time. However, since all actions above take into account the global time and recompute the timestamps of R_1 and R_2 so that they correspond to the correct values stored in the respective registers, the system is sound. For completeness, one can show that if we do not advance time, the values of the timestamps of R_1 and R_2 correspond exactly to the values stored by the registers r_1 and r_2, since the timestamps of facts St_k, encoding instructions, are always zero. Hence, the encoding in our system is complete.

Finally, notice that we do not require critical configurations, we use only one agent A, and no actions above updates values with fresh ones. ∎

Since the reachability problem is undecidable and in the proof we do not make use of any critical states, all of the compliance problems mentioned in Section 2 are undecidable.

COROLLARY 7.2
Given a $TLSTS$ with balanced actions that are linearly-time-advancing, then the plan compliance and the system compliance problems are undecidable.

7.2 Relaxing Time Constraints

Instead of relaxing the timestamps of the facts in the post-condition, we now relax the form of time constraints in the guard of actions and investigate the complexity of the reachability problem for such systems. Recall that in our models, $TLSTS$es, time constraints are necessarily of the form $T_1 \circ T_2 + d$, where $\circ \in \{>, \geq, =, <, \leq\}$ and d is a natural number. We relax this condition by allowing actions to contain constraints of the form $T_1 \circ f(T_1, \ldots, T_n)$, where f is a linear polynomial and $T_1, \ldots, T_n$ are the timestamps appearing in the precondition of the corresponding action. We call this type of actions *linearly-constrained actions*. We show that the reachability problem for $TLSTS$es with balanced and linearly constrained actions is also undecidable.

THEOREM 7.3
Given a $TLSTS$ with balanced and linearly-constrained actions, then the reachability problem is undecidable.

PROOF. As in the proof of Theorem 7.1, we reduce the reachability problem for $TLSTS$es to the termination of an arbitrary Minsky Machine M.

As in the proof of Theorem 7.1, the difference between the timestamps of R_1 (respectively, R_2) and St_k will denote the value of the register r_1 (respectively, r_2). We also use the auxiliary predicate Aux, and a predicate $Update_i^\gamma$ for each instruction γ and $i \in \{1, 2\}$ together with the $Time$ predicate to encode the effects of the instructions of M, such as the instruction to add a value to a register. The initial configuration consists of six facts with three copies of Aux.

$$\mathcal{I} = \{St_1@0, R_1@0, R_2@0, Aux@0, Aux@0, Aux@0, Time@0\}.$$

where we assume w.l.o.g. that the values in both registers is zero. A goal configuration is any configuration containing the facts $G = \{St_{a_0}@T_1, Aux@T_2, Aux@T_3, Aux@T_3\}$.

Each of M's instructions is encoded using a collection of auxiliary actions. Consider the following instruction, γ, that adds the register r_1:

(Add r_1**)** ins_k: $r_1 = r_1 + 1$;**goto** ins_j

This instruction is encoded by the following four balanced and linearly-constrained actions:

(Action γ **1)** $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, Aux@T_4, Aux@T_5, Aux@T_6 \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_j@T, Update_1^\gamma@T, Update_2^\gamma@T$

(Action γ **2)** $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_j@T_4, Update_1^\gamma@T_5 \mid \{T = T_4 + T_1 - T_3 + 1\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, St_k@T_3, St_j@T_4, Aux@T$

(Action γ **3)** $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_j@T_4, Update_2^\gamma@T_5 \mid \{T = T_4 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, St_k@T_3, St_j@T_4, Aux@T$

(Action γ **4)** $Time@T, St_k@T_3, St_j@T_4, Aux@T_5, Aux@T_6 \rightarrow_A$
$Time@T, St_k@T_3, Aux@T, Aux@T, Aux@T$

The first action is applicable only when the current state is k, specified by the fact $St_k@T_3$ in the enabling configuration. It replaces the Aux facts with $St_j@T, Update_1^\gamma@T, Update_2^\gamma@T$. The first fact encodes the new state j. Since the timestamp of St_j is T, whereas the values in the registers are computed with respect to the timestamp of St_k, namely $T_1 - T_3$ and $T_2 - T_3$, we need to update the timestamps of R_1 and R_2 to be relative to T. This is the purpose of the facts $Update_1^\gamma@T, Update_2^\gamma@T$ and of the second and the third action. The second action updates the timestamps of R_1 when the current time is exactly $T_4 + T_1 - T_3 + 1$, that is, the previous value stored in the register r_1 plus 1 and relative to the timestamp of St_j. The third action is similar and corresponds to updating the timestamp of R_2. Only, after the second and third action have been applied can the fourth action be enabled and applied, as this requires two Aux facts in the precondition . The fourth action then simply forgets the previous state, k, by replacing St_k with Aux.

The actions encoding other type of instructions are similar. We show below the encodings of the instructions for Subtracting, instruction for the 0-test for the register r_1 and the JUMP instruction. The remaining actions for register r_2 are similar.

γ **is a Subtract instruction for** r_1**:**

(Action γ **1)** $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, Aux@T_4, Aux@T_5, Aux@T_6 \mid \{T_1 > T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_j@T, Update_1^\gamma@T, Update_2^\gamma@T$

(Action γ **2)** $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_j@T_4, Update_1^\gamma@T_5 \mid \{T = T_4 + T_1 - T_3 - 1\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, St_k@T_3, St_j@T_4, Aux@T$

(Action γ **3)** $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_j@T_4, Update_2^\gamma@T_5 \mid \{T = T_4 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, St_k@T_3, St_j@T_4, Aux@T$

(Action γ **4)** $Time@T, St_k@T_3, St_j@T_4, Aux@T_5, Aux@T_6 \rightarrow_A$
$Time@T, St_k@T_3, Aux@T, Aux@T, Aux@T$

γ is a 0-test instruction for r_1:

(Action γ 1 if) $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, Aux@T_4, Aux@T_5, Aux@T_6 \to_A \mid T_1 = T_3$
$Time@T, R_1@T, R_2@T_2, St_k@T_3, St_{j_1}@T, Aux@T_5, Update_2^\gamma@T$

(Action γ 1 else) $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, Aux@T_4, Aux@T_5, Aux@T_6 \to_A \mid T_1 > T_3$
$Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_{j_2}@T, Update_1^\gamma@T, Update_2^\gamma@T$

(Action γ 2) $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_{j_i}@T_4, Update_1^\gamma@T_5 \mid \{T = T_4 + T_1 - T_3\} \to_A$
$Time@T, R_1@T, R_2@T_2, St_k@T_3, St_{j_i}@T_4, Aux@T$

(Action γ 3) $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_{j_i}@T_4, Update_2^\gamma@T_5 \mid \{T = T_4 + T_2 - T_3\} \to_A$
$Time@T, R_1@T_1, R_2@T, St_k@T_3, St_{j_i}@T_4, Aux@T$

(Action γ 4) $Time@T, St_k@T_3, St_{j_i}@T_4, Aux@T_5, Aux@T_6 \to_A$
$Time@T, St_k@T_3, Aux@T, Aux@T, Aux@T$

γ is a Jump instruction:

(Action γ 1) $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, Aux@T_4, Aux@T_5, Aux@T_6 \to_A$
$Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_j@T, Update_1^\gamma@T, Update_2^\gamma@T$

(Action γ 2) $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_j@T_4, Update_1^\gamma@T_5 \mid \{T = T_4 + T_1 - T_3\} \to_A$
$Time@T, R_1@T, R_2@T_2, St_k@T_3, St_j@T_4, Aux@T$

(Action γ 3) $Time@T, R_1@T_1, R_2@T_2, St_k@T_3, St_j@T_4, Update_2^\gamma@T_5 \mid \{T = T_4 + T_2 - T_3\} \to_A$
$Time@T, R_1@T_1, R_2@T, St_k@T_3, St_j@T_4, Aux@T$

(Action γ 4) $Time@T, St_k@T_3, St_j@T_4, Aux@T_5, Aux@T_6 \to_A$
$Time@T, St_k@T_3, Aux@T, Aux@T, Aux@T$

The auxiliary facts $Aux, Update_i^\gamma$ are necessary for the soundness and completeness of our encoding. In particular, actions can only be applied in the following order:

$$\textbf{Action } \gamma\ \mathbf{1}, n \times Clock, \textbf{Action } \gamma\ \mathbf{i}, m \times Clock, \textbf{Action } \gamma\ \mathbf{j}, h \times Clock, \textbf{Action } \gamma\ \mathbf{4}$$

where n, m, k is a number of time advances, possibly zero, and $\{i, j\} = \{1, 2\}$, that is, either Action γ 1 is applied before Action γ 2 or vice-versa.

We can prove by induction on the size of plans that for a given Minsky machine M and its encoding $\mathcal{T}_M$ as described above, then M reaches the final state a_0 if and only if $\mathcal{T}_M$ reaches a goal configuration from the initial configuration $\mathcal{I}$.

Notice that we do not need any critical configurations.

Since the termination problem is undecidable, so is the reachability problem for $TLSTS$es with balanced and linearly-constrained actions. ∎

We can conclude that since the reachability problem is undecidable and in the proof we do not make use of any critical states, all the compliance problems mentioned in Section 2 are undecidable for systems with balanced and linearly-constrained actions.

COROLLARY 7.4
Given a $TLSTS$ with balanced actions that are linearly-constrained, then the plan compliance and the system compliance problems are undecidable.

8 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 high above the reference values for some substance). 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 make 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 completed 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 5. 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 does not really matter, we can specify the operator representing branching, + , to be commutative as well, 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 [37] 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, 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} as the upper-bound of numbers appearing anywhere in the theory (see Section 5) 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 [31].

9 Related Work

The specification of regulations has been topic of many recent works. In [6, 7, 27], 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

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

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 [27]. De Young *et al.* describe in [11] the challenges of formally specifying the temporal properties of regulations, such as HIPAA and GLPA. They extend the temporal logic introduced in [6] 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 are given in [13]. 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 [17]. In the case of incomplete logs a residual 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 [9], to implement our specifications and analyses.

The Petri nets (PNs) community has investigated many related problems involving time. In particular, the coverability problem of PNs is related to our partial goal reachability problem for $TLSTS$es of a simple form - without branching actions, or critical states, or fresh values [21]. In [10], de Frutos Escrig *et al.* show decidability results for the coverability problem of a type of Timed PNs with discrete time. There seem to be connections between our timestamps of facts and their time (age) associated to tokens as well as connections between our time constraints and their time intervals labeling the arcs in these PNs. However, the complexity of their decision procedures is extremely high, as compared with our upper bounds. Notice that branching actions and critical states are not considered there. Despite these connections, we did not find any work that captures exactly the model presented in this paper.

Real time systems differ from our setting since 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, 24, 35] deal with the specification of real time systems and also explore the complexity of some problems.

Kanovich *et al.* in [24] 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 [12], 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 [35] 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 discrete time, Timed Automaton extend automaton with real-time clocks. There is no evident translation between our systems and Timed Automata, but there seam to be clear correspondence between our planning problem and their reachability problem. Moreover, the reachability problem in Timed Automata with discrete time is shown to be PSPACE-complete [1] Although timed automaton seems 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 or the notion of fresh values.

10 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 planning problems for balanced systems not containing branching actions are PSPACE-complete and that the same problems for balanced systems possibly containing branching actions are EXPTIME-complete. We have also shown that the restirctions on the form of actions and time constraint taken in the deifinition of our model, $TLSTS$, are neccessary to obtain the decidability of the reachability and planning problems.

We also provided the semantics of $TLSTS$es as a linear logic with definitions theory. Our adequacy result capitalized on the completeness of the focusing strategy for this logic.

There are many directions which we intend to follow. In [32], we describe how an assistant can help the participants of clinical investigations to reduce mistakes and comply with policies. We are 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 into a $TLSTS$. We hope that the work described in [14] 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, [3] 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.

Finally, recently we have formalized Progressing Collaborative Systems that may create fresh values [22], inspired by security protocols and administrative and business processes. Such systems are efficient, *i.e.* the processes are always advancing and are completed in a bounded number of transactions. This is reflected in the complexity of the planning problems with progressing behavior. We are currently looking into extending the notion of progressing to systems with time.

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] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[3] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky. Toward patient safety in closed-loop medical device systems. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '10, pages 139–148, New York, NY, USA, 2010. ACM.

[4] D. Baelde. *A linear approach to the proof-theory of least and greatest fixed points*. PhD thesis, Ecole Polytechnique, Dec. 2008.

[5] D. Baelde and D. Miller. Least and greatest fixed points in linear logic. In N. Dershowitz and A. Voronkov, editors, *International Conference on Logic for Programming and Automated Reasoning (LPAR)*, volume 4790, pages 92–106, 2007.

[6] 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.

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

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

[9] 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.

[10] D. de Frutos Escrig, V. V. Ruiz, and O. M. Alonso. Decidability of properties of timed-arc petri nets. In *In ICATPN00*, pages 187–206. Springer-Verlag, 2000.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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`.

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

[18] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Inf. Comput.*, 110(2):327–365, 1994.

[19] M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. *Inf. Comput.* Accepted for Publication.

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

[21] M. Kanovich, P. Rowe, and A. Scedrov. Policy compliance in collaborative systems. In *CSF '09: Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium*, pages 218–233, Washington, DC, USA, 2009. IEEE Computer Society.

[22] M. I. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Bounded memory protocols and progressing collaborative systems. In J. Crampton, S. Jajodia, and K. Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 309–326. Springer, 2013.

[23] M. I. Kanovich, T. B. Kirigin, V. Nigam, A. Scedrov, C. L. Talcott, and R. Perovic. A rewriting framework for activities subject to regulations. In A. Tiwari, editor, *RTA*, volume 15 of *LIPIcs*, pages 305–322. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[24] 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.

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

[26] 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.

[27] P. E. Lam, J. C. Mitchell, and S. Sundaram. A formalization of HIPAA for a medical messaging system. In S. Fischer-Hübner, C. Lambrinoudakis, and G. Pernul, editors, *TrustBus*, volume 5695 of *Lecture Notes in Computer Science*, pages 73–85. Springer, 2009.

[28] R. McDowell and D. Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.

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

[30] M. Minsky. Recursive unsolvability of post's problem of 'tag' and other topics in the theory of turing machines. *Annals of Mathematics*, 1961.

[31] 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.

[32] V. Nigam, T. B. Kirigin, A. Scedrov, C. Talcott, M. Kanovich, and R. Perovic. Towards an automated assistant for clinical investigations. In *Second ACM SIGHIT International Health Informatics Symposium*, 2012.

[33] V. Nigam and D. Miller. Algorithmic specifications in linear logic with subexponentials. pages 129–140, 2009.

[34] V. Nigam and D. Miller. A framework for proof systems. *J. Autom. Reasoning*, 45(2):157–188, 2010.

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

[36] P. Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *Eighth Annual Symposium on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993.

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