Timed Collaborative Systems

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

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

1 Introduction

Time is often a key component used in specifying the rules and the requirements of a collaboration. For a correct collaboration and to achieve a common goal, participants usually should follow strict deadlines and should have quick reactions to some (unexpected) event.

Consider, for example, the scenario where a sponsor, typically a pharmaceutical company, colaborates with health institutions, typically hospitals, to test a new drug on human subjects. An institution provides subjects with samples of the new drugs that can be taken in the institution or at the subjects' homes. However, in order for the correct testing of the drug, the samples have to be taken in well-defined periods of time and the institution must periodically monitor the subjects in order to determine the drug's effectiveness as well as the subjects' health. If any problem with any subject is reported during the testing phase, the competent authority must be promptly informed. In the USA, the responsible authority is the Food and Drug Administration (FDA). Moreover, the correct procedure for such collaboration is regulated by the U.S. Department of Health and Human Services related to drugs for human use, namely, by the Federal Regulations CFR21, Part 312 related to so called Investigational New Drug Application (IND) [23]. For instance, the paragraph 312.32 on IND safety includes explicit time constraints that must be followed in case of any unexpected, serious or life-threatening adverse drug experience:

> " (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 possi-

> ble 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."

There are two types of safety reports, namely, a short notification report transmitted by telephone or facsimile and a detailed written report. According to the paragraph above, once a serious and unexpected event is detected, a telephone or a facsimile transmission safety report, informing the nature of this event, should be send to the FDA in less than 7 calendar days, while a written report, containing full details of this event, should be send before 15 days. FDA performs regular random audits to all the sites participating in clinical trials and the conformity with the regulations is essential for the participation in such studies. If some sponsor or the institution carrying out the tests do not comply with the regulations, serious penalties might be issued. It is therefore important for collaborative systems to comply with regulations, which might involve time explicitly.

Similar time requirements also appear in other regulations such as in the Gramm-Leach-Bliley Act (GLBA) regulating how private information must be handled by financial institutions [15, 18] and in the Health Insurance Portability and Accountability Act (HIPAA) [14, 18] regulating how private infomation must be handled by health institutions.

In [33, 31, 28] models based on multiset rewrite systems were proposed to model collaborative systems. There a snapshot of the state of the world is specified by a multiset of facts, called configuration, while actions act as multiset rewrite rules that remove from and add facts to a configuration changing the state of the world.

This paper extends in three different ways the model used in [33, 31, 28] in order to include time explicitly. First, 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 where subjects, the FDA, and a sponsor collaborate, the timestamp t_1 of the fact $Dose(id)@t_1$ could denote that the subject whose anomynous identication number is id received a dose at time t_1. Alernatively, the timestamp of the fact $Deadline@t_2$ could denote the deadline of the test, that is, the date when the test should be over. Moreover, we keep track of time by assuming that each configuration has a global time, using the special fact $Time@t$, which denotes that the global time in a configuration is t. The global time advances from one configuration to another by using an action that replaces $Time@t$ by $Time@(t+1)$. For our second extension, we allow actions to contain a possibly empty set of time-constraints. These constraints are used as a guard of the action, *i.e.*, an action is applicable only if the associated time-constraints are satisfied. Finally, our third extension allows one to attach similar constraints to initial, goal and critical configurations. Goal configurations are the final configurations that agents want to achieve, *e.g.*, succesfully complete the test of the drug on all subjects, while critical configurations are configurations that the agents want to avoid, *e.g.*, not inform the FDA before the deadlines given above when an unexpected and serious event is detected. Initial configuration describes the state of the system at the beginning

of a collaboration and the attached time constraints could express for example that we start a certain task at a moment T, with guarantying that we get money at the moment $T + 99$ or finish the collaboration before $T + 200$. We later demonstrate that with these extensions one is able to model systems and regulations that mention time explicitly such as the scenario described above.

Besides allowing guards with time constraints, we also allow actions to have non-deterministic effects. In particular, actions are allowed to have a finite number of post-conditions specifying a finite number of possible resulting states. These actions are useful when specifying systems, such as CIs, whose actions may lead to different outcomes, but it is not certain beforehand which one of the outcomes will actually occur. For instance, when carrying out a blood test for the presence of some substance, it is not clear a priori what the test result will be. However, one can classify any result as either *positive* or *negative*. Depending on this result, one would need to take a different set of future actions. For example, if a the blood test is positive, then one might not be suitable for a participating as a subject in a particular CI, but may be suitable for other CIs. We classify actions that have more than one outcome as *branching actions*.

Given a system with explicit time, we are interested to determine whether this system complies with some set of regulations. In particular, we investigate the complexity of the three compliance problems introduced in [33, 31], called *plan*, *semi-critical plan* and *system* compliance problems.

For the complexity results we consider systems that satify the following three restrictions: (1) Time-constraints always compare the timestamps of exactly two facts; (2) The facts created by an action, that is, those facts that appear in it's postcondition, can only have timestamps of the form $T + d$, where T is the current global time and d a natural number; (3) All actions are *balanced* [33, 28], that is, their pre and posconditions have the same number of facts. As discussed in [33, 28], the third restriction intuitively means that agents have a bounded storage capacity, that is, they can remember at any given moment a bounded number of facts.

Under these three assumptions, we show that (1) all three compliance problems are PSPACE-complete if no branching actions are allowed and that (2) the plan compliance problem is EXPTIME-complete if branching actions are allowed.

Since we use natural numbers to denote time, there is an unbounded number of timestamps, which seems to preclude PSPACE membership. However, we can overcome this problem by relying on the conditions above. In particular, instead of natural numbers, we show that it is enough to use a small number of relative timestamps, which specifies the time difference between two facts in a configuration. This is shown in detail in Section 6. We also show that if any one of the conditions is relaxed, then all three compliance problems becomes undecidable. This implies that the three conditions above are indeed necessary for the decidability of the three compliance problems for timed systems.

This paper is structured as follows:

- Section 2 introduces the main vocabulary used throughout the paper as well as the formal definition of our multiset rewrite systems with explicit time, called Timed Local Transition Systems. In this section, the three compliance problems investi-

gated in this paper are adapted from the compliance problems introduced in [33, 31] in order to include explicit time.
- Section 3 describes in detail the restrictions described above that we impose on $TLSTS$es so to obtain a PSPACE-completeness result.
- Section 4 provides a formal semantics based in focused proofs of linear logic with definitions.
- Section 6 enters into the details of our PSPACE and EXPTIME-completeness and undecidability results for the compliance problems involving time.
- Section 7 demonstrates with examples the expressiveness of Timed Local Transition Systems. In particular, we specify a collection of collaborative systems, such as a drug test scenario described above.
- Section 8 comments on how to implement a $TLSTS$ in Maude, an existing rewrite system.
- Finally Section 9 compares our work with related work and Section 10 concludes by pointing to future work.

2 Preliminaries

In this section we extend the definition of local state transition systems introduced in [33, 31, 28] with explicit time.

Timed Local State Transition Systems At the lowest level, we have a first-order signature Σ that consists of a set of sorts together with the predicate symbols $P_1, P_2, \ldots$, function symbols $f_1, f_2, \ldots$, and constant symbols $c_1, c_2, \ldots$ all with specific sorts (or types). The multi-sorted terms over the signature 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 a ground, atomic predicate over multi-sorted terms.

However, in order to accommodate time in our model, we associate to each fact a *natural number* called *timestamp*. *Timestamped facts* are of the form $P(\boldsymbol{c})@t$, where t is the timestamp of the fact $P(\boldsymbol{c})$. The intuitive meaning of a timestamp may depend on the system one is modeling. For instance, in working groups, such as Wikipedia, the timestamp associated to a fact could denote the time when a user has entered a group or the time when a new object has been created. In other settings, such as in access control, it could denote the validity of a resource, such as a permission. 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. A *state*, or *configuration* of the system is a finite multiset W of timestamped facts, and it contains exactly one occurrence of the predicate $Time$. We use both $W\,X$ and W, X to denote the multiset resulting from the multiset union of W and X. For instance, the configuration

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

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

Definition 1. *The* size of a fact *is the total number of term and predicate symbols it contains. We count one for each predicate and function name, and one for each variable or constant symbol. We use $|P|$ to denote the size of a fact P.*

For example, $|P(x, c, x)| = 4$, and $|P(f(x, n), z)| = 5$. We will normally assume in this paper an upper bound on the size of facts, as in [22, 31, 33], which means that the arity of symbols and the depth of terms are bounded.

Following [31, 33], we assume that the global configuration is partitioned into different local configurations each of which is accessible only to one agent. There is also a public configuration, which is accessible to all agents. This separation of the global configuration is done by partitioning the set of predicate symbols in the signature and it will be usually clear from the context. The time predicate $Time$ is assumed to be public.

Actions work as multiset rewrite rules, that is, they replace in a multiset the facts in their pre-condition by the facts appearing in their postcondition. As in [31, 33, 28] we assume that each agent has a finite set of actions.

Time constraints However, we extend actions by attaching to them a finite set of *time constraints*. Time constraints are used to specify the time requirements of a system. Time constraints are arithmetic comparisons involving exactly two timestamps:

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

where a 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. In fact, we later show that the reachability problem is undecidable for systems with non-relative time constraints.

As desribed above, constraints in actions have the form $T1 \circ (T2 + \Delta)$, where $\circ \in \{>, \geq, =, \leq, <\}$ is an arithmetic comparison operator, Δ is a natural number, and $T1$ and $T2$ are variables appearing as timestamps of facts in the precondition of an action, which could include the global timestamp as well. We call the set of constraints associated to an action the *guard of the action*. An action can only be applied if all the constraints in its guard are satisfied, that is, they evaluate to true. Otherwise it cannot be applied.

Branching Actions Additionally, we extend actions by allowing actions to have non-deterministic effects. The actions that we propose in their most general form have the following shape:

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

The subscript A is the name of the agent that owns this action. W is the pre-condition of this rule, while $W_1, \ldots, W_n$ are its post-conditions. All facts in $W, W_1, \ldots, W_n$ are

public and/or belong to the agent A. Υ is the guard of the action consisting of a set of *time constraints*. All free variables appearing in a rule are assumed to be universally quantified. The existentially quantified variables specify the creation of fresh values, also known as nonces in protocol security literature. If $n > 1$, then we classify this action as *branching*, otherwise when $n = 1$ we classify this action as *non-branching*.

The first novelty to previous work [28] is that we allow actions to have a guard of time constraints, denoted by the set $\mathcal{C}$ in the rule above. Time constraints are boolean type operations involving time constraints appearing in the pre-condition the rule, *e.g.*, $T_1 \geq T_2 + 5$. Formally, an action cannot be applied if any of the time constraints in the guard is not satisfied. We elide the vertical bar whenever a rule has no time-constraints, such as in the time action above. The second novelty to previous work [28] is the occurrence of the disjunction symbols, $\oplus$, at the post-conditions of an action. It denotes that if an action is applied, then its outcome can be anyone of the disjuncts. These are intuitively used for specifying actions whose outcome is not certain before the action is applied. For instance consider the following rule specifying when one needs to repeat a urine analysis:

$$\begin{array}{r} Time@T, urine(I, Id, bad, none)@T_1 \mid T_1 \leq T \leq T_1 + 5 \\ \rightarrow_{\text{Nurse}} \ (Time@T, urine(I, Id, bad, ok)@T) \\ \oplus (Time@T, urine(I, Id, bad, high)@T) \\ \oplus (Time@T, urine(I, Id, bad, bad)@T) \end{array}$$

Its pre-condition specifies that one should only repeat a urine analysis for a subject Id if the result of the first test was bad. Moreover, the guard of this rule specifies that the nurse is only allowed to repeat this test in at most 5 days after the first test was performed. Otherwise repeating the test has no value in checking the effectiveness of the drug. Finally, the post-conditions of this rule specify that after one performs this action there can be three possible outcomes. Either the result of the second test is ok, $high$, or again bad. That is, the fact $urine(I, Id, bad, none)@T_1$ is replace by one of the facts below:

$$\begin{array}{c} urine(I, Id, bad, ok)@T, urine(I, Id, bad, high)@T, \\ \text{or } urine(I, Id, bad, bad)@T. \end{array}$$

When specifying administrative systems, one normally makes use of identifiers that uniquely identify, for example, a person or a transaction. Identifiers are also used to anomymize a person's name. For instance, for blinded CIs, a randomization process is performed before the trials starts, to assign either placebo or drug to subjects. One can specify such actions by using the existential quantifiers ($\exists x$) in actions. For instance, the following action could be used to specify the action of randomizing a subject.

$$\begin{array}{r} Time@T, per(N)@T_1, avail(placebo)@T_2, \ \rightarrow_{\text{RandomCo}} \\ \exists Id.[Time@T, sub(Id, N)@T, box(Id, placebo)@T] \end{array}$$

Its pre-condition specifies that the name of the person is N and that there is an available slot for subjects taking placebos. Its post-condition specifies that a fresh value,

Id, should be created and assigned to the subject with name N and that he will take placebo during the trials. Notice that in the post-condition the slot is no longer available since the fact $avail(placebo)$ is deleted. More complicated randomization rules used for double blinded tests can be specified in a similar fashion.

Past obligations can be specified in the pre-condition of actions. As specified by the FDA regulation Part 50, Subpart B, Section 50.20, a subject can only participate on the CIs if he signed a consent form. Therefore, his visits should only be book if this form has been signed. This is specified by the following action:

$$\begin{array}{l} Time@T, consent(N, yes)@T_1, schedule(N)@T_2, \\ sub(Id, N)@T_3, bgn@T_s \mid \mathcal{C} \ \rightarrow_{\text{Nurse}} \\ \quad Time@T, consent(N, yes)@T_1, sub(Id, N)@T_3, bgn@T_s, \\ \quad vital(0, Id, none)@T, visit(1, Id, no)@T_s, \ldots, \\ \quad visit(24, Id, no)@(T_s + 23 \times 28) \end{array}$$

where $\mathcal{C}$ is the time constraint $T_s \geq T \geq (T_s - 42)$. The timestamp, T_s, of the fact bgn specifies when the trials start. The constraint of this rule specifies that one has to book the visits at least 6 weeks before the the trials start. The action's pre-condition specifies that one can only book the visits if the subject has signed the consent form. Its postcondition specifies when the visits are scheduled and that one still has to test at screening time the vital signs of the subject.

Timed Initial, Goal and Critical Configurations We use the notation $W \rhd_{\mathcal{T}} U$ to mean that there is an action in $\mathcal{T}$ which can be applied to the configuration W to transform it into the configuration U. We let $\rhd_{\mathcal{T}}^{+}$ and $\rhd_{\mathcal{T}}^{*}$ denote the transitive closure and the reflexive, transitive closure of $\rhd_{\mathcal{T}}$ respectively. Usually, however, agents do not care about the entire configuration of the system, but only if a configuration contains some particular facts. Therefore we use the notion of partial reachability. We write $W \rightsquigarrow_{\mathcal{T}} Z$ or $W \rightsquigarrow_r Z$ to mean that $W \rhd_r ZU$ for some multiset of facts U to mean that $W \rhd_{\mathcal{T}} ZU$ for some multiset of facts U. For example with the action $r : X \mid \Upsilon \rightarrow_A Y$, we find that if its guard Υ is satisfied, then $WX \rightsquigarrow_r Y$, since $WX \rhd_r WY$. We define $\rightsquigarrow_{\mathcal{T}}^{+}$ and $\rightsquigarrow_{\mathcal{T}}^{*}$ to be the transitive closure and the reflexive, transitive closure of $\rightsquigarrow_{\mathcal{T}}$ respectively. We say that the partial configuration Z is reachable from configuration W using $\mathcal{T}$ if $W \rightsquigarrow_{\mathcal{T}}^{*} Z$. We also consider configurations which are reachable using the actions from all agents except for one. Thus we write $X \rhd_{-A_i} Y$ to indicate that Y can be reached exactly from X without using the actions of agent A_i.

We extend the notion of partial goals in [33] to include time constraints. Intuitively, a partial goal is a configuration which the agents want to reach. In particular, all timestamps appearing in the partial goal must be variables of the form $T, T_1, \ldots, T_n$. As with actions, goal configurations have a finite set of constraints associated, which also have the form $T1 \circ (T2 + \Delta)$, where $\circ \in \{>, \geq, =, \leq, <\}$ is an arithmetic comparison operator, Δ is a natural number, and $T1$ and $T2$ are variables appearing as timestamps of facts in the goal configuration, including the global timestamp.

For simplicity, we use the simpler term goal (respectively, critical, initial) configuration to refer to timed goal (respectively, critical, initial) configuration. We do the same with the compliance problems.

For instance, in the clinical investigations example, a possible goal configuration is when the data of a subject is collected in specified intervals for some number of times. The following goal configuration specifies that the goal is to collect the data of a subject 25 times in intervals of 28 days, but with a tolerance of 5 days.

$$\{Time@T, Data(Id, 1)@T_1, \ldots, Data(Id, 25)@T_{25}\}$$

with the time constraints $T_i + 23 \leq T_{i+1} \leq T_i + 33$ and that $T > T_i$, for $1 \leq i \leq 25$. Formally, any instantiation of the variables $T_1, \ldots, T_{25}$ that satisfy the set of constraints above is considered a goal configuration.

Notice that the global time can also appear in the time constraints. Formally, any instantiation of the variables T_1 and T_2 that satisfy the set of constraints above is considered as a goal configuration.

Similarly, we extend the notion of initial and critical states as done with goal configurations in order to include explicit time. That is, we add to initial and critical configurations a set of constraints. For instance, a configuration is critical for the participants of a clinical investigation when a problem is detected at time T_1, but the written report is not sent to the FDA on time, *i.e.*, within 15 days after the problem is detected:

$$\{Detect(Id)@T_1, Report(Id)@T_2\} \mid \{T_2 > T_1 + 15\}$$

For another example of a critical configuration, one should avoid at all times that, for instance, the lab-technician gets to know a subject's data with the subject's name as specified below:

$$\{Data(Id, N)@T_i, Lab(Id, Name)@T_2\} \mid \{\}$$

where Lab belongs to the lab-technician and the $Data$ is public and contains the data of a subject anonymized by the nonce Id.

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

Given an initial configuration W and a partial configuration Z, we call a *plan* any sequence of actions that leads from configuration W to a configuration containing Z. We classify a plan as *compliant* if it does not reach any critical configuration.

Linear and Branching Plans and Decision Problems As already pointed out, a key concern for both FDA audits and Sponsor/CRO monitors is to try to avoid as much as possible violations of regulation and deviations to the protocol. There are two *non-exclusive* ways of approaching this problem an *a priori* and/or an *a posteriori*. For the *a priori* approach, one can preemptively try to avoid violations and deviations by informing the participants of the actions they need to take. For the *a posteriori* approach, if a violation or a deviation is detected, the Sponsor/COR can take actions to countermeasure the situation. For example, the Sponsor/CRO may need to inform FDA, so to avoid penalties, or exclude a subject from the CI, so to not compromise the test results. Correspondingly, we identify two different decision problems, *plan generation*, an *a priori* approach, and *plan checking*, an *a posteriori* approach.

First, we formalize the notion of branching plans. Since actions may result in different post-conditions due to the $\oplus$ connective, plans may branch. Intuitively, each branch corresponds to one post-condition resulting from applying an action. Moreover, the number of sub-trees is exactly the number of post-conditions of the corresponding action.

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

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

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

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

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

A branching plan considers all possible outcomes of an action and specifies actions to be taken for each possible case and sub-cases. In particular, we will be interested in plans that do not reach any critical configuration and that each branch reaches a goal configuration. These objects are necessary for the *plan generation* problem.

However, once an action is *actually* performed no branching occurs, but the resulting state is obtained by one of the post-conditions. For instance, when an urine test is carried out, then the result is either ok, $high$ or bad. Hence, the input for plan checking is not a branching plan, but a linear one, where the outcomes of all actions are already known. We call these plans *linear plans* or *non-branching plans*. Formally, a linear plan is a *sequence* of configurations each annotated with an action name, act, and an natural number. The natural number specifies the post-condition of act used to obtain the following configuration.

• **Generation of Branching Compliant Plans:** Given a TLSTS, an initial configuration, a set of timed goal configurations and a set of timed critical configurations, is there a branching plan that contains no critical configuration and that all its branches reach a goal configuration? Generate such a plan.

• **Checking Compliance of Linear Plans:** Given a TLSTS, a set of timed critical configurations, and a linear plan, $\mathcal{P}$. Does $\mathcal{P}$ contain any critical configuration?

Branching vs Non-Branching Actions One might wonder about the differences of using a system with a branching action of the form shown in Eq. 2 or a system where this action is replaced by n actions with the same pre-condition, W, but each with a different post-condition $\exists \boldsymbol{x_i}.W_i$. That is, the differences of systems where the non-determinism

is internal and systems where the non-determinism is external. While we do not claim to have a general answer for this question, we believe that each system has its own advantages. For instance, branching plans contain more information, namely a plan for each possible outcome of actions, but non-branching plans might be easier to construct as only one plan for one outcome is built.

The exact trade-off between using branching or non-branching actions depends on the system being modeled. If plans were non-branching, then whenever the actual outcome of an action does not correspond to the one used in the plan, a participant of the collaboration would need to generate a new plan. Constantly generating new plans might very often be inconvenient, as it would generate too much overhead to a participant since he would need to stop temporarily what he is doing to obtain a new plan. On the other hand, generating branching plans might take too long, as an assistant would need to compute a plan for all possible outcomes of actions.

For clinical investigations, however, an *hybrid* approach seems more suitable. In particular, for actions whose outcomes have comparible probability, such as when testing the Urine of a subject, it is better to generate branching plans where a plan for each result is generated. On the other hand, for actions which have an outcome that is much less probable than the others, such as when a serious and unexpected problem is detected, we use non-branching plans. In this way, we minimize the overhead caused by non-branching plans and at the same time, reduce the computational effort in constructing plans.

3 Restricted $TLSTS$

The complexity results appearing in the next sections will consider a restricted form of $TLSTS$es where only two types of actions are allowed. The first type consist of the following action that increments the global time of a configuration:

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

The rule above is the only action that can modify the global time of a configuration and it is the only action that belongs to the agent $clock$. The action above does not have any constraints, which is specified by the empty set $\{\}$.

The second type of actions are those that belong to the other agents of the system and have necessarily the following form:

$$Time@T, W \mid \Upsilon \longrightarrow_A [\exists \boldsymbol{x_1}.Time@T, W_1] \oplus \cdots \oplus [\exists \boldsymbol{x_n}.Time@T, W_n] \tag{4}$$

where Υ is the guard of the action containing a finite set of constraints. We restrict actions so that all variables appearing in Υ are contained in the set of time variables $\{T_1, \ldots, T_n, T\}$ in the pre-condition. Notice that the fact $Time@T$ appears in the pre-condition, W, and in each of the post-conditions $W_1, \ldots, W_n$ exactly once. In that way the action above does not modify the timestamp of $Time$ representing the global time of the configuration, that is, actions are *instantaneous*. We will further impose the following condition on these actions: if $Time@T$ is in the pre-condition W, then all facts created in the post-conditions $W_1, \ldots, W_n$ are of the form $P@(T+d)$, where d

is a natural number, possibly zero. That is, all the created facts have timestamps greater or equal to the global time.

For instance, the following action is not allowed:

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

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

For instance, the following action specifies the action when a check-up is performed:

$$Time@T, Chk(id, ok)@T_1 \mid \{T \leq T_1 + d_2\} \rightarrow Time@T, Chk(id, result)@(T + 1)$$

Here, the $result \in \{ok, problem\}$. In order to collect data for the testing and to assess the health of the subject, the time between two check-ups must not be too long. In fact, they are assumed to be periodic, with periodicity d_2. However, if a subject feels any problem before his next check-up he can come earlier to the hospital so that some assistant can be provided, that is, a new check-up is performed. This is specified by the constraint $T \leq T_1 + d_2$. There are two outcomes of a check-up, either no unexpected and serious problem is detected, denoted by the fact $Chk(id, ok)$, or a problem is detected, denoted by the fact $Chk(id, problem)$. We assume that a check-up takes one hour to be completed, which is specified by the timestamp $T + 1$ in the postcondition of the rule.

Moreover, as in [28, 9, 22] actions can update values with fresh ones. For instance, the following action specifies that when an available slot is available a person p can be selected to join the drug test. We assume that at the beginning there is a fixed number, m, of available slots. This is specified by the facts $Avail(i)$, where $i \in \{1, \ldots, m\}$. Notice that the action does not have time-constraints.

$$\begin{array}{l} Time@T, Per(p)@T_1, Avail(X)@T_2, status((, X), no)@T_3, \\ \quad Chk(X, no)@T_4, Dose(X, no)@T_5 \mid \{\} \rightarrow_A \exists id. Time@T, Sub(id, p)@T, \\ \qquad bgn(id, no)@T, status((, i)d, no)@T, Chk(id, ok)@T, Dose(id, no)@T \end{array}$$

Once a subject is selected, one creates for him an identification id, which is fresh, and a file containing five facts: The fact $Sub(id, p)$ binds the identifier id with the person's name p. We assume here that this is information is kept private. $bgn(id, no/yes)@T$ specifies whether the subject id started taking the drug at time T or did not yet start taking the drug. The fact $Chk(id, result)@T$ denotes the result of the last check-up and that it was performed at time T, where $result \in \{ok, problem\}$. The fact $Dose(id, no/yes)@T$ denotes that the subject took the last dose at time T or that the subject did not yet take any doses. Finally, the fact $status((, i)d, cases)@T$ denotes the status of this subject with the FDA, where $cases \in \{no, notify, report\}$ indicates whether the hospital did not yet need to inform the FDA of any problems involving subject id (ok), or whether the hospital notified the FDA of a problem by telephone or a facsimilie ($notify$), or whether the hospital already sent the written report to the FDA ($written$). The timestamp T indicates the time when any of the two last events has occured.

Often for simplicity, we elide the guard of an action that does not have time constraints. Also we elide the label in the arrow with the agent's name, whenever it is clear from the context to whom an action belongs to.

Definition 2. *A* timed local state transition system *(TLSTS)* $\mathcal{T}$ *is a tuple* $\langle \Sigma, I, R_{\mathcal{T}} \rangle$*, where* Σ *is the signature of the language,* I *is a set of agents, such that* $\mathrm{clock} \in I$*, and* $R_{\mathcal{T}}$ *is a set of actions owned by the agents in* I *of the two forms described above, namely of the forms (4) and (3).*

We classify an action as *balanced* if the number of facts in its precondition is the same as the number of facts in its postcondition. As discussed in [33], if we restrict actions to be balanced, then the size of the configurations in a run remains the same as in the initial configuration. Since we assume facts to have a bounded size, the use of balanced actions imposes a bound on the storage capacity of the agents in the system.

For our complexity results we will focus on a class of *restricted* TLSTSes.

Definition 3. *We classify a* $TLSTS$ *as* restricted *if it satisfies the following conditions:*

1. *Each time constraint compares the values of exactly two timestamps;*
2. *The timestamps of the facts in the postcondition of actions are of the form* $T + d$*, where* T *is the global time and* d *a natural number;*
3. *All actions are balanced.*

Later we will show that if any one of the three conditions above is relaxed, then all compliance problems become undecidable.

Progressing behavior As argued in [30], many administrative or business processes also have a *progressing behavior*: whenever a transaction is performed, it does not need to be repeated at the same time. For instance, at the beginning of a subject's visit, the hospital staff receives a "to-do" list containing the sub-tasks necessary for collecting the subject's data. Once one has "checked" an item on the list, one does not need to return to this item anymore during the same visit. When all items have been checked, the visit is over.

There are many ways of specifying progressing behavior. For instance, in protocol security [22], progressing is enforced syntactically by using protocol state predicates. Here, on the other hand, we adapt our approach described in [30]: we classify a plan as *progressing* if and only if any of its branches only uses an instance of a rule *at most once at a given time*.

Planning Problem In [33, 31], three plan compliance problems were introduced in the setting without explicit time or branching (actions with non-deterministic effects). We now restate those 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 call 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 any goal configuration.

– (Timed plan compliance) 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) 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, and does there exist a plan leading from W to Z?
- (Progressing Timed plan compliance) 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 progressing plan which leads from W to Z?

In [31], the plan compliance problem version without explicit time was called weak plan compliance and a different problem was introduced with the name plan compliance. However, as shown in [29], it turns out that when considering balanced systems, the plan compliance problem introduced in [31] is equivalent to the weak plan compliance problem. For completeness, we also introduce the version of this problem with explicit timed, called timed semi-critical compliance.

- (Timed semi-critical plan compliance) 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 such that for each agent A_i and for each configuration Y along the plan, whenever $Y \rhd^*_{-A_i} V$, then V is not critical for A_i?

The name semi-critical is due to the fact that in that problem one is interested in determining the existence of plans that do not reach states, called semi-critical in [29], from which it is possible to reach a critical state of a particular agent. Notice that semi-critical states are not necessarily critical themselves.

4 Formal Semantics using Linear Logic with Definitions

This Section provides a formal semantics based on linear logic with definitions to the operational semantics of $TLSTS$es. In particular, we provide an encoding 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 [3] of its encoding.

The focused proof system for linear logic LLF is depicted in Figure 1 and was introduced by Andreoli [3]. In order to 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. There are two different sequents in LLF: those containing $\Uparrow$ which belong to the negative phase

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. 1. The focused proof system, LLF, for linear logic [3]. 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.

where only negative formulas are introduced, and those containing $\Downarrow$ which belong to the positive phase and only positive formulas are introduced.

A *definition* is a finite set of *clauses* which are written as $\forall \boldsymbol{x}[P(\boldsymbol{x}) \stackrel{\Delta}{=} B]$: here P is a predicate and every free variable of B (the *body* of the clause) is contained in the list $\boldsymbol{x}$. The symbol $\stackrel{\Delta}{=}$ 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.

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

The proviso for both of these rules is: $\forall \bar{x}[P(\boldsymbol{x}) \stackrel{\Delta}{=} B]$ is a definition clause and θ is the substitution that maps the variables $\boldsymbol{x}$ to the terms $\boldsymbol{c}$, 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:

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

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 formula 1. The proof theory of inference rules such as these is well studied (see, for example, [6, 38, 5]).

We show how to express the semantics of $TLSTS$es as search of 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:

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

where natural numbers are expressed by using the successor function s and the constant z 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 [42], 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 are positive. Therefore, once one focuses on one of the defined atoms above, one does not lose focus anymore and hence a proof of it consists necessarily of a single positive phase. For example, if we focus on the atom $s(z) \leq s(s(z))$ one obtains the following derivation:

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

At the open branch, the definition for the atom $z \leq s(z)$ 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 $s(z) \leq s(s(z))$. 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 $s(z) = z$. For a similar reason, to obtain a proof, one has to instantiate the variables x' and y' with z and $s(z)$, respectively. Otherwise, it is not possible to introduce the resulting equalities.

Using these definitions, we can easily encode a constraint of form $T_1 \circ T_2 + d$ as a the 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, but that uses the constants s and z. For instance, the natural number 2 is translated into the term $s(s(z))$ Notice that also 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. Then a fact $P(\boldsymbol{c})@t$ is encoded as the formula $P'(\boldsymbol{c}, t)$. We extend the definition of $\ulcorner\cdot\urcorner$ for constraints, natural numbers, and timestamped facts to multiset as usual.

To encode an action of the form $W \mid \Upsilon \rightarrow_A \exists \boldsymbol{u}.W'$ in linear logic, we first need to specify the timestamps of the form $T + d_i$ appearing in the postcondition W'. For this, we construct two sets W'_f and W'_c from W': for each fact of the form $Q_i@(T + d_i)$ in W' we add $Q_i@T_i$ in W'_f, where T_i is a new variable, and the formula $Plus(T, \ulcorner d_i \urcorner, T_i)$ to W'_c; and for each fact of the form $Q_i@(T)$ in W' we add the same fact to W'_f and no formula in W'_c. Intuitively, the set W'_c specifies the values for the new time variables used in W'_f to be the same as specified in the original timestamps in W. One could regard the set W'_c as a set of constraints to the new time variables introduced. For instance, the postcondition of the following action,

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

returns the sets $\{Time@T, P(u)@(T'_2), R(x)@T\}$ and $\{Plus(T, s(s(z)), T'_2)\}$.

Now, we are ready to encode actions in linear logic: an action of the form $W \mid \Upsilon \rightarrow_A \exists \boldsymbol{u}.W'$ is encoded as the linear logic formula

$$\forall \boldsymbol{x}[\bigotimes \ulcorner W \urcorner \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes W'_c \otimes q_A \multimap \exists \boldsymbol{t}. \bigotimes \ulcorner W'_f \urcorner \otimes q_A],$$

where $\boldsymbol{x}$ 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'_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 Since we are using a one-sided proof system, we use its negation in the one-sided LLF system with definitions:

$$F^{\perp} = \exists \boldsymbol{x}[(\bigotimes \ulcorner W \urcorner \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes W'_c \otimes q_A) \otimes (\forall \boldsymbol{t} \parr^{7} \ulcorner W'_f \urcorner^{\perp} \parr q_A^{\perp})].$$

Assume now that all atomic formulas have positive polarity, and consequently their negation negative polarity. The focused derivation introducing $F^{\perp}$ has to be necessarily of the form:

$$\dfrac{\dfrac{\vdash \Theta : \Delta \Downarrow \bigotimes \ulcorner W \urcorner \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes W'_c \otimes q_A \qquad \vdash \Theta : \Gamma \Downarrow \forall \boldsymbol{t} \parr \ulcorner W'_f \urcorner^{\perp} \parr q_A^{\perp}}{\vdash \Theta : \Gamma, \Delta \Downarrow (\bigotimes \ulcorner W \urcorner \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes W'_c \otimes q_A) \otimes (\forall \boldsymbol{t} \parr \ulcorner W'_f \urcorner^{\perp} \parr q_A^{\perp})}\ [\otimes]}{\dfrac{\vdash \Theta : \Gamma, \Delta \Downarrow F^{\perp}}{\vdash \Theta : \Gamma, \Delta \Uparrow \cdot}\ [D_2]}\ [n \times \exists]$$

Since $\otimes$ is a positive connective, the left-premise is necessarily introduced by a completely positive phase introducing all tensors in $\bigotimes \ulcorner W \urcorner \otimes \bigotimes \ulcorner \Upsilon \urcorner \otimes \bigotimes W'_c$ until one

[7] If you have problems seeing this symbol (big par), please use the CMLL fonts available at `http://iml.univ-mrs.fr/~beffara/soft/`.

only focuses on atomic formulas. There are then two types of atomic formulas: the first type are atoms that have a definition, such as $Plus$, which when focused on impose the constraints appearing in the action, specified by $\ulcorner\Upsilon\urcorner$, and compute the values of the timestamps of the facts in the postconditions, specified in W'_c. Moreover, as discussed above, these are proved without using any formulas from Δ. 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 $q_A^\perp$. 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:

$$\dfrac{\vdash \Theta : \Gamma, q_A^\perp, \ulcorner W'_f \urcorner^\perp \sigma \Uparrow \cdot}{\vdash \Theta : \Gamma, q_A^\perp, \ulcorner W \urcorner^\perp \sigma \Uparrow \cdot}$$

Notice that if the precondition 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.

The base case 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 R. To check whether this goal configuration is reached, we use the following formula, denoted as $\ulcorner R\urcorner$, $\exists \boldsymbol{x}. \bigotimes \ulcorner Z\urcorner \otimes \ulcorner \Upsilon \urcorner \otimes \top$, where $\boldsymbol{x}$ is the set of time variables appearing in the goal configuration. This formula is necessarily introduced by the following focused derivation:

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

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 $\ulcorner W\urcorner$, that all constraints in Υ are satisfied, and Γ the remaining facts in the sequent. That is, the current configuration is a goal configuration. Notice that with this encoding, we obtain a one-to-one correspondence between focused proofs and runs of the encoded timed local state transition system.

Given the discussion above, we prove the following connection between linear logic with definitions and reachability.

Theorem 1. *Let $\mathcal{T} = \langle \Sigma, I, R_\mathcal{T} \rangle$ be a timed local transition system. Let W be an initial configuration and R be a goal configuration under the signature Σ. Then the sequent $!\ulcorner R_\mathcal{T} \urcorner, Q_I, \ulcorner W \urcorner \vdash \ulcorner R \urcorner$ is provable in linear logic with definitions iff $W \rightsquigarrow^*_\mathcal{T} R$.*

5 Dealing with the Unboundedness of Time

Comparing our *timed* collaborative models here with the results on the *untimed* collaborative systems in our previous work, we meet here with a number of the crucial difficulties. In order to determine the existence of a compliant plan, one needs to face the problem that a priori there is no bound on the size of the time domain used to represent timestamps. For example, when one opens a bank account, one cannot determine a priori when the bank account is going to be closed. Moreover, during the whole period that the bank account is open, the policies regulating bank accounts should be respected. So in principle there is no bound on the timestamps that can appear in a plan. In fact, we assume here that the time domain is the set of natural numbers, which is infinite. Therefore a plan could involve an unbounded number of time units.

For a more 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 seems to preclude the PSPACE membership of the compliance problems previously introduced. [8]

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. In this section, however, we show how to overcome this problem by proposing an equivalence relation between configurations. The key idea is that since time constraints are relative, that is, they involve exactly two timestamps, we do not need to keep track of the concrete values for timestamps, but only enough in order to determine whether a time constraint is satisfied or not. We will focus on a class of $TLSTS$es defined in Section 2, which we call *restriced TLSTSes*. Recall that in restricted $TLSTS$es each time constraint compares the values of exactly two timestamps; also the timestamps of the facts in the postcondition of actions are of the form $T + d$, where T is the global time and d a natural number; and that all actions are balanced.

Later we will show that if any one of the three conditions above is relaxed, then all compliance problems become undecidable.

Truncated time differences In order to prove the PSPACE membership of the three compliance problems described above, we rely on the following the key observation. One does not need to keep track of the value of individual timestamps appearing in a configuration, but rather only of the differences between timestamps. We show that in

[8] In an orthogonal direction, a similar problem occurs when one allows actions that can update nonces with fresh ones. Since plans can be exponentially long, there can be exponentially many nonce names created. How to overcome this problem is described in our previous work [28], where one uses only a small number of nonce names. Therefore, in this section, we will not be concerned with the use of nonces as we assume implicitly that the techinques in [28] have been applied. However, we concentrate on how to solve the question mentioned above on how to bound the number of time constraints needed to check whether a system is plan compliant.

order to check whether a system is compliant one only needs to use a small number of numeric values representing time. In particular, we will store the time differences among the facts, but truncated by an upper bound. Formally, assume given a $TLSTS$ $\mathcal{T}$ and assume that D_{max} is the upper bound on the numbers appearing explicitly anywhere in $\mathcal{T}$, that is, in its time constraints, *i.e.*, the a in Eq. 1, and in its actions. Then the *truncated time difference* of two timestamped facts $P@T_1$ and $Q@T_2$, denoted by $\delta_{P,Q}$, is defined as follows:

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

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

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

Definition 4. *Given a planning problem with the model $\mathcal{T}$, let D_{max} be an upper bound on the the numeric values appearing in $\mathcal{T}$ and in the initial, goal and critical configurations. Let*

$\mathcal{S} = Q_1@T_1, Q_2@T_2, \ldots, Q_n@T_n$ *and* $\widetilde{\mathcal{S}} = Q_1@\widetilde{T}_1, Q_2@\widetilde{T}_2, \ldots, Q_n@\widetilde{T}_n$
be two configurations written in canonical way where the two sequences of timestamps $T_1, \ldots, T_n$ and $\widetilde{T}_1, \ldots, \widetilde{T}_n$ are non-decreasing. (For the case of equal timestamps, we sort the facts in alphabetical order, if necessary.)
Then $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ are equivalent *if for any $1 \leq i < n$ either of the following holds:*

$$T_{i+1} - T_i = \widetilde{T}_{i+1} - \widetilde{T}_i \leq D_{max}$$

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

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

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

According to the definition above, these configurations are equivalent since their truncated time differences are the same. This can be observed by the following *canonical* representation, called *δ-representation*. A δ-representation is constructed from a given configuration by sorting its facts according to their timestamps and sorting facts in alphabetical order (or any other order $\preceq$ among the facts). Ordering of facts simply serves as tie-breaker whenever two facts have the same timestamps, ensuring uniqueness of the δ-representation of each configuration. For a sorted list of facts we then compute the truncated time difference among two consequent facts, $\delta_{Q_i,Q_{i+1}}$.

Definition 5. *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. A* δ-representation *of* $\mathcal{S}$ *is a tuple*

$$\langle Q_1, \delta_{Q_1,Q_2}, Q_2, \ldots, Q_{m-1}, \delta_{Q_{m-1},Q_m}, Q_m \rangle$$

where $\delta_{Q_i,Q_{i+1}}$ *is the truncated time difference of the facts* $Q_1@T_1$ *and* $Q_2@T_2$.

For instance, both configurations $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ above have the following δ-representation:

$$\langle R, 1, P, \infty, Time, 1, Q, 2, S \rangle$$

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

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

Lemma 1. *Let* S *and* $\widetilde{S}$ *be two equivalent configurations from Definition 4.*
$\mathcal{S} = Q_1@T_1, Q_2@T_2, \ldots, Q_n@T_n$ *and* $\widetilde{\mathcal{S}} = Q_1@\widetilde{T}_1, Q_2@\widetilde{T}_2, \ldots, Q_n@\widetilde{T}_n$.
Then the following holds for all i *and* j *such that* $i > j$, *and for all* $a \leq D_{max}$*:*

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

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

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

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

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

If we advance time on both configurations, then the resulting configurations, $\mathcal{S}_1'$ and $\mathcal{S}_2'$, are not equivalent. This is because the truncated time difference $\delta_{\text{Time},P}$ is still ∞ in $\mathcal{S}_1'$,

while it changes to 3 in $\mathcal{S}'_2$. Notice that the same problem does not occur neither with present nor past facts, *i.e.*, those facts whose timestamps are less or equal to the global time.

Definition 6. *Given an upper bound D_{max}, a configuration $\mathcal{S}$ is called* future bounded *if for any future fact P in $\mathcal{S}$, the time difference $\delta_{Time,P} \leq D_{max}$.*

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

As per Definition 4 the initial configuration in a planning problem is future bounded, which from the lemma above implies that all configurations in a plan are also future bounded.

We are now ready to show the main result of this section. Namely, in a restricted $TLSTS$ we can determine when an action is applicable, or when a configuration is a goal configuration or a critical configuration, using its δ-representation.

Theorem 2. *For any given planning problem the equivalence relation between configurations given by Definition 4 is well-defined with respect to the actions of the system (including time advances) and goal and critical configurations. Any plan starting from the given initial configuration can be conceived as a plan over δ-representations.*

Proof (Sketch) Let $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ be two equivalent configurations. Assume that $\mathcal{S}$ is transformed in $\mathcal{S}'$ by means of an action α. By Lemma 1 the configuration $\widetilde{\mathcal{S}}$ also complies with the time constraints required in α, and hence the action α will transform $\widetilde{\mathcal{S}}$ into some $\widetilde{\mathcal{S}}'$. It remains to show that $\widetilde{\mathcal{S}}'$ is equivalent to $\mathcal{S}'$.

We consider our two types of actions. Let the time advance transform $\mathcal{S}$ into $\mathcal{S}'$, and $\widetilde{\mathcal{S}}$ into $\widetilde{\mathcal{S}}'$. From Lemma 2, we have that both $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ are future bounded and therefore $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ are trivially equivalent. For the second type of actions, namely the instantaneous actions belonging to agents, the reasoning is similar. Each created fact in the configuration $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ will be of the form $P@(T + d)$ and $P@(\widetilde{T} + d)$, where T and $\widetilde{T}$ are the global time in $\mathcal{S}$ and $\widetilde{\mathcal{S}}$, respectively. Therefore each created fact has the same difference d to the global time, which implies that these created facts have the same truncated time differences to the remaining facts. Hence $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ are equivalent.

Finally, also from Lemma 1, $\mathcal{S}$ is a goal (respectively, critical) configuration if and only if $\widetilde{\mathcal{S}}$ is a goal (respectively, critical) configuration. □

Definition 7. *A δ-representation of a configuration X is a* goal (resp. critical) *δ-representation if X is a goal (resp. critical) configuration.*

The theorem above establishes that using δ-representations for writing plans is well defined for restricted $TLSTS$es, but it does not establish a bound on the number of δ-representations. To achieve this, we need to use the assumption that all actions are balanced. Recall that balanced actions are actions that have the same number of facts in their pre and post conditions. By using balanced actions, the number of facts in any configuration of a plan is the same as the number of facts in the plan's initial configuration. Hence, we can also establish that there is a finite number of δ-representations. Later in the Section 6 we provide more precise bounds.

Corollary 1. *In the case of a model $\mathcal{T}$ with balanced actions, we can deal with the finite space of representatives of the form*

$$(Q_1, \delta_{12}, Q_2, \delta_{23}, Q_3, \ldots, Q_i, \delta_{i,i+1}, Q_{i+1}, \ldots, Q_m, \delta_{m,m+1}, Q_{m+1})$$

the size of each of the representatives is polynomial with respect to m, k, and $\log_2 D_{max}$, where m is the number of facts in the initial configuration, k is the upper bound on the size of facts, and D_{max} is the upper bound on the numeric values appearing in $\mathcal{T}$ and in the initial, goal and critical configurations.

Remark: Notice that the equivalence relation that we use in this section does not work for the progressing plan compliance problem (see end of Section 2). This is because for that problem the instances of actions used do matter. Therefore, we cannot construct progressing plans using only δ-representations as we cannot guarantee the progressing behavior.

6 Complexity Results

This section enters into the details of the complexity of the both the plan compliance problems and the progressing plan compliance problem described at the end of Section 2. Throughout this section, we assume that all actions are balanced, that is, actions have the same number of facts in their pre and post conditions, except in the last subsection where we consider not necessarily balanced systems.

First we consider the case for non-branching plans. We show that the progressing plan compliance problem is NP-complete when both time and the number of nonces are bounded. Then, we show that the plan compliance, semi-critical plan compliance and the system compliance problems are all PSPACE-complete without assuming any bounds on time nor on the number of nonces. Finally, we consider the case for branching plans. We show that the plan compliance problem is EXPTIME-complete also without assuming any bounds on time nor on the number of nonces. These results are summarized in Table 1.

Additionally, we investigate the complexity of the planning problem for systems that are not restricted. We show that relaxing any of the conditions from Definition 3 that classify restricted $TLSTS$es leads to undecidability. To be more precise, we first consider systems where the timestamps of facts created by an action are of the form $T + f_i(T_1, \ldots, T_n)$, where f_i is a polynomial and T_i are timestamps of facts in the precondition of the action, not only $T + d$, where d is a natural number, as in the restricted

Table 1. Summary of the complexity results for the plan compliance problems. We mark the new results appearing here with a $\star$. We also show that the *progressing* plan compliance problem for balanced non-branching actions is NP-complete provided there is an upper bound on the number of nonces and an upper bound on time.

Plan Compliance Problem		
Balanced Actions	Non-Branching	PSPACE-complete*
	Possibly Branching	EXPTIME-complete*
Not Necessarily Balanced Actions		Undecidable [31]

$TLSTS$es. Next, we relax the restriction on the guard of action 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_i are the timestamps appearing in the precondition of the action, and not only relative as in the restricted $TLSTS$es. Finally, we show that the plan compliance and semi-critical plan compliance problems for timed systems with possibly unbalanced actions are undecidable.

6.1 Progressing Plan Compliance with Non-Branching Actions only

We consider the complexity of progressing plan compliance problem assuming that all actions are non-branching and balanced, an upper bound on the size of facts, k, a fix bound, h, on the number of nonces, and a fix bound, l, on time. That is we assume that global time cannot be greater than l. We show that this problem is NP-complete.

Theorem 3. *Let $\mathcal{T}$ be a $TLSTS$ with balanced actions only. Then the progressing plan compliance problem is NP-complete with respect to the number of facts in the initial configuration, the upper bound on the size of facts, the upper bound on the number of nonces, and the upper bound on time.*

Proof The lower bound follows by encoding a 3-SAT problem which is well-known to be NP-complete [16] to the problem of reachability using transition rules and a bounded number of state variables. The proof given in [30] can be easily adapted to our setting with time. In particular, time does not play an important role in such an encoding.

For the NP upper bound, let $T = \langle \Sigma, I, R_T \rangle$ be a timed local transition system, such that n is the number of rules in R_T, d is the number of constant and function symbols in Σ, k is the upper bound on the size of facts, h is the upper bound on the number of nonces, l is the upper bound on time, and k_2 the upper bound on the number of different variables appearing in a rule. Therefore, there are $D = d + h + l$ terms in the alphabet that can be used for instantiating actions. Since the size of facts is bounded, we do not need to consider terms that have a size greater than k. Therefore we need to consider at most D^k terms. Since, in progressing plans, one is allowed to use only one instance of a rule, the length of plans is bounded by $n \times (D^k)^{k_2} = n \times D^{kk_2}$, which is polynomial on the number of rules and symbols. Assume that W is the initial configuration and that

one can check in *polynomial time* whether on not a configuration is critical or a goal configuration.

We show below that there is a polynomial-time algorithm that checks for compliant plans. Let S_i be the state at step i, so $S_0 = W$; R_i be the set of pairs, $\langle r, \sigma \rangle$, of rules and substitutions used before step i, so $R_0 = \emptyset$; and let the following rule $r_i : W \mid \Upsilon \rightarrow W'$ be used with the substitution σ_i at step i

1. Check if S_{i-1} is a critical state, then FAIL; otherwise continue;
2. Check if S_{i-1} is a goal configuration, then ACCEPT; otherwise continue;
3. Check if $W\sigma_i \in S_{i-1}$, then continue; otherwise FAIL;
4. Check if all time constraints in $\Upsilon\sigma_i$ are satisfied, then continue; otherwise FAIL;
5. Check if $\langle r_i, \sigma_i \rangle \in R_{i-1}$, then FAIL; otherwise continue;
6. $S_i = S_{i-1} \cup W'\sigma_i \setminus W\sigma_i$;
7. $R_i = R_{i-1} \cup \{\langle r_i, \sigma_i \rangle\}$;
8. Increment i.

Since the size of facts is bounded, all steps are done in polynomial time. The only step that may not be apparent is step 5. However, the set R_i is bounded by i, that is, by the length of the computation run. Therefore, the reachability problem is in NP. □

There are some informal similarities between our NP-completeness result and the NP-completeness result for the secrecy problem in protocol security when the number of protocol sessions is bounded [22, 2]. In particular, [22, 2] bound the number of protocol sessions and also bound the number of nonces. Bounding protocol sessions is intuitively also bounding the time of the system. However, there is a fundamental difference between our work in that we are concerned with collaborative systems with privacy, while [22, 2] are concerned in checking the safety of protocols.

6.2 Plan Compliance with Non-Branching Actions only

We now consider the planning problem when actions are non-branching and balanced and when the size of facts is bounded. Here, differently from our NP-completeness result above, there is no bound on the number of nonces nor a bound on time. We show that this problem is PSPACE-complete for each of the three compliances.

The comparison of the complexity results of the compliance problems for balanced systems without time and the restricted $TLSTS$es is shown in Table 2. One can notice that the explicit time has been added to collaborative systems without compromising the complexity of the problems.

Handling Nonces In our previous work [28], we showed that for solving the reachability problem using only balanced actions, it is enough to fix a priori a set of $2mk$ fresh constants, where m is the number of facts in the initial configuration and k is the upper bound on the size of facts. One can then simulate a plan containing an unbounded number of nonces by a plan containing only $2mk$ fresh constants. The main idea follows from the fact that when all actions are balanced, the number of facts in any configuration of a plan is the same as in its initial configuration, *i.e.*, m. Therefore, in any configuration of a plan there are at most mk slots for constant, function, predicate

Table 2. Summary of the complexity results for the planning problems for balanced systems with non-branching actions only. We mark the new results appearing here with a $\star$. For the *progressing* plan compliance problem with nonces we assume an upper bound on the number of nonces and for the timed progressing plan compliance problem we assume an upper bound on the number of nonces and an upper bound on time.

Planning Problems	$LSTS$es (No Time)		Restricted $TLSTS$es
	No fresh values	Possible nonces	Possible nonces
Progressing Plan	NP-complete [30]	NP-complete [30]	NP-complete*
(Weak) Plan	PSPACE-complete [33]	PSPACE-complete[28]	PSPACE-complete*
System	PSPACE-complete [33]	PSPACE-complete[28]	PSPACE-complete*
Semi Critical Plan	PSPACE-complete [33, 44]	PSPACE-complete[28]	PSPACE-complete*

and nonce symbols. Hence, when applying an action that requires nonces, instead of creating a new constant, we can re-use those values from the fixed set of $2mk$ constants that have been forgotten. Since timestamps cannot be instantiated as fresh values, the same idea applies in timed collaborative systems.

PSPACE-hardness The PSPACE-hardness of the compliance problems with explicit time can be inferred directly from the PSPACE-hardness results in [28] for systems without explicit time, where it was shown in [28] that one can faithfully encode a Turing machine with tape of size n using systems with balanced actions. The same idea works in our setting with time. It is easy to modify the encoding in [28]. Timestamps do not play any important role in such encoding. We do not repeat them here.

PSPACE upper bound It is more interesting to show that the plan compliance problem is in PSPACE when the size of facts is bounded and actions are non-branching and balanced. In particular, we will now use all the machinery introduced in Section 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 to a configuration whose δ-representation is the same as a previously reached configuration, there is a cycle of actions which could have been avoided. The following lemma imposes an upper bound on the number of different δ-representations given an initial finite alphabet. Such an upper bound provides us with the maximal length of a plan one needs to consider.

Lemma 3. *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 $\mathcal{T}$, then*

the number of different δ-representations, denoted by $L_T(m, k, D_{max})$, with m facts (counting repetitions) is such that

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

where J and D are, respectively, the number of predicate and the number of constant and function symbols in the initial alphabet Σ.

Proof Let $\langle Q_1, \delta_{Q_1,Q_2}, Q_2, \ldots, Q_{m-1}, \delta_{Q_{m-1},Q_m}, Q_m \rangle$ be a δ-representation with m facts. There are m slots for predicate names and at most mk slots for constants and function symbols. Constants can be either constants in the initial alphabet Σ or nonce names. Following [28], we need to consider only $2mk$ nonce names. 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. □

Clearly, the above upper bound on the number of δ-representations is an overestimate. It does not take into account, for example, the equivalence of configurations that have timestamped facts with multiplicity greater than one. For our purposes, however, it will be enough to assume such a bound.

Although the plan compliance problem is stated as a decision problem, we prove more than just PSPACE decidability. Ideally we are also able to generate a plan in PSPACE when there is a solution. Unfortunately, the number of actions in the plan may already be exponential in the size of the inputs, precluding PSPACE membership of plan generation. For this reason we follow [33] and use the notion of "scheduling" a plan in which an algorithm will also take an input i and output the i-th step of the plan.

Definition 8. *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 actions, then it outputs the i^{th} action of the plan, otherwise it outputs* no.

Following [33], we assume that when given an LSTS, there are three programs, $\mathcal{C}, \mathcal{G}$, and $\mathcal{A}$, such that they return the value 1 in polynomial space when given as input, respectively, a configuration that is critical, a configuration that contains the goal configuration, and a transition that is valid, that is, an instance of an action in the LSTS and whether its guard is satisfied, and return 0 otherwise.

Theorem 4. *Let $\mathcal{T}$ be a restricted $TLSTS$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 transition that is valid, that is, an instance of an action in the $TLSTS$, and return 0 otherwise.

Let m be the number of facts in the initial configuration W. Moreover, assume as inputs an upper bound, k, on the size of facts, an upper bound, D_{max}, on the numeric values appearing in the planning problem, that is in the given $TLSTS$ $\mathcal{T}$, in the initial,

goal and critical configurtions, programs $\mathcal{G}, \mathcal{C}$, and $\mathcal{A}$, as described above, and a natural number $0 \leq i \leq L_T(m, k, D_{max})$.

We modify the algorithm proposed in [28] in order to accommodate explicit time. The algorithm must return "yes" whenever there is compliant plan from the initial configuration W to a goal configuration, *i.e.*, a configuration S such that $\mathcal{G}(S) = 1$. In order to do so, we construct an algorithm that searches non-deterministically whether such configuration *is reachable*. Then we apply Savitch's Theorem to determinize this algorithm. However, instead of searching for a plan using concrete values, we rely on the equivalence described in Section 5 and use δ-representations only. Theorem 2 guarantees us that this abstraction is sound and faithful. From $\mathcal{G}, \mathcal{C}$, and $\mathcal{A}$, it is easy to construct new functions $\mathcal{G}', \mathcal{C}'$, and $\mathcal{A}'$ that use δ-representations instead of configurations. In particular, since time constraints associated to goal and critical configurations are also relative, these can be checked by using the truncated time differences in δ-representations.

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

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

We now show that this algorithm runs in polynomial space. We start with the step-counter i: The greatest number reached by this counter is $L_T(m, k, D_{max})$. When stored in binary encoding, this number takes only space polynomial to the given inputs:
$\log(L_T(m, k, D_{max})) \leq (m-1)\log(D_{max}+2) + m\log(J) + mk\log(D+2mk)$.
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 also be stored in polynomial space to the given inputs. Since our system is balanced, the size of facts is bounded, and the values of the truncated time differences are bounded, the size of any δ-representation, $\langle Q_1, \delta_{Q_1,Q_2}, Q_2, \ldots, Q_{m-1}, \delta_{Q_{m-1},Q_M}, Q_m \rangle$, in a plan is bounded.

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

Theorem 5. *Let $\mathcal{T}$ be a restricted TLSTSwith 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 [33] 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 4 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 configuration with timestamps 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 described in Section 5 and use δ-representations only. Theorem 2 guarantees us that this abstraction is sound and faithful.

We first need to check that none of the critical configurations are 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 in the δ-representation W_i. If no such action exists, then the algorithm outputs "no". Otherwise, we replace W_i by 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 3 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 4, 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 4, skipping the checking of critical states since we have already checked that none of the critical configurations are 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. □

As observed in [29], for systems without explicit timestamps the semi-critical plan compliance problem can be reduced to an instance of the plan compliance problem with a larger set of critical configurations that includes the set of *semi-critical configurations*. Intuitively, a semi-critical configuration for an agent A is a configuration from which a critical configuration for A could be reached by the other participants of the system without the participation of A. Therefore in the semi-critical plan compliance problem, a compliant plan not only avoids critical configurations, but also avoids configurations that are semi-critical. The same is true for systems with explicit time. Formally, the set of semi-critical configurations is given below:

Definition 9. *A configuration X is* semi-critical for an agent A *if a configuration Y that is critical for A is reachable using the actions belonging to all agents except to A,*

i.e., *if* $X \rhd^*_{-A} Y$. *A configuration is simply called* semi-critical *if it is semi-critical for some agent of the system.*

A δ*-representation of a configuration* X *is semi-critical for an angent if* X *is semi-critical for that agent.*

Theorem 6. *Let* $\mathcal{T}$ *be a restricted* $TLSTS$*with balanced non-branching actions. Then the semi-critical 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 The proof is similar to the proof of Theorem 4 and the proof of the PSPACE result of the semi-critical plan compliance for balanced systems in [44]. Again we rely on the fact that NPSPACE, PSPACE, and co-PSPACE are all the same complexity class.

Assume as inputs an initial configuration W containing m facts, an upper bound on the size of facts k, a natural number $0 \leq i \leq L_T(m,k)$, and programs $\mathcal{G}, \mathcal{C}$, and $\mathcal{A}$, that are slightly different to those in Theorem 4. This is because for semi-critical plan compliance it is additionally important to know to whom an action belongs to and similarly for which agent a configuration is critical. Program $\mathcal{A}$ recognizes actions of the system so that $\mathcal{A}(j,r) = 1$ when r is an instance of an action belonging to agent A_j and $\mathcal{A}(j,r) = 0$ otherwise. Similarly, program $\mathcal{C}$ recognizes critical configurations so that $\mathcal{C}(j,Z) = 1$ when configuration Z is critical for agent A_j and $\mathcal{C}(j,Z) = 0$ otherwise. Program $\mathcal{G}$ is the same as described earlier, *i.e.*, $\mathcal{G}(Z) = 1$ if Z contains a goal and $\mathcal{G}(Z) = 0$ otherwise. We modify these algorithms in the same way as described in the proof of Theorem 4 obtaining the algorithms $\mathcal{G}', \mathcal{C}'$, and $\mathcal{A}'$, which accept δ-representations of configurations.

Then we construct the algorithm ϕ that checks if a δ-representation, Z, is semi-critical for an agent. While guessing the actions of a compliant plan at each δ-representation Z reached along the plan we need to check whether for any agent A_j, the remaining agents can reach a δ-representation critical for A_j. More precisely, at δ-representation Z for each agent A_j the following nondeterministic algorithm looks for configurations that are semi-critical for the agent A_j, where $Z_0 = Z$:

1. Check if $\mathcal{C}'(j, Z_i) = 1$, then "yes"; otherwise continue;
2. Guess an action r and an agent $A_l \neq A_j$ such that $\mathcal{A}(l,r) = 1$ and that r is enabled in δ-representation Z_i; if no such action exists then "no";
3. Apply r to Z_i to get δ-representation Z_{i+1}. This is done by updating the positions of facts and the corresponding truncated time differences.

After guessing $L_T(m,k)$ actions, if the algorithm has not yet returned anything, it returns "no". We can then reverse the accept and reject conditions and use Savitch's Theorem to get a deterministic algorithm $\phi(j,Z)$ which accepts if every δ-representation V satisfying $Z \rhd^*_{-A_j} V$ also satisfies $\mathcal{C}'(j,V) = 0$, and rejects otherwise. In other words, $\phi(j,Z)$ accepts only in the case when Z is not semi-critical for agent A_j. Next we construct the deterministic algorithm $\mathcal{C}_2(Z)$ that accepts only in the case when Z is not semi-critical simply by checking if $\phi(j,Z)$ accepts for every j; if that is the case $\mathcal{C}_2(Z) = 1$; otherwise $\mathcal{C}_2(Z) = 0$.

Now we basically use the same approach as for the plan compliance problem considering all semi-critical δ-representations as critical by using the algorithm from the proof of Theorem 4 with the $\mathcal{C}_2$ as the program that recognizes the critical δ-representations.

We now show that algorithm $\mathcal{C}_2$ runs in polynomial space.

The algorithm ϕ stores at most two δ-representations at a time which are of the constant size, *i.e.*, same as the size of δ-representations of the initial configuration W. Also, the action r can be stored with two configurations. At most two agent names are stored at a time. Since the number of agents n is much less than the size of the configuration m, simply by the nature of our system, we can store each agent in space $\log n$. As in the proof of Theorem 4 only a polynomial space is needed to store in binary the values in the step-counter. Since checking if $\mathcal{C}'(j, Z_i) = 1$ and $\mathcal{A}'(j, r) = 1$ can be done in space polynomial to $|W|$, $|\mathcal{C}|$ and $|\mathcal{A}|$, algorithm ϕ and consequently $\mathcal{C}_2$, work in space polynomial to the given inputs.

We combine that with Theorem 4 to conclude that the semi-critical plan compliance problem is in PSPACE. □

6.3 Plan Compliance with possibly Branching Actions

In this section, we consider the plan compliance problem when actions may also be branching. In particular, we show that when actions are balanced then the plan compliance problem is EXPTIME-complete with respect to the number of facts, m, in the future bounded initial configuration, the upper bound, k, on the size of facts, the upper bound, D_{max}, on the numbers appearing explicitly in the theories, and the upper bound, p, on the number of postconditions of an action.

EXPTIME-hard The lower bound for the plan compliance problem can be inferred from a similar lower bound described in [34]. It was shown that one can encode alternating Turing machines [10] 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 future bounded initial configuration, the upper bound, D_{max}, on the numbers appearing explicitly in the theories, and the upper bound, p, on the number of postconditions of an 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 [10], we can infer that the plan compliance problem is in EXPTIME with respect the same parameters.

Theorem 7. *Given a balanced TLSTS $\mathcal{T}$ possibly containing branching actions, the plan compliance problem is in EXPTIME with respect to the number of facts, m, in the future bounded initial configuration, the upper bound, D_{max}, on the numeric values appearing explicitly in $\mathcal{T}$, and the upper bound, p, on the number of postconditions of actions in $\mathcal{T}$.*

Proof We exploit the fact that the complexity classes APSPACE and EXPTIME are equivalent [10] and show that the plan compliance problem can be solved by an alternating Turing machine in PSPACE.

As with the proof of Theorem 4, we will use the equivalence relation described in Section 5 by using the δ-representations of configurations. Theorem 2 ensures that such an abstraction is sound and complete. We also assume that it is possible to check in APSPACE whether a δ-representation is a goal δ-representation or a critical δ-representation and whether an action is valid.

We define the following function FIND(i, W_i), which takes a natural number, i, specifying the depth of a plan and a δ-representation, W_i, of a configuration. Recall from Lemma 3 that the depth of plan is bounded by $L_T(m, k, D_{max})$. Our upper bound algorithm is as follows: Initialize $i = 0$ and W_0 as the δ-representation of the initial configuration W. Then proceed as follows:

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

The fifth step is where we need the extra capabilities of an alternating Turing machine as we require that *all* executions of FIND return ACCEPT. Given the proof of Theorem 4 and the bound, p, on the number of postconditions of actions, it is easy to check that the alternating Turing machine runs in polynomial space. □

6.4 Relaxing Restricted $TLSTS$

In Section 2, we introduce two type of actions: the first type is the one that advances the global time and the second type of actions are those whose post-condition have timestamps of the form $T + d$, where T is the global time and d a natural number. One could think of generalizing the latter type of actions to be of the following form:

$$\begin{aligned} &Time@T, W, P_1(\boldsymbol{c}_1)@T_1, \ldots, P_n(\boldsymbol{c}_n)@T_n \mid \Upsilon \rightarrow_A \\ &\quad \exists u.Time@T, W, P'_1(\boldsymbol{c}'_1)@(T + f_1(T_1, \ldots, T_n)), \ldots, P'_m(\boldsymbol{c}'_m)@(T + f_m(T_1, \ldots, T_n)), \end{aligned} \tag{5}$$

where the timestamps of the facts created add to the global time, T, a polynomial $f_i(T_1, \ldots, T_n)$, which may contain the timestamps, $T_1, \ldots, T_n$, appearing as timestamps of facts in the actions precondition. 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 above, we show that all compliance problems discussed in Section 2 are undecidable already for systems with balanced actions that are linearly-time-advancing.

Theorem 8. *Given a TLSTS whose actions are balanced and linearly-time-advancing, the reachability problem is undecidable.*

Proof The undecidability proof follows by encoding a two counter Minsky machine [39] as a $TLSTS$. Let M be a standard two-counter machine that contains two registers r_1 and r_2 and the following four types of instructions:

- **(Add** r_i**)** ins_k: $r_i = r_i + 1$;**goto** ins_j;
- **(Substract** 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;

where r_i is either r_1 or r_2. We assume that each instruction has a different label ins_k, where ins_1 is the label of the first instruction and ins_0 is the label of the final instruction.

Then for each instruction label ins_k, we use a zero arity predicate Ins_k with the same name, denoting the next instruction to be performed. Following in the machine, we will make sure that there is exactly one instruction predicate in a configuration. Together with the timestamp of this predicate, we use the timestamps of the zero arity predicates R_1 and R_2 to keep track of the value stored in the registers r_1 and r_2, respectively. In particular, the timestamp, T, of the instruction fact $Ins_k@T_1$ is used as the base reference, so if the timestamp of the fact R_1 is T_1, then the value stored by the register is $T_1 - T$. For instance, the following configuration

$$\{Time@7, Ins_2@1, R_1@3, R_2@5\}$$

specifies that the next instruction performed by the machine is ins_2, and the values stored by R_1 and R_2 are, respectively, $3 - 1 = 2$ and $5 - 1 = 4$. The following rules encode M's instructions:

(Add r_1**)** $Time@T, Ins_k@T_1, R_1@T_2, R_2@T_3 \rightarrow_A$
$Time@T, Ins_j@T, R_1@(T + T_2 - T_1 + 1), R_2@(T + T_3 - T_1)$

(Add r_2**)** $Time@T, Ins_k@T_1, R_1@T_2, R_2@T_3 \rightarrow_A$
$Time@T, Ins_j@T, R_1@(T + T_2 - T_1), R_2@(T + T_3 - T_1 + 1)$

(0-test r_1 **if)** $Time@T, Ins_k@T_1, R_1@T_2, R_2@T_3 \mid \{T_1 = T_2\} \rightarrow_A$
$Time@T, Ins_{j_1}@T, R_1@T, R_2@(T + T_3 - T_1)$

(0-test r_1 **else)** $Time@T, Ins_k@T_1, R_1@T_2, R_2@T_3 \mid \{T_1 < T_2\} \rightarrow_A$
$Time@T, Ins_{j_2}@T, R_1@(T + T_2 - T_1), R_2@(T + T_3 - T_1)$

(0-test r_2 **if)** $Time@T, Ins_k@T_1, R_1@T_2, R_2@T_3 \mid \{T_1 = T_3\} \rightarrow_A$
$Time@T, Ins_{j_1}@T, R_1@(T + T_2 - T_1), R_2@T$

(0-test r_2 **else)** $Time@T, Ins_k@T_1, R_1@T_2, R_2@T_3 \mid \{T_1 < T_3\} \rightarrow_A$
$Time@T, Ins_{j_2}@T, R_1@(T + T_2 - T_1), R_2@(T + T_3 - T_1)$

(Jump) $Time@T, Ins_k@T_1, R_1@T_2, R_2@T_3 \rightarrow_A$
$Time@T, Ins_j@T, R_1@(T + T_2 - T_1), R_2@@(T + T_3 - T_1)$

It is easy to show that each action faithfully encodes the corresponding instruction in M. For instance, consider the first action above encoding the instruction **(Add** r_1**)**. At the precondition the value stored in the registers r_1 and r_2 are, respectively, $T_2 - T_1$ and $T_3 - T_1$. In the postcondition, however, since the facts have to advance in time, the timestamp of the fact Ins_j, denoting the next instruction, is changed to the current global time T. Therefore, the timestamp 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 increase by one.

Also notice that the **(0-test** r_i**)** instructions are split into two actions: one where the test is satisfied (**(0-test** r_i **if)** and when the test is not satisfied **(0-test** r_i **else)**. The goal is to reach a configuration that reaches the final instruction ins_0, which is encoded by the fact Ins_0.

For soundness, the only possible problem could the rule 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 resgister, 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 timestamp of facts Ins_k, encoding instructions, are always zero. Hence, the 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 the compliance problems mentioned in Section 2 are undecidable.

Corollary 2. *Given an TLSTS with balanced actions that are linearly-time-advancing, then the plan, system, and semi-critical plan compliance problems are all undecidable.*

Instead of relaxing the timestamps of the facts in the postcondition, we now investigate relaxing the constraints in the guard of actions. In our formalism, these constraints are limited to be of the form $T_1 \circ T_2 + \Delta$, where $\circ \in \{>, \geq, =, <, \leq\}$ and Δ 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 action. We call these type of actions as linearly-contrained actions. We show that the reachability problem for $TLSTS$es with balanced and linearly contrained actions is undecidable.

Theorem 9. *Given TLSTSes whose actions are balanced and linearly-contrained, then the reachability problem is undecidable.*

Proof As in the proof of Theorem 8, we encode the two-counter Minsky machine [39] as a $TLSTS$. Besides the $Time$ predicate, we will require the following zero arity predicate names: Aux, Add, Sub, Jmp, $Zero$, Ins, R_1, and R_2. As before, the difference between the timestamps of R_1 (respectively, R_2) and Ins will denote the value of the register r_1 (respectively, r_2). We use the auxiliary predicate Aux, $Add^k_{i,j}$, $Sub^k_{i,j}$, $Jmp^k_{i,j}$, and $Zero^k_{i,j}$ together with the time predicate to execute the instructions, such as add a value to the register. The initial configuration consists of six facts with two copies of Aux:

$$\{Ins_1@0, R_1@0, R_2@0, Aux@0, Aux@0, Time@0\}.$$

where we assume w.l.o.g. that the values in both registers is zero. Each instruction is encoded by using a collection of auxiliary actions. Below are the the rules for the adding the register r_1, given below:

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

The first three rules handle the case when the value in the register $r_1 + 1$ is smaller than the value in the register r_2. This is handled by the constraint $T_2 > T_1 + 1$ in that rule. On the other hand, the last three rules handle the case when this is not the case. We explain the first three rules. The remaining rules are symmetric. The intruction of adding one to the register r_1 starts by rewriting one of the Aux facts into the next intruction Ins_j. Now, we will use the time to compute the new values for the registers r_1 and r_2, but they will be relative to the timestamp T of the next instruction Ins_j. Notice that we impose that T is greater than T_3, so that the timestamp of the new instruction is greater than the timestamp of the old instruction. This is necessary so that we can identify (in the following rules) which instruction is the old one and which one is the new one. Since the value of the register $r_1 + 1$ is less than the value of r_2, the timestamps of R_1 has to be updated first. This is represented by the fact $Add^1_{1,2}$. The superscript denotes which register we are increasing by one, in this case the register r_1, and the subscript denotes the order in which the timestamps need to be updated, in this case the timestamp of R_1 before the timestamp of R_2. Since the fact $Add^1_{1,2}$ is in the configuration, the only rule applicable is **(Add r_1 2)**. This rule is only applicable when the current time corresponds exactly to the new value of the timestamp of R_1 with respect to the timestamp, T_5, of the next intruction Ins_j, that is, if the global time is $T_5 + T_1 - T_3 + 1$. The constraint $T_3 < T_5$ guarantees that we are not mixing the old instruction with the new one. Once the value of the timestamp of R_1 has been updated, one needs to update the value of the timestamp of the fact R_2. This is denoted by the fact $Add^1_{0,2}$, where the 0 marks that the first timestamp has been updated. Now the rule **(Add r_1 3)** is applicable but only when the global time corresponds to the correct value for the timestamp of R_2, which is specified by the constraint $T = T_5 + T_2 - T_3$. Once this action is applied, we replace the old instruction and the fact $Add^1_{0,2}$ by two Aux facts so that the new instruction can be executed. The remaining three rules behave in a similar way.

(Add r_1 1) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_2 > T_1 + 1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Add^1_{1,2}@T, Ins_j@T$

(Add r_1 2) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Add^1_{1,2}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_1 - T_3 + 1\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, Ins_k@T_3, Add^1_{0,2}@T, Ins_j@T_5$

(Add r_1 3) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Add^1_{0,2}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, Aux@T, Aux@T, Ins_j@T_5$

(Add r_1 4) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_2 \leq T_1 + 1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Add^1_{2,1}@T, Ins_j@T$

(Add r_1 5) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Add^1_{0,1}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, Ins_k@T_3, Add^2_1@T, Ins_j@T_5$

(Add r_1 6) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Add^1_{0,1}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_1 - T_3 + 1\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, Aux@T, Aux@T, Ins_j@T_5$

The rules of the other instructions follows in a similar fashion, but where the . We show below only the rules encoding instructions involving the register r_1. The rules for instructions involving register r_2 are similar.

(Sub r_1 1) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_2 > T_1 - 1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Sub^1_{1,2}@T, Ins_j@T$

(Sub r_1 2) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Sub^1_{1,2}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_1 - T_3 - 1\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, Ins_k@T_3, Sub^1_{0,2}@T, Ins_j@T_5$

(Sub r_1 3) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Sub^1_{0,2}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, Aux@T, Aux@T, Ins_j@T_5$

(Sub r_1 4) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_2 \leq T_1 - 1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Sub^1_{2,1}@T, Ins_j@T$

(Sub r_1 5) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Sub^1_{0,1}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, Ins_k@T_3, Sub^2_1@T, Ins_j@T_5$

(Sub r_1 6) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Sub^1_{0,1}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_1 - T_3 - 1\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, Aux@T, Aux@T, Ins_j@T_5$

(Zero r_1 1 if) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_1 = T_3, T_2 > T_1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Zero^1_{1,2}@T, Ins_{j_1}@T$

(Zero r_1 1 else) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_1 > T_3, T_2 > T_1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Zero^1_{1,2}@T, Ins_{j_2}@T$

(Zero r_1 2) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Zero^1_{1,2}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_1 - T_3\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, Ins_k@T_3, Zero^1_{0,2}@T, Ins_j@T_5$

(Zero r_1 3) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Zero^1_{0,2}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, Aux@T, Aux@T, Ins_j@T_5$

(Zero r_1 4 if) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_1 = T_3, T_2 \leq T_1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Zero^1_{2,1}@T, Ins_j@T$

(Zero r_1 4 else) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_1 > T_3, T_2 \leq T_1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Zero^1_{2,1}@T, Ins_j@T$

(Zero r_1 5) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Zero^1_{0,1}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, Ins_k@T_3, Zero^2_1@T, Ins_j@T_5$

(Zero r_1 6) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Zero^1_{0,1}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_1 - T_3\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, Aux@T, Aux@T, Ins_j@T_5$

(Jmp 1) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_2 > T_1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Jmp^1_{1,2}@T, Ins_j@T$

(Jmp 2) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Jmp^1_{1,2}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_1 - T_3\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, Ins_k@T_3, Jmp^1_{0,2}@T, Ins_j@T_5$

(Jmp 3) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Jmp^1_{0,2}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, Aux@T, Aux@T, Ins_j@T_5$

(Jmp 4) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Aux@T_4, Aux@T_5 \mid \{T > T_3, T_2 \leq T_1\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Jmp^1_{2,1}@T, Ins_j@T$

(Jmp 5) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Jmp^1_{0,1}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_2 - T_3\} \rightarrow_A$
$Time@T, R_1@T_1, R_2@T, Ins_k@T_3, Jmp^2_1@T, Ins_j@T_5$

(Jmp 6) $Time@T, R_1@T_1, R_2@T_2, Ins_k@T_3, Jmp^1_{0,1}@T_4, Ins_j@T_5 \mid \{T_3 < T_5, T = T_5 + T_1 - T_3\} \rightarrow_A$
$Time@T, R_1@T, R_2@T_2, Aux@T, Aux@T, Ins_j@T_5$

For our encoding, it is important to notice that due to the different auxiliary constants, such as $Add^j_{i,k}$, there are only a fixed number of ways the rules are applied. For instance, if the instruction to be executed is **(Add r_1)** and if the value in the register r_1 plus one is less than the value of the register r_2, then one necessarily executes the sequence of rules **(Add r_1 1)**, **(Add r_1 2)**, followed by **(Add r_1 3)**. Notice as well that a new instruction is executed only when the old one is completed. This is because in

order to start a new instruction, one requires two Aux facts, which can only happen when such a sequence of rules is applied. Moreover, after these rules are executed, the timestamps of the facts $R_1@T_1$, $R_2@T_2$, and $Ins_j@T_3$ are exactly as expected. That is, the value $T_1 - T_3$ and $T_2 - T_3$ are exactly the values in the registers r_1 and r_2, respectively. This is because of the time-constraints in these rules appearing in these rules, *e.g.*, $T = T_5 + T_1 - T_3 + 1$ and $T = T_5 + T_2 - T_3$. Hence we have a sound and complete encoding of any two-counter Minsky machine. □

6.5 Decision problems for not necessarily balanced $TLSTS$

We next show that the plan compliance and semi-critical plan compliance problems for timed systems with possibly unbalanced actions is undecidable. This is shown by faithfully encoding the two counter Minsky Machine in a similar way as in [31].

Theorem 10. *The plan compliance problem and the semi-critical plan compliance problem for general TLSTSes are undecidable.*

Proof (Sketch) We modify the encoding in [31] to accomodate timestamped facts by introducing timestamps but at the same time keeping the time still, *i.e.*, not using the action that advances the global time. A standard two-counter Minsky machine M in the initial configuration $(a_1; n, 0)$ is translated into to an LSTS T_M given by the following actions for agents A and B without constraints:

$$
\begin{aligned}
S_j^A@T &\rightarrow_A R^A@T\ C_2(b_j)@T \\
S_j^A@T &\rightarrow_A R^A@T\ R_1@T\ C_2(b_j)@T \\
S_j^A@T\ R_1@T &\rightarrow_A R^A@T\ C_2(b_j)@T \\
S_j^A@T &\rightarrow_A R^A@T\ C_2(b_{\tilde{j}})@T \\
S_j^A@T\ R_1@T &\rightarrow_A R^A@T\ R_1@T\ C_2(b_k)@T \\
R^A@T\ C_1(a_i)@T &\rightarrow_A S_j^A@T \\
\\
S_j^B@T &\rightarrow_B R^B@T\ C_1(a_k)@T \\
S_j^B@T &\rightarrow_B R^B@T\ R_2@T\ C_1(a_k)@T \\
S_j^B@T\ R_2@T &\rightarrow_B R^B@T\ C_1(a_k)@T \\
S_j^B@T &\rightarrow_B R^B@T\ C_1(a_{\tilde{k}})@T \\
S_j^B@T\ R_2@T &\rightarrow_B R^B@T\ R_2@T\ C_1(a_l)@T \\
R^B@T\ C_2(b_j)@T &\rightarrow_B S_j^B@T
\end{aligned}
$$

by the initial state $R^A@0\ R_1^n@0\ R^B@0\ C_1(a_1)@0\ Time@0$, by critical configurations $C_1(a_{\tilde{i}})@T$ and $C_2(b_{\tilde{j}})@T$ and by the goal configuration $C_1(a_0)@T$, where a terminating computation of M is one that ends in a configuration $(a0; *, *)$, that is, the final state a_0 with any values in the counters. As per definition of TLSTSes, we also need to add the agent $Clock$ with no critical configurations attached and his action:

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

As in [31] it can be shown that the computation of Minsky machine M terminates if and only if T_M has a matching compliant plan. The soundness of the encolding follows the argument in [31] exactly. For the completeness we additionally point out that the rules, goal and critical configurations of T_M do not have any constraints attached. Therefore we can easily transform any compliant plan into another compliant plan that does not use the action for the progress of time by removing such actions form the plan and consequently keeping all timestamps 0. □

7 Examples

In this section we present several examples that motivated us to extend our previous work on models for collaborative systems [28] with explicit time. We are particularly interested in properties that involve time, namely *explicit time intervals* and *past provisions*. We introduced Timed Local State Transition Systems (TLSTS), which allow one to mention time explicitly, and argue by examples that TLSTSes can be used for expressing temporal properties of given scenarios, for example for specifying clinical investigation protocols and the relevant regulations. Later in Section 8, we also demonstrate that TLSTS specifications can also be easily implemented in Maude [13], an existing computational tool based on rewriting. Therefore, in principle, the construction of a prototype of an automated asistant is within reach.

Types of Properties While studying the FDA regulations [23], we noticed that many clauses mention time explicitly. Such clauses served as motivation for extending with explicit time our previous work on models for collaborative systems [28]. We briefly describe two types of properties that we have identified, namely *explicit time intervals* and *past provisions*. Similar properties in the context of GLBA financial regulations are considered in [18]. (The emphasis in the quotes below is ours.)

Explicit Time Intervals Consider the following clause appearing in the Federal Regulations CFR21, Part 312 [23] for *Investigational New Drug Applications* (INDs).

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

The clause above mentions explicitly two different time intervals. The first is that that one must send a detailed safety report to the FDA within 15 days, while the second obligation is that one must notify FDA of such an event within 7 days. These time intervals are used to specify *future obligations*, such as the obligation of sending a safety report.
Protocols mention explicit time intervals as well. For example, a protocol might specify that if some parameter of an urine test is three times above the upper limit, then the same test is repeated with a fresh sample *within 5 days* in order to make sure that the first result is not an isolated result.

Past Provisions Regulations and protocols also often enforce that to apply some action one must have satisfied some conditions in the past. A typical example of such requirement is that a subject can only participate in a CI if he has signed an informed consent. This is specified in Part 50 – Protection of Human Subjects, Subpart B, Section 50.20, quoted below:

> " Except as provided in 50.23 and 50.24, no investigator may involve a human being as a subject in research covered by these regulations *unless the investigator has obtained the legally effective informed consent* of the subject or the subject's legally authorized representative. [$\cdots$] "

That is, a test can only be performed on a subject if he has signed an informed consent.

7.1 Investigational New Drug Application Example

There is little doubt that drugs have improved in several ways the quality of life of the population. One can now rely on a huge variety of drugs for different symptoms, from simple pain killers to drugs for diabetes or HIV patients. A great deal of this success is due to the careful understanding of the effectiveness of a drug through experimentation. Since drugs may affect people's health, before a drug is made available to the general public, one is required to perform experiments in order to determine its effectiveness. At the final stages of testing, one is often required to test the drug on human subjects. These procedures are called *Clinical Investigations* (CIs) [23]. Only when CIs are successfully carried out and the effectiveness of the drug is experimentally validated may the drug be approved by public agencies, *e.g.*, the Food and Drug Administration (FDA), to be made available to the general public.

There are two key concerns while carrying out CIs. The first concern is of assuring that the subjects' health is not compromised by the test. In order to protect the subjects' health, CIs are rigorously regulated and audited by (FDA) inspectors. Violations of FDA regulations may imply heavy penalties, both financial as well as of bad public relations.[9] The second key concern is to collect enough data to determine the effectiveness of the drug. Without such data, it is most likely that the drug will not be allowed by public agencies (such as the FDA) to be commercialized.

Normally, a pharmaceutical company (Sponsor) that wants its drug to be tested hires a Clinical Research Organization (CRO) which is specializing in carrying out CIs. Then a detailed plan, called a *protocol*, is elaborated by specialists explaining how CIs should be carried out in order to obtain the most conclusive results without compromising the health of the subjects involved. For instance, the protocol normally contains the number of subjects that need to be used in the CI, or the type of CI, *e.g.*, blind, double-blinded, comparison with placebo, etc, as well as the duration of the CI.[10] In order to carry out a CI, the Sponsor/CRO collaborate with health institutions, typically hospitals, which have the necessary means, *i.e.*, competent people and the equipment, to perform the clinical research and collect the required data. At each site, a Principal Investigator is assigned and is in charge of the clinical trial carried out at the site.

[9] FDA maintains a list available online of the name of companies that have in the past violated regulations [24].

[10] To illustrate the complexity of some CIs, typical protocols are more than 100 pages long.

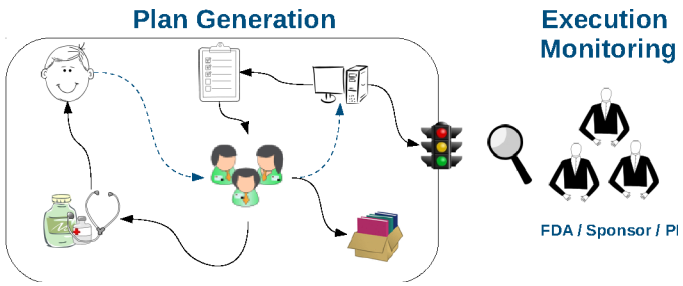

Fig. 2. Diagram illustrating the two main applications of an automated assistant. Here the smiley figure is a subject, the green symbols the study team, the to-do list a plan, the computer our assistant, the lower-left symbols the medical procedures, while the box contains the physical files, *e.g.*, reports, and the three black suited agents are the FDA inspector, the Sponsor's monitor, and the health institution's PI.

During a CI it is the responsibility of the Sponsor/CRO and the PI from the health institution to (1) make sure that there are no violations of the regulations imposed by public agencies, such as the FDA, which could lead to penalties, and (2) that there are no deviations from the protocol, which could lead to data that is not conclusive for determining the effectiveness of the drug. Deviations could also compromise the safety of the subjects, which should be avoided at all costs. Furthermore, health institutions that have a record of deviations in previous CIs may be penalized by the market by not being hired for carrying out future CIs.

Although all participants of a CI, *e.g.*, physicians, study coordinators, pharmacists, and nurses, are informed of and even trained in the protocol and in the regulations, errors do occur. At each site it is common to have more than one ongoing trial, typically 5-10 trials, with different procedures, and they take place along with normal hospital duties. Furthermore, as all subjects are not enrolled in a study at the same time, it is difficult to have a clear overview of the process. Therefore, the process is prone to human error. It is easy to lose track, for example, that some subject has to be called for follow up or for an unscheduled visit. In order to avoid such errors, a monitor from the Sponsor/CRO that has in-depth knowledge of both the regulations and of the protocol monitors the ongoing trials on site and *manually* checks the available records in the health institutions to identify situations that could lead to regulation violations or to protocol deviations. A much better approach would be to have an automated assistant that helps to correctly carry out CIs.

It has been shown that computer tools can be used to support carrying out administrative tasks. For instance, the tool Maude [13, 27], which we use in this work, has been successfully used for planning and workflow formalisms. Other formalisms include the PROforma clinical knowledge representation and workflow language and successors used for modeling health care processes, clinical guidelines, error handling and safe delegation [26]. However, to our knowledge there is no automated assistant on location that helps the medical team or the monitors in carrying out CIs.

For CIs, we identify two possible direct uses for such a tool. The first application is called *plan generation*. Plan generation preemptively avoids violations and deviations

from occurring. For instance, once a nurse inputs a new test result in the system, she can be informed of the sequence of actions that need to be performed next, *e.g.*, carry out additional tests. The second application is called *execution monitoring*, that is, to check whether there have been violations and/or deviations in the past. When such violations or deviations are detected, the Sponsor/CRO/PI could react by, for instance, informing the corresponding authority, *e.g.*, FDA, and excluding incompetent sites or personnel from the CI or by excluding subjects from the trial.

Figure 2 illustrates how an automated tool could help to carry out CIs. For plan generation, when the subject arrives the health institution, the staff register the subject's visit in the automated tool. Then the tool generates a plan with the sequence of tasks that have to be performed during the visit. These tasks may include carrying out exams and providing drugs or filling out reports, such as safety reports. Moreover, whenever a task is finished, the staff may input the results in the tool and new plans might be generated. Since the staff is always informed by the tool of the actions that need to be taken, the chances of errors are reduced.

On the other hand, since the tool is aware of the actions taken by the staff, it can also detect when deviations and violations occur by execution monitoring. Once such events occur, the monitor or the PI can be informed, so that they can take the necessary actions, such as double-check the physical records. For instance, the tool could trigger an alarm sign (the traffic light sign) informing the responsible people that a problem has occurred. Such a tool could also be used by FDA inspectors in their first steps of an audit visit to have an overall view of the hospital procedures and the steps taken so far. The FDA audits can then decide which records and processes they want to take a closer look at.[11]

While we are still far from completely understanding the challenges of developing a full scale automated assistant, we provide the first steps in that direction. Our main contribution is to propose a model that can be used to specify regulations and protocols and that at the same time can be implemented in existing tools.

A simplified Clinincal Investigation scenario Next we present a simplified example of a Clinincal Investigation scenario. The following protocol synopsis for the trial contains enough details to illustrate the main tasks and concerns of the parties involved, in particular of the study team on location. The table specifies the visits that need to be perforemed together with the corresponding tests and some conditions in which additional tests need to be performed.

[11] One could imagine more sophisticated provenance mechanisms that allow inspectors to even see on his monitor the electronic/scanned versions of the physical files.

STUDY SYNOPSIS

Protocol Number:	RP20110516
Version Number:	1
Protocol Title:	A Randomized, Multicenter, Double-Blind, Placebo-Controlled, Comparison Study evaluating the Efficacy and Safety of XXIP in Female Subjects between the age 20 – 50 with children between the age 1 -20 , suffering from intermittent headaches
Study Phase:	3
Rationale for the Study:	A double-blind, placebo-controlled, safety and efficacy study in 500 female subjects between the age 20 – 50 with children suffering from intermittent headaches.
Study Design:	Multicenter, parallel-group, randomized, placebo-controlled, double-blind study.
Study Location:	North America, Europe, and rest of world
Study Objectives:	The primary objective of this study is to determine whether the compound XXIP when compared with placebo, is effective in reducing the proportion of intermittent headaches in subjects with children between the 1 - 20 at 2 years. The secondary objectives of this study are: To determine whether XXIP, when compared with placebo, is effective in: • reducing the total number of headaches in two years; • reducing the rate of headache episodes at 1 year
Number of Planned Subjects:	500 subjects
Study Population:	Female subjects between the age 20 – 50 that have two children between the age of 0 – 20 without any major clinical diseases who experienced at least 10 headaches that needed medical treatment (prescription or/and over the counter medicine) within 1 year of screening.

Treatment Groups:	Subjects will be randomized into 1 of 2 groups in a 1:1 ratio. Subjects will receive 2 capsules orally 3 times a day on the day of the first headache occurrence after headache symptoms. Group 1 250 subjects will receive XXIP, 240 mg TID (2 capsules [120 mg each] 3 times a day). Group 2 250 subjects will receive placebo 2 capsules 3 times daily. Throughout the study the subject will not be allowed in using any other pain relief medication to treat headaches unless consulted with the medical monitor (prescription and/or over the counter) for the duration of the study. Patients taking any psychiatrical medication are not eligible to participate in the study.
Visit Schedule:	1st Year of Study • Screening Visit • Baseline Visit • Study site visits once every 4 weeks, Visits 1 through 12 • Premature Study Withdrawal Visit (if necessary) • Unscheduled Relapse Assessment Visits (if necessary) within 72 hours of the onset of any headaches 2nd Year of Study • Study site visits once every 4 weeks, Visits 13 through 24 • End of Study Visit on Visit 25 • Premature Study Withdrawal Visit (if necessary) • Unscheduled Relapse Assessment Visits (if necessary)
Efficacy Assessments:	Clinical assessments, Psychology assessment, Quality of life questionnaires
Safety Assessments:	Physical examination, including vital signs, 12-lead electrocardiogram (ECG), blood chemistry, hematology, urinalysis, adverse event monitoring.
Additional Assessments:	SF-36, EQ-5D Quality of Life questionnaire, psychology/psychiatry assessment
Statistical Analysis:	"not subject of this paper"
Interim Analysis:	"not subject of this paper"

1 STUDY ACTIVITIES RP20110516

1.1 Study Activities RP20110516—Chart 1 of 3

Tests and Assessments	Screening Visit (within 6 Weeks of Baseline)	Baseline Visit (Day 1)	Visit 1 (Week 4 ±5d)	Visit 2 (Week 8 ±5d)	Visit 3 (Week 12 ±5d)	Visit 4 (Week 16 ±5d)	Visit 5 (Week 20 ±5d)	Visit 6 (Week 24 ±5d)	Visit 7 (Week 28 ±5d)	Visit 8 (Week 32 ±5d)	Visit 9 (Week 36 ±5d)	Visit 10 - 24 (±5d)	Unscheduled Visits	Premature study withdrawal	End of study visit Visits	Follow up visit[5] (every 45 days)
Informed Consent	X															
Randomization		X														
Medical History	X															
Physical Examination	X	X						X					X	X	X	
Neurological examination	X												X	X	X	
Transcranial Doppler Ultrasound	X[1]	X											X	X	X	
Psychological Evaluation	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Vital Signs	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
12-Lead ECG		X						X					X	X	X	
Hematology	X	X	X	X	X			X			X		X	X	X	
Blood Chemistry	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
Serum Pregnancy Test	X															
Urine Pregnancy Test		X	X	X	X	X	X	X	X	X	X	X	X	X	X	
Urinalysis	X	X	X[2]	X[2]	X[2]	X[2]	X[2]	X[2]	X[2]	X[2]	X[2]	X[2]	X[2]	X[2]	X[2]	
Nephrology evaluation	X	X	X[3]	X[3]	X[3]	X[3]	X[3]	X[3]	X[3]	X[3]	X[3]	X[3]	X[3]	X[3]	X[3]	
SF-36 and EQ-5D	X	X		X		X		X		X		X	X	X	X	X
Psychiatric Evaluation	X												X	X	X	
Dispense Study Treatment[4]		X	X	X	X	X	X	X	X	X	X	X	X	X	X	
Cocomitant Therapy, Adverse Events		Monitor and record throughout the study														

[1] – done only if pathology found in neurological head exam to confirm no desease

[2]– if values are 3 times above the upper limit of the refent values , a second test has to be performed within 5 days

[3] – done only if the second urinalysis shows values above the upper limit of the referent values

[4] – study drug should be dispenced only if subjct suffers from headache

[5] - to be performed only in subjects who were discontinued before the end of the study

TLSTS encoding The encoding of a simplified scenario where an hospital performs a the test of a new drug is shown in Figures 4, 5, 6 and 7. Figure 3 shows the signature of this example, that is lists the predicate names and the constants with the intended semantics.

We assume that at the beginning there is a fixed number of subjects, m, to be selected. This number is usually big enough so that one can collect enough data to determine the effectiveness of the drug and is specified in the protocol. We assume that m

- emp – Denotes an empty memory slot in the system.
- $Time$ – Time of the current state;
- $avail(placebo/drug)$ – There is an available slot for either a subject that will take only placebo or a subject that will take the drug;
- $per(name)$ – A person's name;
- $sub(id, name)$ – The subject called $name$ has been assigned the identification number id;
- bgn– Specifies when the drug trial goes into effect, *i.e.* when it effectively begins;
- $consent(name, yes/no)$ – Fact specifying that the person called $name$ has or has not signed the consent form;
- $scrTests(id, blank/ok/notOk)$ – Denotes that the subject with identification number id either needs to perform screening tests, or that the test have been done and the results are in conformity with the protocol requirements, or that they are not satisfactory;
- $visit(0, id, no/yes/done)$ – Denotes that the subject with identification number id either needs to perform baseline visit tests, or that the baseline visit has started, or that the baseline visit finshed, *i.e.* tests have been done and the drug provided;
- $schedule(Id)$ – Denotes that the next visit for the subject with identification number Id needs to be booked;
- $visit(1-24, id, no/yes/done)$ – Denotes that number of visits that the subject with identification number id has made and how many still remain;
- $urine(1-24, id, bad/high/ok/none, bad/high/ok/none)$ – A subject that participates the whole test during two years will have at least 24 urine analysis test performed. There are three possible results. The result might be under the expected normal levels (ok), or slightly above the normal levels ($high$), or three times the upper limit levels (bad)or more. If the first result is bad, then one has to repeat the test. If the result of the second test is again above the upper limit, that is, either $high$ or bad, then the nephrology evaluation has to be performed.
- $nephro(1-24, id, result)$ – Similarly as with Urine analysis, but one is not required to repeat the test.
- $vital(0-24, id, result/none)$ – Similar to Nefro evaluation, however, one also performs a vital signs test at the screening phase;
- $dose(0-24, id, yes/no, B)$ – Denotes whether the subject was given the box of doses or not. The identification number id should match the box number B. Here we are assuming that subjects takes a box every visit.
- $status(id, ok/hold/stop)$ – Specifies the status of a subject. There are three types of status. Either the subject is ok, that is, he does not have any problems related to the drug trial, or he has had some problem related to the drug trial and he is put on hold for some type, or he had a serious problem and is excluded from the test.
- $fda(id, resolved/report/notify/no, ;)$

Fig. 3. Signature of the CI example

Time Action:

$$Time@T \;\rightarrow\; Time@(T+1)$$

Screening Visit

Select Candidate:

$$Time@T, per(N)@T_1, place@T_2, bgn@T_s \mid T_s \geq T \;\rightarrow_N\; Time@T, sub(blank, N)@T, consent(N, no)@T, bgn@T_s$$

Informed Consent:

$$Time@T, consent(N, no)@T \;\rightarrow_N\; Time@T, consent(N, yes)@T$$

Anonymize:

$$Time@T, sub(blank, N)@T, consent(N, yes)@T \;\rightarrow_N\; \exists Id.Time@T, sub(Id, N)@T, consent(N, yes)@T, scrTests(Id, blank)@T$$

Screening Tests:

$$Time@T, scrTests(Id, blank)@T \;\rightarrow_N\; [Time@T, scrTests(Id, ok)@T] \oplus [Time@T, scrTests(Id, notOk)@T]$$

Select Subject:

$$Time@T, scrTests(Id, ok)@T \;\rightarrow_N\; Time@T, scrTests(Id, ok)@T, visit(0, Id, no)@T$$

Reject Candidate:

$$Time@T, scrTests(Id, notok)@T, consent(N, yes)@T, sub(Id, N)@T \;\rightarrow_N\; Time@T, per(N)@T, place@T$$

Fig. 4. Actions specifying the selection of a candidate.

is given. We also assume that at the beginning no person has been selected as a subject and that therefore there are m free slots, denoted by facts of the form $place@0$. The first action of the screening visit, "Select candidate", in Figure 4 specifies the selection of a candidate for the trial, denoted by the fact $per(p)$, where p is the candidate's name. If this person satisfies the study requirements, *i.e.* passes the screening visit, he will fill one of the available slots. The action can only be triggered once the trial has gone in effect, which is specified by the fact $bgn@T$ and the time constraint of the action, $T_s \geq T$, which ensures that at the current time the trial is in effect. The postcondition of the action specifies that the candidate is a potential subject in the trial and that he needs to sign an informed consent form. Signing of the informed consent is specified by the next action, resulting in the fact $consent(N, yes)$ denoting that the person named N has signed the informed consent form.

In order to anomynize the persons name, a fresh value Id is created and assinged to the person, specified by the fact $sub(Id, N)$. The Id is a unique identifier for the subject. It will ensure confidentiality during the screening visit. The fact $scrTests(Id, blank)$, denoting that the person with the identifier Id needs to perform the screening tests, contains only the identifier while the name of the person is not be available to the technicians performing the laboratory tests. Results of the test performed during the screening visit are either satisfactory for the trial (denoted by the fact $scrTests(Id, ok)$) or do not meet the study requirements specified by the protcol (denoted by the fact $scrTests(Id, notOk)$). In the former case the candidate is included in the study and needs to schedule for the baseline visit next (denoted by the fact $visit(0, Id, no)$), while

Baseline Visit

Randomization:

$Time@T, sub(Id, N)@T_1, visit(0, Id, no)@T_2, avail(X)@T_3 \mid T \leq (T_2 + 6 \times 7) \rightarrow_{\text{RandomCo}}$
$\exists B.[Time@T, sub(Id, N)@T, visit(0, Id, yes)@T, baselineTests(Id, no)@T,$
$box(Id, B)@T, blind(Id, B, X)@T]$

Baseline Tests:

$Time@T, baselineTests(Id, no)@T, \rightarrow_N Time@T,$
$baselineTests(Id, done)@T, status(Id, ok)@T, dose(0, Id, no, none)@T$

Provide Dose:

$Time@T, dose(0, Id, no, none)@T, box(Id, B)@T_2 \rightarrow_N Time@T,$
$dose(0, Id, yes, B)@T, box(Id, B)@T_2, schedule(Id)@T$

Visit End:

$Time@T, schedule(Id)@T, visit(0, Id, yes)@T \rightarrow_N Time@T,$
$visit(0, Id, done)@T, visit(1, Id, no)@(T + 28), status(Id, ok)@T$

where X is either $placebo$ or $drug$.

Fig. 5. Actions encoding time progress, screening and baseline visits

in the later case the candidate is rejected and therefore there is a free place in the study made available for another candidate (a fact $place@T$ is created). Protocol permiting, even the same person might try to enrole in the study at the later date.

The time constraint of the first action ("Randomization") in the set of baseline visit rules, shown in Figure 5, makes sure that the baseline visit is performed within the specified deadline, that is within 6 weeks of the screening visit. A randomization company will assign either the drug or the placebo to each of the subject. Drug and placebo distribution will conform with the given ratio between placebo and the drug (50%-50% in our example). This is specified by the multiple facts $avail(placebo)$ and $avail(drug)$ specified in the initial configuration. Furthermore, the distribution is purely casual since, when both the facts $avail(placebo)$ and $avail(drug)$ are present in the configuration, either of these facts can be "used" for firing an instance of the "Randomization" action. The choice of which instance of the rule will be fired is nondeterministic. As a result of the randomization process, each subject will be given a number B that will appear on the treatment box (denoted by the fact $box(Id, B)$). The trial in our example is double-blind, therefore the randomization company will store the confidential information on who is getting what (denoted by the fact $blind(Id, B, X)$), *i.e.* what is hidden behind the number B, the drug or placebo, while the treatment box will only be marked with the number B. Again, because of the confidentiality concerns, only the identifier will be shown to the laboratory personnel during the baseline test, see fact $baselineTests(Id, blank)$. Once all the baseline tests have been performed (denoted by the fact $baselineTests(Id, done)$), the subject can be given the first box of treatment, which is represented by the action "Provide dose." The action models the dispension of the correct box for the subject in question and points to scheduling of the next visit by the fact $schedule(Id)@T$. The visit ends by scheduling of the first regular visit denoted

by the fact $visit(1, Id, no)@(T + 28)$, where the timestamp shows that the visit shuold take place in about 28 days.

Scheduled Visits

Visit:

$$Time@T, visit(I, Id, no)@T_1, status(Id, ok)@T_2 \mid \{T_1 + 5 \geq T \geq T_1 - 5\} \\ \rightarrow_N Time@T, visit(I, Id, yes)@T, status(Id, ok)@T_2, urine(I, Id, none, none)@T, vital(I, Id, none)@T$$

Visit Hold:

$$Time@T, visit(I, Id, no)@T_1, status(Id, hold)@T_2 \mid C \\ \rightarrow_N Time@T, status(Id, hold)@T_2, visit(I, Id, yes)@T, schedule(Id)@T$$

Vital Signs:

$$Time@T, vital(I, Id, none)@T \rightarrow_N Time@T, vital(I, Id, result)@T$$

Urine Analysis:

$$[Time@T, urine(I, Id, none, none)@T] \rightarrow_N [Time@T, urine(I, Id, ok, none)@T] \\ \oplus [Time@T, urine(I, Id, high, none)@T] \\ \oplus [Time@T, urine(I, Id, bad, none)@T]$$

Repeat Analysis:

$$Time@T, urine(I, Id, bad, none)@T_1 \mid T_1 \leq T \leq T_1 + 5 \rightarrow_N \\ [Time@T, urine(I, Id, bad, ok)@T] \\ \oplus [Time@T, urine(I, Id, bad, high)@T, nephro(I, Id, none)@T] \\ \oplus [Time@T, urine(I, Id, bad, bad)@T, nephro(I, Id, none)@T]$$

Nephrology Eval:

$$Time@T, nephro(I, Id, none)@T \rightarrow_N Time@T, nephro(I, Id, result)@T$$

Test End 1:

$$Time@T, vital(I, Id, result)@T_1, urine(I, Id, OkHigh, none)@T_2 \rightarrow_N Time@T, \\ vital(I, Id, result)@T_1, urine(I, Id, OkHigh, none)@T_2, dose(I, Id, no, none)@T$$

Test End 2:

$$Time@T, vital(I, Id, result)@T_1, urine(I, Id, bad, ok)@T_2 \rightarrow_N Time@T, \\ vital(I, Id, result)@T_1, urine(I, Id, bad, ok)@T_2, dose(I, Id, no, none)@T$$

Test End 3:

$$Time@T, vital(I, Id, result)@T_1, urine(I, Id, bad, HighBad)@T_2, nephro(I, Id, result)@T_3, \rightarrow_N \\ Time@T, vital(I, Id, result)@T_1, urine(I, Id, bad, HighBad)@T_2, \\ nephro(I, Id, result)@T_3, dose(I, Id, no, none)@T$$

Provide Dose:

$$Time@T, dose(I, Id, no, none)@T, box(Id, B)@T_2 \rightarrow_N Time@T, \\ dose(I, Id, yes, B)@T, box(Id, B)@T_2, schedule(Id)@T$$

Visit End:

$$Time@T, schedule(Id)@T, visit(0, Id, done)@T_1, visit(I, Id, yes)@T_2 \rightarrow_N Time@T, \\ visit(0, Id, done)@T_1, visit(I, Id, done)@T, visit(I + 1, Id, no)@(T_1 + I \times 28)$$

where $C = \{T_2 + 5 \geq T \geq T_2 - 5, T < T_3 + d_h\}$ and where $OkHigh$ is either ok or $high$ and $HighBad$ is either $high$ or bad.

Fig. 6. Actions encoding scheduled visits

The first rule in Figure 6 ("Visit") has a time constraint attached that ensures that the scheduled visits take place every 4 weeks with the allowed deviation of + \ - 5 days. The rule initiates the I-th visit by replacing the fact $visit(I, Id, no)$ with the fact $visit(I, Id, yes)$. It also creates facts $vital(I, Id, none)$ and $urine(I, Id, none, none)$ which denote that the vital signs test and the urine test for that visit need to be done. "Vital signs" rule represents the process of taking vital signs and storing of the results (denoted by the fact $vital(I, Id, result)$). "Urine Analysis" rule is a branching rule modeling all possible outcomes of the urine test. As per protocol specification, if the result of this test is bad, that is if the values are 3 times above the upper limit of the refent values, a second urine test needs to be perfomed and that is represented with the "Repeat Analysis" rule. Then, if the second urine analysis shows values above the upper limit of the referent values, the nephrology evaluation needs to be perfomed, *i.e.* the "Neprolgy Eval" rule is applied. Rules "Test End 1", "Test End 2" and "Test End 3" encode checking if all required tests have been performed. Three rules are required because, under specified conditions, additional urine anylysis and the nephrology evaluation have to be done. In the first case, the results of the urine tests were not bad and no additional test were necessary, while in the second case the first urine test resutls were bad but the second test resutls were ok. Finally, in the third case, even the nephrology evaluation was required. All of the three rules check if the vital signs have been taken as well. Only in case all of the above conditions are satisfied, can the new box of doses be given to the corresponding subject (denoted by the fact $dose(I, Id, no, none)$). The action "Provide Dose" can then be applied, giving the right box to the subject (denoted by the fact $dose(I, Id, yes, B)$ where identification number and box number B match the fact $box(Id, B)$). This rule also points to scheduling of the next visit. Finally, the "Visit End" rule models the completion of the visit (denoted by the fact $visit(I, Id, done)$) and schedules the next visit. Notice that the visit cannot be finalized ($done$) in all the required test haven't been performed and the doses provided.

The "Visit hold" rule models scheduled visits for patients that are on hold at the time of the visit (denoted by the fact $status(Id, hold)$), since scheduled visits need to be done even for patients that are on hold. The status of the patients that have had no serious adverse problems have the status ok, while the patients that have been excluded from the trial have the status $stop$. For example, if the subject had an allergic reaction to the drug he might be excluded from the trial, but in case there is doubt on what had caused the allergy, the subject can be out on hold for some time till the allergy symptoms disappear and then it will be checked again if the same allergic reaction appears when the subject is given the drug.

Safety Reports

Notify FDA: $Time@T, fda(Id, no, Num)@T_1 \rightarrow_N Time@T, fda(Id, notify, Num)@T$

Safety FDA: $Time@T, fda(Id, notify, Num)@T_1 \rightarrow_{PI} Time@T, fda(Id, safety, Num)@T$

Fig. 7. Actions related to safety reports

The last two actions shown in Figure 7 specify, respectively, the actions taken when a serious adverse problem is detected: in the former the FDA is notified (by telephone or facsimile) of the problem, while the latter a detailed written report is sent to the FDA. These are denoted, respectively, by the facts $fda(Id, notify, Num)$ and $fda(Id, safety, Num)$.

The goal of this collaboration is to test the new drug and collect enough data to assess its effectiveness. On the other side, a key concern for FDA inspectors as well as for PI/Sponsor/CRO is to try to avoid violations of regulations and deviations from the protocol. Such situations are represented through critical configurations, while a a possible goal configuration is such that enough subjects have completed all required visits and on time. This can be specified by the following goal configuration where we assume that at least 450 subjects need to finish the study correctly:

$$\left\{ \begin{array}{c} visit(0, id_1, done)@T_{1,1}, \ldots, visit(24, id_1, done)@T_{24,1} \\ \ldots \\ visit(0, id_{500}, done)@T_{1,500}, \ldots, visit(24, id_{500}, done)@T_{24,500} \end{array} \right\}$$

and the set of time-constraints of the form

$T_{1,j} + (i-1) \times (28) - 5 \leq T_{i,j} \leq T_{1,j} + (i-1) \times (28) + 5$

for $1 \leq i \leq 24$ and $1 \leq j \leq 450$. The constraints specify that the time of the visits occurred every 4 weeks after the time of the first visit (also called baseline visit) $T_{1,j}$ with a tolerance of 5 days.

Timed critical configurations can be used to specify future obligations. For instance, the future obligation that a second urine test should be carried out in at least 5 days if the result of the first test was very high (three times the upper-limit) is specified by the following critical configuration:

$\langle \{Time@T, urine(I, Id, bad, none)@T_1\}, \{T > T_1 + 5\} \rangle$.

If the second test has not been carried out within the next five days, then the configuration is critical, that is, it should be avoided.

Similarly, all the scheduled visits need to be performed on time, with the + \ - 5 days time frame, therefore the following configuration is critical:

$\langle \{Time@T, visit(I, Id, no)@T_1\}, \{T > T_1 + 5\} \rangle$.

For another example, the FDA regulation CFR21, Part 312 specifies that one must notify FDA if any serious or unexpected event has been detected within 7 days. This future obligation can be specified by critical configurations of the following form:

$\{Time@T, detected(Id, Num)@T_1, fda(ID, none, Num)@T_2\}$

with the time constraint $T > T_1 + 7$. This critical configuration specifies that a problem has been detected at time T_1, but the FDA was neither notified nor a safety report was sent in 7 days time.

Other critical configurations include those specifying that exams should be performed in the same day as the subject's visits, except for the second Urine test. The reader can find them in the Maude implementation of this scenarion available at [40].

7.2 GLBA and HIPAA

Often in legislation one requires some conditions involving time. We take a closer look at the Gramm-Leach-Bliley Act (GLBA) [15] and Health Insurance Portability and Accountability Act (HIPAA) [14], which specify how private data should be manipulated

$$Time@T, Sent(no, fi, c, form)@T_1 \mid \{\} \rightarrow_{fi} Time@T, Sent(yes, fi, c, form)@T$$

$$Time@T, Sent(yes, fi, c, form)@T_1, Rec(no, c, fi, form)@T_2 \mid \{\} \rightarrow_{mail} Time@T, Sent(yes, fi, c, form)@T_1, Rec(yes, c, fi, form)@T$$

$$Time@T, Rec(yes, c, fi, form)@T_1, Rec(no, fi, c, ans)@T_2 \mid \{\} \rightarrow_{c} Time@T, Rec(yes, c, fi, form)@T_1, Rec(yes, fi, c, allow)@T$$

$$Time@T, Rec(yes, c, fi, form)@T_1, Rec(no, fi, c, ans)@T_2 \mid \{\} \rightarrow_{c} Time@T, Rec(yes, c, fi, form)@T_1, Rec(yes, fi, c, disallow)@T$$

$$Time@T, Sent(yes, fi, c, form)@T_1, Rec(no, fi, c, ans)@T_2 \mid \{T \geq T_1 + d_{tol}\} \rightarrow_{fi} Time@T, Sent(yes, c, fi, form)@T_1, Rec(yes, fi, c, allow)@T$$

$$Time@T, Rec(yes, fi, c, allow)@T_1, Sent(no, fi, natp, npi(c))@T_2 \mid \{\} \rightarrow_{fi} Time@T, Rec(yes, fi, c, allow)@T_1, Sent(yes, fi, natp, npi(c))@T$$

Fig. 8. Actions of an $TLSTS$ specifying the possible interations between consumers and financial institution with respect to the disclosure of non-public information.

by, respectively, financial and health care institutions. As discussed in [18], in order to model some of their clauses one needs to take into account the time when events happen We show how to specify in our model some of such conditions.

Consider the following GLBA paragraph number 6802(b)(1) quoted below:

> "A financial institution may not disclose nonpublic personal information to a nonaffiliated third party unless $[\cdots]$ the consumer is given the opportunity, before the time that such information is initially disclosed, to direct that such information not be disclosed to such third party."

This paragraph specifies that a financial institution cannot, for instance, make public the personal information of one of its client if it did not give the consumer opportunity of disclosure. An opportunity of disclosure could be a letter with a form, which the consumer could fill allowing the disclosure or not. However, notice that the clause above does not enter into the details of how much time the financial institution must provide the consumer to, for example, fill the form and send it back to the financial institution. As in [18], we assume that the financial institution must wait at least d_{tol} time units after the form is *received* by the consumer.

To model such paragraph, we use the following two public predicates $Sent$ and Rec, described below:

- Facts of the form $Sent(yes/no, snd, rcv, data)$ denote whether or not (yes/no) a sender (snd) sent to a receiver (rcv) some information $(data)$. For instance, the fact $Sent(no, fi, c, form)$ denotes that the financial institution, fi, did not yet send the form, $form$, to the consumer, c;

– Similarly, facts of the form $Rec(yes/no, rcv, snd, data)$ denote whether or not (yes/no) a message with some information $(data)$ sent by the sender (rcv) was received by the receiver (rcv). For example, the fact $Rec(yes, fi, c, disallow)$ denotes that the financial institution, *fi*, received from the c a message disallowing the financial institution to disclose his non-public information.

Using these predicates, we can specify the following three critical configurations that are forbidden by the paragraph above:

$$Sent(yes, fi, natp, npi(c))@T_1, Sent(no, fi, c, form)@T_2 \quad \{\}$$
$$Sent(yes, fi, natp, npi(c))@T_1, Rec(yes, fi, c, disallow)@T_2 \quad \{T_1 \geq T_2\}$$
$$Sent(yes, fi, natp, npi(c))@T_1, Rec(yes, c, fi, form)@T_2, Rec(no, fi, c, ans)@T_3 \quad \{T_1 \leq T_2 + d_{tol}\}$$

The first and second configurations specify that the financial institution (*fi*) should not sent the consumer's non-public information $(npi(c))$ to the non affiliated third party $(natp)$ if the financial institution did not yet sent the disclosure form $(form)$ to the consumer or if the financial institution had already received a message from the consumer explicitly disallowing the disclosure of his non-public information. The last configuration specifies that the financial institution must provide the consumer at least d_{tol} time units to respond after he received the form before disclosing his non-public information.

Figure 8 contains a possible set of the actions specifying the exchange of messages between a financial institution, *fi*, and a consumer, c. In the first action, *fi* sends a disclosure form, $form$, to the consumer by mail. In the second action the consumer receives the form and the post-office sends to *fi* a confirmation that the consumer has received the form. The third and fourth actions specify that once the form is received by the consumer, he can return to *fi* a message allowing or not the disclosure of his public information, $npi(c)$, to a non-affiliated third party, $natp$. The fifth rule is the most important for us. It specifies, incorrectly, that if the consumer did not reply d_{tol} time units after the financial institution sent the form, then the financial institution assumes that the consumer allows the disclosure of his non-public information. Notice that this clause is incorrect because if the consumer receives the form more than d_{tol} units after the form was sent, due to some mailing problem, then in this case the consumer was not given the opportunity to allow or not the disclosure of his non-public information, hence not satisfying GLBA's paragraph above. In order to comply with this paragraph, the financial institution must wait d_{tol} time units after the form is *received by the consumer* and not when the form is sent by the financial institution. Thus, the corrected version of this action would be the following:

$$Time@T, Rec(yes, c, fi, form)@T_1, Rec(no, fi, c, ans)@T_2 \mid \{T \geq T_1 + d_{tol}\} \rightarrow_{fi} Time@T, Rec(yes, c, fi, form)@T_1, Rec(yes, fi, c, allow)@T$$

Finally, the last clause specifies that the financial institution can send some of the consumer's non-public information $(npi(c))$ to the non affiliated third party $(natp)$ if it is assumed that the consumer allows the disclosure of such information.

The goal of the system is to decide whether the non-public information of a consumer can be send to a non affiliated third party or not. These are specified by the

following two goal configurations:

$$Rec(yes, fi, c, allow)@T_1, Sent(yes, fi, natp, npi(c))@T_2 \quad \{T_2 \geq T_1\}$$
$$Rec(yes, fi, c, disallow)@T_1, Sent(no, fi, natp, npi(c))@T_2 \quad \{\}$$

Another paragraph in the GLBA legislation which requires time is the paragraph 6803(a) quoted below.

> At the time of establishing a customer relationship with a consumer and not less than annually during the continuation of such relationship, a financial institution shall provide a clear and conspicuous disclosure to such consumer [·], of such financial institutions policies and practices with respect to [disclosing non-public personal information].

It establishes that financial institutions have to provide a consumer with their policies and practices involving disclosing non-public information at least once a year. As with the previous paragraph, this prohibits some configurations to be reached, namely, those where the financial institution does not inform a customer annually about its practices and policies. This is specified similarly as before by using the predicates, $Sent$ and Rec, described above. Since we are only interested in the time difference between when two consecutive policies are received by the consumer, we only need to keep track of two received facts: $Rec(yes, c, fi, p_o)$ and $Rec(yes, c, fi, p_n)$, where p_o and p_n are respectively the old and new policy received by the consumer. Whenever a new policy is received by the consumer, one updates these facts by keeping only the newer one and replacing the older one with the fact $Sent(no, fi, c, new)@T$, indicating that the financial institution did not yet send a new policy to the consumer since time T. This is formally expressed the following action:

$$Time@T, Rec(yes, c, fi, p_n)@T_1, Rec(yes, c, fi, p_o)@T_2 \mid \{T_2 < T_1\} \rightarrow_{fi}$$
$$Time@T, Rec(yes, c, fi, p_n)@T_1, Sent(no, fi, c, new)@T.$$

There if a new policy, p_n, is received by the consumer, then the financial institution does not need to need to keep that the consumer received the old policy p_o. So this is replaced by the fact $Sent(no, fi, c, new)$. Sending a new policy is specified by the following action, which updates a value with a fresh one and assumes that the message is received instantaneously by the consumer, such as in the case where the finacial institution sends the policy by e-mail instead of conventional mail:

$$Time@T, Sent(no, fi, c, new)@T_1 \mid \{\} \rightarrow_{fi} \exists p_n. Time@T, Rec(yes, c, fi, p_n)@T$$

Given this, we can specify when a system is compliant with the paragraph above by using the following critical configurations:

$$Rec(yes, c, fi, p)@T_1, Rec(yes, c, fi, p_n)@T_2 \quad \{T_2 > T_1 + 365\}$$
$$Rec(yes, c, fi, p)@T_1, Sent(no, fi, c, new)@T_2, Time@T \quad \{T > T_1 + 365\}$$

The first configuration specifies the case when the financial institution sent the next policy in a time after the limit of 365 days, and the second configuration specifies the

case when the finantial institution failed to send a new policy to the consumer before the limit of 365 days.

As a second example, consider the following paragraph number 164.510(a)(3) quoted below:

> "The covered health care provider must inform the individual and provide an opportunity to [opt-out of] uses or disclosures for directory purposes as required by paragraph (a)(2) of this section when it becomes practicable to do so."

As before with the GLBA clause, the paragraph above is also a negative statement since it obliges the health provider to provide the client an opportunity to opt-out of uses or disclosure under some conditions, namely, when "it becomes practicable to do so". For example, if the client is in a coma, it is not practical to do so. But when he goes out the coma, then the health provider needs to provide the opportunity to opt-out. This can also be specified as a critical configuration:

$$No\_Opt\_out(u_1)@T_1, User(u_1, out\_of\_comma)@T_2, Time@T \quad \{T \geq T_2 + 1\}$$

If the provider did not provide an opportunity to opt-out of uses or disclosure one time unit after the user is out a a coma, then this is a critical configuration.

7.3 Collaborating Software Companies Example

When agents collaborate with each other, time is often one of the key components taken into account for not only the success of the collaboration, but also for defining the rules of the collaboration. Consider the following scenario where two software companies start collaborating in order to develop a product jointly. Assume that the product has to be launched in tradeshow and that the final goal is to develop the product before the tradeshow begins. However, since each company has its own set of software libraries (or secrets) and it is expected that new libraries are developed during the collaboration, one needs to specify how these intelectual properties are going to be shared among the companies. For instance, they could agree that no library developed before the start of the collaboration can be used in the development of the product, due to legal complications, but that all software issued during the collaboration are shared by the two companies. That is, the libraries that resulted from the collaboration can also be used by the companies while the companies are collaborating. Thus, the time when a software was developed has to be taken into account in order to distinguish what is allowed from what is not allowed in the collaboration.

Figure 9 contains the set of all actions necessary to specify the small example described above. The critical and goal configurations are those explained in Section 2. Intuitively, the main product is composed of sub-products that need to be developed. Here, for simplicity, we assume that the sub-products are not composed of further sub-products. These could be easily added to the system in a similar way. Moreover, each sub-product passes through three development phases, namely, the analysis phase, the coding phase, and the testing phase. The first three actions specify how many time units are needed to go from one phase to another, denoted by the constants d^s_{ana}, d^s_{cod}, and

$$Time@T, Sub\_prod(s, start, none)@T_1, \mid \{T \geq T_1\}$$
$$\rightarrow_A Time@T, Sub\_prod(s, analysis, none)@(T + d^s_{ana}),$$

$$Time@T, Sub\_prod(s, analysis, none)@T_1, \mid \{T \geq T_1\}$$
$$\rightarrow_A Time@T, Sub\_prod(s, coding, none)@(T + d^s_{cod}),$$

$$Time@T, Sub\_prod(s, coding, none)@T_1, \mid \{T \geq T_1\}$$
$$\rightarrow_A Time@T, Sub\_prod(s, testing, none)@(T + d^s_{tes}),$$

$$Time@T, Sub\_prod(s, testing, none)@T_1, \mid \{T \geq T_1\}$$
$$\rightarrow_A Time@T, Sub\_prod(s, ready, none)@(T + 1),$$

$$Time@T, Start@T_1, Lib(p)@T_2, Sub\_prod(s, status, none)@T_3 \mid \{T_1 \leq T_2, T \geq T_1, T \geq T_3\}$$
$$\rightarrow_A Time@T, Start@T_1, Lib(p)@T_2, Sub\_prod(s, ready, p)@(T + 1)$$

$$Time@T, Sub\_prod(s, ready, none)@T_1, P(*)@T_3 \mid \{T \geq T_1\}$$
$$\rightarrow_A \exists p_{new}\, Time@T, Sub\_prod(s, ready, p_{new})@T, Lib(p_{new})@T$$

$$Time@T, Sub, Product(not\_ready)@T_p \mid \Upsilon_p$$
$$\rightarrow_A Time@T, Sub, Product(ready)@(T + d_p)$$

Fig. 9. Here $status \in \{analysis, coding, testing\}$ is one of the different development phases of a software; the natural numbers $d^s_{ana}, d^s_{cod}, d^s_{tes}$, and d_p are the number of time-units necessary for the corresponding development phase; for instance d^s_{cod} is the time needed to code the sub-product s; Sub is the multiset containing facts of the form $Sub\_prod(s_i, ready, none)@T_i$ for all the sub-products s_i; and the set of constraints Υ_p is composed of the constraints $T \geq T_i$ for all sub-products s_i.

d^s_{tes}. Notice that the constraint specifies that one is only allowed to move from one phase to another when the current time, T, is greater or equal to the the time, T_1, when the previous phase is over. This is specified by the constraint $T \geq T_1$. Here, we model a waterfall development model. One could easily add other actions to model other development models.

Fifth and sixth rules from Figure 9 involve the use of a library that was previously developed. The fifth rule specifies that one is able to use library p to develop a sub-product s if p was developed after the collaboration started and if the library is available, *i.e.*, the current time is greater or equal to the time that the development of the software is finished. When a library is used, the development of the sub-product is finished in just one time unit and is ready to be used in the final product.[12] The sixth rule specifies that when a sub-product is finished it can be used to help the development of other sub-products. For this one creates a new library with a fresh name p_{new}, so that this library is uniquely identified. In this rule, we follow [44, 28] and use facts of the form $P(*)$ to denote a free memory space. So, in this case, we consume an empty slot in the server, for instance, to store a new library. If such space is not available, that is, there are no empty facts left, then the action cannot be fired and no library is created.

Finally, once all the sub-poducts are ready, one can assemble the main product. This is specified by the last action of the specification. Notice that we use time constraints to specify that this rule is applicable only when all sub-products are ready, that is, when the current time is greater or equal to the timestamps of all the sub-products.

The goal of the collaboration is given below. It specifies that ten time units before the deadline, one already knows whether the product is going to be ready five time units before the deadline. The goal is reached because the marketing department has time enough to begin preparing the launch of the final product and there is a safety gap of five time units for any final polishing of the product.

$$Deadline@T_1, Product(ready)@T_2, Time@T \quad \{T_1 \geq T_2 + 5, T_1 > T + 10\}$$

A possible critical configuration of this scenario is when one uses in a sub-product a library that was developed before the start of the collaboration, specified below:

$$Sub\_prod(s, status, p)@T_1, Lib(p)@T_2, Start@T_3 \quad \{T_3 > T_2\}$$

Alternatively, if the product is not ready before the deadline, then it is also a critical configuration. This is specified as follows:

$$Product(not\_ready)@T_1, Deadline@T_2, Time@T \quad \{T \geq T_2\}$$

The constraint specifies that the global time T is greater or equal to the time of the deadline, that is, the global time passed the deadline. Since the product is not yet ready, this state is critical and must be avoided.

[12] Notice that one could add more information to the predicates Lib and Sub_prod, such as the problem that a library and the sub-product solve. In this way, we improve the rule above so that the action is only applicable when problem solved by the library and the sub-product coincide.

7.4 Group-Centric Collaborations

In [35, 36], a new model of collaboration, called Group-Centric, was introduced. They use groups to specify collaboration. Groups consist of users and objects, where users at sometime might join or quit a group and similarly an object might be inserted or deleted from a group. Following ideas from access control, the authors in [35] specify the operations and properties involving the users and objects of groups. For members, there are two basic types of way a user can join or leave a group

- Strict Join (SJ) and Liberal Join (LJ): A user that strictly joins a group g_1 at time t has access only to the object in g_1 that are inserted at time t or after. This constrasts with LJ, where the user also has access to objects inserted before t.
- Strict Leave (SL) and Liberal Leave (LL): Similarly to the previous case, when a user leaves strictly a group, he loses the access to all the objects in the group, whereas when a user leaves liberally a group, he does not lose the access to those objects.

Similarly, there are two different ways an object can be added or removed from a group:

- Strict Add (SA) and Liberal Add (LA): When an object, o, is strictly added to a group at time t, only users that joined before time t can have access to o. On the other hand, when an object is liberally added to a group, then the object can be access by all or some that joined (*e.g.*, LJ) afterwards.
- Strict Remove (SR) and Liberal Remove (LA): An object that is strictly removed cannot be accessed by any user, while an object that is liberally removed can be accessed by all the users that had access to the object at remove time may retain access to it.

We now show how to model all these preperties using $TLSTS$es. For this, we will require the following facts:

- $User(u, g, member)@t$ – This fact denotes that at time t the user u has joined or left the group g according to the constant $member \in \{sj, lj.sl, ll\}$, where sj denotes strict join, lj liberal join, sl strict leave, and ll liberal leave. For instance, the fact $User(u_1, g_1, sj)@2$ denotes that at time 2, the user u strictly joined the group g_1;
- $Object(o, g, in)@t$ – This fact denotes that at time t the object o was added to or removed from the group g according to the constant $in \in \{sa, la, sr, lr\}$, where sa denotes strict add, la liberal add, sr strict remove, and lr liberal remove. For instance, the fact $Object(o, g, la)@4$ denotes that at time 4, the object o was liberally added to the group g;
- $Auth(u, o, g, yes/no)@t$ – This fact denotes that the user u has or not authorization to the object o in the group g at time t.

Given the definition above, we can specify critical states where a user should not have access to an object.

– A strictly joined user cannot have access to a previously strictly added object:

$$User(u, g, sj)@T_1, Object(o, g, sa)@T_2, Auth(u, o, g, yes)@T \quad \{T_1 > T_2\}$$

– A user that strictly left a group does not have access to any objects in the group:

$$User(u, g, sl)@T_1, Object(o, g, \_)@T_2, Auth(u, o, g, yes)@T_3 \quad \{\}$$

where _ denotes that any value for an argument.

– A user that liberally left a group does not have access to any object added to a group after he left the group:

$$User(u, g, ll)@T_1, Object(o, g, add)@T_2, Auth(u, o, g, yes)@T_3 \quad \{T_2 > T_1\}$$

where $add \in \{sa, la\}$.

One has to repeat this execise and specify the critical states for when and how an object is added or removed from a group.

Some of these properties can also be enforced by the constraints in actions. For instance, the following action where the user u, that is a strictly joined member of group g, has access to the o:

$$User(u, g, sj)@T_1, Object(o, g, add)@T_2, Auth(u, o, no)@T_3, Time@T \mid \{T_2 \geq T_1\} \rightarrow_u User(u, g, sj)@T_1, Object(o, g)@T_2, Auth(u, o, yes)@T, Time@T$$

where $add \in \{sa, la\}$.

In [35], they also specify more complex properties which require one to check the history of a user. For instance, a user when rejoining a group could be allowed to reacquire all the previous authorizations that he had before leaving the group. These type of properties seem to require one to check the previous configurations in a plan and are left out of the scope of this paper.

8 Implementation

We now turn our attention on how to use the rewrite engine Maude [12] to rapidly prototype *TLSTS*es. In particular, we show how to implement actions with multiple postconditions, that is, actions in general, and show how to use Maude to generate branching plans and check linear plans (see end of Section 2). One can find the Maude source files implementing the Clinical Investigation Scenarion discussed in Section 7 in [40].

Configurations We start by specifying the signature of an TLSTS, *i.e.*, the set of constants and predicate symbols. For instance, the code below specifies that the zero arity operator `Time` is of sort (or type) `Fact` and that `Per` is a unary operator whose argument is of sort `Person`.

```
op Time : -> Fact [ctor] .
op Per : Person -> Fact [ctor] .
```

The keyword `ctor` specifies when an operator is a constructor of a sort. For instance, `Time` is a constructor of the sort `Fact`. Other constructors of the sort `Fact` can be specified in a similar fashion.

Timestamped facts are specified by using the `@` operator which are used to attach a natural number to facts.

```
op _@_ : Fact Nat -> TFact [ctor] .
```

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

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

The last line also specifies that configurations are multisets by attaching the keywords `assoc` and `comm`, which specify that the operator constructing configuration 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 emptyset, is the identity of an operator. For instance, it is used to identify the configurations `none Time none Per(john)` and `Time Per(john)`.

Timed Critical and Timed Goal Configurations Timed critical and timed goal configurations are specified as equational theory. For instance, the following equational theory specifies in Maude the critical configuration when the FDA is not notified 7 days after a serious and unexpected problem is detected:

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

Maude automatically replaces a configuration by `true` if it satisfies the condition specified by the equation above.Other critical configuration can be specifies in a similar way. Given this equational theory for critical configurations, one can determine whether a configuration `C` is critical by using an expression of the form `critical C == true`. This expression returns true if and only if the configuration `C` is critical. Similarly, an equational theory for timed goal configuration can be specified.

Actions with Multiple Postconditions Actions are specifies as rewrite rules in Maude. In order to accommodate branching plans, we need two new operator `noPlan`, denoting when a branching plan has no leaves, and + used to construct the list of leaves of a branching plan. The leaves of a branching plan belong to the sort `BConf`.

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

The + is used to specify the different outcomes of an action. For instance, the following Maude rule specifies the action of performing a Urine test discussed in Section 2, where test has three possible outcomes, `ok`, `high`, or `bad`:

```
crl[urine]:
{(C:Conf time@T urine(1,24,bad,none)@T1)}
  => {(C:Conf time@T urine(1,24,bad,ok)@T)} +
    {(C:Conf time@T urine(1,24,bad,high)@T)} +
    {(C:Conf time@T urine(1,24,bad,bad)@T)}
  when T1 <= T ∧ T <= T1 + 5 .
```

The conditional rule above, labeled `urine`, specifies that when a urine test is applied then three different leaves are created, one for each possible result of an Urine test. The facts appearing in the configuration `C` are left untouched. Moreover, the boolean operations appearing at the end of the rule above specify exactly the time constraints appearing in the corresponding action shown in Section 2.

Plan Generation With the machinery described above, one can specify in Maude an TLSTS. Once a system is specified, Maude provides powerful meta-level reasoning techniques which enable one to automatically check whether a linear plan is compliant or to construct compliant branching plans.

Given a Maude program $\mathcal{P}$ specifying a $TLSTS$, we can derive from it a new program that can be used for constructing a compliant branching plan. We rewrite each unconditional rule in $\mathcal{P}$ of the form

```
rl[name]:
{S} => {S1} + {S2} + ··· + {Sn} .
```

to the following conditional rule, where one is only allowed to fire a rule if all the postconditions are not critical configurations.

```
crl[name]:
{S} => {S1} + {S2} + ··· + {Sn}
  if not (critical(S1) == true) ∧
    not (critical(S2) == true) ∧
    ··· ∧
    not (critical(Sn) == true) .
```

Similarly, we rewrite all conditional rules of the form

```
crl[name]:
{S} => {S1} + {S2} + ··· + {Sn}
  if T .
```

to the following conditional rule:

```
crl[name]:
{S} => {S1} + {S2} + ··· + {Sn}
  if T ∧ not (critical(S1) == true) ∧
    not (critical(S2) == true) ∧
    ··· ∧
    not (critical(Sn) == true) .
```

That is, we hard-code into the conditions of the rules that no critical configuration can be reached. Hence, one can safely search a compliant plan with the derived rules without worrying about critical confgurations. In fact, the search space is pruned by using the derived rules.

Since each branch of a branching plan does not depend on the remaining branches, we do not need to keep track of branches that have already reached a goal configu-

ration. In fact, we can forget about this branches while searching for a plan. This is accomplished by the following rule:

```
crl[simplify]:
{S} => noPlan
  if goal(S) == true .
```

In order to improve performance, it is best to use the rule above eagerly. Since Maude attempts rule from top to bottom, it is better to have this rule as the first rule in the Maude code.

With the resulting program, we can search and generate a compliant by using the search engines available in Maude. There are two types of search commands. The first is a breadth-first search command called `search` and the other is a depth-first search command called `rewrite`. Our experiments show that a depth-first search strategy is best. This seems to be related to the fact that we hard-code in the conditions of the rules that critical states are not reachable. For instance, given a initial configuration $init$, which can be specified by using an equational theory, one can ask Maude to find a plan by inputting the following command:

```
search [1] init =>+ noPlan .
```

In order for Maude to write down the plan, one invokes Maude's `META-LEVEL` module and writing the following command:

```
rew metaSearch(
  upModule('FDA, false),
  upTerm (init),
  upTerm (noPlan),
  nil, '+, unbounded, 0) .
```

where `FDA` is the name of the module where all rewrite rules are specified. For further details on Maude's `META-LEVEL` module, we refer the reader to Maude's manual [11]. Although Maude's output does provide all the details necessary for constructing a plan, its format is not very user-friendly. We leave ways of improving, such as by post-processing a plan into a work-flow, as future work.

Execution Monitoring While for plan generation one needed to consider all postconditions of actions, for execution monitoring the actual postcondition that occurred is already determined. Hence, in order to represent these plans, actions in the initial specification that have multiple postconditions must be "broken" into different rules, as follows. The following rule

crl[name]:
{S} => {S1} + {S2} + ⋯ + {Sn}
  if $\mathcal{T}$.

is rewritten into the following rules:

```
crl[name1]:
{S} => {S1} if T .

crl[name2]:
{S} => {S2} if T .
...
crl[namen]:
{S} => {Sn} if T .
```

Since the actions in the resulting program do not have the branching operator, they only produce linear plan. Maude allows to apply a particular instance of an action to a configuration by using the command `srew`, which has the following syntax:

$$\texttt{srew } \mathcal{C} \texttt{ using } \mathcal{R}\texttt{[}X_1 \texttt{ <- } t_1 \texttt{ ... } X_n \texttt{ <- } t_n\texttt{] as}$$

where $\mathcal{C}$ is the configuration for which one wants to apply the rule called $\mathcal{R}$ contaning the variables $X_1, \ldots, X_n$, which should be replaced by the terms $t_1, \ldots, t_n$. If the action is applicable, Maude returns the resulting configuration. Therefore, whenever one inputs a new entry in the assistant, a new configuration is computed.

However, as before, the previous configuration $\mathcal{C}$ is lost if we use the command above. We, therefore, rely on the meta-level capabilities of Maude to keep track of all configurations that have been reached. This is done by using the meta-level version of the command `srew`, which returns a trace. A trace is simply a list of configuration and actions.

Finally to perform plan checking in Maude, we traverse the stored trace to check whether any configuration is critical. This can be done also in the meta-level and the equational theories specifying critical configuration, shown above. Given a trace $\mathcal{T}$, we perform plan checking by using the following operator:

```
op check-critical : Trace -> Bool .

ceq check-critical ( L ) = check-critical (tail (L) )
  if not (critical(downTerm( first (head( L )), error)) == true) .
ceq check-critical ( L ) = true
  if (critical(downTerm( first (head( L )), error)) == true) .
```

The operator above returns true if there is a critical configuration in the given trace. More complicated operators could be definied, such as operators that also return the critical configuration or the step in the trace.

Summary In [40], one can find the Maude source files implementing the Clinical Investigation scenarion discussed in Section 7. In the source files, we also report some experiments involving different parameters for the scenario, such as by varying the time between visits, or when a visit should end. Although Maude is able to find a single branching plan quite quickly, depending on how the parameters are set, it still takes some time to find all possible plans. We believe that this is because of the interleaving of actions and the presence of the + operator.

We are currently pursuing some directions to improve performance. One option would be to replace the + in the post-conditions of rules and replace them by one rule for each post-condition, as we did for execution monitoring. Some of our experiments

suggest that Maude can find all (linear) plans rather quickly. However, the number of plans increases very fast and it is still not clear how to merge these plans to build a branching plan.

Another way of improving the performance is to bound time. Since the study team might not be interested on plans that contain the actions that need to be taken on visits that are far away, one might not need to construct them. We are currently investigating all these options.

9 Related Work

The specification of regulations has been topic of many recent works. In [7, 8, 37], a temporal logic formalism for modeling collaborative systems is introduced. In this framework, one relates the scope of privacy to the specific roles of agents in the system. For instance, a patient's test results, which normally should not be accessible to any agent, are accessible to the agent that has the role of the patient's doctor. We believe that our system can be adapted or extended to accommodate such roles depending on the scenario considered. In particular, it also seems possible to specify in our framework the health insurance scenario discussed in [37]. De Young *et al.* describe in [18] the challenges of formally specifying the temporal properties of regulations, such as HIPAA and GLPA. They extend the temporal logic introduced in [7] with fixed point operators, which seem to be required in order to specify these regulations.

While temporal logic seems suitable for specifying the temporal properties that need to be satisfied by the plans/traces of a system, it does not provide the details on how plans are constructed. In contrast, since it is based on multiset rewriting, our formalism also provides the means for specifying the actions of a system. Temporal properties, on the other hand, are specified by using critical and goal configurations as well as time constraints. We believe that, with these constructs, one is able to specify most of the properties needed in for instance FDA's regulation for clinical investigations. Moreover, we can also use existing rewriting tools, such as Maude [12], to implement our specifications.

Another application of systems with explicit time is explored in [19], namely for specifying access control policies. De Young *et al.* introduce [19] a hybrid proof system which allows one to reason with propositions annotated with time intervals. Although the main application of their systems is for proof authentication frameworks, their systems has many connections to our system. For instance, to capture the notion of state, [19] borrows the notion of linear formulas from linear logic to specify states and allows for time constraints to be specified. However, they do not enter into the details of the type of constraints used, but just require that some general properties are satisfied, such as transitivity. Here, on the other hand, we fix the type of constraints that can appear in order to analyse the complexity of systems. This allows us to specify the semantics of $TLSTS$es by using existing linear logic systems with definitions, such as the one appearing [5]. It seems possible, however, to encode an $TLSTS$ into the system proposed in [19] in a similar way as done in our encoding in linear logic with definitions. However, instead of encoding the validity of constraints as definitions, we use constraints judgments in [19]. Rewrite rules are encoded as linear im-

plications. Moreover, our timestamps are encoded as singleton time intervals. Although our timestamped facts are annotated with a single timestamp, time intervals can also be expressed in our framework by the use of fresh values. For instance, to create a new time interval of d time units one can use a rule whose post condition is of the form: $\exists x.Begin(x)@T, End(x)@(T+d)$. Since the value replacing x is fresh, the time interval is uniquely identified in a plan. Therefore, attaching intervals to facts as in [19] does not necessarily increase the expressiveness of the language. For instance, all the examples shown in [19] seem to be encodable as possibly unbalanced $TLSTS$es.

Real time systems differ from our setting in that dense time domains, such as the real numbers, are required, while in our intended applications, such as clinical investigations, discrete numbers suffice. The models introduced in [1, 32, 43] deal with the specification of real time systems and also investigate the complexity of some problems.

Kanovich *et al.* in [32] 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 [19], their model is limited since one is not allowed to specify properties which involve different timestamps, such as a fact P is valid at time t_1 and Q is valid at time t_2 greater than t_1. In our framework this is possible by using, for instance, goals/critical states with time constraints $\{P@T_1, Q@T_2\} \quad \{T_2 \geq T_1 - 5\}$. Finally, predicates are assumed to be unary, whereas we allow for predicates of a bounded arity in the size of facts.

Ölveczky and Meseguer in [43] also identify conditions 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.

[21] proposes to use an extension of linear temporal logic as specification language for policies and procedures. [21] also uses as running example the specification of FDA regulations and is interested in checking whether a given trace complies with FDA regulations. This problem seems to be related to the problem of execution monitoring that we propose here. One also shows that in [21] that checking whether a trace is compliant is EXPTIME-hard and in EXPSPACE. As with other temporal logic approaches, it seems hard to implement systems that are specified in such languages.

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

10 Conclusions and Future Work

This paper introduced a model based in multiset rewriting that can be used for specifying policies and systems which mention time explicitly. We also investigated the complexity of the plan compliance problem. In particular, we have shown that the plan compliance problem for balanced systems not containing branching actions is PSPACE-complete and the same problem for balanced systems possibly containing branching actions is EXPTIME-complete. Finally, we have shown that the plan compliance problem for progressing balanced systems not containing branching actions is NP-complete when there is a bound on the number of nonces and a bound on time.

There are many directions which we intend to follow. In [41], we describe how an assistant can help the participants of clinical investigations to reduce mistakes and comply with policies. We are currently implementing a small scale prototype in Maude [41] in order to collect more feedback from the health care community. The model introduced in this paper provides the theoretical foundations for this prototype. One main challenge, however, is to specify procedures in a modular fashion. One might need to specify intermediate languages that are closer to the terminology and format used in the specification of CIs, but that are still precise enough to translate them to a TLSTS. We hope that the work described in [20] may help us achieve this goal.

Once a clinical investigation is carried out and data about subjects is collected, data is analyzed by specialist to determine the effectiveness of the drug. Since by using an automated assistant data is stored electronically, we expect to build interfaces to other existing tools which help specialists in their tasks by, for example, performing statistical analysis of data. On the other hand, as discussed in [17], this easier handling of data might also help lead to privacy violations. Therefore, one must also provide means of handling data without inflicting privacy policies.

We would also like to extend our model to include dense times. This would allows us to specify policies for which real-times are important. For instance, [4] describes how one can reduce human errors by connecting medical devices and configuring them according to some hospital policies.

Finally, another interesting direction, which we still want to investigate, is that of checking whether some given plan complies with regulation. Such a work would help FDA audits as well as sponsors to monitor CIs and detect mistakes as early as possible. Similar work appears in [21, 25], but using models based in temporal logic.

To accomplish our ultimate goal of constructing a practical assistant tool for CIs, we identify and list below the following tasks in both practical and theoretical domains. We plan to tackle them in the near future.

Decidability of the Plan Generation Problem – We identify foundational challenges to this project, such as complexity results for the plan generation problem. We already have some initial steps in this direction; namely, we have identified conditions on TLSTSes for which we believe that the plan generation is PSPACE-complete. The details can be found in the companion technical report [40]. Besides complexity results, we are also investigating implementation optimizations, such as ways to minimize search space.

Specify Complete Protocols – This paper proved the concept that it is possible to specify regulations and protocols. However, at the end, we would like to be able to

specify complete protocols. In order to reduce the gap between the specification of protocols and its formalization, one might need to specify intermediate languages that are closer to the terminology and format used in protocols, but that is still precise enough to translate it to a TLSTS.

Human Computer Interface (HCI) – A crucial step on building a tool that is going to be at the end useful for the participants of a CI is to develop an intuitive HCI. We believe that our fruitful collaboration with CI specialists will continue to play a key role to accomplish this goal, in particular, in order to further understand the terminology and the procedures used.

Handling of Stored Data – Once a CI is carried out and data about subjects is collected, data is analyzed by specialist to determine the effectiveness of the drug. Since by using an automated assistant data is stored electronically, we expect to build interfaces to other existing tools which help specialists in their tasks by, for example, performing statistical analysis of data. This seems related to the on-going project described in [4] of connecting medical devices.

Privacy – On the other hand, as discussed in [17], the easier handling of data might also help lead to privacy violations. Since TLSTS is a model for collaborative system with privacy [28], we expect it to also be a suitable model for specifying and enforcing privacy policies.

Although we are years from achieving our ultimate goal, we believe that this is, nevertheless, a project worth investigating because of its theoretical and engineering challenges as well as of its potential impact on helping testing drugs, while taking care of the subjects' health. This project also shows that computer science and engineering have much to contribute on improving health practices.

Our PSPACE upper bounds seem to work even if allow different extensions to TLSTSes. For instance, we could allow several actions that advance time of the following form:

$$Time@T, W \mid \Upsilon \longrightarrow_{clock} Time@T + d, W$$

where W is a multiset of facts, Υ is a set of constraints only involving timestamps appearing in the precondition of the rule, and d is a natural number.

In another direction, we could also consider a scenario where instead of a global clock, each agent has its own clock. These models seems to be normally used to specify real-time systems.

A future work direction is to extend our PSPACE results to accommodate dense time domains, instead of natural numbers. We are also interested in implementing model-checkers and tools for modeling timed collaborative systems.

References

1. R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *SFM*, pages 1–24, 2004.
2. R. M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *CONCUR '00: Proceedings of the 11th International Conference on Concurrency Theory*, pages 380–394, London, UK, 2000. Springer-Verlag.
3. J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

4. 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.
5. D. Baelde. *A linear approach to the proof-theory of least and greatest fixed points*. PhD thesis, Ecole Polytechnique, Dec. 2008.
6. 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.
7. 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.
8. A. Barth, J. C. Mitchell, A. Datta, and S. Sundaram. Privacy and utility in business processes. In *CSF*, pages 279–294, 2007.
9. I. Cervesato, N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *CSFW*, pages 55–69, 1999.
10. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28:114–133, January 1981.
11. M. Clavel, F. Durán, S. Eker, , S. E. Lincoln, N. Martí-Oliet, J. Meseguer, J. F. Quesada, and C. Talcott. Maude manual version 2.6. Available at `http://maude.cs.uiuc.edu/maude2-manual/maude-manual.pdf`, 2011.
12. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. The Maude system. volume 1631, pages 240–243, 1999.
13. 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.
14. U. S. Congress. Health insurance portability and accountability act of 1996, privacy rule. Available at `http://www.access.gpo.gov/nara/cfr/waisidx_07/45cfr164_07.html`, August, 2002.
15. U. S. Congress. Gramm-Leach-Bliley act, financial privacy rule. Available at `http://www.law.cornell.edu/uscode/usc_sup_01_15_10_94_20_I.html`, November 1999.
16. S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.
17. A. Datta, N. Dave, J. C. Mitchell, H. Nissenbaum, and D. Sharma. Privacy challenges in patient-centric health information systems. In *Usenix Workshop on Health Security and Privacy*, 2010.
18. 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.
19. H. DeYoung, D. Garg, and F. Pfenning. An authorization logic with explicit time. In *CSF*, pages 133–145, 2008.
20. N. Dinesh, A. K. Joshi, I. Lee, and O. Sokolsky. Permission to speak: A logic for access control and conformance. *J. Log. Algebr. Program.*
21. 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.
22. 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.
23. 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`.

24. FDA. Warning letters. Available at http://www.fda.gov/ICECI/EnforcementActions/WarningLetters/default.htm.
25. D. Garg, L. Jia, and A. Datta. Policy auditing over incomplete logs: Theory, implementation and applications. To appear in CCS'11, 2011.
26. M. A. Grando, M. Peleg, and D. Glasspool. A goal-oriented framework for specifying clinical guidelines and handling medical errors. *Journal of Biomedical Informatics*, 2009.
27. S. Iida, G. Denker, and C. Talcott. Document logic: Risk analysis of business processes through document authenticity. *Journal of Research and Practice in Information Technology*, 2011.
28. M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. In *FAST*, 2010.
29. M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. http://www.math.upenn.edu/~scedrov/docs/tr-bounded.pdf, Aug. 2010.
30. M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Progressing collaborative systems. In *FCS-PrivMod*, 2010.
31. 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.
32. 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.
33. M. I. Kanovich, P. Rowe, and A. Scedrov. Collaborative planning with confidentiality. *J. Autom. Reasoning*, 46(3-4):389–421, 2011.
34. 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.
35. R. Krishnan, R. S. Sandhu, J. Niu, and W. H. Winsborough. Foundations for group-centric secure information sharing models. In *SACMAT*, pages 115–124, 2009.
36. R. Krishnan, R. S. Sandhu, J. Niu, and W. H. Winsborough. Towards a framework for group-centric secure collaboration. In *CollaborateCom*, pages 1–10, 2009.
37. 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.
38. R. McDowell and D. Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.
39. M. Minsky. Recursive unsolvability of post's problem of 'tag' and other topics in the theory of turing machines. *Annals of Mathematics*, 1961.
40. 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.
41. V. Nigam, T. B. Kirigin, A. Scedrov, C. Talcott, M. Kanovich, and R. Perovic. Towards an automated assistant for clinical investigations. To appear in the Second ACM SIGHIT International Health Informatics Symposium, 2012.
42. V. Nigam and D. Miller. Algorithmic specifications in linear logic with subexponentials. pages 129–140, 2009.
43. P. C. Ölveczky and J. Meseguer. Abstraction and completeness for Real-Time Maude. *Electr. Notes Theor. Comput. Sci.*, 176(4):5–27, 2007.
44. P. Rowe. *Policy compliance, confidentiality and complexity in collaborative systems*. PhD thesis, University of Pennsylvania, 2009.