ACSIJ Advances in Computer Science: an International Journal, Vol. 3, Issue 3, No.9 , May 2014
ISSN : 2322-5157
www.ACSIJ.org

ACSIJ
WWW.ACSIJ.ORG

# Progressing Collaborative Systems

**Max Kanovich[1], Tajana Ban Kirigin[2], Vivek Nigam[3], and Andre Scedrov[4]**

[1]**University College London, UCL-CS, UK, m.kanovich@ucl.ac.uk**
[1]**Queen Mary, University of London, UK, mik@dcs.qmul.ac.uk**
[2]**University of Rijeka, HR, bank@math.uniri.hr**
[2]**Federal University of Paraíba, Brazil, vivek.nigam@gmail.com**
[4]**University of Pennsylvania, USA, Email: scedrov@math.upenn.edu**

**Abstract**

This paper builds on existing multiset rewriting models for collaborative systems. We formalize Progressing Collaborative Systems by restricting repetitions of actions. Namely, we consider processes where instances of actions are used only a bounded number of times. Administrative processes usually involve such progressing behavior, *i.e.* whenever a transaction is performed, it does not need to be repeated. We investigate the complexity of the reachability problem and the planning problem for Progressing Collaborative Systems that may create fresh values. We show that these problems are NP-complete when actions are balanced and the size of facts is bounded.

***Keywords****: Computational Complexity, NP-completeness, Collaborative Systems, Progressing, Fresh Values*

## 1. Introduction

This paper formalizes *Progressing Collaborative Systems that may create fresh values*. Our model is based on existing multiset rewriting models for collaborative systems [16, 19, 14] where agents work together in order to achieve a common goal. They perform actions, specified by rewrite rules, which change the system's state of the world or configuration. Configuration of the system is specified by a multiset of facts. Since agents do not completely trust each other, they want to avoid some undesired states [19].

In [15], we introduced the class of Progressing Collaborative Systems, which was inspired by administrative and business systems as well as protocol sessions. Namely, in the execution of security protocols, once one step of a protocol session is taken, the same step is not repeated.

Similarly, many administrative and business processes not only have a bounded number of transactions, but also have a progressing behavior: whenever a transaction is performed, it does not need to be repeated. Such a process is always advancing and it is completed within a bounded number of transactions. Thus, the system is always progressing.

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 has his 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 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 small number of times.

In [15], however, we limited ourselves to systems that did not feature the creation of fresh values, such as nonces in security protocols. This is a serious limitation as administrative processes often do generate fresh values. For instance, when a new transaction is issued, a fresh value is assigned to it, so that it can be uniquely identified (see the discussion in [14]).

Combining the progressing condition with the creation of fresh values turned out to be surprisingly challenging because of a subtle interaction between the two features. In order to overcome this difficulty, we restrict our attention to balanced systems with facts of bounded size. Intuitively, these conditions impose a bound on the memory of the system. These restrictions also allow us to use similar ideas as those described in [14]. Moreover, our definition of progressing in systems that can create fresh values is motivated by the solution for handling nonces in our PSPACE-completeness result in [14].



Copyright (c) 2014 Advances in Computer Science: an International Journal. All Rights Reserved.

The main idea is that agents do not need to create new nonces. Instead, they can simply *re-use nonce names* from a set of nonce names fixed a priori. Whenever a nonce name is used, it is picked in a way that it appears to be fresh to the participants of the system. Using above machinery, we are able to show that the related reachability and planning problems are both NP-complete (Theorem 3 and Theorem 4).

The paper is organized as follows. We introduce, in Section 2, the model of collaborative systems. Section 3 contains the formalization of Progressing Collaborative Systems that may create nonces. We argue that its precise formalization is only meaningful when bounding the memory of the participants of the system. In Section 4 we prove the NP-completeness of the reachability and of the planning problem for progressing systems. Finally, in Sections 5 and 6 we comment on related work and conclude by pointing to future work.

This paper contains the extended and improved material on progressing collaborative systems from the conference papers [17] and [15]. Besides containing the proofs and more detailed explanation, we additionally consider confidentiality issues and investigate the complexity of the planning problem for progressing systems.

## 2. Preliminary: Collaborative Systems with confidentiality and bounded memory

In this section we review the main vocabulary and concepts for multiset rewriting models of collaborative systems introduced in [16, 19, 14].

Assume fixed a sorted first-order alphabet consisting of constant symbols, $c_1, c_2, \ldots$, function symbols, $f_1, f_2, \ldots$, and predicate symbols, $P_1, P_2, \ldots$ 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.

A *fact* is a ground, atomic formula over multi-sorted terms. Facts have the form $P(t_1, \ldots, t_n)$ where $P$ is an $n$-ary predicate symbol, where $t_1, \ldots, t_n$ are terms, each with its own sort.

The *size of a fact* is the total number of term and predicate symbols it contains. We count one for each predicate, function, constant, and variable symbols. We use $|F|$ to denote the size of a fact $F$. For example, $|P(x,c)| = 3$, and $|Q(f(z,x,n),z)| = 6$.

In the remainder of this paper we will assume an upper bound on the size of facts, as in [5, 11, 19, 14]. As we argue later in this section, the combination of a bound on the size of facts and the use of balanced actions imposes a *bound on the memory* of the system. This will be key for the decidability of the problems that we deal with in this paper.

A *state* or *configuration* of the system is a finite multiset of grounded facts, *i.e.*, facts that do not contain variables. Intuitively configurations specify the state of the world. For example, the following multiset of facts

$$\{\, Nurse(\mathrm{Tom}, \mathrm{id}_1, \mathrm{blood}), Nurse(\mathrm{Bob}, \mathrm{id}_2, \mathrm{blood}), \\ TestResult(\mathrm{id}_1, \mathrm{ok}), Lab(\mathrm{id}_2, \mathrm{blood}) \,\}$$

denotes that the nurse has collected and labeled blood samples from two patients and that the results are ready for only one of those samples.

Configurations are modified by actions, which are in general multiset rewrite rules of the following form:

$$X_1, \ldots, X_n \longrightarrow \exists \vec{x}. Y_1, \ldots, Y_m \tag{1}$$

where the $X_i$s and $Y_j$s are facts. The collection $X_1, \ldots, X_n$ is called the *pre-condition* of the rule, while $Y_1, \ldots, Y_m$ is called its *post-condition*.

We assume that all free variables are universally quantified at the head of the rule. By applying the rule for a ground substitution ($\sigma$), the pre-condition $(X_1\sigma, \ldots, X_n\sigma)$ to which this substitution has been applied is replaced with the post-condition $(Y_1\sigma, \ldots, Y_m\sigma)$ to which the same substitution has been applied.

In this process, the existentially quantified variables ($\vec{x}$) appearing in the post-condition are replaced by fresh constants, also called *nonces* in protocol security literature. The rest of the configuration remains untouched. For example, we can apply the rule

$$P(x), Q(y) \rightarrow \exists z. R(x,z), Q(y)$$

to the configuration $V, P(t), Q(s)$ to get $V, R(t,c), Q(s)$, where the constant $c$ is new, and $V$ is some multiset of facts. Notice the role of fresh values, for example in the rule

$$Nurse(x, \mathrm{blank}, \mathrm{blood}) \rightarrow \exists t. Nurse(x, t, \mathrm{blood})$$

which denotes labeling of blood samples. Fresh values ensure that each sample has a unique identification number assigned.

Given a multiset rewrite system $T$, one is often interested in the *reachability problem*:

Is there a sequence of (0 or more) rules from $T$ which transforms configuration $W$ into $Z$? If this is the case then we say that $Z$ is reachable from $W$ in $T$, and we call any such sequence of actions a *plan*.

### 2.1 Policy Compliances

Kanovich *et al.* [16, 19] proposed policy compliances to model confidentiality issues in collaborative systems. Since agents need to communicate and exchange data in order to achieve a final goal, it is often necessary for them to share some private information with other agents. For example, a customer needs to share his credit card details with the personnel at a store or in a restaurant in order for a credit card payment to be effected.



Copyright (c) 2014 Advances in Computer Science: an International Journal. All Rights Reserved.

Although agents might be willing to share some private information with some of the agents, they might not be willing to do the same with others.
For example, a client could share his id number or the credit card pin number with the bank, which is trusted, but not with another client. One is, therefore, interested in determining whether the given system complies with some *confidentiality policies*: when a credit card payment is made, the credit card details should not be improperly shared or used for other people's payments.

In order to specify private and shared information in multiset rewrite systems, [16, 19] introduced Local State Transition Systems (LSTS). In LSTSes the system configuration is partitioned so that there exists a public configuration, which is accessible to all agents, and a number of local configurations each of which is accessible only to one agent. Intuitively, local configurations contain the private data of the agents of the system, while the public configuration contains the data available to all agents. The separation of the global configuration is done by partitioning the set of predicate symbols in the alphabet.
Following this intuition, the set of rules of the system is partitioned as well. Each rule belongs to an agent and, when applied, it can only modify this agent's own local configuration and the public configuration. This is formalized by restricting the facts that can be mentioned in the rule.
Typically, predicate symbols and rules are annotated with the name of the agent that owns it or with *pub* if it is public. For instance, the fact $P_A(\vec{t})$ belongs to the agent $A$, the fact $R_{pub}(\vec{t})$ is public, while the rule

$$P_A(x), Q_A(x,y), R_{pub}(t) \rightarrow_A \exists z. S_A(x,z), Q_A(z,y), R_{pub}(t)$$

belongs to agent $A$. Agents exchange their private data through public facts. For example by the following rules:

$$\begin{array}{rl} P_A(x), F_{pub}(t,s) & \rightarrow_A \quad P_A(x), P_A(s), F_{pub}(x,s) \\ Q_B(x,y), F_{pub}(t,s) & \rightarrow_B \quad Q_B(t,y), F_{pub}(t,y) \end{array}$$

agents $A$ and $B$ can exchange information. We adopt the same approach to specify private and shared information. For simplicity we omit the annotation when it is clear from the context.

Formally, our system $T$ is a multiset rewrite system specified by a signature $\Sigma$, a set of agents $I$ and the set of rules $R$ owned by the agents of the system. *Initial configuration* is the multiset of facts denoting the initial state of the system, while the *goal configuration* represents the final goal, such as the multiset of facts denoting that all scheduled patients have been visited and diagnosed by the doctor. Agents may also have confidentiality policies, given as multisets of facts, specifying situations that are undesired or unacceptable for that agent. We call a *critical configuration* any configuration that conflicts with some confidentiality policy of any agent of the system. Critical configurations of the system are simply given as the union of critical configurations of all agents.

For example, a configuration containing the facts

$$Nurse(\text{Tom}, \text{id}_1, \text{blood}), Nurse(\text{Sam}, \text{id}_1, \text{blood})$$

should be critical because it denotes that different patients are assigned the same identification number. Similarly, configuration containing the facts

$$TestResults(id, result), Chart(\text{Tom}, id)$$

is critical, since, according to hospital policies and regulations, the test results should never publicly leak together with the patient's name.

We assume that critical configurations are closed under nonce renaming, since a nonce that may appear in a critical configuration denotes a fresh value, not any particular constant, *e.g.* nonce *id* in above facts represents any identification number.

Any plan that does not reach any critical configuration is classified as *compliant*.

In [16, 19] various compliances were proposed in order to best suit the characteristics of the given collaborative system. We relate the progressing behavior to the compliance called weak plan compliance[1] and the following *planning problem*:

Given a system $T$, an initial configuration $W$, a goal configuration $Z$, and a set of critical configurations, is there a compliant plan which leads from $W$ to $Z$?

Here one models a rather high level of trust between agents by only requiring the existence of a compliant plan. Agents are expected to follow a plan that is compliant, meaning that the plan is safe with respect to given critical states of all agents. However, if some agent doesn't follow a compliant plan, then it might be possible for him to reach a state which is critical for some agent. This approach to collaboration is suitable, for example, for credit card payments when customers have to trust the personnel not to misuse the card, although they could possibly do so. It could also be adequate when the goal of a collaboration should be reached before some deadline.
We study the policy compliance in relation to progressing behavior and investigate the complexity of the corresponding planning problem in Section 4.2.

## 2.2 Balanced Actions and Fresh Values

An important condition for formalizing the notion of progressing is that of balanced actions, introduced in [19].

We classify an action as *balanced* if the number of facts in its pre-condition is the same as the number of facts in its post-condition. That is, $n = m$ in Eq.(1) :

$$X_1, \ldots, X_n \longrightarrow \exists \vec{x}. Y_1, \ldots, Y_n$$

[1]The remaining two types of policy compliance, called plan compliance and system compliance are left out of the scope of this paper as we are of the opinion that they do not particularly suit progressing behavior.



Copyright (c) 2014 Advances in Computer Science: an International Journal. All Rights Reserved.

If we restrict all actions in a system to be balanced, then the number of facts in each configuration in a plan is the same as in the initial configuration. This is because when a balanced action is applied to a configuration, the same number of facts that are removed from the configuration are also inserted into it.

The main motivation for using balanced actions is to bound the memory *i.e.* the storage capacity of agents. Since all configurations in a plan have the same number of facts, the number of facts that are known to agents at any time is bounded.

However, even with the use of balanced actions, it does not mean that there is a bound on the number of symbols (or terms) present in any configuration. One also needs a *bound on the size of facts*. Otherwise, an agent would be able to store as many symbols as he wants by using for instance a pairing rule:

$$M(t_1), M(t_2) \rightarrow M(\langle t_1, t_2 \rangle), M(t_2)$$

Notice that this action is balanced, but the number of symbols that appear in the post-condition is greater than the number of symbols that appear in the pre-condition.

Without bounding the size of facts, one would be able to store an unbounded number of symbols in the term level. In this way it would be possible for a system configuration, to contain as much information as needed. On the other hand, bounding the size of facts implies that the pairing rule would not be allowed if the size of the fact $M(\langle t_1, t_2 \rangle)$ is greater than the upper-bound.

These two conditions, namely an upper-bound on the size of facts and the restriction to balanced actions, are crucial for our decidability results, as the known results imply:

- Kanovich, Rowe and Scedrov have shown [16] that for *unbalanced systems* the reachability problem is undecidable even if the size of facts is bounded;
- It was also shown [11] that *if one does not bound the size of facts*, then the reachability problem is undecidable even if the system is balanced.

In contrast, our previous work [14] shows that the reachability problem is PSPACE-complete if we assume both a balanced system and an upper-bound on the size of facts. The key insight for this result was showing how to handle the fact that a plan may contain an exponential number of nonces (for more details on plans of exponential length see [14]), which seems to preclude PSPACE membership. We circumvent the problem of requiring too many fresh values in a plan by reusing obsolete constants instead of creating new values.

In more details, the argument is roughly the following: Since all actions of the system are balanced, the number of facts in any configuration is the same as the number of facts in the initial configuration, say $m$. Moreover, as we assume an upper bound on the size of facts, say $k$, then any configuration in a plan contains at most $mk$ symbols. We can then a priori fix a polynomial number of nonce names, namely, $2mk$ names, so that whenever one needs a fresh nonce, one can find a name in this set of $2mk$ names that will appear fresh to the participants. It suffices to take a nonce name that does not appear anywhere in the configuration. It may well be the case that some name in this set of nonce names is used many times in a plan. However, to the participants, at that point, the name used seems fresh as no participant remembers it. For further details we point to our previous work [14].

The idea of reusing names for fresh values will be key both for our proposal of progressing systems with nonce generation in Section 3 and for the complexity results in Section 4.

## 3. Progressing Collaborative Systems

We introduced the notion of progressing in [15]. Here we are interested in systems where all agents have bounded memory and are also able to update nonces.

Progressing is inspired by the nature of security protocols, as well as many administrative processes. Namely, once one step of a protocol session is taken, the same step is not repeated. Similarly, whenever one initiates some administrative task, one receives a "to-do" list with the activities or tasks that have to be performed or achieved. Once an item on the list has been "checked", one does not need to return to this item anymore. When all the items have been checked, the process ends. Such a process is always advancing and it is completed within a bounded number of transactions.

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 small number of times.

Additionally, such processes often manipulate a bounded number of values. Consider, for example, the simple process where a bank customer needs a new PIN number: The bank will assign the customer a new PIN number, which is often a four digit number and hence bounded. Even when a customer is allowed to choose a PIN number or some password, it has to satisfy some conditions, e.g. all its characters must be alphanumeric and, in practice, the password is bounded since users are never able to use an unbounded password due to buffer sizes, etc. Consequently, protocols and administrative processes have a polynomial number of steps with respect to the given inputs. In other words they can be considered as *efficient*: one does not need to perform an exponential number of actions to conclude such processes.



Copyright (c) 2014 Advances in Computer Science: an International Journal. All Rights Reserved.

To formally capture this intuition, we defined progressing in [15] as follows: A sequence of actions is progressing if *an instance* of an action appears at most once. In our previous work [15] no nonces were allowed. Then, an instance of an action is obtained by a substitution which replaces all variables appearing in the pre- and post-condition of the action with constants. Assuming a finite signature, *i.e.*, a finite number of constant symbols, there is a finite number of instances of any action. This notion of progressing reflects the requirement that progressing processes are efficient, since, in order to check whether a process can be completed or not, one only needs to consider polynomial number of actions w.r.t. the number of symbols in the signature. For instance, the well known Towers of Hanoi problem has no progressing plans [14], since any solution is of exponential length, which implies that one and the same action is necessarily used an exponential number of times.
In [15] we show that the progressing reachability problem for systems that do not create nonces is NP-complete.

We now extend the notion of progressing to systems that also allow the creation of fresh values since in administrative systems one often needs to generate fresh values. For instance, whenever a new administrative process is initiated, one creates a fresh identifier different to the identifiers of all the existing processes. In this way, actions needed for different processes are not mixed up. In [14], we provide further illustrative examples for the need of fresh values in administrative processes.
However, extending the notion of progressing to systems that can create nonces turned out to be quite challenging. Namely, with the creation of fresh values, one may capture processes which cannot be efficiently carried out. In order to demonstrate this, let us try to *naively* extend the above progressing definition as follows: A sequence of actions that may create fresh values is progressing if an instance of an action, with the same constants and the same nonces, appears at most once. Unfortunately, such a definition of progressing is not satisfactory. When a nonce is created, it is fresh, meaning that it hasn't appeared in the system as yet. Consequently, every application of an action that creates a nonce is a new instance of that action.
For instance, we can adapt the encoding of the Towers of Hanoi, so that each move creates a new nonce. Then, each action is a different instance, because a different nonce is used. Therefore, according to the above naive definition, the Towers of Hanoi puzzle would be progressing, which is clearly not what we want.

In order to extend the notion of progressing we shouldn't allow unbounded nonce generation. Instead we need to somehow limit the use of nonces, but how many nonces is enough? This question is answered for the case when systems are *balanced*, as already discussed in Section 2: One can simulate any plan that uses an unbounded number of nonces by fixing a priori a polynomial number of nonce names [14] with respect to the number of facts in the initial configuration and the upper-bound on the size of facts. We formalize these intuitions next.

## 3.1 Progressing in Systems with Fresh Values

We now extend the notion of progressing for systems that can create fresh values. From now on we assume that the system is balanced and the size of facts is bounded.
The next definition will be central to our new notion of progressing. Consider for example the following two instances of an action, where the $t_i$s are terms and $n_j$s are nonce names which do not appear in the alphabet:

$$X_1(t_1)X_2(t_2,t_3,n_1)X_3(n_1,n_2) \rightarrow X_4(t_1)X_2(t_2,t_1,n_3)X_5(n_1,n_3)$$

$$X_1(t_1)X_2(t_2,t_3,n_4)X_3(n_4,n_5) \rightarrow X_4(t_1)X_2(t_2,t_1,n_6)X_5(n_4,n_6).$$

These instances only differ in the nonce names used: the same fresh value, $n_3$ in the former instance and $n_6$ in the latter, appear in the same facts exactly at the same places, and similarly, for the pairs of nonces $(n_1, n_4)$, and $(n_2, n_5)$. Inspired by a similar notion in $\lambda$-calculus [7], and $\alpha$-equivalence among configurations in [14], we regard instances of actions that differ only in the nonce's names used, as equivalent.

**Definition 1** *Two* instances *of an action, $r_1$ and $r_2$, are* equivalent *if there is a bijection $\sigma$ that maps the set of all nonce names appearing in one instance to the set of all nonce names appearing in the other instance, such that $r_1\sigma = r_2$.*

The two instances given above are equivalent because of the following bijection $\{(n_1,n_4),(n_2,n_5),(n_3,n_6)\}$. It is easy to show that the above relation among instances of actions is indeed an equivalence relation.

**Definition 2** *Given a system $T$, an initial configuration $W$ and a polynomial $f(m,k)$, we say that a sequence of actions is* progressing *if it contains at most $f(m,k)$ equivalent instances of any action, where $m$ is the number of facts in $W$ and $k$ is the upper bound on size of facts.*

Notice that, by Definition 2, not every computation could be considered as progressing. Since a nonce name may only be used by the same action a polynomial number of times in a plan, not every problem that has a solution will have a progressing solution. This is formalized by the polynomial $f$, reflecting that the process is efficient.
For example, in any solution to the Towers of Hanoi puzzle, one and the same nonce name has to be updated an exponential number of times by the only action from the representation of this puzzle given in [14]. Therefore this problem has no progressing solution as per Definition 2, as expected.



Copyright (c) 2014 Advances in Computer Science: an International Journal. All Rights Reserved.

Our new notion of progressing extends progressing from [15], as they coincide when systems do not allow fresh values. We point out once more that, if nonces are allowed, we only conceive progressing in balanced systems, while progressing with no nonces is clear for any multiset rewriting system, even the unbalanced ones. This is because nonce update from [14] is only possible for balanced systems.

As before, we are interested in the reachability and the planning problem, but only allow progressing plans:

- (*Progressing reachability problem*) Given a system $T$ and configurations $W$ and $Z$, is there a *progressing* plan from $W$ to $Z$?
- (*Progressing planning problem*) Given a system $T$, an initial configuration $W$, a goal configuration $Z$, a set of critical configurations, an upper-bound on the size of facts, an alphabet with a finite number of constant and function symbols, and a polynomial with two parameters, is there a *compliant progressing plan* from $W$ to $Z$?

## 4. Complexity Results

In this section we investigate the progressing reachability problem when actions can create fresh values. We then turn to the progressing planning problem.

### 4.1 Complexity of the Progressing Reachability Problem

Recall that reachability is generally undecidable [16, 11], and is PSPACE-complete for balanced systems with facts of bounded size [14].
We now show that the progressing condition improves the complexity of reachability, as stated by the next theorem.

**Theorem 3** *Given a multiset rewrite system $T$ with only balanced actions that can create fresh values, an initial and a final configuration, an upper-bound, $k$, on the size of facts, an alphabet with a finite number of constant and function symbols, and a polynomial $f$ with two parameters, the progressing reachability problem is NP-complete.*

**Proof** We infer the NP lower bound from the encoding of the 3-SAT problem from [15], which is well-known to be NP-complete [10].
For the NP upper bound, assume given an initial configuration with $m$ facts and a polynomial $f$ with two parameters. Moreover, let $n$ be the number of rules in $T$, $d$ the number of constant and function symbols, $k$ the upper bound on the size of facts and $l$ the upper bound on the number of different variables appearing in a rule in $T$. We assume $k$ and $l$ to be much smaller than $d$ and $m$.
Following [14], we can assume that all nonces are used from a set of $2mk$ nonce names that is fixed a priori. Hence, the number of constants in the system is $d+2mk$. As actions are applied, instead of fresh values being created, nonces are updated. Obsolete nonce names are picked from the fixed set of $2mk$ nonce names. They are, therefore, different from any nonce in the configuration and can be considered fresh.
Since the size of facts is bounded, we do not need to consider terms that have the size greater than $k$. Therefore we need to consider at most $(d+2mk)^k$ terms.
Since in a progressing plan, one is allowed to use only a polynomial number of instances of any given rule, the length of a plan is bounded by

$$f(m,k) \times n \times \left((d+2mk)^k\right)^l = f(m,k) \times n \times (d+2mk)^{kl}.$$

which is polynomial in the size of the configurations, number of rules and symbols.
Assume that $W$ is the initial and $Z$ is the goal configuration. We show that we can check in nondeterministic polynomial time, whether there exists a plan that solves the progressing reachability problem.
Let $S_i$ be the configuration at step $i$, so $S_0 = W$,and let $Q_i$ be the multiset of pairs, $\langle r, \sigma \rangle$, of rules and substitutions used before step $i$, so $Q_0 = \emptyset$.
The following algorithm checks for valid computations:

1. Check if $Z \subseteq S_{i-1}$, then ACCEPT; otherwise continue;
2. Guess an action $r_i : X_i \rightarrow Y_i$ from $T$, and a substitution $\sigma_i$;
3. Check if $X_i\sigma_i \in S_{i-1}$, then continue; otherwise FAIL;
4. Check if the multiplicity of $\langle r_i, \sigma_i \rangle$ in $Q_{i-1}$ is greater than $f(m,k)$, then FAIL; otherwise continue;
5. $S_i = S_{i-1} \cup \{Y\sigma_i\} \setminus \{X\sigma_i\}$;
6. $Q_i = Q_{i-1} \cup \{\langle r_i, \sigma_i \rangle\}$;
7. 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 4. However, the set $Q_i$ is bounded by the length of the computation.
Therefore, the progressing reachability problem is in NP. □

### 4.2 Complexity of the Progressing Planning Problem

We now investigate the policy compliance in relation to progressing behavior and show that the NP-completeness result for the reachability problem extends to the planning problem. If no restrictions on the systems are made, the planning problem is undecidable [16], even if actions do not create nonces. As discussed in [16, 14], the restriction to balanced actions provides decidability of planning problem. In [16, 19, 14] plans were allowed to contain an instance of an action as many times as needed. Here, however, we assume that the system is progressing, *i.e.*, we only consider progressing plans.



Copyright (c) 2014 Advances in Computer Science: an International Journal. All Rights Reserved.

We assume that when given a system $T$, there is a program $\mathscr{C}$ that returns the value 1 in polynomial time when given as input a configuration that is critical in $T$ and returns 0 otherwise.

**Theorem 4** *Given a system $T$ with only balanced actions that can create fresh values, an initial and a goal configuration, a finite set of critical configurations, an upper-bound on the size of facts, an alphabet with a finite number of constant and function symbols, and a polynomial $f$ with two parameters, the progressing planning problem is NP-complete.*

**Proof** The proof is similar to the proof of Theorem 3. We adapt the upper bound algorithm to ensure that the plan is compliant.
Let $m$ be the number of facts in the given initial configuration $W$, $Z$ be the goal configuration and $k$ be the upper bound on the size of facts. Let $S_i$ be the configuration at step $i$, so $S_0 = W$, and let $Q_i$ be the multiset of pairs, $\langle r, \sigma \rangle$, of rules and substitutions used before step $i$, so $Q_0 = \emptyset$.
The following algorithm checks for compliant plans:

1. Check if $\mathscr{C}(S_{i-1}) = 1$, then FAIL; otherwise continue;
2. Check if $Z \subseteq S_{i-1}$, then ACCEPT; otherwise continue;
3. Guess an action $r_i : X_i \to Y_i$ from $T$, and a substitution $\sigma_i$;
4. Check if $X_i\sigma_i \in S_{i-1}$, then continue; otherwise FAIL;
5. Check if the multiplicity of $\langle r_i, \sigma_i \rangle$ in $Q_{i-1}$ is greater than $f(m,k)$, then FAIL; otherwise continue;
6. $S_i = S_{i-1} \cup \{Y\sigma_i\} \setminus \{X\sigma_i\}$;
7. $Q_i = Q_{i-1} \cup \{\langle r_i, \sigma_i \rangle\}$;
8. Increment $i$.

In step 1. we check that the plan does not contain any critical configuration and in step 5. we make sure that only a polynomial number of instances of any action is used. Since the length of progressing plans is polynomial in the number of facts in configurations, number of actions and symbols, the above algorithm is NP w.r.t. the number of facts in configurations, upper-bound on the size of facts, the number of actions and constant and function symbols in the alphabet, the polynomial $f$ and the size of the program $\mathscr{C}$. Therefore, the progressing planning problem is in NP. □

## 5. Related Work

As already discussed, we build on the framework described in [16, 19, 14]. In particular, we formalize the notion of progressing in collaborative systems with fresh values. We tighten the upper bounds of the planning problem from [16, 19, 14] by using the progressing assumption, moreover show that our results also apply to systems with fresh values.

Our paper is closely related to frameworks based on multiset rewriting systems used to specify and verify security properties of protocols [1, 2, 6, 9, 11, 22]. Our NP-completeness results for progressing systems are different from the NP results obtained for protocol insecurity in [2, 22]. While here we are concerned with systems where agents are in a *closed room* and collaborate, they consider systems in an *open room* where an intruder tries to attack the participants of the system by manipulating the transmitted messages.
While no bound on the size of adversary messages is assumed in [2, 22], the protocol and adversary theories used in [2, 22] are intended for protocol security. Moreover, their NP complexity results rely exactly on the nature of adversary rules, namely on the composition and decomposition adversary rules, and do not apply to general collaborative systems (see [18]). On the other hand, for our NP upper bound for the progressing reachability and the progressing planning problem, we need to assume a bound on the size of facts. This condition normally appears in the specification of administrative processes, where only tokens are used and no function symbols [18]. Although lifting this bound could make sense if no function symbols were allowed, the bound on the depth of terms is deeply embedded in the semantics of our balanced systems.

In [3, 4, 20], 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, patient's test results, which normally should not be accessible to every agent in the system, 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.

Harrison *et al.* present a formal approach to access control [13]. They show that if no restrictions are imposed to the systems, the reachability problem is undecidable. However, 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, their result is different from ours since they do not add the notions of progressing nor of balanced actions to their system.

Much work on reachability related problems has been done within the Petri nets community, see *e.g.*, [12]. Specifically, we are interested in the *coverability problem* which is closely related to the reachability problem in multiset rewrite systems. To the best of our knowledge, no work that captures exactly the balanced condition or the progressing with nonce creation has yet been proposed. In these cases, it does not seem possible to provide direct, *faithful* reductions between our systems and Petri nets.



Copyright (c) 2014 Advances in Computer Science: an International Journal. All Rights Reserved.

## 6. Conclusions

The main contribution of this paper is the formalization of progressing for systems that can create fresh values and have bounded memory. We believe that this fragment will provide foundations for a useful class of systems, namely for systems such as administrative and business processes where the same instance of an action should not be performed a large amount of times, *e.g.* an exponential number of times. Additionally, we prove the NP-completeness of the progressing reachability problem and the progressing planning problem.

There are many interesting directions to follow from this work, which we intend to pursue. For instance, in [18, 17] we introduced the dimension of time into our rewriting models, and we plan to research progressing in timed systems.

Together with Carolyn Talcott, we are investigating the use of the computational tool Maude [8] for the specification and model-checking of regulated processes, such as administrative processes [18, 21]. In particular, we are investigating whether our NP-completeness proof can improve Maude's performance in model-checking Progressing systems.

**Acknowledgments:**

We thank John Mitchell and Elie Bursztein for the intuition that the set of balanced actions needs to be restricted in some way. We also acknowledge fruitful discussions with Paul Rowe, Carolyn Talcott, Anupam Datta and Dale Miller as well as their helpful suggestions and comments. Scedrov was partially supported by NSF, ONR, and by MURI program through AFOSR. Nigam was partially supported by the Alexander von Humboldt Foundation and CNPq. Kanovich was partially supported by the EPSRC

**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] 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.

[4] Adam Barth, John C. Mitchell, Anupam Datta, and Sharada Sundaram. Privacy and utility in business processes. In *CSF*, pages 279–294, 2007.

[5] 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.

[6] 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.

[7] Alonzo Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

[8] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude: A High-Performance Logical Framework*. Springer, 2007.

[9] 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.

[10] 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.

[11] 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.

[12] Javier Esparza and Mogens Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.

[13] 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.

[14] Max Kanovich, Tajana Ban Kirigin, Vivek Nigam, and Andre Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. *Inf. Comput.* Accepted for Publication.

[15] Max Kanovich, Tajana Ban Kirigin, Vivek Nigam, and Andre Scedrov. Progressing collaborative systems. In *FCS-PrivMod*, 2010.



Copyright (c) 2014 Advances in Computer Science: an International Journal. All Rights Reserved.

[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 I. Kanovich, Tajana Ban Kirigin, Vivek Nigam, and Andre Scedrov. Bounded memory protocols and progressing collaborative systems. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 309–326. Springer, 2013.

[18] Max I. Kanovich, Tajana Ban Kirigin, Vivek Nigam, Andre Scedrov, Carolyn L. Talcott, and Ranko Perovic. A rewriting framework for activities subject to regulations. In Ashish Tiwari, editor, *RTA*, volume 15 of *LIPIcs*, pages 305–322. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[19] Max I. Kanovich, Paul Rowe, and Andre Scedrov. Collaborative planning with confidentiality. *J. Autom. Reasoning*, 46(3-4):389–421, 2011.

[20] 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.

[21] Vivek Nigam, Tajana Ban Kirigin, Andre Scedrov, Carolyn L. Talcott, Max I. Kanovich, and Ranko Perovic. Towards an automated assistant for clinical investigations. In Gang Luo, Jiming Liu, and Christopher C. Yang, editors, *IHI*, pages 773–778. ACM, 2012.

[22] 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.

**Max Kanovich** is a Professor of Computer Science at the University College London. He obtained his Ph.D. in Mathematics and Physics from Moscow (Lomonosov) University in 1971 and Doctor of Science (Dr. Hab) in Mathematics and Physics from Moscow (Lomonosov) University in 1989. He was a Lecturer, Associate Professor, and Full Professor in Mathematics and Computer Science at the Tver' University, Russian University for the Humanities, University of Pennsylvania, and Queen Mary College, University of London.

**Tajana Ban Kirigin** is a lecturer at the Department of Mathematics, University of Rijeka. She obtained her Ph.D. in Mathematics from the Faculty of Natural Sciences and Mathematics, Department of Mathematics, University of Zagreb, Croatia, in 2011.

**Vivek Nigam** is a lecturer at the Federal University of Paraíba, Brazil. He obtained his Ph.D. from the École Polytechnique, France, in 2009 and was as a postdoctoral researcher at the University of Pennsylvania from 2009 to 2010 and then a postdoctoral research at the Ludwig Maximilian University in Munich, Germany from 2010 until 2012.

**Andre Scedrov** is a Professor of Mathematics and Computer and Information Science at the University of Pennsylvania. He obtained his Ph.D. in Mathematics from State University of New York at Buffalo in 1981. Before coming to Penn in 1982 he was a T.H. Hildebrandt Research Assistant Professor of Mathematics at University of Michigan, Ann Arbor.



Copyright (c) 2014 Advances in Computer Science: an International Journal. All Rights Reserved.