Progressing Collaborative Systems

Max Kanovich[1], Tajana Ban Kirigin[2], Vivek Nigam[3], and Andre Scedrov[3]

[1] Queen Mary, University of London, UK
mik@dcs.qmul.ac.uk
[2] University of Rijeka, HR
bank@math.uniri.hr
[3] University of Pennsylvania, Philadelphia, USA
{vnigam,scedrov}@math.upenn.edu

Abstract. This paper builds on existing models for collaborative systems with confidentiality policies. The actions in these models are balanced, namely, they have an equal number of facts in their pre- and post-conditions. Here we consider a further restriction that each instance of an action is used at most once in a process. Administrative processes usually involve such progressing behavior, that is, whenever a transaction is performed, it does not need to be repeated. We investigate the complexity of the decision problem whether there exists a sequence of transitions from an initial state to a final state that avoids any critical states, e.g., states which conflict with the given confidentiality policies. We show that this problem is NP-complete when balanced actions do not involve fresh values and when the system is progressing. The same problem is shown to be PSPACE-complete when the system is not necessarily progressing, and PSPACE-hard when the system is progressing, but when actions may update values with fresh ones. The bounds hold even when balanced actions change only one fact in a configuration. We implement some examples in logic-based verification tools and model-check that they comply with certain policies.

Keywords: Collaborative Systems, Critical Configurations, Policy Compliance, Multiset Rewriting, Complexity, Model Checking, Linear Logic.

1 Introduction

Many administrative or business processes have not only a bounded number of transactions, but also have a *progressing behavior*: whenever a transaction is performed, it does not need to be repeated. For instance, whenever one initiates some administrative task, one receives a "to-do" list containing the sub-tasks necessary for accomplishing the final goal. Once one has "checked" an item on the list, one does not need to return to this item anymore. When all items have been checked, the process ends.

For a more concrete example, consider the scenario where a patient needs a medical test, *e.g.*, a blood test, to be performed in order for a doctor to correctly diagnose the patient's health. This process may involve several agents, such as a secretary, a nurse, and a lab technician. Each of these agents have their own set

of tasks. For instance, the patient's initial task could be to make an appointment and go to the hospital. Then, the secretary would send the patient to the nurse who would collect the patient's blood sample and send it to the lab technician, who would finally perform the required test. Such administrative processes are usually progressing: once a patient has made an appointment, he does not need to repeat this action again. Even in cases where it may appear that the process is not progressing, it is. For example, if the patient needs to repeat the test because his sample was spoiled, then a different process is initiated with possibly a new set of actions: the secretary is usually allowed to give the patient priority in scheduling appointments. Moreover, it is not realistic to imagine that one would need to reschedule infinitely many times, but only a very small number of times. For another example, when submitting a grant proposal, or even this paper, one must submit it before some deadline, otherwise the grant is not accepted. The use of deadlines is another form of bounding processes: since actions take some positive time to be performed, it must be the case that the whole operation is completed within a bounded number of transactions.

Administrative processes not only have a bounded number of transactions, but also manipulate a bounded number of values. Consider, for example, the simple process where a bank customer needs a new PIN number: The bank will either assign the customer a new PIN number, which is often a four digit number and hence bounded. Alternatively, the bank will allow the customer to select a password satisfying some conditions, *e.g.*, all its characters must be alphanumeric and it has to be of some particular length, and hence again bounded. Even when the bank lets the customer select a password of any particular length, in practice this password is bounded since users are never able to use an unbounded password due to buffer sizes, etc.

Frameworks based on multiset rewrite systems have been recently proposed for modeling collaborative systems [16, 17]. In such systems, participants, or agents collaborate in order to reach a state which contains some common goal. They perform actions, specified by rewrite rules, which change the system's state of the world, specified by a multiset of facts. A sequence of actions that leads from an initial state to a state containing the common goal is often called a *plan*. In such systems, agents have a limited storage capacity, that is, at any moment an agent can remember at most a bounded number of facts. In order to reflect this intuition, these frameworks use actions with same number of facts in their pre- and post-conditions, called *balanced* actions. With such actions all states in a run have the same size as the initial state.

Since information can be passed from one agent to another and agents do not necessarily trust each other completely, one is interested in showing that some critical states cannot be reached. Such critical states do not appear only in the domain of computer security, but in *information security* in general: a configuration where an agent's sensitive information, such as a password, is leaked to other agents is an example of such critical state that should be avoided. In [16, 17], this issue is address by different notions of policy compliance for a system, among them the following two: The first, called *system compliance*, is satisfied if

there is no plan for which one reaches a critical state of any agent. The second, called *weak plan compliance*, is satisfied if there is a plan for which no critical state is reached. In [17], it is shown that the problem of checking whether a system satisfies such policy compliances is PSPACE-complete under the condition that actions are balanced.

This paper builds on the framework in [16, 17]. On the one hand, we make two extensions to the framework: We allow balanced actions to update values with fresh ones, and we introduce among the available agents a leader who is trusted by all other agents. Each agent can also interact with the leader directly. On the other hand, we impose two restrictions to the framework: We allow balanced actions to change exactly one fact, a, to another fact, a', and in the process, one is allowed to check the presence of a fact, b, which can also be seen as checking for a condition. If we do not allow actions to create fresh values, then the same set of constants is used in the whole run of the system. Furthermore, we do not allow the same instance of a balanced action to be used more than once. This restriction incorporates the assumption that systems are progressing: each transition rule can be seen as checking one of the check-boxes necessary to achieve a final goal.

Under the restrictions above we show that in our model the weak compliance problem is NP-hard. It follows from [2, 20] that it is also in NP. Interestingly, if we remove the progressing restriction, that is, if we allow actions to be used more than once, then the same problem becomes PSPACE-hard. The PSPACE upper bound follows from [17]. The same problem, however, is also PSPACE-hard if the system is progressing, that is, each instance of a balanced action is used at most once, but if we also allow balanced actions to update old values with fresh ones. The upper bound for this case with balanced actions is open and left for future work. We prove our PSPACE-hard lower bounds by encoding Turing machines in our framework. The main challenge here is to *faithfully* simulate the behavior of a Turing machine that uses a sequential, non-commutative tape in our formalism that uses commutative multisets. This contrasts with the encoding of Turing machines in [15, p. 469] where the tape is encoded using non-commutative matrices.

Finally, we formalize the different policy compliances in intuitionistic logic. We use them in the two scenarios briefly described above called *grant proposal* and *medical test* where the latter is from [17] and the former is new. Then, we use the logic interpreters Bedwyr [4] and XSB [21] to model-check that these examples satisfy, respectively, the system compliance and the weak plan compliance.

The rest of the paper is organized as follows. We introduce, in Section 2, the models used for collaborative systems, including the extensions and restrictions that we propose. In Section 3, we state and prove the complexity results for different types of systems. Later, we discuss the two scenarios and their implementation in Section 4. Finally, in Sections 5 and 6, we comment on related work and conclude by pointing to future work.

2 Preliminaries

In this section we review the main vocabulary and concepts introduced in [16, 17] and also extend their definitions to accommodate a leader and actions that can update values with fresh ones.

2.1 Local State Transition Systems

At the lowest level, we have a signature Σ of predicate symbols $p_1, p_2, \ldots$, and constant symbols $c_1, c_2, \ldots$. For simplicity of exposition we restrict our terms to be constants and variables only (no function symbols). If we allow function symbols up to a fixed bounded depth, then the results of this paper continue to hold unchanged. A *fact* is a ground, atomic predicate over multi-sorted terms. Facts have the form $p(\boldsymbol{t})$ where p is an n-ary predicate symbol and $\boldsymbol{t}$ is an n-tuple of terms, each with its own sort. A *state*, or *configuration* of the system is a finite multiset W of facts. We use both WX and W, X to denote the multiset resulting from the multiset union of W and X.

As in [16, 17], we assume that the global configuration is partitioned into different local configurations each of which is accessible only to one agent. There is also a public configuration which is accessible to all agents. This separation of the global configuration is done by partitioning the set of predicate symbols in the signature. We typically annotate a predicate symbol with the name of the agent that owns it or with *pub* if it is public. For instance, the fact $p_A(\boldsymbol{t})$ belongs to the agent A, while the fact $p_{pub}(\boldsymbol{t})$ is public. However, differently from [16, 17], we allow an agent to privately communicate through a private channel with a privileged agent called *leader*. This is accomplished by further partitioning the predicates in the signature. As before, we annotate predicate symbols belonging to a private channel between an agent A and the leader l with Al. Whenever an agent and the leader need to communicate between each other, *e.g.*, request a new password, they can modify the facts in their corresponding private channel.

As in [16, 17], each agent has a finite set of actions or rules which transform the global configuration. Here, as in [7, 12], we allow agents to have more general actions which can create fresh values (*nonces*). Our actions have the forms:

$$X_A X_{Al} X_{pub} \rightarrow_A \exists \boldsymbol{t}. Y_A Y_{Al} Y_{pub} \quad \text{and} \quad X_l X_{Al} X_{pub} \rightarrow_l \exists \boldsymbol{t}. Y_l Y_{Al} Y_{pub}.$$

The agent who owns an action is specified by the subscript on the arrow. An agent that is not the leader owns only actions of the former form, where it also owns the local facts and the facts in the private channel. The leader, on the other hand, owns only rules of the latter form, where it also owns the local facts and any fact in any private channel to any agent. Actions work as multiset rewrite rules. All free variables in a rule are treated as universally quantified. $X_A X_{Al} X_{pub}$ are the pre-conditions of the action and $Y_A Y_{Al} Y_{pub}$ are the postconditions of the action. By applying the action for a ground substitution (σ), the pre-condition applied to this substitution ($X_A\sigma X_{Al}\sigma X_{pub}\sigma$) is erased and replaced with the post-conditions applied to the same substitution

$(Y_A \sigma Y_{Al} \sigma Y_{pub} \sigma)$. In this process, the existentially quantified variables ($\boldsymbol{t}$) appearing in the post-condition are replaced by fresh variables (also known as eigenvariables). The rest of the configuration remains untouched. Thus, for instance, we can apply the action $p_A(x), q_{pub}(y) \rightarrow_A \exists z.r_A(x,z), q_{pub}(y)$ to the global configuration $V, p_A(t), q_{pub}(s)$ to get the global configuration $V, r_A(t,c), q_{pub}(s)$, where the constant c is new. Later, we will impose restrictions to the types of actions allowed as discussed in the Introduction.

Definition 1. *A* local state transition system T *is a tuple* $\langle \Sigma, I, l, R_T \rangle$*, where* Σ *is the signature underlying the language,* I *is a set of agents,* $l \in I$ *is the leader, and* R_T *is the set of (local) actions owned by those agents.*

We use the notation $W \rhd_T U$ to mean that there is an action in T which can be applied to the state W to transform it into the state U. We let $\rhd_T^+$ and $\rhd_T^*$ denote the transitive closure and the reflexive, transitive closure of $\rhd_T$ respectively. Usually, however, agents do not care about the entire configuration of the system, but only if a configuration contains some particular facts. For example, in the medical test scenario, the patient is only interested to know if his test results are ready and not in those of any other agent. Therefore we use the notion of partial goals. We write $W \rightsquigarrow_T Z$ to mean that $W \rhd_T ZU$ for some multiset of facts U. For example with the action $r : X \rightarrow_A Y$, we find that $WX \rightsquigarrow_r Y$, since $WX \rhd_r WY$. We define $\rightsquigarrow_T^+$ and $\rightsquigarrow_T^*$ to be the transitive closure and the reflexive, transitive closure of $\rightsquigarrow_T$ respectively. We say that the partial configuration Z is reachable from state W using T if $W \rightsquigarrow_T^* Z$. Finally, given an initial configuration W and a partial configuration Z, we call a plan any sequence of actions that leads from state W to a state containing Z.

2.2 Connections to Linear Logic

Similar to [17, 8], we now formalize the informal notion of actions that can create fresh values, described in the previous section, in the precise, standard terms of linear logic [14]. In particular, one can capture precisely the intended semantics of such actions by using the notion of focusing introduced by Andreoli in [3].

We encode a rule of the form $X_A X_{Al} X_{pub} \rightarrow_A \exists \boldsymbol{t}. Y_A Y_{Al} Y_{pub}$ as the linear logic formula $\forall \boldsymbol{x}[\bigotimes\{q_A X_A X_{Al} X_{pub}\} \multimap \exists \boldsymbol{t}. \bigotimes\{q_A Y_A Y_{Al} Y_{pub}\}]$, where $\boldsymbol{x}$ are the free variables appearing in the rule and where 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 $\ulcorner R_T \urcorner$ is the set with the encoding of all the transition rules in R_T, and the set of propositions used to mark a rule to an agent is defined as $Q_I = \{q_A : A \in I\}$. One feature of this particular encoding is that the creation of nonces is specified by using standard quantifiers.

The following theorem connects linear logic provability to state reachability. In fact, by using focused cut-free proofs [3], we obtain a stronger result, namely, a one-to-one correspondence between the set of plans of an LSTS and the set of focused derivations obtained from its encoding.

Theorem 1. *Let $T = \langle \Sigma, I, l, R_T \rangle$ be a local transition system. Let W and R be two states under the signature Σ. Then the sequent $!\ulcorner R_T \urcorner, Q_I, W \vdash \bigotimes\{Q_I, R\} \otimes \top$ is provable in linear logic iff $W \rightsquigarrow_T^* R$.*

2.3 Policy Compliances

In order to achieve a final goal, it is often necessary for an agent to share some private knowledge with some other agent. For example, in the medical scenario, the patient needs to share his name with the secretary in order for the test to be run. However, although agents might be willing to share some private information with some agents, they might not be willing to do the same with other agents. For example, a client could share his password with the bank, which is trusted, and not with another client. One is, therefore, interested in determining if a system complies with some *confidentiality policies*, such as the secretary should not know the test results of the patient or that the deadline for sending a proposal has not passed. We call a *critical state* any configuration that conflicts with some given confidentiality policies, and we classify any plan that does not reach any critical state as *compliant*.

We review two of the three policy compliances proposed in [16, 17]:[1]

- (System compliance) Given a local state transition system T, an initial configuration W, a (partial) goal configuration Z, and a set of confidentiality policies, is no critical state reachable, and does there exist a plan leading from W to Z?
- (Weak plan compliance) Given a local state transition system T, an initial configuration W, a (partial) goal configuration Z, and a set of confidentiality policies, is there a compliant plan which leads from W to Z?

While system compliance requires that for any sequence of actions no critical state is reachable from the initial configuration, weak plan compliance only requires the existence of a plan that does not reach a critical state. Which compliance is more suitable will depend on the process used. In some cases, such as in the medical scenario, one might require system compliance: according to hospital policies, it should never be possible that, for example, the secretary gets to know the test results of the patient. In other cases, however, such as in grant proposal scenario, one might only require weak plan compliance: there is a plan for which the grant is sent to the funding agency before the deadline.

2.4 Subcases of Local State Transition Systems, Plans, and Policy Compliances

Now, we identify some restrictions to the actions of local state transition systems (LSTS). These restrictions are useful to obtain classes of LSTSes for which solving the policy compliance problems becomes feasible. For instance, if no restrictions are made, one can show that weak plan compliance is undecidable [16], even if actions do not create nonces.

[1] The third type of policy compliance, called plan compliance, is left out of the scope of this paper.

We classify a rule as *balanced* if the number of facts in its precondition is the same as the number of facts in its postcondition, and we classify a rule as *monadic* if it changes exactly one state variable. In particular, we will be interested in monadic rules of the following three forms:

$$a \to a', \qquad ab \to a'b, \qquad \text{and} \qquad ab \to \exists \boldsymbol{t}.a'b.$$

The first type of rule, called *context-free*, changes one state variable, a, to another state variable, a', without checking for the presence of any other variable. The second and third type of rules, called *context-sensitive*, check for the presence of a state variable b, which can be seen as a condition for applying such a rule. Moreover, in the third type of rule new constants, $\boldsymbol{t}$, are created, while in the first and second type of rules no new constants are created.

In the remaining of this paper, we only consider LSTSes with monadic, and, hence, balanced rules. Therefore, the size of configurations is always the same as in the initial configuration, and whenever an action is used, only one state variable is changed. As discussed in [16], the restriction of balanced actions provides decidability of weak plan compliance. On the other hand, the restriction of monadic actions is new and it will be explored later in this paper.

In [16, 17], plans were allowed to use an instance of an action as many times needed. Here, however, in order to accommodate the assumption that a collaborative system is progressing, we define progressing plans: we classify a plan as *progressing* if it only uses an instance of a rule *at most once*. We extend this classification to weak-compliance if one only allows compliant progressive plans.

While the progressing condition naturally appears in the specification of security of protocols, note that here we differ from [12] because we use only balanced actions. In particular, in [12], the intruder can copy facts, *i.e.*, the intruder's memory is unbounded.

3 Complexity Results

In this Section we discuss complexity results for the weak plan compliance problem in local state transition systems. We assume that the set of critical configurations is decidable in polynomial time.

We start, mainly for completeness, with the simplest form of monadic actions, namely, the context-free ones. The following result can be inferred from [12, Proposition 5.4].

Proposition 1. *Given a local state transition system with only context-free monadic actions, the weak plan compliance problem is in P.*

Now, we investigate the complexity of the progressing weak plan compliance problem when one is allowed to use only monadic actions that cannot create nonces. These restrictions reflect the assumptions discussed in the Introduction, namely, that many systems have a constant number of names and that they have a progressing behavior. For instance, the examples of the medical test and grant proposal used in Section 4 are in this class of LSTSes.

We show that this problem is NP-complete.

Theorem 2. *Given a local state transition system with only monadic actions of the form $ab \to a'b$, the progressing weak plan compliance problem is NP-complete.*

Proof We start by proving the NP-hard lower bound for the partial reachability problem. In our proof, we do not use critical states and use only one single agent. In particular, we reduce the 3-SAT problem which is well-known to be NP-complete [11] to the problem of reachability using transition rules and a bounded number of state variables. Consider the formula below in conjunctive normal form with three variables per clause, classified as 3-CNF, to be the formula for which one is interested in finding an model: $C = (l_{11} \vee l_{12} \vee l_{13}) \wedge \cdots \wedge (l_{n1} \vee l_{n2} \vee l_{n3})$, where each l_{ij} is either an atomic formula v_k or its negation $\neg v_k$. We encode the 3-SAT problem by using two sets of rules. The first one builds an interpretation for each variables, v_k, appearing in C as follows

$$v_k \to_A t_k \qquad \text{and} \qquad v_k \to_A f_k,$$

where the first rule rewrites the variable v_k to true, denoted by t_k, and the second to false, denoted by f_k. The second set of rules checks if the formula C is satisfiable given an interpretation:

$$\begin{array}{ll} S_{(v_k \vee l_{j2} \vee l_{j3}) \wedge C}, t_k \to_A S_C, t_k & S_{(\neg v_k \vee l_{j2} \vee l_{j3}) \wedge C}, f_k \to_A S_C, f_k \\ S_{(l_{j1} \vee v_k \vee l_{j3}) \wedge C}, t_k \to_A S_C, t_k & S_{(l_{j1} \vee \neg v_k \vee l_{j3}) \wedge C}, f_k \to_A S_C, f_k \\ S_{(l_{j1} \vee l_{j2} \vee v_k) \wedge C}, t_k \to_A S_C, t_k & S_{(l_{j1} \vee l_{j2} \vee \neg v_k) \wedge C}, f_k \to_A S_C, f_k \end{array}$$

where $1 \leq j \leq n$ and v_k is a variable appearing in C. Thus, we have a total of $(2 \times m + 6 \times n)$ rules and a total of $(3 \times m + n + 1)$ state variables, where m and n are, respectively, the number of variables and clauses in C.

Given a 3-CNF formula C and the set of rewrite rules, R_T, shown above, we prove soundness and completeness as shown below.

($\Rightarrow$) If C is satisfiable, we rewrite the variables in C according to the model of C by using the first set of rules in R_T and then check for satisfiability using the second set of rules in R_T.

($\Leftarrow$) It is similar to the previous direction. A state containing $S_\emptyset$ is reached only if a partial interpretation for the variables in C is build by using the first set of rules in R_T. This interpretation can be completed by assigning, for example, false to all other variables and will be necessarily a model of C.

The NP upper bound can be inferred from [2, 20]. In fact, an NP upper bound can also be inferred from [2, 20] even if we relax systems to have non-monadic actions, that is, where actions can change more than one fact. □

Interestingly, the NP-hardness result obtained above is replaced by a PSPACE-hardness result if we allow actions to be used as many times needed, even when LSTSes are restricted to have only monadic actions that do not create nonces. This shows that the notion of progressing is indeed important to guarantee the lower complexity. This result also improves the result in [17, Theorem 6.1] since in their encoding they allowed any balanced actions including non-monadic ones, whereas here we use only monadic actions. The main challenge here is to simulate operations over a non-commutative structure (tape) by using a commutative one (multiset).

Theorem 3. *Given a local state transition system with only actions of the form $ab \to a'b$, the weak-plan compliance problem is PSPACE-complete.*

Proof The upper bound for this problem can be inferred directly from [17].

In order to prove the lower bound, we encode a non-deterministic Turing machine M that accepts in space n within monadic actions, whenever each of these actions is allowed any number of times. In our proof, we do not use critical states and need just one agent A.

For each n, we design a local state transition system T_n as follows:

First, we introduce the following propositions: $R_{i,\xi}$ which denotes that *"the i-th cell contains symbol ξ"*, where $i=0,1,..,n+1$, ξ is a symbol of the tape alphabet of M, and $S_{j,q}$ denotes that *"the j-th cell is scanned by M in state q"*, where $j=0,1,..,n+1$, q is a state of M.

Given a *machine configuration* of M in space n - that M scans j-th cell in state q, when a string $\xi_0\xi_1\xi_2..\xi_i..\xi_n\xi_{n+1}$ is written left-justified on the otherwise blank tape, we will represent it by a configuration of T_n of the form (here ξ_0 and ξ_{n+1} are the end markers):

$$S_{j,q}R_{0,\xi_0}R_{1,\xi_1}R_{2,\xi_2}\cdots R_{n,\xi_n}R_{n+1,\xi_{n+1}}. \tag{1}$$

Second, each instruction γ in M of the form $q\xi \to q'\eta D$, denoting *"if in state q looking at symbol ξ, replace it by η, move the tape head one cell in direction D along the tape, and go into state q'"*, is specified by the set of $5(n+2)$ actions of the form:

$$\begin{array}{lll} S_{i,q}R_{i,\xi} \to_A f_{i,\gamma}R_{i,\xi}, & f_{i,\gamma}R_{i,\xi} \to_A f_{i,\gamma}h_{i,\gamma}, & f_{i,\gamma}h_{i,\gamma} \to_A g_{i,\gamma}h_{i,\gamma}, \\ g_{i,\gamma}h_{i,\gamma} \to_A g_{i,\gamma}R_{i,\eta}, & g_{i,\gamma}R_{i,\eta} \to_A S_{i_D,q'}R_{i,\eta}, & \end{array} \tag{2}$$

where $i=0,1,..,n+1$, $f_{i,\gamma}$, $g_{i,\gamma}$, $h_{i,\gamma}$ are auxiliary atomic propositions, $i_D := i+1$ if D is *right*, $i_D := i-1$ if D is *left*, and $i_D := i$, otherwise.

The idea behind this encoding is that by means of such five monadic rules, applied in succession, we can simulate any successful non-deterministic computation in space n that leads from the initial configuration, W_n, with a given input string $x_1x_2..x_n$, to the accepting configuration, Z_n.

The *faithfulness* of our encoding heavily relies on the fact that any machine configuration includes exactly one machine state q. Namely, because of the specific form of our actions in (2), any configuration reached by using a plan $\mathcal{P}$, leading from W_n to Z_n, has exactly one occurrence of either $S_{i,q}$ or $f_{i,\gamma}$ or $g_{i,\gamma}$. Therefore the actions in (2) are necessarily used one after another as below:

$$S_{i,q}R_{i,\xi} \to_A f_{i,\gamma}R_{i,\xi} \to_A f_{i,\gamma}h_{i,\gamma} \to_A g_{i,\gamma}h_{i,\gamma} \to_A g_{i,\gamma}R_{i,\eta} \to_A S_{i_D,q'}R_{i,\eta}.$$

Moreover, any configuration reached by using the plan $\mathcal{P}$ is of the form similar to (1), and, hence, represents a configuration of M in space n.

Passing through this plan $\mathcal{P}$ from its last action to its first v_0, we prove that whatever intermediate action v we take, there is a successful non-deterministic computation performed by M leading from the configuration reached to the

accepting configuration represented by Z_n. In particular, since the first configuration reached by $\mathcal{P}$ is W_n, we can conclude that the given input string $x_1x_2..x_n$ is accepted by M. □

In order to obtain a faithful encoding, one must be careful, specially, with commutativity. If we attempt to encode these actions by using, for example, the following four monadic rules

$$\begin{array}{ll} S_{i,q}R_{i,\xi} \rightarrow_A f_{i,\gamma}R_{i,\xi}, & f_{i,\gamma}R_{i,\xi} \rightarrow_A f_{i,\gamma}h_{i,\gamma}, \\ f_{i,\gamma}h_{i,\gamma} \rightarrow_A f_{i,\gamma}R_{i,\eta}, & f_{i,\gamma}R_{i,\eta} \rightarrow_A S_{i_D,q'}R_{i,\eta}, \end{array}$$

then such encoding would not be faithful because of the following conflict: $(f_{i,\gamma}R_{i,\xi} \rightarrow_A f_{i,\gamma}h_{i,\gamma})$ and $(f_{i,\gamma}R_{i,\eta} \rightarrow_A S_{i_D,q'}R_{i,\eta})$.

It is also worth noting that one cannot always use a set of five monadic rules similar to those in (2) to faithfully simulate non-monadic actions of the form $ab \rightarrow cd$. Specifically, one cannot always guarantee that a goal is reached after all five monadic actions are used, and not before. For example, if our goal is to reach a state with c and we consider a state containing both c and d as critical, then with the monadic rules it would be possible to reach a goal without reaching a critical state, whereas, when using the non-monadic rule, one would not be able to do so. This is because, after applying the rule $ab \rightarrow cd$, one necessarily reaches a critical state. In the encoding of Turing machines above, however, this is not a problem since all propositions of the form $S_{i,q}$ do not appear in the intermediate steps, as illustrated above.

We return to the problem of progressing weak plan compliance, where an instance of an action can be used at most once, but now, we allow monadic actions to create nonces. That is, we trade the absence of the progressing condition for the ability to create nonces. It turns out that such problem is also PSPACE-hard. Therefore, the progressing condition alone is not enough to guarantee an NP complexity, but one also needs to forbid the creation nonces.

Theorem 4. *Given a local state transition system with only monadic actions of the form* $ab \rightarrow \exists \boldsymbol{t}.a'b$, *the progressing weak plan compliance problem is PSPACE-hard.*

Proof (Sketch) The proof goes in a similar fashion as the lower bound proof of Theorem 3. However, we cannot use the same encoding appearing in (2). Since only one instance of any rule in the LSTS can be used, we would only be allowed to encode runs that use an action of M once. In order to overcome this problem, here, instead of using propositional rules, we use $6(n+2)$ first-order actions of the form:

$$\begin{array}{ll} S_{i,q}(t)R_{i,\xi} \rightarrow_A \exists t_n.f_{i,\gamma}(t_n)R_{i,\xi}, & f_{i,\gamma}(t)R_{i,\xi} \rightarrow_A f_{i,\gamma}(t)h_{i,\gamma}(t), \\ f_{i,\gamma}(t)h_{i,\gamma}(t) \rightarrow_A g_{i,\gamma}(t)h_{i,\gamma}(t), & g_{i,\gamma}(t)h_{i,\gamma}(t) \rightarrow_A g_{i,\gamma}(t)R_{i,\eta}, \\ g_{i,\gamma}(t)R_{i,\eta} \rightarrow_A S_{i_D,q'}(t)R_{i,\eta}, & S_{i,q}(t) \rightarrow_A S_{i,q}. \end{array} \quad (3)$$

where $i=0,1,..,n+1$. The initial state contains a variable $S_{i,q}(c)$ with some constant c and the goal state is the accepting configuration with a propositional variable $S_{j,q}$ (of arity zero). Intuitively, the first five rules above are used in the

same way as before to encode M's actions of the form $S_{i,q}R_{i,\xi} \rightarrow_A S_{i_D,q'}R_{i,\eta}$, but, now, we create a new constant, t_n, everytime we apply the first rule. This allows us to encode runs where the same action of M is used more than once, since, for each use of this action, we use a different instance of the rules in (3). Moreover, since in the accepting state one is not interested in the constant t appearing in the variables $S_{i,q}(t)$, we use the last rule in (3) when the accepting state is reached. Notice that after this last action is used, no other rule in (3) is applicable. □

In [12], Durgin *et al.* show that the reachability problem for when rules can create a bounded number of new constants is DEXP-complete. However, in their system, rules were not balanced, whereas here rules are monadic, hence balanced. One can, however, show that if we allow only a bounded number of nonces to be created, then the complexity of the progressing weak plan compliance problem using a system with balanced actions returns to NP. The upper bound for this problem when actions are balanced and when an arbitrary number of nonces can be used remains open.

4 Model-checking for Policy Compliances

In the past years, model-checking for properties in a finite system has been successfully reduced to the search for sequent calculus (cut-free) proofs of intuitionistic logic theories [22] (a.k.a. logic programming). We now briefly describe how we can model-check whether an LSTS satisfies a policy by encoding it in intuitionistic logic. In particular, given an LSTS, we encode configurations, actions, and policy compliances as described below:

Specifying configurations: We use a list of terms in the logic to encode a configuration of system. For instance, the list of terms

```
[(pat john blood), (sec id nUsed 1), (sec id nUsed 2), (sec id nUsed 3), (doc wait john blood)]
```

corresponds to the configuration where the patient John requires a blood test, `(pat john blood)`, the secretary has three available ID numbers, *e.g.*, `(sec id nUsed 1)`, and the doctor is expecting John's blood test results, `(doc wait john blood)`. A public fact is encoded using a term whose head is `pub` and likewise we encode a fact in the private channel of an agent a to the leader by using a term whose head is `al`. In the example above, there are no public facts nor facts in any private channel.

Specifying actions: We use intuitionistic logic clauses to specify the actions of an LSTS. Given a configuration, `L1`, these clauses specify when an action is applicable and what is the resulting post-configuration, `L2`. For example, the following three clauses specify three different actions in the medical scenario [17]. The first one belongs to the patiend `pat` and the other two to the secretary `sec`. The terms that start with upper-case letters are variables that are bound in the clause by universal quantifiers.

```
arr L1 L2 pat := member (pat N T) L1, remEle (pat N T) L1 L3, append [(pub toSec N T)] L3 L2.
arr L1 L2 sec := member (pub toSec N T) L1, remEle (pub toSec N T) L1 L3, append [(sec N T)] L3 L2.
arr L1 L2 sec := member (sec N T) L1, member (sec used I N T) L1,
                 remEle (sec id nUsed I) L1 L3, append [(sec used I N T)] L3 L2.
```

The last clause, for instance, specifies the following action belonging to the secretary:

$$sec(id, nUsed, I), sec(N, T) \rightarrow_{sec} sec(used, I, N, T), sec(N, T)$$

where the secretary assigns an unused ID number (I) to a patient (N). We use three auxiliary predicates in order to specify this action: `member`, `append`, and `remEle`. The first two predicates are the usual specifications for when, respectively, an element belongs to a list and when a list is the result of appending two lists. For instance, the predicate `(member (sec N T) L1)` checks if the fact `(sec N T)` belongs to the configuration `L1`. In the clause above, this predicate is used to specify that its corresponding action is aplicable only if the action's precondition is present in the configuration `L1`. The predicate `(append [(sec used I N T)] L3 L2)`, on the other hand, checks if the configuration `L2` is the result of adding the fact `(sec used I N T)` to the configuration `L3` and it is is used, together with the predicate `(remEle E L1 L2)`, to construct the resulting configuration `L2`. Finally, the third predicate `(remEle E L1 L2)` specifies when the list `L2` is the result of removing exactly one occurrence of the fact `E` from the list `L1`. For example, the predicate `(remEle a [a,a,b,a] [a,a,b])` is derivable since the list `[a,a,b]` is the result of removing exactly one occurrence of the element `a` from `[a,a,b,a]`.

Specifying policy compliances: As is usual in logic programming, we use the following two clauses to specify recursively when a state is reachable from another:

```
path L1 L2 := arr L1 L2 A    and    path L1 L2 := arr L1 L3 A, path L3 L2.
```

The former clause contains the base case when a state is reachable by performing a single action and the latter clause contains the recursion. Notice that the agent, `A`, is not relevant to determine whether a state is reachable.

The clause below specifies when an LSTS has a system compliant plan from an initial state, `W`, when given a partial goal `G`.

```
scomp W G := pi S\ (critical S, path W S => false), path W Z, subset G Z.
```

Here the symbol `(pi S\)` denotes the universal quantification of the variable `S`. To show that an LSTS is system compliant, one needs to check for two conditions: The first condition is that all critical states, `S`, must not be reachable from the initial state `W`. In the clause above, this condition is specified by the expression `pi S\(critical S, path W S => false)`. Intuitively, the interpreter shows by finite-failure that, for all instances of a critical state `S`, the formula `path W S`, specifying reachability, is not provable. Here, the critical states of the system are also specified by using clauses. For instance, the following clause specifies that any state where the secretary knows John's test results is a critical state.

```
critical L := member (sec john testResults) L
```

The second condition is that there must be a path from the initial state, `W`, to another state, `Z`, such that `Z` contains the partial goal `G`. This is specified by the expression `(path W Z, subset G Z)`, where the predicate `subset G Z` specifies when all elements of `G` are also elements of `Z`.

Finally, the following clauses specify recursively whether a weak plan compliant plan exists from an initial state, `W`, to a state containing the goal `G`.

$$
\begin{aligned}
time(T) &\rightarrow_{time} time(T+1)\\
time(T), Al(noBudget, A) &\rightarrow_A time(T), A(office, writeBudget, T+TW)\\
time(T), A(office, writeBudget, T) &\rightarrow_A time(T), A(coPI, no\ title, budget)\\
pub(title), A(coPI, no\ title, budget) &\rightarrow_A pub(title), A(coPI, title, budget)\\
time(T), A(coPI, title, budget) &\rightarrow_A time(T), A(uni, title, budget, T+TU)\\
time(T), A(uni, title, budget, T) &\rightarrow_A time(T), Al(coPI, budget)
\end{aligned}
$$

Fig. 1. The set of actions involving the writing of a budget for a coPI called A and of the agent time. Here TW and TU are, respectively, the time needed for A's accounting office to write a budget and for the Dean of A's university to approve the final budget.

```
wcomp W G := subset G W  and  wcomp W G := arr W Z A, (critical Z => false), wcomp Z G.
```

The former clause specifies that there is, trivially, such a plan if the initial state, `W`, contains the goal `G`. The second clause contains the recursion and specifies the existence such a plan if an agent `A` can perform an action (`arr W Z A`) yielding a configuration `Z` that is not critical (`critical Z => false`) and that one also has a weak plan compliant plan from `Z` to the same goal `G` (`wcomp Z G`). We also point out that we can also specify, in a similar way, the third type of policy compliance described in [16]. However, this is left out of the scope of this paper.

We now specify a new example of a collaborative process, called *grant proposal*, where a leader is involved. There, different agents or researchers collaborate to write a proposal, which includes a budget and a technical text. Among the agents, we distinguish one called PI (principal investigator) and we call the remaining coPIs (co-principal investigators). The PI is responsible for sending the complete project to the funding agency and for coordinating the coPIs. The task is to find a plan so that all coPIs finish writing their part of the text and budget and send them to the PI well before the deadline of the proposal, so that the PI can wrap up and send the final project to the funding agency. The critical states of this example is any state where the time has passed the deadline.

Some of the actions belonging to a coPI are depicted in Figure 1. At the beginning all coPIs do not have a budget (*noBudget*). A coPI starts at sometime by requesting to its accounting office to write the budget (*writeBudget*), which on the other hand takes TW time units to do so. This is specified by the second action. When the time is reached, in the third action, the office sends the budget to the coPI. At this point, however, the coPI does not know the title of the project (*no title*) and therefore cannot send the final proposal to the Dean's office (*uni*) for final approval. Only when the title is made public by the PI, a coPI can do so and the Dean's office requires TU time units to approve the budget. These are specified by the fourth and fifth actions. Finally, when the budget is approved, the budget is made available only to the PI by using the private channel to the leader *Al*. Here, time is another agent of the system whose unique action is to move time forward by incrementing the value in the fact $time(T)$. In our implementation, we included some more actions to the scenario, such as the action where a coPI can also revise the budget and send it back to its accounting office for modifications.

It is easy to check that the scenario described above is progressing. Each action corresponds to checking a box, that is, once it is performed, it is not repeated. This is enforced syntactically by using the time agent who moves time forward. Since actions either require some time, *e.g.*, TW and TU, to be performed or necessarily occur after another action is performed, *e.g.*, the coPI can only send the budget when the PI has made the title public, an instance of an action can never be repeated. For instance, if the example above is extended so that the coPI and its office send several versions of the budget back and forward, with revisions, all these actions are different instances of the same rules each with a different time value. Moreover, since time is discrete and all actions need to be performed until a deadline is reached, there cannot be infinitely many revisions. Hence, any computation run in this system is bounded.

We implemented the example above and the medical scenario described in [17] in the logic interpreters Bedwyr [4] and XSB [21].[2] The fragment of intuitionistic logic underlying Bedwyr subsumes the one underlying XSB. For instance, one cannot specify system compliance in XSB, but one can do so in Bedwyr. On the other hand, weak plan compliance can be specified in both systems. However, since Bedwyr is still a prototype, its performance is far exceeded by XSB which is already in its third version. Therefore, whenever possible, we opt to use XSB. For instance, we implemented the medical scenario, which requires system compliance, in Bedwyr, and the grant proposal scenario, which requires only weak plan compliance, in XSB. In a Centrino Duo 1.2 GHz machine with 1.5 GB of memory and running Linux, Bedwyr took less than two minutes to prove system compliance for the medical scenario, while XSB took less than ten seconds to prove weak plan compliance for a grant-proposal scenario with ten coPIs.

5 Related Work

As previously discussed, we build on the framework described in [16, 17]. In particular, here we introduce the notion of progressing collaborative systems and investigate the use of actions that can create nonces. We tighten the lower bounds from [16, 17] by using the progressing assumption, and we also provide new results for when nonces can be created. In [5, 6, 18], a temporal logic formalism for modeling collaborative system is introduced. In this framework, one relates the scope of privacy to the specific roles of agents in the system. For instance, in our medical scenario, the 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, the health insurance scenario discussed in [18] has many connections with our medical scenario and it seems possible to implement it in our framework.

[2] The source code can be found by following the link `http://www.math.upenn.edu/~vnigam/implementations/pcs.zip`.

Harrison *et al.* present a formal approach to access control [15]. In their proofs, they faithfully encode a Turing machine in their system. However, differently from our encoding, they use a non-commutative matrix to encode the sequential, non-commutative tape of a Turing machine. We, on the other hand, encode Turing machine tapes by using commutative multisets. Specifically, they show that if no restrictions are imposed to the systems, the reachability problem is undecidable. However, if actions are not allowed to create nonces, then the same problem is PSPACE-complete. Furthermore, if actions can delete or insert exactly one fact and in the process one can also check for the presence of other facts and even create nonces, then it is NP-complete, but in their proof they implicitly impose a bound on the number of nonces that can be created. Although related to our case with LSTSes containing monadic actions, their result is different from ours since they do not add the notions of progressing systems nor of balanced actions to their system.

Our paper is closely related to frameworks based on multiset rewriting systems used to specify and verify security properties of protocols [1, 2, 9, 10, 12, 20]. While here we are concerned with systems where agents are in a *closed room* and collaborate, there, the concern was with systems in an *open room* where an intruder tries to attack the participants of the system by manipulating the transmitted messages. This difference is reflected in the assumptions used by the frameworks. In particular, the security research considers a powerful intruder that has an unbounded memory and that can, for example, copy messages. On the other hand, we assume here that each agent has a bounded memory, technically imposed by the use of balanced actions. Therefore, the lower bounds obtained here are tighter than the results obtained in those papers.

Much work on reachability related problems has been done within the Petri nets (PNs) community, see *e.g.*, [13]. Specifically, we are interested in the *coverability problem* which is closely related to the partial goal reachability problem in LSTSes [16]. To our knowledge, no work that captures exactly the conditions in this paper has yet been proposed. For instance, [13, 19] show that the coverability problem is PSPACE-complete for 1-conservative PNs. While this type of PNs is related to LSTSes with balanced actions, it does not seem possible to provide direct, *faithful* reductions between LSTSes and PNs.

6 Conclusions and Future Work

In this paper we introduced an important class of collaborating systems called progressing collaborative systems. These systems seem to capture well many administrative processes, namely, those in which the same action does not need to be performed more than once. We obtain exact lower bounds for the weak plan compliance problem when using such systems under different conditions, tightening results from the literature. We also investigate systems with balanced actions that can create nonces. First, we use focused proofs in linear logic to formalize the operational semantics of such actions. Second, we provide lower bounds for the weak plan compliance problem for these systems. Finally, we

formalize the policy compliance problems and two examples in intuitionistic logic. Then, we use the logic interpreters Bedwyr and XSB to model-check that specific confidentiality policies are satisfied.

There are many interesting directions to follow from this work, which we intend to pursue. For instance, the upper bound for the weak compliance problem when actions are balanced and can create nonces is left open. Also, the upper and lower bounds of the other policy compliance problems, such as system compliance and plan compliance, are open for the different types of LSTSes considered in this paper.

On the implementation side, we hope to provide the means to model-check systems with actions that can create nonces. It is also interesting to leverage the work in [5, 6, 18] and specify policies in temporal logic, instead of intuitionistic logic. We expect to do so in the near future.

Acknowledgments: We thank John Mitchell for suggesting to us the grant proposal scenario and Elie Bursztein for the intuition that the set of balanced actions needs to be restricted in some way. We also acknowledge the fruitful discussions with Paul Rowe, Carolyn Talcott, Anupam Datta, and Dale Miller as well as their helpful suggestions and comments.

Scedrov, Nigam, and Kanovich were partially supported by ONR Grant N00014-07-1-1039, by AFOSR MURI "Collaborative policies and assured information sharing", and by NSF Grants CNS-0524059 and CNS-0830949.

References

1. Roberto M. Amadio and Denis 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.
2. Roberto M. Amadio, Denis Lugiez, and Vincent Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theor. Comput. Sci.*, 290(1):695–740, 2003.
3. Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.
4. David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. The Bedwyr system for model checking over syntactic expressions. In Frank Pfenning, editor, *cade07*, number 4603, pages 391–397. Springer, 2007.
5. Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*, pages 184–198, 2006.
6. Adam Barth, John C. Mitchell, Anupam Datta, and Sharada Sundaram. Privacy and utility in business processes. In *CSF*, pages 279–294, 2007.
7. Iliano Cervesato, Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *CSFW*, pages 55–69, 1999.
8. Iliano Cervesato and Andre Scedrov. Relating state-based and process-based concurrency through linear logic (full-version). *Inf. Comput.*, 207(10):1044–1077, 2009.
9. Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An np decision procedure for protocol insecurity with xor. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 261, Washington, DC, USA, 2003. IEEE Computer Society.

10. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 271, Washington, DC, USA, 2003. IEEE Computer Society.
11. Stephen 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.
12. Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.
13. Javier Esparza and Mogens Nielsen. Decidability issues for petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.
14. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
15. Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. On protection in operating systems. In *SOSP '75: Proceedings of the fifth ACM symposium on Operating systems principles*, pages 14–24, New York, NY, USA, 1975. ACM.
16. Max Kanovich, Paul Rowe, and Andre 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.
17. Max Kanovich, Paul Rowe, and Andre Scedrov. Collaborative planning with confidentiality. *Journal of Automated Reasoning, Special Issue on Computer Security: Foundations and Automated Reasoning*, 2010. To appear. This is an extended version of a previous paper which appeared in CSF'07.
18. Peifung E. Lam, John C. Mitchell, and Sharada Sundaram. A formalization of hipaa for a medical messaging system. In Simone Fischer-Hübner, Costas Lambrinoudakis, and Günther Pernul, editors, *TrustBus*, volume 5695 of *Lecture Notes in Computer Science*, pages 73–85. Springer, 2009.
19. Y.E. Lien N.D. Jones, L.H. Landweber. Complexity of some problems in petri nets. *Theoretical Computer Science*, 4:277–299, 1977.
20. Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is np-complete. *Theor. Comput. Sci.*, 299(1-3):451–475, 2003.
21. Konstantinos Sagonas, Terrance Swift, David S. Warren, Juliana Freire, Prasad Rao, Baoqiu Cui, Ernie Johnson, Luis de Castro, Rui F. Marques, Steve Dawson, and Michael Kifer. *The XSB Version 3.0 Volume 1: Programmer's Manual*, 2006.
22. Alwen Tiu. Model checking for π-calculus using proof search. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653, pages 36–50. Springer, 2005.