Analysis of Dynamic Policies ⋆

A. Prasad Sistla, Min Zhou

University of Illinois, Department of Computer Science, Chicago, USA

Abstract

The paper considers the problem of modeling and analyzing dynamic policies in Trust Management Systems. It presents a general frame work and a specification language FCTL for specifying properties in such systems. Various important security properties such as simple safety, liveness, role containment can all be elegantly specified in this logic. It presents upper and lower bounds for checking classes of formulas specified in this logic for dynamic systems specified in the RT_0 Role based Trust management system.

Key words: Dynamic Access Control Policies, Security Properties, Trust Management Systems, Role Containment, Tree Automata, First Order CTL.

1 Introduction

With the rapid increase in the Internet based distributed and network computing, it is of paramount importance to develop methods for secure access and sharing of resources in such systems. Toward this end, many Trust Management (TM) Systems [5,22,21,12,14] have been proposed in the literature. These systems allow distributed specification of a policy by multiple trusted entities, giving access rights to resources to various principals in the system. In such systems there is a dire need for formal mechanisms and tools for analyzing the current state of a policy. A number of techniques have been developed in the literature [8,18] for analyzing specific properties of the current state of a policy.

More recently, there has been much interest in analyzing security properties in dynamic systems, i.e., systems in which the policy may change with some restrictions imposed on the changes [23,20]. In this paper, we consider such

⋆ This work is partially supported by the NSF grant CCR-0205365.

dynamic systems modeled as *structures* that specify how policy changes. Essentially, a structure is a directed graph whose nodes denote states. Each state is composed of two parts, a *system state* and a *policy*. The policy part specifies access rights to various individuals. The system state part captures external system information that affects policy changes. For example, time, past access patterns or levels of security/threat may be captured as part of the system state. The edges of the structure denotes how policy changes with changes in system state. Thus, structures form natural models of dynamic policies.

Apart from the works of [23,20] where dynamic policies have been explicitly considered, there have been implicit dynamic policies that have been defined in literature. For example, the SPKI/SDSI system [14] allows implicit change in policy through valid time intervals associated with certificates and also through certificate revocation. The valid time interval implicitly specifies addition of the corresponding certificate to the policy at the beginning of the valid time interval and its deletion at the end of the valid time interval. Similarly, a certificate can be revoked by placing it on a revocation list. Such dynamic aspects are modeled in our formalism by capturing time and certificate revocation list as part of the system state. The *mandatory security controls* specified by the *Orange Book* of the military [7] specifies a dynamic policy based on classification and access patterns. In this scheme, an individual after reading from a classified document is prevented from writing on to an unclassified file, in order to prevent leakage of classified information. Similarly, the *Chinese Wall* policy [6] also defines a dynamic policy. In this scheme, resources are divided in to groups, called *conflict groups*. The first access of an individual to a group can be to any object in that group. However, any subsequent accesses by that individual to that group have to be to the same object. Policies defined in both of the above schemes can be modeled as structures where the past access patterns of individuals form the system state. Thus, structures are powerful enough to model any dynamic policy defined in existing schemes proposed in the literature.

In addition to introducing structures as models of dynamic policies, we also introduce a general purpose logic, called FCTL (First order CTL) for specifying properties of such systems. FCTL is an extension of CTL [9,10] that allows first order quantifiers. CTL has been extensively used in automatic analysis and verification of concurrent programs. In [23,20], the authors consider six different properties, called *simple safety*, *simple availability*, *bounded safety*,*liveness*, *mutual exclusion* and *universal role containment*. All these properties and many more important properties can be expressed in FCTL using formulas of the form $AG(f)$ or of the form $EF(f)$ where f is a simple first order formula and AG (EF) is temporal operator that asserts that f holds on all (on some) reachable states.

In this paper, we consider two types of dynamic systems — *role restricted* *systems* and *finite state systems*. Finite state systems are systems in which the number of states, and hence the number of policies, is finite. Role restricted systems were introduced in [23,20] and form a class of infinite state systems. They are defined using the trust management language RT_0 [22]. Each system in this class is given by an initial policy, specified in RT_0, and by a pair of subsets of *roles* which specify growth and shrink restrictions on roles; the initial policy can be changed to any policy as long as the role restrictions are respected. The need for checking of the above mentioned security properties in such systems has been very well articulated in [23,20] and needs no repeating here.

Among the above mentioned properties, checking universal containment is technically the most challenging one. From a practical point of view, this is a very important property as many critical properties can be formulated as universal role containment properties. (For example, "all the principals belonging to a role X can access a resource R" is such a property). In the above work, this problem has been shown to be solvable in non-deterministic exponential time (NEXPTIME); this means that all the known algorithms for this problem take double exponential time. In this paper we show that this problem is solvable in deterministic exponential time, thus achieving an exponential speed up over existing algorithms. The techniques we employ depend on non-trivial results on the derivability in the RT_0 language. We also show that this problem is EXPTIME-hard, thus showing that the above bound is tight. It is interesting to note that the universal role containment for the subset of RT_0 that does not include intersection inclusion has been shown to be PSPACE-complete in [23,20].

We also show that the problem of checking formulas of the form $EF(f)$ or $AG(f)$, where f is a positive or a negative formula, can be done in polynomial time by considering the *minimal* or the *maximal* states (Positive formulas are normalized formulas that have only existential quantifiers and in which all membership atomic formulas appear positively). All the previously mentioned properties, excepting universal role containment, and others such as *principal conflict* can all be expressed by formulas of the above form. This result generalizes the corresponding result of [23,20] and shows that all such properties can be efficiently checked.

We also show that the problem of checking a formula of the form $EF(f)$, where f is a simple first order formula, can be decided in non-deterministic exponential time (i.e., in the complexity class NEXPTIME). This implies that the problem of checking a formula of the form $AG(f)$ is in co-NEXPTIME. This result generalizes the result of [23,20] where the above result has been shown for the case when f asserts role containment as indicated above.

We also consider the problem of checking arbitrary FCTL formulas in finite state systems. We show that traditional model checking algorithms combined with static policy analysis algorithms can be used for checking FCTL properties. We also show how a formalism similar to timed automata can be used to specify dynamic finite state systems where the policy can be changed according to some timing conditions.

In summary, the main contributions of the paper are the following: (1) a frame work for modeling and analyzing dynamic security and the FCTL logic for specifying the security properties; (2) an upper bound result showing that universal role containment can be checked in exponential time in role restricted systems and a lower bound result showing that this bound is tight; (3) upper bounds for checking formulas of the form $AG(f)$ and $EF(f)$ in role restricted systems where f is a first order formula; (4) efficient upper bounds for checking formulas of the form $AG(f)$ and $EF(f)$ in role restricted systems where f is a positive or a negative formula; (5) application of the frame work to finite state systems and also to time dependent systems.

The rest of this paper is organized as follows. Section 2 gives the syntax and semantics of the RT_0 language and the FCTL logic together with some examples. Section 3 gives the lower bound and upper bound results for the role containment problem and also presents other decision algorithms. Section 4 shows how finite state systems can be handled. Section 5 contains comparison to related work. Section 6 contains concluding remarks. Section 7 is the appendix which contains proofs of all lemmas and theorems.

2 Preliminaries

2.1 Syntax and Semantics of the TM Language

Here we define the RT_0 Trust Management Language that was introduced in [22]. We assume that we have an infinite set $\mathcal{K}$ of principals; each such principal is given by a public key. We also assume that we have an infinite set of role names which are simply identifiers. A *role* is a principal followed by a role name separated by a dot, e.g., $A.r$ and $X.u$. A role defines a set of principals that are members of this role. Each principal A has the authority to define the members of the role of the form $A.r$. An access control permission is represented by a role, i.e., all members of the role have that access permission. The language RT_0 has the following four types of policy statements.

(1) *Simple Member*: $A.r \leftarrow K$. This means that K is a member of the role $A.r$.

(2) *Simple Inclusion:* $A.r \leftarrow B.u$. This asserts that every member of $B.u$ is also a member of $A.r$.
(3) *Linking Inclusion:* $A.r \leftarrow A.r_1.r_2$. Here $A.r_1.r_2$ is called a *linked role*. This statement asserts that $A.r$ includes all members of $B.r_2$ for every B who is a member of $A.r_1$. Note that the same principal appears on both sides of $\leftarrow$. However,through out the paper, we will allow linking inclusions with different principals on both sides $\leftarrow$; such statements can be translated in to the above type by using intermediate roles.
(4) *Intersection Inclusion:* $A.r \leftarrow B_1.r_1 \cap B_2.r_2 \cap ... \cap B_k.r_k$ where $k \geq 2$. This asserts that every principal C, who is a member of $B_i.r_i$ for each $1 \leq i \leq k$, is also a member of $A.r$.

The sub-language of RT_0, where the intersection inclusion statements have exactly two roles on the right hand side, is called BRT_0. It has been show in [23,20] that every RT_0 policy $\mathcal{P}$ can be converted into an equivalent policy $\mathcal{P}'$ in BRT_0 such that the size of $\mathcal{P}'$ is at most polynomial in the size of $\mathcal{P}$. In this paper we consider policies specified in BRT_0. An example policy specified in BRT_0, which is adapted from [23,20], is given in subsection 7.1 of the appendix. A *policy* is a set of policy statements. Let $\mathcal{P}$ be a policy. We define $Principals(\mathcal{P})$ to be the set of principals that appear in some statement of $\mathcal{P}$. We define $Roles(\mathcal{P})$ to be the set of roles $K.r$ such that $K \in Principals(\mathcal{P})$ and r is a role name appearing in some statement in $\mathcal{P}$. The semantics of a policy is given by a function $\mathcal{F}_{\mathcal{P}} : Roles(\mathcal{P}) \rightarrow 2^{Principals(\mathcal{P})}$ that assigns for each $K.r \in Roles(\mathcal{P})$ a set of principals $\mathcal{F}_{\mathcal{P}}(K.r)$. Such semantics is defined by a least fix point that satisfies all the membership and inclusion statements (i.e., the simple inclusions, the intersection inclusions and the linking inclusions) given in $\mathcal{P}$. The semantics can also be defined by translation to a set of datalog rules as given in [23,20]. For a policy $\mathcal{P}$, we let size of $\mathcal{P}$ denote the sum of the lengths of each statements in $\mathcal{P}$. If n is the size of $\mathcal{P}$ then the number of statements and the cardinality of $Principals(\mathcal{P})$ is bounded by n, while the cardinality of $Roles(\mathcal{P})$ is bounded by n^2.

2.2 Syntax and Semantics of FCTL

Structures: In order to model dynamic systems, we assume that we have a set of variables V, which essentially are system variables, whose values can change. The system variables allow us to model the fact that changes in policies may depend on external system aspects. For example, one of the system variables can denote the value of time and another might denote the security/threat level. A system state is a mapping that associates appropriate values to the variables in V. A state s is a pair $(\mathcal{P}, t)$ where $\mathcal{P}$ is a policy and t is a system state. For the state s, $Principals(s)$ and $Roles(s)$ are defined to be simply $Principals(\mathcal{P})$ and $Roles(\mathcal{P})$ respectively. When V is the empty set then the state is given by the policy $\mathcal{P}$ and in this case, we take the state to be the policy itself. We model such dynamic systems by a structure $M = (S, R, s_0)$ where S is a set of states and $R \subseteq S \times S$ is a total binary relation, i.e., for every $s \in S, \exists t \in S$ such that $(s,t) \in R$ and s_0 is the initial state. $(s,t) \in R$ means that the state can change from s to t in one step. A path is an infinite sequence of states obtained by traversing along the arcs given by R. Essentially, a structure denotes how policies may change with system states.

Role Restricted Systems: In [23,20] the authors define a class of dynamic systems specified by an initial policy $\mathcal{P}$ (specified in RT_0) and a role restriction pair $(\mathcal{G}, \mathcal{S})$ where $\mathcal{G}, \mathcal{S}$ are subsets of $Roles(\mathcal{P})$. We call such a system as a *role restricted* system. Formally, we capture such a dynamic system by a structure (S, R, s_0) defined as follows: the set V of system variables is the empty set; $s_0 = \mathcal{P}$ and S is the set of policies $\mathcal{P}'$ obtained from $\mathcal{P}$ by addition and deletion of statements with the restriction that no statement in $\mathcal{P}$ that defines a role in $\mathcal{S}$ is deleted and no new statement that defines a role in $\mathcal{G}$ is added; $R = \{(s_0, s) : s \in S\} \cup \{(s,s) : s \in S\}$. Note that S contains s_0 and policies in S may contain principals and roles not present in s_0. Note that S is an infinite set as the total number of possible roles is infinite. Also, every state in S is reachable from s_0 in one step and each state is reachable from itself. Given an initial policy $\mathcal{P}$ and a restriction pair $(\mathcal{G}, \mathcal{S})$, we let $Struct(\mathcal{P}, \mathcal{G}, \mathcal{S})$ denote the above structure.

Time Varying Systems: There is a another class of dynamic systems where policies change with time. Such systems can be modeled by a structure with a single variable *time* that takes values given by natural numbers. Each state in such a system defines a policy at a particular time. SPKI/SDSI is a system which specifies policies using certificates with valid time intervals. Since we are considering policies specified in RT_0, we assume that each certificate is specified by a statement of RT_0. Each certificate is only valid during the valid time interval associated with it. Roughly speaking, we can capture such a system by the structure (S, R, s_0) where $S = \{(\mathcal{P}, i) : 0 \leq i < \infty$ and $\mathcal{P}$ is the set of certificates that are valid at time $i\}$; $R = \{((\mathcal{P}, i), (\mathcal{P}', i{+}1)) : 0 \leq i < \infty$, $\mathcal{P}, \mathcal{P}'$ are the sets of certificates valid at times $i, i+1$ respectively $\}$.

Systems with multiple security levels: Consider a system which has different policies depending on the security/threat level. For such a system, we have a single system variable *security_level* which takes values zero through four, denoting five different levels of security with zero being the lowest level. The system has five policies $\mathcal{P}_0, ..., \mathcal{P}_4$ corresponding to the five security levels. For $0 \leq i \leq 4$, let s_i denote the state $(\mathcal{P}_i, t_i)$ where $t_i(security\_level) = i$. The set of states of the system S is given by $\{s_i : 0 \leq i \leq 4\}$. The system can go from a lower level of security to any higher level of security in one transition. However, from a higher level of security it can go down only by one level in one transition. Thus, the set of transitions $R = R_0 \cup R_1$ where

$R_0 = \{(s_i, s_j) : 0 \leq i < j \leq 4\}$ and $R_1 = \{(s_i, s_{i-1}) : 0 < i \leq 4\}$. The initial state can be s_0 indicating lowest security/threat level.

Suppose we want to impose further constraints on the system as follows. When ever, the system goes to a new security level then it should stay at that level for at least 10 units of time before it can go to the next lower level; this timing requirement is not needed if the system moves to a higher security level. The behavior of such system can be captured by introducing another variable, say *duration*, that is reset to zero whenever the security changes. This variable gets incremented in each clock tick , which is captured by a transition which increments only *duration* with out changing any other components of the state. The transitions to lower security level are present only in those states in which the *duration* variable has a value greater than or equal to 10.

FCTL definition: Now, we define First order CTL logic(FCTL) to specify the properties of dynamic systems that are modeled by structures. FCTL is an extension of CTL [10] obtained by adding first order quantifiers to it. We assume that we have a set I of first order variables which range over principals. The set of FCTL formulas are defined inductively starting with atomic formulas and using standard propositional connectives, temporal operators and existential first order quantifier. An atomic formula is either a condition on system variables or is of one of the following forms— $i \in K.r$, $K' \in K.r$, $i = j$, $i = K$— where $K.r$ is a role, K' is a principal and $i, j \in I$. We also use $true, false$ as atomic formulas with the standard semantics. Every atomic formula is a FCTL formula. More complex formulas of FCTL are built inductively as follows. If g and h are FCTL formulas then $E[g\, U\, h]$, $A[g\, U\, h]$, $EX(g)$, $\neg g$, $g \wedge h$ and $\exists i(g)$ are also FCTL formulas.

A variable $i \in I$ appearing in a formula f is said to be bound in f, if every occurrence of i in f is in the scope of a quantifier over i. Otherwise, we say i is a free variable of f. Let $free\_var(f)$ denote the set of free variables of f. An evaluation ρ of f is a mapping from $free\_var(f)$ to the set of principals $\mathcal{K}$. We denote by $\rho[i \mapsto k]$ the evaluation which maps i to k and every other variable j to $\rho(j)$.

Let f be FCTL formula. An interpretation for f is a triple (M, s, ρ) where $M = (S, R, s_0)$ is a structure, s is a state in S and ρ is an evaluation for f. We do not mention M when it is understood from the context. We define what it means for a formula f to be satisfied in an interpretation (M, s, ρ) for f inductively on the structure of f as follows. (s, ρ) satisfies $i \in K.r$ iff $\rho(i) \in \mathcal{F}_{\mathcal{P}}(K.r)$ where $\mathcal{P}$ is the policy of the state s (recall that the function $\mathcal{F}_{\mathcal{P}}$ defines the semantics of $\mathcal{P}$). (s, ρ) satisfies $i = j$ iff $\rho(i) = \rho(j)$. (s, ρ) satisfies p where p is an atomic formula which is a condition over the system state iff the system state component of s satisfies p. (s, ρ) satisfies $E[gUh]$ iff there is a path $s_1 = s, s_2, \cdots$, such that there is some s_i along the path, (s_i, ρ)

satisfies h, and for each $j < i$, (s_j, ρ) satisfies g. This means that for some path starting from s, g continues to hold until h holds. The satisfaction of $A[gUh]$ in (s, ρ) is defined similarly excepting that we require that g continue to hold until h in all the paths starting from s. (s, ρ) satisfies $EX(g)$ iff there exists a s_1 which is a successor of s (i.e., $(s, s_1) \in R$), (s_1, ρ) satisfies g. (s, ρ) satisfies $\exists i\, g(i)$ iff $(s, \rho[i \mapsto k])$ satisfies $g(i)$ for some $k \in Principals(\mathcal{P})$ where $\mathcal{P}$ is the policy of s. The meaning of $\wedge, \neg$ are defined in the normal way.

For convenience of expression, we introduce derived operators as follows: $f \vee g \equiv \neg(\neg f \wedge \neg g)$, $\forall x f(x) \equiv \neg \exists x \neg f(x)$, $AXf \equiv \neg EX \neg f$, $AFf \equiv A[trueUf]$, $EFf \equiv E[trueUf]$, $AGf \equiv \neg EF \neg f$, $f \rightarrow g \equiv \neg f \vee g$. It is to be noted that the existential quantifier in $\exists i\, \phi(i)$ ranges over the principals of the corresponding policy associated with the state. It is easily seen that $EF(f)$ (resp. $AG(f)$) holds in (s, ρ) if f holds in some (t, ρ) (resp., in all (t, ρ)) such that t is reachable from s in M. We also define bounded temporal operators $AF_{\leq k}$, for each integer $k \geq 0$ as follows: $AF_{\leq 0}(g) = g$ and for $k > 0$, $AF_{\leq k}(g) = g \vee AX(AF_{\leq (k-1)}(g))$. Essentially, $AF_{\leq k}(g)$ states that with in k steps g holds.

2.3 Expressing some important Properties

We show that many important dynamic security properties can be expressed in FCTL. For two roles $A.r, X.u$, let $X.u \supseteq A.r$ be the abbreviation for the formula $\forall i(i \in A.r \rightarrow i \in X.u)$. This formula states that every member of $A.r$ is also a member of $X.u$. The first four of the following properties are considered in [23,20]. Note that the last of these properties uses nested temporal operators illustrating the need for such formulas in complex systems. Many other important properties considered in the literature are given subsection 7.1 of the appendix.

- **Simple Safety:** There exists a reachable state in which principal K has access to resource R (i.e., indicated by the membership of K in $A.r$). $EF\,(K \in A.r)$.
- **Simple Availability:** In every reachable state, principal K has access to resource R. $AG\,(K \in A.r)$.
- **Bounded Safety:** In every reachable state, the set of all principals that have access to resource R is bounded by the finite set $\{K_1, ..., K_l\}$. $AG\,\forall i(i \in A.r \rightarrow (i = K_1 \vee ... \vee i = K_l))$.
- **Containment:** In every reachable state, every member of $A.r$ is also a member of $X.u$. $AG\,(X.u \supseteq A.r)$.
- **Bounded Revocation:** In every reachable state, if the security is at the highest level (i.e., level 4) then with in k steps the access rights of principal

K to resource R are revoked.
$AG\,(security\_level = 4 \supset AF_{\leq k}(\neg(K \in A.r)))$.

3 Decision Problems and Complexity Analysis for Role Restricted Systems

In this section we consider the decision problems for checking formulas of the form $AG(f)$ and $EF(f)$, where f is a closed first order formula, in role restricted systems. Recall that a role restricted system is given by a state, i.e., policy $\mathcal{P}$ and a restriction pair $(\mathcal{G}, \mathcal{S})$. We consider the problem of checking formulas of the form $AG(f)$ and $EF(f)$ in the initial state $\mathcal{P}$ of the structure $Struct(\mathcal{P}, \mathcal{G}, \mathcal{S})$. From the semantics, we see that $AG(f)$ is satisfied at the initial state of $Struct(\mathcal{P}, \mathcal{G}, \mathcal{S})$ iff every reachable state, i.e.,every policy $\mathcal{P}'$ obtained by adding and deleting statements to $\mathcal{P}$ in accordance with the role restrictions specified by $(\mathcal{G}, \mathcal{S})$ satisfies f.

In subsections 3.1,3.2 we show that the problem of checking if a formula of the form $AG(f)$, where f asserts role containment, holds in a role restricted dynamic system is EXPTIME-hard and is in EXPTIME respectively. In subsection 3.3, we show that checking if $AG(f)$ holds in a dynamic system for an arbitrary first order formula is in co-NEXPTIME. In subsection 3.4, we show that checking formulas of the form $AG(f)$ and $EF(f)$ where f is a positive or negative formula can be decided in polynomial time.

3.1 Lower bound for Checking Role containment

In this section we show that the problem of checking formulas of the form $AG(X.u \supseteq A.r)$, i.e., checking dynamic role containment, is EXPTIME-hard. We prove this by showing that the problem of deciding language containment for finite tree automata is reducible to checking formulas of the above form in dynamic systems. The language containment problem for tree automata has been shown to be EXPTIME-complete in [26].

Below, we give the reduction. First we define tree automata. A tree automaton is like a regular automaton except that inputs to it are labeled trees. Each node of the tree is labeled with an input symbol. The automaton starts at the root of the tree in the initial state. When ever it is at a node of the tree that has k children, it branches out into k copies and each copy moves onto the appropriate child. The state of each copy is given by the transition relation and is determined by the state of the automaton when it was at the parent and the input symbol that labels the parent. A copy at a leaf node is accepting if its state and the corresponding input symbol belong to the transition relation. An input tree is accepted if all the copies at the leaf nodes are accepting.

Now, we give a formal definition of a tree automaton. A m-ary tree is a finite tree in which each node has at most m children. We represent each node of the tree as a finite sequence of elements from the set $\{0, 1, ..., m-1\}$ so that the null sequence, denoted by $NULL$, is the root node, and if σ is any node with k-children then its children are the sequences $\sigma 0, \sigma 1, ..., \sigma(k-1)$ in that order from left to right. Thus a tree is a prefix closed non-empty subset of $\{0, 1, ..., m-1\}^*$ satisfying the following property: (a) for every $(b_0, b_1, ..., b_{i-1}, b_i) \in T$ and for every c such that $0 \leq c \leq b_i$, $(b_0, b_1, ..., b_{i-1}, c) \in T$. The leaf node is a node without any children. The arity of a node σ in T is the number of its children and is denoted by $arity(\sigma)$.

Let Δ be any set. A labeled tree over Δ is a pair (T, ρ) where $\rho : T \rightarrow \Delta$. An input to a tree automaton is a labeled tree over an input alphabet Σ. We assume that the input alphabet Σ is partitioned into sets $\Sigma_0, ..., \Sigma_m$. We require that, in an input tree, a node with k children (i.e., a node of arity k) be labeled only with an element from Σ_k. The transitions of the automaton are given by relations $\delta_0, ..., \delta_m$ where δ_k is the set of transitions on input symbols from Σ_k; in addition, ϵ-transitions (i.e., internal transitions) are given by the set δ_{-1}. Thus, a m-way tree automaton $\mathcal{A}$ is a sequence $(Q, \Sigma_0, ..., \Sigma_m, \delta_{-1}, \delta_0, ..., \delta_m, q_0)$ of length $2m+5$ where

- Q is a finite set of states,
- $(\Sigma_0, ..., \Sigma_m)$ is an alphabet sequence,
- for each i such that $0 < i \leq m$, $\delta_i \subseteq Q \times \Sigma_i \times Q^i$, $\delta_0 \subseteq Q \times \Sigma_0$ and $\delta_{-1} \subseteq Q \times \{\epsilon\} \times Q$,
- $q_0 \in Q$ is the initial state.

Note that, for $k \geq 0$, each member of δ_k gives a k-tuple of next states corresponding to the k children of a node. The operation of automaton $\mathcal{A}$, on an input tree, can be described intuitively as follows. Initially, a single copy of $\mathcal{A}$ is in state q_0 and is scanning the root of the tree. Subsequently, it splits in to multiple copies, according to the transition relations, each of which behaves as follows. Suppose a copy $\mathcal{C}$ of the automaton is in state q and is scanning a node σ labeled with a symbol $a \in \Sigma_k$. Assume $k > 0$. Then $\mathcal{C}$ non-deterministically chooses a tuple $(q, a, q_1, ..., q_k) \in \delta_k$, splits into k copies— $\mathcal{C}_1, .., \mathcal{C}_k$— the i^{th} copy $\mathcal{C}_i$ changes to state q_i and moves on to the i^{th} child of σ, i.e.,the child $\sigma(i-1)$. $\mathcal{C}$ accepts iff all the copies $\mathcal{C}_1, ..., \mathcal{C}_k$ accept. If $k = 0$ then $\mathcal{C}$ accepts iff $(q, a) \in \delta_0$. Each copy also has an option of changing state instantaneously according to the ϵ-transitions given by δ_{-1}, i.e., if $(q, \epsilon, q') \in \delta_{-1}$ then a copy $\mathcal{C}$ in state q can change to state q' without moving. The automaton $\mathcal{A}$ accepts the input tree if the initial copy accepts. The appendix gives a formal definition of acceptance in terms of runs of the automaton.

The language containment problem for tree automata has been shown to be EXPTIME-complete in [26].

Lemma 1 *The language containment problem for tree automata is polynomial time reducible to the problem of checking if a role restricted system satisfies a formula of the form $AG(f)$ where f asserts role containment. More formally, there exists a polynomial time algorithm that takes two tree automata $\mathcal{A}, \mathcal{B}$ and outputs a policy $\mathcal{P}$, a restriction pair $(\mathcal{G}, \mathcal{S})$ and two roles $A.r, X.u$ such that the formula $AG(X.u \supseteq A.r)$ is satisfied for the role restricted system iff $L(\mathcal{A}) \supseteq L(\mathcal{B})$.*

3.2 Upper Bound for Role Containment

In this section, we consider the problem of checking formulas of the form $AG(X.u \supseteq A.r)$ for a role restricted system given by an initial state $\mathcal{P}$ and a restriction rule $(\mathcal{G}, \mathcal{S})$. We give an exponential time bounded algorithm for the case when $\mathcal{S} = Roles(\mathcal{P})$; that is, none of the statements of $\mathcal{P}$ can be deleted. Using this algorithm, we can give an exponential time bounded algorithm for the general case when $\mathcal{S} \subset Roles(\mathcal{P})$. This general algorithm enumerates all possible subsets of statements that define roles in $Roles(\mathcal{P}) - \mathcal{S}$; for each such subset Y, it deletes the statements of Y from $\mathcal{P}$ to obtain another state $\mathcal{P}'$ and then checks the role containment property for the dynamic system with initial state $\mathcal{P}'$ and restriction rule $(\mathcal{G}, Roles(\mathcal{P}'))$. Theorem 1 shows that the later problem can be decided using an algorithm that runs in time $2^{O(n^4)}$ where n is the size of $\mathcal{P}'$. Since there are at most 2^n possible subsets of statements that define roles in $Roles(\mathcal{P}) - \mathcal{S}$, we see that the over all time taken by the resulting algorithm is $2^{O(n)+O(n^4)}$, which is $2^{O(n^4)}$. Thus, in the reminder of this subsection, we assume that $\mathcal{S} = \emptyset$.

Theorem 1 *Given an initial state s and a restriction rule $(\mathcal{G}, Roles(s))$ and a pair of roles $X.u, A.r \in Roles(s)$, there exists an algorithm that runs in time $2^{O(n^4)}$ and that checks if the formula $AG(X.u \supseteq A.r)$ holds at state s in the structure $Struct(s, \mathcal{G}, Roles(s))$ where n is the sum of the size of the state s and the cardinality of $\mathcal{G}$.*

The reminder of this subsection is devoted to proving the above theorem. Through out this subsection we fix the initial state s given by the policy $\mathcal{P}$. We also fix the restriction rule $(\mathcal{G}, Roles(s))$.

We say that a state t is *reachable* from s if t is obtained from s by adding new statements defining roles not in $\mathcal{G}$. We say that t is a *simple* reachable state if it is reachable from s and all the statements in $t - s$ are simple member statements and $role\_names(t) = role\_names(s)$. Note that there may be some principals in t that are not in s. Clearly $AG(X.u \supseteq A.r)$ is not satisfied at state s iff there exists a reachable state t and a witness principal K such that $K \in \mathcal{F}_t(A.r) - \mathcal{F}_t(X.u)$; in this case, the witness K may be a new principal not in $Principals(s)$. To over come this problem, we assume that there exists a principal E and a role name l such that $E.l \leftarrow E$ is a statement in s and no other statement refers to E and l, and $E.l \notin \{A.r, X.u\}$. We can always satisfy this condition by adding a dummy rule of the above form. With this assumption, it is not difficult to see that $AG(X.u \supseteq A.r)$ is not satisfied at state s iff there exists a witness principal $K \in Principals(s)$ and a reachable state t such that $K \in \mathcal{F}_t(A.r) - \mathcal{F}_t(X.u)$. It has been shown in [23,20] that $AG(X.u \supseteq A.r)$ is not satisfied at state s iff there exists a simple reachable state t satisfying the above condition. Thus it is enough if we look for simple reachable states.

We say that a state t is an *extension* of the state s, if $t = s \cup Q$ where Q is some set of simple member statements of the form $K.l \leftarrow K'$ such that $K.l \in Roles(s)$ and $K' \in Principals(s)$. Observe that if t is an extension of s and size of s is n then size of t is bounded by n^3; this is seen from the fact that the cardinality of $Roles(s)$ is bounded by n^2 and that of $Principals(s)$ is bounded by n. Also, note the difference between states that are extensions of s and simple states that are reachable from s. For an extension of s, the newly added statements may be defining roles that are in $\mathcal{G}$ (and hence an extension state may not be a reachable state). However, all the roles and principals in these statements should be roles and principals of s. However, in case of simple reachable states these roles are not roles in $\mathcal{G}$, but they need not be roles of s; similarly, the principals in these statements need not be principals of s.

Let s' be any simple reachable state from s. Note that s' may have new principals and roles not in s. The new statements in s' may directly or indirectly introduce new members into the roles in $Roles(s)$, some of which may be from $Principals(s)$. We define $dual(s')$ to be the state $s \cup Q$ where Q is the set of all simple member statements of the form $K.l \leftarrow K'$ such that $K' \in Principals(s)$, $K.l \in Roles(s)$ and $K' \in \mathcal{F}_{s'}(K.l)$. It is to be noted that $dual(s')$ is an extension of s and it may not be reachable from s. Let $Dspace(s)$ be the set of states $\{dual(s') \ : s'$ is a simple reachable from $s\}$. It is to be noted that $s \in Dspace(s)$. The following lemma follows from [23,20].

Lemma 2 *The state s satisfies the formula $AG(X.u \supseteq A.r)$ iff for every state $t \in Dspace(s)$, $\mathcal{F}_t(X.u) \supseteq \mathcal{F}_t(A.r)$.*

Let n be the sum of the size of s and the cardinality of $\mathcal{G}$. In the reminder of this section we show how to construct $Dspace(s)$ in time $2^{O(n^4)}$. Since the size of any extension of s is bounded by n^3,we see that the number of elements in $Dspace(s)$ is bounded by 2^{n^3}. The condition $\mathcal{F}_t(X.u) \supseteq \mathcal{F}_t(A.R)$ can be checked in time bounded by a polynomial in the size of t. Thus, if we can enumerate elements in $Dspace(s)$ in time $2^{O(n^4)}$ then we can check if s satisfies

$AG(X.u \supseteq A.r)$ in time $2^{O(n^4)}$.

Tree Automata Corresponding to States

For any state t, let $alphabet(t)$ be the sequence ($Principals(t)$,$role\_names(t)$, $\{\cap\}$). With the state s and a role $K.l \in Roles(s)$, we associate a tree automaton denoted by $Aut(s, K.l)$ constructed as follows. The automata $Aut(s, K.l)$ and $Aut(s, K'.l')$ for different roles $K.l$ and $K'.l'$ are identical except for the starting state. The alphabet sequence of $Aut(s, K.l)$ is $alphabet(s)$. Note that an input tree node can be labeled with elements from $Principals(s)$, $role\_names(s), \{\cap\}$ iff the arity of the node is $0, 1$ or 2 respectively. Thus, a node labeled with an element of $Principals(s)$ is a leaf node, a node labeled with a role name has a single child and a node labeled with $\cap$ has two children. We describe the states and transitions of the automaton intuitively. This automaton also has ϵ-transitions. The set of states of the automaton is $Roles(s) \cup Principals(s)$. It is to be noted that we are using members of $Principals(s)$ as both states and as input symbols; the usage will be clear from the context. We define an initial set of transitions directly from the policy statements in s. After this, we perform a fix point computation to get the final set of transitions. This fix point computation is similar to the computation used in [20,18] for computing string automata corresponding to policies that do not include intersection inclusion statements. The initial set δ' of transitions are defined as follows. Each transition is a pair, triple or a quadruple (respectively) if the input symbols is a principal, role name or is $\cap$ respectively; ϵ-transitions are also triples. For each $K \in Principals(s)$, $(K, K) \in \delta'$; this means that a single node tree labeled with K is accepted from state K. There are no other transitions from state K. For each role $K.l \in Roles(s)$, there is a transition from the state $K.l$ on input l to the state K, i.e.,$(K.l, l, K) \in \delta'$. For each simple member rule of the form $K.l \leftarrow K'$ in s, $(K.l, K') \in \delta'$; this means that from state $K.l$ the single node tree with label K' is accepted. For each rule of the form $K.l \leftarrow K_1.l_1$, the ϵ-transition $(K.l, \epsilon, K_1.l_1) \in \delta'$; this means every tree accepted from $K_1.l_1$ is also accepted from $K.l$. For each rule of the form $K.l \leftarrow K_1.l_1.l_2$, there is a transition from the state $K.l$ to the state $K_1.l_1$ on input l_2, i.e., $(K.l, l_2, K_1.l_1) \in \delta'$; this means that for every tree T accepted from $K_1.l_1$, the tree T' whose root node is labeled with l_2 and which has T as its sole sub-tree is accepted from $K.l$. For each rule of the form $K.l \leftarrow K_1.l_1 \cap K_2.l_2$, there is a transition from $K.l$ on input $\cap$ to the pair of states $(K_1.l_1, K_2.l_2)$, i.e., $(K.l, \cap, K_1.l_1, K_2.l_2) \in \delta'$; this means that for every pair of trees T_1, T_2 accepted from $K_1.l_1$ and $K_2.l_2$ respectively, the tree T, whose root is labeled with $\cap$ and which has T_1, T_2 as its sub-trees, is accepted from $K.l$.

We say that a string $(a_0, a_1, ..., a_{n-1})$ is accepted from a state q of the automaton if the string represented as a tree having a single path whose nodes are labeled with $a_0, ..., a_{n-1}$ in that order is accepted from q. The set δ of transitions of the automaton $Aut(s, K.l)$ is computed from δ' by iterating the following steps until a fix point is reached. If the string (l_2, K_2) is accepted from a state R, which is a role, then add the ϵ-transition from R to $K_2.l_2$ to the set of transitions. For each role R and for each $(R, \cap, R_1, R_2)$ in the current set of transitions and for each principal K that is accepted by the automaton starting from state R_1 and also from state R_2, add (R, K) to the set of transitions (here R_1, R_2 are roles).

A formal description of the above fix point algorithm as "Algorithm A" is given in the appendix. Let δ be the final set of transitions after termination of the above algorithm. Clearly, the cardinality of δ is $O(n^3)$ where n is the size of the state s.

The set δ can be partitioned in to $(\delta_{-1}, \delta_0, \delta_1, \delta_2)$ where δ_{-1} is the set of ϵ-transitions in δ, and δ_0, δ_1, δ_2 are the sets of transitions on input symbols that are principals, on input symbols that are role names, on input symbol $\cap$, respectively. For any $R \in Roles(s)$, let $Aut(s, R)$ denote the tree automaton $(Q, Principals(s), role\_names(s), \{\cap\}, \delta_{-1}, \delta_0, \delta_1, \delta_2, R)$ where the set of states $Q = Roles(s) \cup Principals(s)$. Note that for any $R, R' \in Roles(s)$, $Aut(s, R)$ and $Aut(s, R')$ are identical except for the starting states.

The automaton $Aut(s, R)$ constructed as above satisfies the following properties. $\mathcal{F}_s(R)$ is exactly the set of $K \in Principals(s)$ such that the single node input tree labeled with K is accepted by $Aut(s, R)$. If there is an ϵ-transition from R to R', i.e., $(R, \epsilon, R') \in \delta$, then $\mathcal{F}_s(R) \supseteq \mathcal{F}_s(R')$. If there is a transition from R to R' on l, i.e., $(R, l, R') \in \delta$, then $\mathcal{F}_s(R)$ contains the set $\mathcal{F}_s(K.l)$ for each $K \in \mathcal{F}_s(R')$. If there is a transition from R to R', R'' on input $\cap$, i.e., $(R, \cap, R', R'') \in \delta$ then $\mathcal{F}_s(R) \supseteq \mathcal{F}_s(R') \cap \mathcal{F}_s(R'')$.

It is to be noted that for each input tree to the automaton $Aut(s, K.l)$, all its leaf nodes are labeled by principals. If the tree is not a single node tree then the parent of every leaf node is labeled with a role name or is a node labeled with $\cap$. If a leaf node is labeled with a principal K_1 and its parent is labeled with a role name l_1, then we define $K_1.l_1$ to be the role associated with the leaf node.

Partial orders on trees and sets of trees

We say that an input tree is proper if it is not a single node tree and all the parents of leaf nodes are role names. Let $S = (T, \eta)$ be a proper tree. A node $x \in T$ is a called a *regular node* of S if $\eta(x) \neq \cap$. A node x is called a *first node* if it is a regular node and none of its proper ancestors is a regular node. We define a relation $\rightarrow_S$ on the set of regular nodes of S as follows: $x \rightarrow_S y$ if y is a proper descendant of x and all the intermediate nodes on the unique path from x to y are non-regular nodes. We define a partial order $\preceq$ on the set of proper trees as follows. For two proper trees $S = (T, \eta)$, $S' = (T', \eta')$,

$S \preceq S'$ if there exists a function h from the regular nodes of S' to the regular nodes nodes of S such that for every regular node x of S', $\eta'(x) = \eta(h(x))$ and the following conditions are satisfied.

- If x is a first node of S' then $h(x)$ is a first node of S.
- For every y such that $x \rightarrow_{S'} y$, it is the case that $h(x) \rightarrow_S h(y)$.

It is to be noted that any function h satisfying the above conditions maps the leaf nodes of S' to the leaf nodes of S. Intuitively, $S \preceq S'$ means that S' is contained in S. For two sets X, Y of proper input trees, we define a partial order $\sqsubseteq$ as follows: $X \sqsubseteq Y$ iff for every $S \in X$ there exists a $S' \in Y$ such that $S \preceq S'$.

Composition of trees and sets of trees

We say that a proper input tree is growth unrestricted if every role associated with its leaf nodes is growth unrestricted, i.e., it is not in $\mathcal{G}$. For the state s and a role $K.l \in \mathit{Roles}(s)$, let $\mathit{GU\_trees}(s, K.l)$ denote the set of growth unrestricted input trees that are accepted by $\mathit{Aut}(s, K.l)$.

Now, we define some operations on trees and sets of trees. Let $S_1 = (T_1, \rho_1)$ and $S_2 = (T_2, \rho_2)$ be labeled binary trees. We define $\mathit{comp}(S_1, S_2)$ to be the tree whose root is labeled with $\cap$ and having S_1, S_2 as left and right subtrees respectively. Formally $S = (T, \rho)$ where $T = \{NULL\} \cup \{0x : x \in T_1\} \cup \{1x : x \in T_2\}$ and ρ is defined as follows: $\rho(NULL) = \cap$; for every $x \in T_1$, $\rho(0x) = \rho_1(x)$; for every $x \in T_2$, $\rho(1x) = \rho_2(x)$.

Let X_1, X_2 be sets of labeled binary trees. Now, we define an operation $\circ$ on such sets as follows. $X_1 \circ X_2 = \{\mathit{comp}(S_1, S_2) : S_1 \in X_1, S_2 \in X_2\}$. For a sequence $X_1, ..., X_k$ of sets of labeled binary trees, we let $X_1 \circ X_2 ... X_k$ denote the set denoted by $(...(X_1 \circ X_2) \circ X_3)...) \circ X_k)$; that is the operation $\circ$ is carried out by association to the left. For a sequence $X = (X_1, ..., X_k)$ of sets of labeled binary trees, we let $\mathit{product}(X)$ denote the set of trees denoted by $X_1 \circ ... \circ X_k$.

Construction of the set $\mathit{Dspace}(s)$

For any state t, let $\mathit{closure}(t)$ be the state $t \cup \{R \leftarrow K : R \in \mathit{Roles}(t), K \in \mathcal{F}_t(R)\}$. Roughly speaking, t and $\mathit{closure}(t)$ are equivalent. We say that a role $K.l$ in a state t is unbounded if there exists at least one growth unrestricted tree that is accepted by the automaton $\mathit{Aut}(t, K.l)$. Let $\mathit{GU\_roles}(t)$ be the set of roles in t that are unbounded.

A role-principal sequence (in short rp-sequence) σ is a sequence of pairs of the form (R_1, L_1), (R_2, L_2), ..., (R_m, L_m) such that for each i, $1 \leq i \leq m$, $R_i \in \mathit{Roles}(s)$ and $L_i \in \mathit{Principals}(s)$ and such that no pair repeats. Note that the length of such a sequence is bounded by n^3 where n is the size of

s. For an rp-sequence σ as given above, let $u_1, ..., u_m, u_{m+1}$ be a sequence of states such that $u_1 = s$ and for each i, $1 \leq i \leq m$, $u_{i+1} = u_i \cup \{R_i \leftarrow L_i\}$. For each i such that $1 \leq i \leq m$, let $X_i = GU\_trees(u_i, R_i)$. For any $K \in Principals(s)$, let $Roles(\sigma, K)$ be the set $\{R_i : L_i = K\}$ and $guseq(\sigma, K)$ be the sequence $(X_{i_1}, .., X_{i_l})$ where $i_1 < ... < i_l$ are all the indices j such that $L_{i_j} = K$. For a principal K, let $Product(\sigma, K)$ denote the set of growth unrestricted trees given by $Product(guseq(\sigma, K))$ and let $RGU\_trees(\sigma, K)$ denote the set which is the union of sets in $\{GU\_trees(u_{m+1}, R') : R' \in Roles(s)$ and $K \notin \mathcal{F}_{u_{m+1}}(R')\}$. We say that σ is an *acceptable* rp-sequence for s if for each principal K such that $Roles(\sigma, K) \neq \emptyset$,
(*) it is not the case that $Product(\sigma, K) \sqsubseteq RGU\_trees(\sigma, K)$.

If σ is an acceptable rp-sequence for s then, for each i, $1 \leq i \leq m$, it is the case that $R_i \in GU\_roles(u_i)$; if this were not the case then X_i will be the empty set and $Product(\sigma, L_i)$ will be the empty set and it is going to be the case that $Product(\sigma, K) \sqsubseteq RGU\_trees(\sigma, K)$ is vacuously satisfied. For an acceptable rp-sequence σ as given above, we let $final(\sigma)$ be the state $closure(u_{m+1})$. The following lemma states that for every acceptable rp-sequence σ, $final(\sigma) \in Dspace(s)$. It also shows that acceptability of a rp-sequence can be checked in exponential time. The algorithm to check the acceptability of a rp-sequence acts as follows: for each $K \in Principals(s)$, it constructs two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, which are of size polynomial in the size s, such that $L(\mathcal{A}_1) = Product(\sigma, K)$ and $L(\mathcal{A}_2) = RGU\_trees(\sigma, K)$ and checks that $L(\mathcal{A}_1) \sqsubseteq L(\mathcal{A}_2)$. The later condition ,as we show, can be checked in time exponential in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$. The details of this algorithm together with proof of the following lemma are given in the appendix.

Lemma 3 *If σ is an acceptable rp-sequence then $final(\sigma) \in Dspace(s)$. Further more, there exists an algorithm that takes as input a state s, a rp-sequence σ and decides if σ is an acceptable rp-sequence for s and the algorithm runs in time $2^{O(n^4)} + O(m)$ where n is the size of s and m is the length of σ.*

Lemma 4 *The set of states enumerated by algorithm B is exactly the set $Dspace(s)$.*

Algorithm B: Now we give an algorithm that computes the set $Dspace(s)$. Let n be the size of s. The algorithm enumerates all rp-sequences of length at most n^3. For each sequence it checks if it is acceptable and for each such sequence σ, it includes $final(\sigma)$ into $Dspace(s)$. By a simple combinatorial analysis, it is easy to see that the number of of rp-sequences of length at most n^3 is bounded by $2^{O(n^4)}$; this is because the number role-principal pairs is bounded by n^3. Using lemma 3, we see that the total time to check for acceptability of all such sequences is $2^{O(n^4)} \cdot 2^{O(n^4)}$ which is $2^{O(n^4)}$. Thus the elements of $Dspace(s)$ can be enumerated in time $2^{O(n^4)}$ and hence we can check if the state s satisfies $AG(X.u \supseteq A.r)$ in $Struct(s, \mathcal{G}, Roles(s))$ in time

$2^{O(n^4)}$.

Example: We illustrate the above algoritm by the following simple example. Let s be the policy given by the following set of statements.

$K_1.l_1 \longleftarrow K_2.l_2.l'_2$

$K_2.l_2 \longleftarrow K_3.l_3.l'_3$

$K_3.l_3 \longleftarrow K_4.l_4 \cap K_5.l_5$

$K_4.l_4 \longleftarrow U$

$K_6.l_6 \longleftarrow U.l'_3.l'_2$

$Principals(s)$ is the set $\{K_1, K_2, K_3, K_4, K_5, K_6, U\}$. Let $\mathcal{G}$ be all the roles excepting $U.l'_3$ and $K_5.l_5$ and $\mathcal{S}$ be all the roles. Using the above algorithm, we can check that this dynamic policy satisfies $AG(K_6.l_6 \supseteq K_1.l_1)$ at state s. Below, we give some acceptable and some unacceptable rp-sequences.

Consider the rp-sequence consisting of $(K_6.l_6, K_3)$. Here the length m of this sequence is one. Using the terminology given above, we have $u_1 = s$, $u_2 = s \cup \{K_6.l_6 \leftarrow K_3\}$. It should be easy to see that $GU\_trees(v, R)$ is the empty set for every role R other than $U.l'_3$, $K_6.l_6$ and $K_5.l_5$, and for $v = u_1$ and $v = u_2$. For $v = u_1, u_2$, $GU\_trees(v, U.l'_3)$, $GU\_trees(v, K_6.l_6)$ and $GU\_trees(v, K_5.l_5)$ consist of the single trees given by the sequences (l'_3, U), (l'_2, l'_3, U) and (l_5, K_5), respectively. From this it should be easy to see that the above sequence is an acceptable sequence.

Now consider the sequence σ given by $(K_6.l_6, K_3)$, $(K_5.l_5, U)$. Here $m = 2$, u_1, u_2 are same as above and $u_3 = u_2 \cup \{K_5.l_5 \leftarrow U\}$. It should be easy to see that $Product(\sigma, K_3)$ consists of the single sequence (l'_2, l'_3, U). It can be verified that $GU\_trees(u_3, K_1.l_1)$ contains the tree given by the sequence (l'_2, l'_3, U), but $K_3 \notin \mathcal{F}_{u_3}(K_1.l_1)$. From this, it should be easy to see that the acceptability condition (*) is not satisfied for the above sequence for the case $K = K_3$. The above sequence when extended by adding $(K_1.l_1, K_3)$ becomes acceptable.

It is to be noted that in the above example all relevant growth unrestricted trees are sequences. However, it is not difficult to modify the example so that some of the relevant growth unrestricted trees are not sequences.

3.3 Upper Bounds for formulas of the form $AG(f)$

In this section we consider the problem of checking if a role restricted system given by an initial state $\mathcal{P}$ and a restriction pair $(\mathcal{G}, \mathcal{S})$ satisfies a FCTL formula of form $AG(f)$ where f has no temporal operators. Essentially we show that this problem is in co-NEXPTIME, thus generalizing the same result given for role containment in [23,20]. This result is stated as the following theorem. Its proof is based on the properties of first order monadic logics and is given in the appendix.

Theorem 2 *Given an initial state $\mathcal{P}$, a restriction pair $(\mathcal{G}, \mathcal{S})$, and a FCTL formula of the form $AG(f)$ where f has no temporal operators and has no free variables , then the problem of determining the role restricted dynamic system given by $\mathcal{P}$ and $(\mathcal{G}, \mathcal{S})$ satisfies $AG(f)$ is in the complexity class co-NEXPTIME. (NEXPTIME is the class of languages that can be decided in non-deterministic exponential time).* □

3.4 Checking Monotonic Formulas

Consider any FCTL formula f that has no temporal operators. Every such formula can be converted into an equivalent formula in which negations apply only to atomic formulas and all double negations are eliminated. This conversion can be done by using D'Morgan's rules and by using the equivalences $\neg\forall x(g) \equiv \exists x(\neg g)$ and $\neg\neg(g) \equiv g$. We call such formulas as normalized formulas. An occurrence of an atomic formula h in a normalized formula is said to be negative if the occurrence is $\neg h$, otherwise it is said to be positive. An atomic formula is said to be a membership atomic formula if it is of the form $x \in F$ or $K' \in K.l$ where x is a first order variable, $K.l$ is a role and K' is a principal. A *positive formula* is a normalized formula in which all the quantifiers are existential and all occurrences of membership atomic formulas are positive. A *negative formula* is a normalized formula in which all quantifiers are universal and all occurrences of membership atomic formulas are negative. Note that a negative formula is the negation of a positive formula.

Now we consider checking FCTL formulas of the form $AG(f)$ and $EF(f)$ where f is a positive formula in a dynamic system with initial state s and restriction rule $(\mathcal{G}, \mathcal{S})$. Notice that if we can check formulas of the above type efficiently then we can also check the cases when f is a negative formula by using the identities $AG(f) = \neg EF(\neg f)$ and $EF(f) = \neg AG(\neg f)$. (For example, to check $AG(f)$ for the case when f is negative, we simply check if the formula $EF(\neg f)$ at state s and answer 'yes' iff the answer to the later check is 'no'; it is to be noted that $\neg f$ after normalization is a positive formula and it falls under the previous case.)

Let the role restricted system be given by the initial state s and the restriction pair $(\mathcal{G}, \mathcal{S})$. The minimal state reachable from s is the state obtained from s by deleting all statements that define roles not in $\mathcal{S}$; note that statements defining such roles can be deleted. It has been show on [23,20] that for certain properties such as $AG(f)$ where f specifies boundedness can be checked by checking f in the minimal state. (In [23,20], a minimal state is given by a lower bound program). We generalize this for the case when f is any closed positive formula in the following lemma.

Lemma 5 *For a dynamic system with initial state s and restriction pair $(\mathcal{G}, \mathcal{S})$ and for a positive closed formula f, the formula $AG(f)$ is satisfied at s iff f is satisfied in the minimal state.* $\square$

Examples: Among the properties specified in sub-section 2.3, the simple availability property is of the form $AG(f)$ where f is positive formula and liveness is of the form $EF(g)$ where g is a negative formula. Hence both these formulas can be checked by considering the minimal state.

Our generalization allows one to analyze more properties efficiently than [23,20]. For example, consider the property that in every reachable state there are at least two principals in the role $K.l$. Such a property is expressed by $AG(f)$ where $f = \exists\, i, \exists\, j(\neg(i = j) \wedge i \in K.l \wedge j \in K.l)$. Note that f is a positive formula since all membership conditions appear positively in f. It is not clear how such a property can be directly checked using the method given in [23,20] since their availability analysis allows us only to check if a given group of principals are present in a role in all reachable states.

Now we consider the problem of checking $EF(f)$ for the case when f is positive. We assume that all the roles appearing in f also appear in s. Let s and $(\mathcal{G}, \mathcal{S})$ be as given above. As in [23,20], we define a maximal state. For this, we introduce a set of new principals. Let m be an integer defined as follows. If the symbol $=$ appears in f then m is the sum of the number of variables and the number of principals appearing in f, otherwise $m = 1$. The new set of principals is $\Gamma = \{T_1, ..., T_m\}$. We define a maximal state s' by adding the following statements to s. For each role $K.l \in (Roles(s) - \mathcal{G})$ and principal $K' \in \Gamma \cup Principals(s)$, we add the statement $K.l \leftarrow K'$. Also for each role name r appearing in s and for each principal $T' \in \Gamma$ and each principal $K' \in \Gamma \cup Principals(s)$, we add the statement $T'.r \leftarrow K'$. We call s' as the maximal state and it satisfies the properties given by the following lemma.

Lemma 6 *For a dynamic system with initial state s and restriction pair $(\mathcal{G}, \mathcal{S})$ and for a positive closed formula f, the formula $EF(f)$ is satisfied at s iff f is satisfied in the maximal state s' as given above.* $\square$

Examples: Among the properties specified in sub-section 2.3, the simple safety property is of the form $EF(f)$ where f is a positive formula. The properties specified by bounded safety, principal conflict 1 and mutual exclusion are of the form $AG(f)$ where f is a negative formula. All these properties can be checked by considering the maximal state.

Thus the problem of checking formulas of the form $AG(f)$ and $EF(f)$, where f is a positive formula, is reduced to that of checking f in a single state. In general the problem of checking if a positive formula is satisfied in a state can be shown to be NP-complete as follows. We can easily give an algorithm in the class NP to check this satisfiability. First we move all the existential quantifiers to the beginning of the formula; this might require renaming the variables. Corresponding to each first order variable we guess a principal mentioned in the state s and substitute this principal for that variable and remove all the existential quantifiers. We simply check if the resulting formula is satisfied in the state s. This can be done in polynomial time. This decision procedure is in NP. The lower bound is proved by reducing the satisfiability problem for propositional formulas as follows. Let g be a propositional formula in Conjunctive Normal Form. For each propositional variable p appearing in g, we introduce two first order variables i_p and j_p. We replace every occurrence of p in g by $i_p = j_p$. We close the resulting formula by adding existential quantifiers on all first order variables. Let f be the resulting formula. Now let s be any state with a single role and having n principals in it where n is the number of first order variables in f. It should be easy to see that the propositional formula g is satisfiable iff f is satisfied in the state s.

We can adopt a deterministic, method given in [13], for checking first order formulas in a single state of a SPKI system, to the RT_0 TM system. The complexity of the resulting algorithm will be polynomial in the size of the state but exponential in the length of the formula.

4 Dynamic Policy Analysis in Finite State Systems

Model Checking based approach for Dynamic Policy Analysis

A finite state system is one where the number of reachable states is bounded. Such systems can be modeled by structures as given in section 2. However, they may be implicitly specified using transition systems [24]. We assume that we have a set V of system variables. We assume that the number of system variables is finite and each of them can only have a bounded number of values. Thus the number of system states is finite. A transition system is given by a tuple $(V, \Lambda, \Theta, \mathcal{P})$ where V is a set of system variables, Λ is a finite set of transitions, Θ is predicate specifying initial system state and $\mathcal{P}$ is the initial policy. We assume that Θ is satisfied by exactly one system state, i.e., it essentially specifies values of the system variables. A state s is a pair $(u, \mathcal{Q})$ where u is a system state and $\mathcal{Q}$ is policy. Each element of Λ is a transition which is a pair of the form (G, A) where G is called the guard and A is called the action. The guard is a predicate which is of the form $G_1 \wedge G_2$ where G_1 is a condition involving system variables and G_2 is a FCTL formula without temporal operators. Here G_2 specifies a condition on the current policy. The action part is a set of atomic actions where each atomic action is an assignment for a system variable or is an operation that adds or deletes a policy statement.

Given such a transition system, we can operationally specify its semantics as follows. We say that a guard $G = G_1 \wedge G_2$ is satisfied in a state $(u, \mathcal{Q})$ if G_1 is satisfied in the system state u and G_2 is satisfied by the policy $\mathcal{Q}$. We say that a transition τ is enabled in a state if its guard is satisfied in that state. An initial state is a state of the form $(u, \mathcal{P})$ where u satisfies the initial condition. The system initially starts in the initial state. It repeatedly executes the following. It non-deterministically chooses a transition that is enabled and executes the action part of the transition atomically. This execution of the action part may change the state. This process is repeated for ever. It is to be noted that the system state captures the external world. In many transitions only the system state may change while no changes are made to the policy, i.e., no additions or deletions of policy statements. It is not difficult to see that the behavior of such a dynamic system can be modeled by a structure as give in section 2. Since the number of systems states is finite, the resulting structure is finite. We call this structure as the reachability graph.

Such a system satisfies an FCTL formula iff the initial state in the above structure satisfies the formula. Using the model checking techniques of [10], we can give an algorithm that constructs the reachability graph and check a formula in the graph efficiently. In this procedure, we need to evaluate atomic formulas involving system variables and the policy in each state. This can be done using algorithms for static policy analysis. If the formula that we check have no first order quantifiers and similarly all the guards conditions referring to policy do not have quantifiers, then this model checking algorithms runs in time linear in the number of reachable states and linear in the length of the formula and polynomial in the number of principals and the sum of the lengths of all the policy statements.

We can handle certain types of formulas more efficiently. As given in subsection 3.4, we can check formula of the form $AG(f)$ where f is a positive formulas and does not refer to system variables by evaluating f only in the minimal states in the reachability graph. A state $s = (u, \mathcal{Q})$ is minimal if there is no other reachable state of the form $(u', \mathcal{Q}')$ such that $\mathcal{Q}' \subset \mathcal{Q}$. There can be multiple minimal states. We need to evaluate f in all such minimal states. Similarly formulas of the form $EF(f)$ can be checked by evaluating f only in maximal states.

Timed systems

Many times policies change with time. For example in SPKI Trust Management systems certificates are issued for certain time durations specified by upper and lower bounds. One can imagine situations where certificates are issued periodically to be valid during certain times in the day, say from 7AM to 5PM every day. One way of specifying such polices is assuming a system clock as a system variable which is implicitly updated or is explicitly updated using a tick transition in the system. Then we can have transitions that add and delete policy statements based on the clock values. For example, we can have a transition that adds a policy statement when $clock.hour = 7AM$ and another transition that deletes it when $clock.hour = 5PM$. This defines a policy statement that is in the policy from $7AM$ to $5PM$ every day. Similarly, we can specify policy statements that are issued only once during a particular time period. Here we may need to refer to the time as well as date. For example, suppose there is a server in CS department which can be used by both faculties and students. But since some maintenance work needs to be done, only faculties are allowed to log on this server from $11PM$ to $12PM$ every Monday. In the formulas we want to verify we can use the system clock in conditions. So one can query if principal K can access resource R during a particular time period. Adding, such clocks may seem that the over all system is not a finite state system. However, as long as we use the clocks in comparisons of the form $Clock = c$ or $Clock \geq c$ or $Clock \mod k = c$ where c, k are constants checking FCTL formulas is decidable.

Another formalism that can be used for handling time is the timed automaton model [2]. Such automata have clock variables, states and transitions between states. This model assumes an external clock that runs continuously and each of the clock variables is updated at the same rate as the external clock. The transitions from the state take place on some conditions over the clock variables. We can use timed automata for modeling dynamic policies by associating a policy with each state of the automaton. Taking a transition implies changing the policy based on timing conditions. We can use FCTL to specify properties of such dynamic systems and analyze them using the techniques of [2].

5 Related Work

The work that is closely related to ours is that given [23,20]. There the authors introduce dynamic security analysis and present various results for role restricted systems described by RT_0 and its sub-languages. They consider specific properties such as boundedness, role containment,etc. We present a general frame work based on structures for modeling such dynamic systems. We also present a general purpose language for specifying properties of such systems. We can specify all properties considered in that work, but more. We also show that the problem of checking role containment is EXPTIME-hard and is solvable in exponential time. Further we generalize a number of results given in that work.

The work presented in [25] also considers security analysis in dynamic access control systems. There the authors consider simple role based systems. In their model, each system consists of a set of roles, a set of resources and a set of principals. The system state is given by a *membership* relationship between roles and users, and an *access* relationship between resources and roles. A principal can access a resource if he/she belongs to a role that has access to that resource. Changes to system states are allowed through transitions. Each transition specifies addition of a Principal or deletion of a principal to a role, if the principal currently belongs to certain roles and does not belong to certain other roles. In such a model the authors show that the problem of checking simple safety is PSPACE-complete. They give polynomial time bounded algorithms for the case when certain restrictions are placed on the system. Our model has richer structure for roles and role memberships are defined through different types of rules. In our case, changes in policy are permitted as long as the role restrictions are obeyed.

Languages based on First Order Logic have been considered for policy analysis in [15]. In this work the authors use the logic to specify policies and to check their consistency. On the contrary we only use the logic for specifying properties. Further we use branching time temporal operators, while they do not use temporal operators. We consider dynamic policy analysis, while they consider static policy analysis. There has been other work such as [13] where the authors use a first order temporal logic for analyzing policies specified by SPKI certificates. The logic used there is a linear temporal logic which cannot specify properties requiring existential quantifiers on future, while here we use branching time logic which allows specification of such properties. Since FCTL is state oriented, we believe it is more appropriate to analyze dynamic policies than linear time logic which is computation oriented. We also consider general dynamic systems specified in RT_0 frame work and give complexity results and decision algorithms.

There has been other work on static policy analysis for SPKI Trust Management systems and other systems. For example, Jha and Reps [18] propose using temporal logic to reason about certificate chain derivations in a given SPKI system. In that work the authors primarily consider static policy analysis but not dynamic policy analysis as we consider here.

Flexible access control policies based on logic programming are proposed in [17]. This formalism supports positive as well as negative authorizations allows conflict resolution etc. However, it can be used primarily for defining static policies. A mechanism for specifying time dependent access control policies has been presented in [3]. This mechanism uses deductive temporal rules with periodicity and order constraints. Compared to these methods, our scheme for defining dynamic policies allows explicit change in policy by executing different rules. Each execution takes the system to a new state which is automatically consistent as we do not allow negative authorizations.

There are other logics for Trust Management Systems such as SPKI/SDSI which model the name resolution, authorization and other features of SPKI/SDSI: Martin Abadi's logic to model local name spaces of SDSI [1], Halpern and van der Meyden's Logic of Local Name Containment to characterize SDSI name resolution which was extended to deal with other SPKI issues like revocation, expiry dates, and tuple reduction [16]. Ninghui Li [19] proposes a logic program to describe the name resolution algorithm which also handles authorization certificates and threshold subjects. The purpose of our logic is only to specify properties that one wants to check in a dynamic system. There has also been other work in Trust Management Systems such as [5], [4]. In these works dynamic policy analysis is not considered.

6 Conclusion

In this paper, we have introduced a general frame work for modeling dynamic policy. We also introduced a general purpose branching time temporal logic FCTL for specifying properties of such systems. We presented a number of results for deciding if a formula holds for a system. We showed that the role containment problem is EXPTIME-hard. We also showed an exponential upper bound for this. We also generalized the results of [23,20] for the case when the properties are specified by any first order formula. For example, we showed that checking of properties such as $AG(f)$ and $EF(f)$ where f is a positive or negative first order formula can be done efficiently. We also generalized the result of [23,20] showing that checking formula of the form $AG(f)$ where f is any first order formula can be done is in the complexity class co-NEXPTIME.

We have also illustrated how our methods can be used for finite state systems. More work needs to be done here. Also a better upper bound for checking formulas of the form $AG(f)$ where f is an arbitrary first order formula needs to be investigated.

Acknowledgements: We are thankful to Dr. Venkatakrishnan for his insightful comments on the paper.

References

[1] M. Abadi. On SDSI's linked local name spaces. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 122(2):183–235, 1994.

[3] E. Bertino, C. Bettinit, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Transactions on Database Systems*, 23(3):213–285, 1998.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The role of trust management in distributed systems security. pages 185–210, 1999.

[5] M. Blaze, J. Feigenbaum, and K. Keromytis. KeyNote: Trust management for public-key infrastructures. In B. Christianson, B. Crispo, W. S. Harbison, and M. Roe, editors, *Security Protocols—6th International Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–66, Cambridge, United Kingdom, 1999. Springer-Verlag, Berlin Germany.

[6] D. F. C. Brower and M. J. Nash. The chinese wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206–214, Oakland, California, May 1989.

[7] D. R. Clark, , and D. R. Wilson. A comparison of commercial and millitary security policies. In *IEEE Symposium on Security and Privacy*, Oakland, California, April 1987.

[8] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4), 2001.

[9] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of Workshop on Logics of Programs—Springer-Verlag LNCS*, volume 131, pages 52–71, Yorktown Heights, NY, May 1981. Springer-Verlag.

[10] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions On Programming Languages And Systems*, 8(2):224–263, 1986.

[11] A. K. Chandra, D. C. Kozen, L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[12] C.M.Ellision, B.Frantz, B.Lampson, R.L.Rivest, B.M.Thomas, and T.Ylonen. Spki certificate theory. In *RFC 2693*, Sept. 1999.

[13] A. K. Eamani and A. P. Sistla. Language based policy analysis in trust management systems. In *Proceedings of the Fourth PKI Conference*, April 2005.

[14] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. *RFC 2693: SPKI Certificate Theory*. The Internet Society, Sept. 1999. See `ftp://ftp.isi.edu/in-notes/rfc2693.txt`.

[15] J. Halpern and V. Weizman. Using first order logic to reason about policies. In *Proceedings of The 16th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2003.

[16] J. Y. Halpern and R. van der Meyden. A logic for sdsi's linked local name spaces. *Journal of Computer Security*, 9:105–142, 2001.

[17] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subramanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):214–247, 2001.

[18] S. Jha and T. Reps. Analysis of spki/sdsi certificates using model checking. In *15th IEEE Computer Security Foundations Workshop*, pages 129–144, Cape Breton, Canada, 24–26 June 2001. IEEE Computer Society Press.

[19] N. Li. Local names in SPKI/SDSI. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 2–15. IEEE Computer Society Press, jul 2000.

[20] N. Li, J. Mitchell, and W. H. Winsborough. Beyond Proof-of-Compliance: Safety and availability analysis in trust management. *Journal of the ACM*, 52(3), May 2005.

[21] N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages*, January 2003.

[22] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 114. IEEE Computer Society, 2002.

[23] N. Li, W. H. Winsborough, and J. C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 123, Washington, DC, USA, 2003. IEEE Computer Society.

[24] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. Springer-Verlag, 1991.

[25] A. Sasterkar, P. Yang, S. D. Stroller, and C. R. Ramakrishnan. Policy analysis for administrative role based access control. In *CSFW 2006: Proceedings of the Computer Security Foundation Workshop*, Venice, Italy, 2006.

[26] H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. COMPUT.*, 19(3):424–137, 1990.

7 Appendix

In this section we give some of the formal definitions proofs of some of the lemmas and theorems.

7.1 Appendix: Preliminaries

The following is an example of a policy specified in BRT_0. This example is adapted from [23,20].

$$SA.access \longleftarrow SA.manager \quad (1)$$

$$SA.access \longleftarrow SA.delegatedAccess \cap HR.employee \quad (2)$$

$$SA.manager \longleftarrow HR.manager \quad (3)$$

$$SA.delegatedAccess \longleftarrow SA.manager.access \quad (4)$$

$$HR.employee \longleftarrow HR.manager \quad (5)$$

$$HR.employee \longleftarrow HR.programmer \quad (6)$$

$$HR.manager \longleftarrow Alice \quad (7)$$

$$HR.programmer \longleftarrow Bob \quad (8)$$

$$Alice.access \longleftarrow Bob \quad (9)$$

The following are many other important properties that are considered in the literature. Their specification in FCTL is given next to each one of them.

- **Liveness:** There exists a reachable state in which no principal has access to resource R.
  $EF\, \forall i\, \neg(i \in A.r)$
- **Principal Conflict 1**: Two given principals K_1, K_2 should not be members of role $X.u$ at the same time.
  $AG\,(K_1 \in X.u \;\rightarrow\; \neg(K_2 \in X.u))$
- **Principal Conflict 2**: Once principal K_1 is assigned to role $X.u$, there is no possibility that K_2 will be assigned to role $X.u$. This is a stronger requirement than the earlier property.
  $AG\,(K_1 \in X.u \;\rightarrow\; AG(\neg(K_2 \in X.u)))$.
- **Mutual Exclusion:** In every reachable state, no principal belongs to two given roles $X.u, A.r$.
  $AG\,\forall i(\neg(i \in X.u) \vee \neg(i \in A.r))$.
- **Repeated Access**: Once a principal is granted the right for accessing resource R, he/she will be granted this right eventually again (i.e., he/she will

be repeatedly granted access).
$AG\forall i(i \in A.r \rightarrow AXAF\ i \in A.r)$

7.2 Appendix: Lower bound for checking role containment

First we give some definitions about tree automata. Let $\mathcal{A} = (Q, \Sigma_0, ..., \Sigma_m, \delta_{-1}, \delta_0, ..., \delta_m, q_0)$ be an m-way tree automaton. The elements of δ_{-1} are ϵ-transitions. For any state q, we define $\epsilon\_closure(q)$ to be the set of all states that can be reached from q using zero or more ϵ-transitions. Formally, $\epsilon\_closure(q)$ is the smallest set X such that $q \in X$ and for every $q_1, q_2 \in Q$, if $q_1 \in X$ and $(q_1, \epsilon, q_2) \in \delta_{-1}$ then $q_2 \in X$. For each i such that $0 < i \leq m$, we also define the closure δ_i^* of δ_i as follows: $\delta_i^* = \{(q, a, q_1, ..., q_i) : \exists (q, a, r_1, ..., r_i) \in \delta_i$ such that $q_j \in \epsilon\_closure(r_j)$ for each j, $1 \leq j \leq i\}$.

Let $\mathcal{A}$ be a m-way tree automaton as defined above and $\Sigma = \cup_{0 \leq i \leq m} \Sigma_i$. An input tree to $\mathcal{A}$ is a labeled tree (T, ρ) over Σ such that T is a m-ary tree and $\rho : T \rightarrow \Sigma$ such that for every node $\sigma \in T$, $\rho(\sigma) \in \Sigma_{arity(\sigma)}$. Note that the label of any node of an input tree denotes its arity also. Now we define runs of $\mathcal{A}$ on an input tree. Intuitively, $\mathcal{A}$ starts at the root node $NULL$ in the initial state q_0. Suppose $arity(NULL) = k$ and $(q_0, \rho(NULL), q'_0, ..., q'_{k-1}) \in \delta_k$. $\mathcal{A}$ splits into $arity(NULL)$ number of copies and the i^{th} copy moves onto the i^{th} child of $NULL$ and goes into the state q'_i. This process is repeated until $\mathcal{A}$ reaches the leaf nodes. Also each copy can take any ϵ-transition without making any move. With this intuition in mind, we define a run of $\mathcal{A}$ on an input tree (T, ρ) to be a labeled tree (T, η) over Q such that $\eta(NULL) \in \epsilon\_closure(q_0)$ and for every non-leaf node σ the following condition is satisfied: if $arity(\sigma) = k$ then $(\eta(\sigma), \rho(\sigma), \eta(\sigma 0), \eta(\sigma 1), ..., \eta(\sigma(k-1)))$ is in δ_k^*; note that $\sigma 0, \sigma 1, ..., \sigma(k-1)$ are the children of σ. We say that the run (T, η) is an *accepting* run, if for every leaf node σ of T, $(\eta(\sigma), \rho(\sigma)) \in \delta_0$. An input tree (T, ρ) is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on (T, ρ). We let $L(\mathcal{A})$, called the language of $\mathcal{A}$, to be the set of input trees accepted by $\mathcal{A}$.

For the tree automaton $\mathcal{A}$, we define size of $\mathcal{A}$ to be the sum of cardinalities of $Q, \Sigma, \delta_{-1}, \delta_0, ..., \delta_m$. Now we give the proof of lemma 1 which is restated here.

Lemma 1: The language containment problem for tree automata is polynomial time reducible to the problem of checking if a role restricted system satisfies a formula of the form $AG(f)$ where f asserts role containment. More formally, there exists a polynomial time algorithm that takes two tree automata $\mathcal{A}, \mathcal{B}$ and outputs a policy $\mathcal{P}$, a restriction pair $(\mathcal{G}, \mathcal{S})$ and two roles $A.r, X.u$ such that the formula $AG(X.u \supseteq A.r)$ is satisfied for the role restricted system iff $L(\mathcal{A}) \supseteq L(\mathcal{B})$.

Proof: It is enough if we give the reduction for the case when the two automata are binary tree automata, i.e., $m = 2$. Let $\mathcal{A} = (Q, \Sigma_0, \Sigma_1, \Sigma_2, \delta_{-1}, , \delta_0, \delta_1, \delta_2, q_0)$ and $\mathcal{B} = (Q', \Sigma_0, \Sigma_1, \Sigma_2, \delta'_{-1}, \delta'_0, \delta'_1, \delta'_2, q'_0)$ be two binary tree automata over the same alphabet sequence $(\Sigma_0, \Sigma_1, \Sigma_2)$. Without loss of generality, we assume that both the automata do not have ϵ-transitions, i.e., $\delta_{-1} = \delta'_{-1} = \emptyset$. Let $\Sigma = \cup_{0 \le i \le 2} \Sigma_i$. Without loss of generality, we assume that $Q \cap Q' = \emptyset$. Let $\delta = \cup_{0 \le i \le 2} \delta_i$ and $\delta' = \cup_{0 \le i \le 2} \delta'_i$.

For any policy $\mathcal{P}$, with each of its roles, we can associate a set of finite trees whose internal nodes are labeled with role names or the symbol $\cap$ and whose leaves are labeled with roles. For example, consider the policy consisting of the two statements $K_0.l_0 \leftarrow K_1.l_1.l_2$ and $K_0.l_0 \leftarrow K_1.l \cap K_2.l_2$. There are three trees associated with the role $K_0.l_0$. These consist of the tree with a single node labeled with $K_0.l_0$, a tree whose root is labeled with l_2 and whose only child is a leaf node labeled with $K_1.l_1$, a third tree whose root is labeled with $\cap$ and which has two children labeled with K_1, l_1, $K_2.l_2$ both being leaf nodes. In general, these sets of trees can be defined inductively using the statements of the policy. Members of these sets whose leaves are labeled with roles which are not growth restricted, are called growth unrestricted trees. Growth unrestricted trees can be used to generate additional principals in a role by adding appropriate policy statements. Corresponding to the two automata, $\mathcal{A}$ and $\mathcal{B}$, we define a policy $\mathcal{P}$ and a restriction pair $(\mathcal{G}, \mathcal{S})$ as follows. The principals of $\mathcal{P}$ correspond to the states and transitions of the two automata. Its role names include those that correspond with the input alphabet of the automata together with five additional role names l, u_0, u_1, v_0, v_1. We define $\mathcal{P}$ so that, for any $q \in Q$ and $q' \in Q'$, the sets of growth unrestricted trees associated with the roles $K_q.l$ and $K_{q'}.l$ correspond to the sets of trees accepted by the automata $\mathcal{A}$ and $\mathcal{B}$ starting in states q, q' respectively. The policy is defined so that in every reachable state the set of principals in the role $K_{q_0}.l$ is a super-set of the set of principals $K_{q'_0}.l$ iff $L(\mathcal{A}) \supseteq L(\mathcal{B})$.

The formal definition of the policy $\mathcal{P}$ is as follows. For each $q \in Q \cup Q'$, we have a principal K_q. Also for each tuple $\tau \in \delta \cup \delta'$, we have a principal L_τ. Finally, we have a special principal L_*. Also for each $a \in \Sigma$, we have a role name a. When ever we refer to an element $a \in \Sigma$, it will be clear from the context whether that reference is to an input alphabet or to a role name. We have five additional role names l, u_0, u_1, v_0, v_1.

The statements of the policy $\mathcal{P}$ consist of the following.

(1) For each $\tau \in \delta \cup \delta'$ which is a transition from a state q on an input $a \in \Sigma$, i.e.,τ is of the form $(q, a, ...)$, we have the following linked inclusion statement: $K_q.l \leftarrow L_\tau.l.a$
(2) For each $\tau \in \delta_1 \cup \delta'_1$ where $\tau = (q, a, r)$ we have the following simple inclusion statement: $L_\tau.l \leftarrow K_r.l$
(3) For each $\tau \in \delta_2 \cup \delta'_2$ where $\tau = (q, a, r_0, r_1)$ we have the following

intersection inclusion statement: $L_\tau.l \leftarrow K_{r_0}.u_0 \cap K_{r_1}.u_1$

(4) For each $q \in Q \cup Q'$, we have the two linked inclusion statements $K_q.u_0 \leftarrow K_q.l.v_0$ and $K_q.u_1 \leftarrow K_q.l.v_1$

(5) Finally, for each $\tau \in \delta_0 \cup \delta_0'$, we have the following simple inclusion. $L_\tau.l \leftarrow L_*.l$

Now we specify the restriction pair. The set $\mathcal{G}$ of growth restricted roles is exactly the set of all roles excepting the single role $L_*.l$. The set $\mathcal{S}$ of shrink restricted roles is the set of all roles. Now we have the following claim.

Claim: The formula $AG(K_{q_0}.l \supseteq K_{q_0'}.l)$ is satisfied at the initial state $\mathcal{P}$ of the dynamic system $Struct(\mathcal{P}, \mathcal{G}, \mathcal{S})$ iff $L(\mathcal{A}) \supseteq L(\mathcal{B})$.

Proof of the Claim: Note that policy $\mathcal{P}$ does not contain any simple member statement. As a consequence for every role $U \in Roles(\mathcal{P})$, $\mathcal{F}_{\mathcal{P}}(U) = \emptyset$. The claim is proved in two parts. In the first part we show that if in every reachable state $\mathcal{P}'$ from $\mathcal{P}$, $\mathcal{F}_{\mathcal{P}'}(K_{q_0}.l) \supseteq \mathcal{F}_{\mathcal{P}'}(K_{q_0'}.l)$ then $L(\mathcal{A}) \supseteq L(\mathcal{B})$. In the second part we prove the reverse implication.

Now, assume that in every state $\mathcal{P}'$ reachable from $\mathcal{P}$ in $Struct(\mathcal{P}, \mathcal{G}, \mathcal{S})$, $\mathcal{F}_{\mathcal{P}'}(K_{q_0}.l) \supseteq \mathcal{F}_{\mathcal{P}'}(K_{q_0'}.l)$. For any state $r \in Q$, let $\mathcal{A}_r$ denote the automaton identical to $\mathcal{A}$ excepting that its initial state is r; similarly, for any $r' \in Q'$, let $\mathcal{B}_{r'}$ denote the automaton identical to $\mathcal{B}$ excepting that its initial state is r'. Let $S = (T, \rho)$ be an input tree to the automaton $\mathcal{A}$ and hence to $\mathcal{B}$. Recall that each element of T is a member of $\{0,1\}^*$ with $NULL$ denoting the empty string and for any $x \in T$, $\rho(x) \in \Sigma$. Now we define a new policy $\mathcal{P}_S$ by adding a set Δ_S of new statements to $\mathcal{P}$ as follows. First, corresponding to each node x of T, we introduce one or two new principals as follows: if x is either a leaf node or a node with single child, we introduce just one principal W_x; if x has two children, then we introduce two principals W_x, W_x'. We also introduce another new principal W_*. Let Δ_S be the set of member statements given by $\Delta_0' \cup \Delta_1' \cup \Delta_2'$ where $\Delta_0', \Delta_1', \Delta_2'$ are as given below:

- $\Delta_0' = \{L_*.l \leftarrow W_*\} \cup \{W_*.a \leftarrow W_x\ : \rho(x) = a\ and\ x \in T$ is a leaf node$\}$;
- $\Delta_1' = \{W_y.a \leftarrow W_x\ : x, y \in T, \rho(x) = a\ and\ x$ is a parent of y and has only one child $\}$;
- $\Delta_2' = \{W_y.v_0 \leftarrow W_x',\ W_z.v_1 \leftarrow W_x'\ : x, y, z \in T, x$ has two children and y, z are its left and right child respectively $\} \cup \{W_x'.a \leftarrow W_x\ : x \in T, \rho(x) = a$ and x has two children$\}$.

Now let $\mathcal{P}_S = \mathcal{P} \cup \Delta_S$. Note that the only role of $\mathcal{P}$ that appears on the left side of a statement in Δ_S is $L_*.l$. Hence $\mathcal{P}_S$ is a reachable policy from $\mathcal{P}$ in $Struct(\mathcal{P}, \mathcal{G}, \mathcal{S})$. Now, by a simple induction on the height of the tree T, it can be shown that for every $q \in Q$ the following (**) holds. (This inductive proof is based on the definition of $\mathcal{P}$ and is left to the reader.) :
(**) $W_{NULL} \in \mathcal{F}_{\mathcal{P}_S}(K_q.l)$ iff the automaton $\mathcal{A}_q$ accepts the input tree S.

Similarly, for every $q' \in Q'$, $W_{NULL} \in \mathcal{F}_{\mathcal{P}_S}(K_q.l)$ iff the automaton $\mathcal{B}_{q'}$ accepts S.

Now consider any input tree $S = (T, \rho)$ accepted by $\mathcal{B}$. By the property (**), we see that $W_{NULL} \in \mathcal{F}_{P_S}(K_{q'_0}.l)$. By our assumption $\mathcal{F}_{P_S}(K_{q_0}.l) \supseteq \mathcal{F}_{P_S}(K_{q'_0}.l)$. Hence $W_{NULL} \in \mathcal{F}_{P_S}(K_{q_0}.l)$. By the property (**), we see that S is accepted by $\mathcal{A}$. Hence $L(\mathcal{A}) \supseteq L(\mathcal{B})$.

Now we prove the claim in the other direction. Assume that $L(\mathcal{A}) \supseteq L(\mathcal{B})$. Now consider any policy $\mathcal{P}' = \mathcal{P} \cup \Delta$ such that the only role of $\mathcal{P}$ that appears on the left side of a rule in Δ is the growth unrestricted role $L_*.l$. Now consider a fix point computation that computes the values of $\mathcal{F}_{\mathcal{P}'}(R')$ for each role R' of $\mathcal{P}'$. This computation maintains a set variable $Y_{R'}$ for each role R' of $\mathcal{P}'$, initializes all these variables to the empty set and repeatedly adds principals to them using the statements in $\mathcal{P}'$ until a fix point is reached. In each step of this computation, a statement t in $\mathcal{P}'$ is considered, its right hand side is evaluated and the result is added to the variable $Y_{R'}$ where R' is the role on the left hand side of t. (Such fix point algorithm can easily be constructed from the translation of the policy $\mathcal{P}'$ to a logic program [21,22]). The final value of $Y_{R'}$ gives the value of $\mathcal{F}_{\mathcal{P}'}(R')$. Now consider a role $K_q.l$ of the policy $\mathcal{P}$ and any principal $X \in Y_{K_q.l}$. For each such pair $(K_q.l, X)$, we associate a derivation tree $S(K_q.l, X)$ which is an input tree to the automaton $\mathcal{A}$ satisfying the following properties:

(i) If $q \in Q$ then the tree $S(K_q.l, X)$ is accepted by $\mathcal{A}_q$; similarly if $q \in Q'$ then $S(K_q.l, X)$ is accepted $\mathcal{B}_q$;
(ii) If q' is any other state in Q (or Q') such that $\mathcal{A}_{q'}$ (respectively, $\mathcal{B}_{q'}$) accepts $S(K_q.l, X)$ then $X \in Y_{K_{q'}.l}$.

From the existence of the such trees $S(K_q.l, X)$, it can be shown that $\mathcal{F}_{\mathcal{P}'}(K_{q_0}.l) \supseteq \mathcal{F}_{\mathcal{P}'}(K_{q'_0}.l)$. To see this, consider any $X \in \mathcal{F}_{\mathcal{P}'}(K_{q'_0}.l)$. From the completeness of the fix point algorithm, we see that $X \in Y_{K_{q'_0}.l}$ after termination. By property (i), $S(K_{q'_0}.l, X)$ is accepted by $\mathcal{B}$ and hence by $\mathcal{A}$. By property (ii), $X \in Y_{K_{q_0}.l}$ after termination. By soundness of the fix point algorithm, we see that $X \in \mathcal{F}_{\mathcal{P}'}(K_{q_0}.l)$.

The tree $S(K_q.l, X)$ is defined inductively as follows according to the following cases. In all these cases, the inductive step for properties (i) and (ii) holds.

- A pair of statements of the form $K_q.l \leftarrow L_\tau.l.a$ and $L_\tau.l \leftarrow L_*.l$ are used to add X to $Y_{K_q.l}$. (Note it must be that for some X', $X' \in Y_{L_*.l}$ and $X \in Y_{X'.a}$.) In this case, $S(K_q.l, X)$ is the single node tree labeled with a. From the way we defined $\mathcal{P}$, we see that $(q, a) \in \delta_0 \cup \delta'_0$ and hence it should easy to see that propertie (i) holds. Property (ii) is also seen to hold similarly, since $(q', a) \in \delta_0 \cup \delta'_0$.

- A pair of statements of the form $K_q.l \leftarrow L_\tau.l.a$ and $L_\tau.l \leftarrow K_r.l$ are used to add X to $Y_{K_q.l}$. It must be that for some X', $X' \in Y_{K_r.l}$ and $X \in Y_{X'.a}$. In this case, $S(K_q.l, X)$ is the tree whose root is labeled with a and whose only sub-tree is $S(K_r.l, X')$. In this case, $(q, a, r) \in \delta_1 \cup \delta'_1$.
- A pair of statements of the form $K_q.l \leftarrow L_\tau.l.a$ and $L_\tau.l \leftarrow K_{r_0}.u_0 \cap K_{r_1}.u_1$ together with the statements $K_{r_0}.u_0 \leftarrow K_{r_0}.l.v_0$ and $K_{r_1}.u_1 \leftarrow K_{r_1}.l.v_1$ are used to add X to $Y_{K_q.l}$. It must be that there exist principals X', X'' such that $X \in Y_{X'.a}$, $X' \in Y_{K_{r_0}.u_0}$, $X' \in Y_{K_{r_1}.u_1}$, $X' \in Y_{X''.v_0}$ and $X' \in Y_{X''.v_1}$. In this case, $S(K_q.l, X)$ is the tree whose root is labeled with a and having $S(K_{r_0}.u_0, X')$, $S(K_{r_1}.u_1, X')$ as the left and right sub-trees respectively. Also, in this case, $(q, a, r_0, r_1) \in \delta_2 \cup \delta'_2$.

$\square$

7.3 Appendix: Upper bound for Role Containment

Here we give the algorithm that computes the set of transitions for the automaton $Aut(s, K.l)$. The initial set δ' of transitions is defined in subsection 3.2. This algorithm maintains a variable Tr that contains a set of transitions. It also maintains a binary variable *Change* which denotes if a new transition has been added to Tr. In the algorithm, for any role R, we let $\mathcal{A}_R$ denote the automaton with $Principals(s) \cup Roles(s)$ as the set of states, Tr as its set of transitions and R as its initial state.

Algorithm A**:**
$Tr \leftarrow \delta'$;
$Change \leftarrow True$;
While *Change*
  $Change \leftarrow False$;
  For each role R and each string l_2, K_2,
    If the string l_2, K_2 is accepted by $\mathcal{A}_R$
      Add the ϵ-transition from R to $K_2.l_2$ to Tr;
  For each role R and for each transition in Tr on input $\cap$
    to any pair of states (R_1, R_2) and for each principal K_4 such that
    K_4 is accepted by both the automata $\mathcal{A}_{R_1}$ and $\mathcal{A}_{R_2}$
      Add the transition (R, K_4) to Tr;
  If any new transition is added in this iteration **then** $Change \leftarrow True$;
End While

The following lemma relates the automata $Aut(s, R)$ with the function $\mathcal{F}_s$.

Lemma 7 *For every $R, R', R'' \in Roles(s)$ and $l \in role\_names(s)$, the following properties hold.*

(1) $\mathcal{F}_s(R)$ *is exactly the set of* $K \in Principals(s)$ *such that the single node input tree labeled with* K *is accepted by* $Aut(s, R)$.

(2) If there is an ϵ*-transition from* R *to* R'*, i.e.,* $(R, \epsilon, R') \in \delta$*, then* $\mathcal{F}_s(R) \supseteq \mathcal{F}_s(R')$.

(3) If there is a transition from R *to* R' *on* l*, i.e.,* $(R, l, R') \in \delta$*, then* $\mathcal{F}_s(R)$ *contains the set* $\mathcal{F}_s(K.l)$ *for each* $K \in \mathcal{F}_s(R')$.

(4) If there is a transition from R *to* R', R'' *on input* $\cap$*, i.e.,* $(R, \cap, R', R'') \in \delta$ *then* $\mathcal{F}_s(R) \supseteq \mathcal{F}_s(R') \cap \mathcal{F}_s(R'')$.

For any policy s and role $R \in Roles(s)$, we show that the size of the automaton $Aut(s, R)$ is $O(n^3)$ where n is the size of the policy s. The set $Roles(s)$ can be partitioned into two sets called *active*, *passive* roles. Active roles are roles that appear on the left side $\leftarrow$ in some statement in s. Clearly the number of active roles is at most n and the number of passive roles is at most n^2. In each passive roles there is only one transition. The number of initial transitions from all the active roles is bounded by the number of statements in s and hence at most n. In the fix point computation, at most n^2 ϵ-transitions can be added to each active active role and at most n additinal transitions of the form (R, K_4) can be added to each active role R. Hence the total number of transitions from all active roles is $O(n^3)$ and the total number of transitions from all passive roles is $O(n^2)$. Hence the size of $Aut(s, R)$ is $O(n^3)$.

The following lemma states that given two tree automata $\mathcal{A}_1, \mathcal{A}_2$ there exists an exponential time bounded algorithm that checks if $L(\mathcal{A}_1) \sqsubseteq L(\mathcal{A}_2)$.

Lemma 8 *Given two automata* $\mathcal{A}, \mathcal{B}$ *over the alphabet sequence* $alphabet(s)$*, there exists an algorithm that checks if* $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$ *and that runs in time* $2^{O(n)}$ *where* n *is the sum of the sizes of* $\mathcal{A}$ *and* $\mathcal{B}$.

Proof: We prove this lemma by giving the construction of an Alternating Turing Machine (ATM) M that takes as input the descriptions of $\mathcal{A}$ and $\mathcal{B}$ and accepts if it is not the case that $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$; further more, M uses space $O(n)$. From the results of [11], it follows that there exists a deterministic algorithm of time complexity $2^{O(n)}$ for the above problem. Let $\delta_{\mathcal{A}}, \delta_{\mathcal{B}}$ be the sets of transitions of $\mathcal{A}$ and $\mathcal{B}$ respectively.

Recall that, for any input tree S, $NULL$ is the root node of S. If the root node has one child then this node is 0; if it has two children then they are $0, 1$. In the former case, we let S_0 represent the unique input tree corresponding to the sub-tree of S rooted at its lone child; in the later case, we let S_0, S_1 represent the two unique input trees corresponding to the two sub-trees of S rooted at $0, 1$ respectively.

Let S be an input tree over the alphabet sequence $alphabet(s)$. For such a tree S, let $ext(S)$ denote the set of S' such that $S \preceq S'$. Let $Astates(S)$ denote the set of all states q, of $\mathcal{A}$, such that S is accepted by $\mathcal{A}$ when it is started in the state q. Let $Bstates(S)$ denote the set of all states r of $\mathcal{B}$ such that some tree in $ext(S)$ is accepted by $\mathcal{B}$ when it is started in state r. Note the difference in the definitions of $Astates(S)$ and $Bstates(S)$.

For any set X' of state of $\mathcal{A}$, let $reverse\_\epsilon\_closure(X')$ be the set of states from which some state in X' can be reached using zero or more ϵ-transitions in $\delta_{\mathcal{A}}$. Formally this is the smallest set X such that $X' \subseteq X$, and for every state q of $\mathcal{A}$, if there exists a state $q' \in X$ such that $(q, \epsilon, q') \in \delta_{\mathcal{A}}$ then $q \in X$.

The set $X = Astates(S)$ satisfies the following properties:

(a1) If S is a tree consisting of a single node that is labeled by some $K \in Principals(s)$, then $X = reverse\_\epsilon\_closure(X')$ where $X' = \{q : (q, K) \in \delta_{\mathcal{A}}\}$;
(a2) If the root node of S is labeled by some $l \in role\_names(s)$ then $X = reverse\_\epsilon\_closure(X')$ where $X' = \{q : \exists\, q' \in X_0,\ (q, l, q') \in \delta_{\mathcal{A}}\}$ and $X_0 = Astates(S_0)$;
(a3) if the root of S is labeled by $\cap$ then $X = reverse\_\epsilon\_closure(X')$ where $X' = \{q : \exists\, q' \in X_0, \exists\, q'' \in X_1,\ (q, \cap, q', q'') \in \delta_{\mathcal{A}}\}$, $X_0 = Astates(S_0)$ and $X_1 = Astates(S_1)$.

For any set C' of states of $\mathcal{B}$, let $closure(C')$ denote the smallest set C containing C' and satisfying the following two closure conditions: for any q, if for some $q', q'' \in C$, $(q, \cap, q', q'') \in \delta_{\mathcal{B}}$ then $q \in C$; for any q, if for some $q' \in C$, $(q, \epsilon, q') \in \delta_{\mathcal{B}}$ then $q \in C$. Note that the second condition is a reverse closure condition with repsct to ϵ-transitions. Let $Y = Bstates(S)$. The set Y is closed under the above conditions, i.e., $Y = closure(Y)$; this is because for any $S', S'' \in ext(S)$, the labeled tree, whose root is labeled with $\cap$ and whose left and right sub-trees correspond to S' and S'', is also in $ext(S)$. The set Y satisfies the following additional conditions:

(b1) If S is a tree consisting of a single node that is labeled by some $K \in Principals(s)$, then $Y = closure(Y')$ where $Y' = \{q : (q, K) \in \delta_{\mathcal{B}}\}$;
(b2) If the root node of S is labeled by some $l \in role\_names(s)$ then $Y = closure(Y')$ where $Y' = \{q : \exists q' \in Y_0,\ (q, l, q') \in \delta_{\mathcal{B}}\}$ and $Y_0 = Bstates(S_0)$;
(b3) if the root of S is labeled by $\cap$ then $Y = closure(Y')$ where $Y' = Bstates(S_0) \cup Bstates(S_1)$. (The correctness of (b3) is seen from the following facts. For $i = 0, 1$, $S \preceq S_i$ and hence $Bstates(S_i) \subseteq Bstates(S)$; For any $q' \in Bstates(S_0)$, $q'' \in Bstates(S_1)$ and any q, such that $(q, \cap, q', q'') \in \delta_{\mathcal{B}}$, it is the case that $q \in Bstates(S)$.)

The ATM M guesses an input tree S and checks that S is accepted by $\mathcal{A}$ and that no tree in $ext(S)$ is accepted by $\mathcal{B}$. This is done as follows. M maintains two set variables X, Y. It performs the following steps.

(1) It first non-deterministically guesses a subset of states of $\mathcal{A}$ and checks

that the initial state of $\mathcal{A}$ is present in this; it initializes X to this value. Similarly it also guesses a subset of states of $\mathcal{B}$, but in this case, checks that the initial state of $\mathcal{B}$ is not present in this set; it initializes Y to be this set. Intuitively, X and Y represent $Astates(S)$ and $Bstates(S)$ respectively.

(2) It non-deterministically guesses a label $z \in Principals(s) \cup role\_names(s) \cup \{\cap\}$ (z is supposed to be label of the root of the tree/sub-tree it is guessing).

- If $z = K$ for some $K \in Principal(s)$ then it checks conditions (a1) and (b1) are satisfied; it accepts if these conditions are satisfied, otherwise it rejects.
- If $z = l$ for some $l \in role\_names(s)$ then M non-deterministically guesses X_0, Y_0 which are subsets of states of $\mathcal{A}$ and $\mathcal{B}$ respectively, and then checks that conditions (a2) and (b2) are satisfied. If any of these two conditions is not satisfied then it rejects. Otherwise it assigns X_0, Y_0 to the variables X, Y respectively and goes to step (2).
- If $z = \cap$ then M does the following. It non-deterministically guesses sets X_0 and X_1 which are subsets of states of $\mathcal{A}$. Similarly it non-deterministically guesses subsets Y_0 and Y_1 which are subsets of states of $\mathcal{B}$. It checks that conditions (a3) and (b3) are satisfied. If either of them is not satisfied then it rejects. Otherwise, it universally chooses a value $i \in \{0, 1\}$, assigns X_i to X and Y_i to Y, and goes to step (2).

It is not difficult to see that M accepts it's input iff it is not the case that $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$. M needs to maintain sets of states X, X_0, X_1 and Y, Y_0, Y_1 which take $O(n)$ space. It is not difficult to see that conditions a1 thru a3 and b1 thru b3 can all be checked using space $O(n)$. Hence M uses space $O(n)$. □

Let $\mathcal{A}_1, \mathcal{A}_2$ be two binary tree automata. We define another automaton $\mathcal{A}_1 \circ \mathcal{A}_2$ so that $L(\mathcal{A}_1 \circ \mathcal{A}_2) = L(\mathcal{A}_1) \circ L(\mathcal{A}_2)$. Let Q_1, Q_2 be the sets of states of $\mathcal{A}_1, \mathcal{A}_2$ respectively. Without loss of generality we assume that Q_1, Q_2 are disjoint. The set Q of states of $\mathcal{A}_1 \circ \mathcal{A}_2$ is exactly $Q_1 \cup Q_2 \cup \{q_0\}$ where q_0 is a state that does not appear in $Q_1 \cup Q_2$. Also q_0 is the initial state of $\mathcal{A}_1 \circ \mathcal{A}_2$. The set of transitions of $\mathcal{A}_1 \circ \mathcal{A}_2$ is the union of the set of transitions of $\mathcal{A}_1$ and $\mathcal{A}_2$ together with the additional transition $(q_0, \cap, q_1, q_2)$ where q_1, q_2 are the initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. It is easy to see that $L(\mathcal{A}_1 \circ \mathcal{A}_2) = L(\mathcal{A}_1) \circ L(\mathcal{A}_2)$. We define $\mathcal{A}_1 \circ \mathcal{A}_2 \circ ...\mathcal{A}_k$ by association to the left. It is to be note that the size of $\mathcal{A}_1 \circ \mathcal{A}_2$ is proportional to the sum of the sizes of $\mathcal{A}_1, \mathcal{A}_2$.

Now, we restate and prove lemma 3. For convenience, we reproduce the definition of an acceptable rp-sequence and the associated notation. A role-principal sequence (in short rp-sequence) σ is a sequence of pairs of the form (R_1, L_1), (R_2, L_2), ..., (R_m, L_m) such that for each i, $1 \leq i \leq m$, $R_i \in Roles(s)$ and $L_i \in Principals(s)$ and such that no pair repeats. For an rp-sequence σ as given above, let $u_1, ..., u_m, u_{m+1}$ be a sequence of states such that $u_1 = s$ and

for each i, $1 \leq i \leq m$, $u_{i+1} = u_i \cup \{R_i \leftarrow L_i\}$. For each i such that $1 \leq i \leq m$, let $X_i = GU\_trees(u_i, R_i)$. For any $K \in Principals(s)$, let $Roles(\sigma, K)$ be the set $\{R_i : L_i = K\}$ and $guseq(\sigma, K)$ be the sequence $(X_{i_1}, .., X_{i_l})$ where $i_1 < ... < i_l$ are all the indices j such that $L_j = K$. For a principal K, let $Product(\sigma, K)$ denote the set of growth unrestricted trees given by $Product(guseq(\sigma, K))$ and let $RGU\_trees(\sigma, K)$ denote the set which is the union of sets in $\{GU\_trees(u_{m+1}, R') : R' \in Roles(s) \text{ and } K \notin \mathcal{F}_{u_{m+1}}(R')\}$. We say that σ is an *acceptable* rp-sequence for s if for each principal K such that $Roles(\sigma, K) \neq \emptyset$,
(*) it is not the case that $Product(\sigma, K) \sqsubseteq RGU\_trees(\sigma, K)$.

Recall that $final(\sigma) = closure(u_{m+1})$.

Lemma 3: If σ is an acceptable rp-sequence then $final(\sigma) \in Dspace(s)$. Further more, there exists an algorithm that takes as input a state s, a rp-sequence σ and decides if σ is an acceptable rp-sequence for s and the algorithm runs in time $2^{O(n^4)} + O(m)$ where n is the size of s and m is the length of σ.

Proof: First we show that there exists an algorithm of exponential complexity that checks if a given rp-sequence is acceptable for s. The difficult part is checking condition (*) in the definition of acceptable rp-sequence. First we check that there are no repetitions in the sequence σ; if so we immediately reject. Clearly $m \leq n^3$. Let σ be the sequence $(R_1, L_1), ..., (R_m, L_m)$ and $u_1, ..., u_{m+1}$ be the sequence of states such that $u_1 = s$ and $u_{i+1} = u_i \cup \{(R_i, L_i)\}$ for $1 \leq i \leq m$.

In order to check if σ is acceptable, we will be constructing automata that accept the sets of trees of the form $GU\_trees(v, R)$ where v is u_i for some i, $1 \leq i \leq m+1$ and $R \in Roles(s)$. Let v be any of $u_1, ..., u_{m+1}$ and R be any role in $Roles(s)$. We briefly describe the construction of an automaton $Aut'(v, R)$ that accepts $GU\_trees(v, R)$. $Aut'(v, R)$ is constructed by modifying $Aut(v, R)$ as follows. Recall that the set of states of $Aut(v, R)$ is the set $Roles(v) \cup Principals(v)$. From $Aut(v, R)$, we remove all transitions from states in $Roles(v)$ on input symbols which are principals; that is we remove all transitions of the form (R', K) for each $R' \in Roles(v)$ and each $K \in Principals(v)$. (Note that transitions of the form (K, K) are retained; these are transitions from states belonging to $Principals(v)$ on input symbols that are principals). Recall that for every role $K.l \in Roles(v)$, the transition $(K.l, l, K)$ is in $Aut(v, R)$. We remove all such transitions if $K.l \in \mathcal{G}$, i.e., is growth restricted; however, such a transition is retained if $K.l \notin \mathcal{G}$, i.e., is growth unrestricted. These are the only changes. The resulting automaton is $Aut'(v, R)$. It can easily be shown that the set of trees accepted by $Aut'(v, R)$ is $GU\_trees(v, R)$.

For each i, $1 \leq i \leq m+1$, and for each active role R, the size of $Aut'(u_i, R)$ is $O(n^3)$ as analyzed below. Looking at the construction of $Aut'(u_i, R)$, we see that this automaton has at most n^2 transitions of the form $(K.l, l, K)$; it has at most n^3 ϵ-transitions (because there are at most n^2 such transitions from each active role of s, and none from passive roles of s); it has at most n transitions of all other types as these come directly from the statements of s and no such transitions are added during the fix point computation in the construction of $Aut(u_i, R)$.

For each $R = K.l$ which is a passive role of s, the automaton $Aut'(u_i, R)$ can be simplified as follows. Note that the start state of $Aut'(u_i, R)$ is R. Since R is a passive role of s, there is at most one transition from R, which is $(K.l, l, K)$. This transition is present only in the case when R is growth unrestricted. Thus $Aut'(u_i, R)$ can be simplified to an automaton of size $O(n)$.

For each $K \in Principals(s)$, let $guseq(\sigma, K)$, $Product(\sigma, K)$ and $RGU\_trees(\sigma, K)$ be as defined above. We check for acceptability of σ as follows. For each $K \in Principals(s)$, we build automaton $\mathcal{B}_K$ that accepts the set $Product(\sigma, K)$, the automaton $\mathcal{C}_K$ that accepts $RGU\_trees(\sigma, K)$ and then check that the property $L(\mathcal{B}_k) \sqsubseteq L(\mathcal{C}_K)$ is not satisfied. When constructing $\mathcal{B}_K$, we take the product (with respect to the operator $\circ$) of the automata $Aut'(u_{i_1}, R_{i_1})$, ..., $Aut'(u_{i_l}, R_{i_l})$ where $i_1 < i_2 ... < i_l$ are all the indices j such that $L_j = K$. There are at most n^2 automata and hence size of $\mathcal{B}_K$ is bounded by the sum of their sizes. At most n of these automata correspond to active roles of s and the size of each of them is $O(n^3)$; the remaining correspond to passive roles of s and the size of each of these is $O(n)$ as explained earlier. Thus the size of $\mathcal{B}_K$ is $O(n^4)$. Automaton $\mathcal{C}_K$ accepts $RGU\_trees(\sigma, K)$ which is the union of the languages accepted by the automata in $\{Aut'(u_{m+1}, R) : R \notin Roles(\sigma, K)$ and $K \notin \mathcal{F}_{u_{m+1}}(R)\}$. It is straightforward to see how $\mathcal{C}_K$ can be obtained by having a new initial state from which an ϵ-transition is taken to the initial states of each of the automaton in the above set. By similar reasoning as in the case of $\mathcal{B}_K$, we see that the size of $\mathcal{C}_K$ is $O(n^4)$. Using lemma 8, we can check whether $L(\mathcal{B}_K) \sqsubseteq L(\mathcal{C}_K)$ in time $2^{O(n^4)}$.

The above is repeated for each $K \in Principals$. Clearly, the over all time taken for checking the acceptability of σ with respect to s is $2^{O(n^4)} + O(m)$.

Now we prove the first part of the lemma. Let σ be an acceptable rp-sequence for s and be as given in the above definition. Let $v = final(\sigma)$. Now we show that $v \in Dspace(s)$. Let u_i for $1 \leq i \leq m+1$, X_i for $1 \leq i \leq m$ be as given above. Let $K \in Principals(s)$ be such that K appears in some pair in σ, i.e., $Roles(\sigma, K) \neq \emptyset$. Since it is not the case that $Product(\sigma, K) \sqsubseteq RGU\_trees(\sigma, K)$, there exists a growth unrestricted tree S_K in $Product(\sigma, K)$ such that

(**)for every $S' \in RGU\_trees(\sigma, K)$, it is not the case that $S_K \preceq S'$.

For each K such that $Roles(\sigma, K) \neq \emptyset$, we choose one S_K satisfying (**). Clearly the number of these chosen trees is bounded by m. Note that each tree S_K is a pair of the form (T_K, ρ_K) where T_K is the set of tree nodes and ρ_K labels each node in T_K with symbols from $Principals(s) \cup Role\_names(s) \cup \{\cap\}$. From the definition of $\circ$ operation, it should be easy to see that, for each $i = 1, ..., m$, if $L_i = K$ then there exists a $S'' \in GU\_trees(u_i, R_i)$ such that $S_K \preceq S''$. Now, we define another label with each node in each tree S_K. This labeling for nodes in S_K is given by the function η_K. We also define a set Δ_K of simple member statements. Recall that a node $x \in T_K$ is called a regular node if $\rho_K(x) \neq \cap$; it is called a first node if it is a regular node and none of its proper ancestors is a regular node.

We say that two regular nodes $x, y \in T_K$ are *immediately related* if x is a parent of y and $\rho_K(x) = \cap$. We let the relation *related* denote the reflexive, symmetric and transitive closure of "immediately related" (i.e., is an equivalence relation); thus any sub-tree of S consisting of nodes all labeled with $\cap$ are related, and their immediate children are also related to them even if they are not labeled with $\cap$. For any node $x \in T_K$, we let $T_{K,x}$ denote the sub-tree consisting of x and all its descendants; we let $S_{K,x}$ denote the labeled tree $(T_{K,x}, \rho_{K,x})$ where $\rho_{K,x}$ is the restriction of ρ_K to $T_{K,x}$. Note that $S_{K,x}$ is also a growth unrestricted tree. For any node $x \in T_K$, $\eta_K(x)$ is going to be a principal defined as follows. The values of $\eta_K(x)$ are defined so that related nodes are given the same label. If x is a leaf node then $\eta_K(x) = \rho_K(x)$. If x is a first node of S_K or is related to a first node then $\eta_K(x) = K$. If x is not related to a first node and is not a leaf node then $\eta_K(x)$ is a principal not in $Principals(s)$; in this case, the values of $\eta_K(x)$ are defined so that if x and y are related then $\eta_K(x) = \eta_K(y)$ and if x, y are not related then $\eta_K(x) \neq \eta_K(y)$. Also if x, y are nodes in two different trees $S_K, S_{K'}$ $(K \neq K')$ and both $\eta_K(x), \eta_{K'}(y)$ are not in $Principals(s)$ then $\eta_K(x) \neq \eta_K(y)$; that is, the new principals used in different trees are different. Now we define a set Δ_K of simple member statements as follows. $\Delta_K = \{K'.l' \leftarrow K'' : \exists x \in T_K$ such that $\rho_K(x) = l', \eta_K(x) = K'', \eta_K(y) = K'$ where y is the child of $x\}$. Note that each element of Δ_K corresponds to an edge in the tree. Let Δ be the union of all Δ_K such that $Roles(\sigma, K) \neq \emptyset$.

The following properties of the labeling can be easily seen. If $x, y \in T_K$ are related nodes and x is an ancestor of y then $\eta_K(x) = \eta_K(y)$ and $S_{K,x} \preceq S_{K,y}$. The only nodes in S_K that are labeled with principals in $Principals(s)$ are nodes related to the leaf nodes or to the root node.

Let $t = s \cup \Delta$. Now we show that $final(\sigma) = dual(t)$. Let $v = final(\sigma)$. Observe that $v = closure(u_{m+1})$. Hence, for every $R' \in Roles(s)$, it is the case that $GU\_trees(u_{m+1}, R') = GU\_trees(v, R')$. It should be quite easy to

see that for every $R' \in Roles(s)$ and every $K' \in Principals(s)$, if $K' \in \mathcal{F}_v(R')$ then $K' \in \mathcal{F}_t(R')$. We show that, for every $R' \in Roles(s)$ and every $K' \in Principals(s)$, if $K' \in \mathcal{F}_t(R')$ then $K' \in \mathcal{F}_v(R')$. We consider the fix point computation of $\mathcal{F}_t$. In this computation, for each $R' \in Roles(t)$, a set $Y_{R'}$ is maintained. In each iteration of this fix point computations, each statement in t is considered and the sets are updated by evaluating the right hand side of the statement and adding the result to the set $Y_{R'}$ where R' is the left hand side of the statement. By induction on the number of iterations, we show the following,
(A) for each $R' \in Roles(s)$, if a principal $K'' \in Y_{R'}$ then the following hold: if $K'' \in Principals(s)$ then $K'' \in \mathcal{F}_v(R')$; if $K'' \notin Principals(s)$ then there exists a node $x \in T_K$ with $\eta_K(x) = K''$, for some K appearing in a pair in σ, and a growth unrestricted tree $S' \in GU\_trees(v, R')$ such that $S_{K,x} \preceq S'$.

We prove this by induction on the number of iterations of the fix point computation. Assume that it holds after the k^{th} iteration. We show that it holds after the $(k+1)st$ iteration. Now, we consider the different cases based on the type of statement that is considered. Assume that the left hand side of the statement is R' and $R' \in Roles(s)$. If this is not the case then property (A) continues to hold after execution of this statement. Assume that R' is $K'.l'$.

Suppose the statement considered is $R' \leftarrow K''$. Without loss of generality, we assume that K'' is added to $Y_{R'}$ in this step. If this statement is in s then $K'' \in Principals(s)$ and hence property (A) continues to hold. Consider the case when the above statement is not in s and hence is in Δ. This means that $R' \notin \mathcal{G}$. From the definition of Δ, we see that there exist two nodes x, y in some S_K such that x is a parent of y, y is a leaf node and $\eta_K(y) = K'$, $\rho_K(x) = l'$ and $\eta_K(x) = K''$. Hence $S_{K,x}$ is given by the string (l', K'). Obviously this growth unrestricted tree is in $GU\_trees(s, R')$ and hence is in $GU\_trees(v, R')$. As a consequence, (A) is satisfied if $K'' \notin Principals(s)$. Now consider the case when $K'' \in Principals(s)$. Clearly, $K = K''$ and x is a first node in $S_{K''}$. Now, we prove that $K'' \in \mathcal{F}_v(R')$ through contradiction. Assume that $K'' = K \notin \mathcal{F}_v(R')$. Hence $S_{K,x} \in RGU\_trees(\sigma, K)$. However, $S_K \preceq S_{K,x}$ and this contradicts (**).

In all the remaining cases, the statement considered is also in s. Now, if the statement considered in the $(k+1)st$ iteration is of the form $R' \leftarrow R''$ then it is straightforward to show that (A) continues to hold. Now consider the case when the statement considered is of the form $R' \leftarrow R_1 \cap R_2$. Suppose that principal K_1 is added to $Y_{R'}$. If $K_1 \in Principals(s)$ then it easy to see that (A) continues to hold. So, assume that $K_1 \notin Principals(s)$. This means that K_1 is in both Y_{R_1} and Y_{R_2}. By the induction hypothesis, it can be seen that there exists a tree S_K, for some K, so that for each $i = 1, 2$, there exist a node y_i in T_K such that $\eta_K(y_i) = K_1$ and there exists a growth unrestricted tree $U_i \in GU\_trees(v, R_i)$ such that $S_{K,y_i} \preceq U_i$. (Note that y_1, y_2 have to be in the same tree S_K since both are labeled with K_1 and $K_1 \notin \mathit{Principals}(s)$). Let x be the least common ancestor of y_1 and y_2 in T_K. It should be easy to see that, for each $i = 1, 2$, for each proper ancestor z of y_i which is a descendant of x (possibly same as x) , it is that case that $\rho_K(z) = \cap$ and $\eta_K(z) = K_1$. Now consider the growth unrestricted tree U whose root is labeled with $\cap$ and which has U_1, U_2 as its left and right sub-trees. It should be easy to see that $U \in \mathit{GU\_trees}(v, R')$ and $S_{K,x} \preceq U$. Hence (A) continues to hold.

Now consider the case when the statement considered is of the form $R' \leftarrow K_1.l_1.l_2$. Suppose that principal K_3 is added to $Y_{R'}$ in this iteration. This means there exists a $K_2 \in Y_{K_1.l_1}$ such that $K_3 \in Y_{K_2.l_2}$. Suppose both K_3, K_2 are in $\mathit{Principals}(s)$ then it is easy to show that (A) continues to hold. Assume that $K_2 \notin \mathit{Principals}(s)$ but $K_3 \in \mathit{Principals}(s)$. By the induction hypothesis, we see that there exists a node y in some tree S_K for some K such that $\eta_K(y) = K_2$ and a tree $S'' \in \mathit{GU\_trees}(v, K_1.l_1)$ such that $S_{K,y} \preceq S''$. Since $K_2 \notin \mathit{Principals}(s)$, it has to be that the statement $K_2.l_2 \leftarrow K_3$ is in Δ_K. Hence y has a parent x such that $\eta_K(x) = K_3$ and $\rho_K(x) = l_2$. Now consider the growth unrestricted tree S', whose root is labeled with l_2 and which has S'' as its sub-tree. Since $S_{K,y} \preceq S''$ and $\rho_K(x) = l_2$, it should be easy to see that $S_{K,x} \preceq S'$. Since the statement $R' \leftarrow K_1.l_1.l_2$ is also in s and hence in v, we see that $S' \in \mathit{GU\_trees}(v, R')$. Since $\eta_K(x) \in \mathit{Principals}(s)$, we see that x is a first node of S_K. As a consequence, it is easy to see that $S_K \preceq S_{K,x}$ and hence $S_K \preceq S'$. Also $K_3 = K$. Now, if $K \notin \mathcal{F}_v(R')$ then $S' \in \mathit{RGU\_trees}(\sigma, K)$ and this would contradict (**). Hence $K \in \mathcal{F}_v(R')$ and (A) continues to hold.

Now, consider the case when $K_3 \notin \mathit{Principals}(s)$ and $K_2 \in \mathit{Principals}(s)$. By the induction hypothesis, there exists a node $x \in S_K$ for some K such that $\eta_K(x) = K_3$ and there exists a $S' \in \mathit{GU\_trees}(v, K_2.l_2)$ such that $S_{K,x} \preceq S'$. By the induction hypothesis $K_2 \in \mathcal{F}_v(K_1.l_1)$. Also in the automaton $\mathit{Aut}(v, R')$ there is a transition from R' to $K_1.l_1$ on the input symbol l_2. Since $K_2 \in \mathcal{F}_v(K_1.l_1)$, we see that there is an ϵ-transition in $\mathit{Aut}(v, R')$ from R' to $K_2.l_2$. As a consequence S' is also in $\mathit{GU\_trees}(v, R')$ and the induction hypothesis continues to hold. Now consider the case when $K_3 \notin \mathit{Principals}(s)$ and $K_2 \notin \mathit{Principals}(s)$. By the induction hypothesis, we see that there exists a node y in some S_K such that $\eta_K(y) = K_2$ and there exists some $S'' \in \mathit{GU\_trees}(v, K_1.l_1)$ such that $S'' \preceq S_{K,y}$. Also, it has to be the case that $K_2.l_2 \leftarrow K_3$ is in Δ. As a consequence, we see that for the parent x of y, $\eta_K(x) = K_3$ and $\rho_K(x) = l_2$. Now consider the growth unrestricted S' whose root is labeled with l_2 and which has S'' as its sole sub-tree. It is not difficult to see that $S_{K,x} \preceq S'$. Also, the tree $S' \in \mathit{GU\_trees}(v, R')$; this is because there is a transition in $\mathit{Aut}(v, R')$ from the state R' to the state $K_1.l_1$ on the input symbol l_2. Putting the above together, we see that (A) continues to hold. $\square$

Here we state and prove lemma 4. First we need some definitions.

Reduction of input trees: Let $S = (T, \eta)$ be an input tree to the automaton $Aut(s, R)$ where R is any role in $Roles(s)$. Let Δ be a set of simple member statements (note that Δ may contain new principals and roles names not appearing in s). We can define a notion of reducing such a tree with respect to Δ as follows. S reduces to S' if the later is obtained by a sequence of one or more of the following operations.

- If the tree has a leaf node x whose parent is labeled with a role name and the role associated with x is $K_1.l_1$ and the simple member statement $K_1.l_1 \leftarrow K_2$ is in Δ then x and its parent are replaced by a single node labeled with K_2.
- If the tree has two sibling leaf nodes x, y both labeled with the same principal K_1 (note that in this case, their parent is labeled with $\cap$) then both x, y and their parent are replaced by a single node labeled with K_1.

For an input tree S and Δ as given above, let $Red\_Principals(S, \Delta)$ be the set of principals to which S reduces to with respect to Δ, i.e., it is the set of K such that S reduces to a tree with a single node labeled with K. Note that there may be some new principals in this set, i.e., principals not present in s.

Lemma 4: The set of states enumerated by algorithm B is exactly the set $Dspace(s)$.

Proof Sketch: The proof of soundness of the algorithm, i.e., that every enumerated state is in $Dspace(s)$, follows from lemma 3. We give the completeness proof. Let s' be any simple state reachable from s. We show that $dual(s')$ will eventually be enumerated. Recall that each statement in $s' - s$ is of the form $K.l \leftarrow K'$ where $K.l$ is a growth unrestricted role of s or K is a new principal, i.e., $K \notin Principals(s)$. Consider a fix point computation of the function $\mathcal{F}_{s'}$ starting with the function $\mathcal{F}_s$. Such a computation maintains a set Y_R corresponding to each role $R \in Roles(s')$. If $R \in Roles(s)$ then Y_R is initialized to $\mathcal{F}_s(R)$, otherwise it is initialized to the empty set. In each iteration, the fix point computation adds some new element to one of these sets. Without loss of generality, we assume that exactly one such element is added to only one of the sets in each iteration. Assume that this fix point computation terminates after p iterations. Corresponding to this fix point computation we will construct a rp-sequence inductively. Actually, we define a sequence $\sigma_0, \sigma_1, ...\sigma_i, ...$ where each σ_i is a rp-sequence where σ_0 is the empty sequence. We also define a sequence $v_0, v_1, ..., v_j, ...$ of extensions of s where $v_0 = closure(s)$. These sequences are defined inductively.

Let K_j be the principal added to the set Y_{R_j} during iteration j. For each $j > 0$, if $R_j \in Roles(s)$ and $K_j \in Principals(s)$ and the statement $R_j \leftarrow K_j$ is not present in v_{j-1}, then $v_j = closure(v_{j-1} \cup \{R_j \leftarrow K_j\})$ and σ_j is obtained by concatenating (R_j, K_j) to σ_{j-1}; otherwise, $v_j = v_{j-1}$ and $\sigma_j = \sigma_{j-1}$. We show that σ_p is an acceptable rp-sequence for s and $final(\sigma_p) = dual(s')$.

Claim: For each j, $0 \leq j \leq p$, the following properties hold.

(1) For every $R' \in Roles(s)$ and $K' \in Principals(s)$, if K' is present in the set $Y_{R'}$ after j iterations then $K' \in \mathcal{F}_{v_j}(R')$.

(2) For every $R' \in Roles(s)$ and $K' \notin Principals(s)$, if K' is present in the set $Y_{R'}$ after j iterations then there exists a $S \in GU\_trees(v_j, R')$ such that $K' \in Red\_Principals(S, s' - s)$

(3) If $j > 0$ and $R_j \in Roles(s)$ and $K_j \in Principals(s)$ and the statement $R_j \leftarrow K_j$ is not present in v_{j-1}, then there exists $S \in GU\_trees(v_{j-1}, R_j)$ such that $K_j \in Red\_Principals(S, s' - s)$

Proof of Claim: We prove by induction on j. Note that for $j = 0$, i.e., before the start of the fix point computation, conditions (1) and (2) are satisfied since each $Y_{R'}$ is initialized to $\mathcal{F}_s(R')$ for $R' \in Roles(s)$, and is initialized to the empty set in the other cases. Condition (3) is vacuously satisfied for $j = 0$. It is trivial to see that condition (1) is always satisfied because if $K_j \notin v_{j-1}$ then the statement $R_j \leftarrow K_j$ is added to v_j. So we only prove the induction step for (2) and (3). Since the states v_j are monotonically non-decreasing, if any tree $S \in GU\_trees(v_j, R')$ then $S \in GU\_trees(v_{j+1}, R')$ for every $R' \in Roles(s)$. Assume that all the three properties hold up to $j = k$. Now consider the case when $j = k + 1$. If $R_{k+1} \notin Roles(s)$ then all the conditions continue to be satisfied for $j = k + 1$. So assume that $R_{k+1} \in Roles(s)$. Assume that R_{k+1} is $L'.l'$. We prove the induction step based on the type of statement that is considered. Suppose the statement considered is a simple member statement. This means that $R_{k+1} \leftarrow K_{k+1}$ is in $s' - s$. (Note that if this statement were present in s then K_{k+1} will already be in $Y_{R_{k+1}}$ contradicting our assumption). Clearly $R_{k+1} \notin \mathcal{G}$. Hence the growth unrestricted tree S given by (l', K') is in $GU\_trees(s, R_{k+1})$ and hence in $GU\_trees(v_k, R_{k+1})$. Since $R_{k+1} \leftarrow K_{k+1}$ is in $s' - s$, it follows that $K_{k+1} \in Red\_Principals(S, s' - s)$. Hence conditions (2) and (3) continue to be satisfied whether K_{k+1} is in $Principals(s)$ or not.

Now the proof of the induction step for the cases when the statement considered is either simple inclusion or intersection inclusion is fairly straightforward and are left to the reader. It can be shown that in both these cases no statement will be added to v_k because $K_{k+1} \in \mathcal{F}_{v_k}(R_{k+1})$. Now consider the case when the statement considered is $R_{k+1} \leftarrow K'.l'.l''$. This means there exists a $K'' \in Y_{K'.l'}$ and $K_{k+1} \in Y_{K''.l''}$ after the k^{th} iteration. For the case when K_{k+1} and K'' are in $Principals(s)$, the induction step is straightforward.In this case also, no statement will be added to v_k to get v_{k+1}.

Now consider the case when $K'' \notin Principals(s)$ but $K_{k+1} \in Principals(s)$. By the induction hypothesis for (2), we see that there is some tree S' in

$GU\_trees(v_k, K'.l')$ such that $K'' \in Red\_Principals(S', s' - s)$. Let S be the labeled tree in which the root is labeled with l'' and having S' as its sole sub-tree. Since there is a transition in $Aut(v_k, R_{k+1})$ from state R_{k+1} to the state $K'.l'$ on the input symbol l'', we see that $S \in GU\_trees(v_k, R_{k+1})$. Since $K''.l'' \notin Roles(s)$, it has to be that the statement $K''.l'' \leftarrow K_{k+1}$ is in $s' - s$. From this and the fact that $K'' \in Red\_Principals(S', s' - s)$, it follows that $K_{k+1} \in Red\_Principals(S, s' - s)$. Thus condition (3) continues to be satisfied. Condition (2) is vacuously satisfied. Now consider the case when both K'' and K_{k+1} are not in $Principals(s)$. In this case, conditions (2) is satisfied due to similar reasoning for condition (3) in the previous case. Condition (3) is vacuously satisfied. Now consider the case when $K'' \in Principals(s)$ but $K_{k+1} \notin Principals(s)$. In this case condition (3) is vacuously satisfied for $j = k + 1$. By the induction hypothesis, for (1) we see that $K'' \in \mathcal{F}_{v_k}(K'.l')$. From this, we see that there is an ϵ-transition in $Aut(v_k, R_{k+1})$ from R_{k+1} to $K''.l''$. Also using the induction hypothesis for (2), we see that there is a growth unrestricted tree in $S \in GU\_trees(v_k, K''.l'')$ such that $K_{k+1} \in Red\_Principals(S, s' - s)$. From the above observations we see that $S \in GU\_trees(v_{k+1}, R_{k+1})$ and condition (2) continues to be satisfied. $\square$

We see that whenever a (R_j, K_j) is added to σ_j at that time K_j is added to Y_{R_j}. Also whenever K_j is added to Y_{R_j} and $R_j \in Roles(s)$ and $K_j \in Principals(s)$, then either $R_j \leftarrow K_j$ is already in v_j or is added to v_j. These are the only elements added to v_j. Hence $final(\sigma_p)$ is going to be identical to the $dual(s')$. Now, let $(R_1, L_1), ..., (R_m, L_m)$ be σ_p and $u_1, ..., u_{m+1}$ be the states such that $u_1 = s$ and for all $i, 1 \leq i \leq m$, $u_{i+1} = u_i \cup \{R_i \leftarrow L_i\}$. From the claim given above we see that there exist labeled trees $S_1, ..., S_m$ such that for all $i, 1 \leq i \leq m$, $S_i \in GU\_trees(u_i, R_i)$ and $L_i \in Red\_Principals(S_i, s' - s)$. Now, consider any $K \in Principals(s)$. Let $1 \leq i_1 < i_2 < ... < i_r \leq m$ be all the integers such that $L_{i_j} = K$ for all $1 \leq j \leq r$. Let S' denote the single tree in $\{S_{i_1}\} \circ \{S_{i_2}\} \circ ... \circ \{S_{i_r}\}$. Now, recall that $RGU\_trees(\sigma_p, K)$ denotes the set which is the union of sets in $\{GU\_trees(u_{m+1}, R') : R' \in Roles(s)$ and $K \notin \mathcal{F}_{u_{m+1}}(R')\}$. Now we show that the following condition (C) holds.
(C) There does not exist some $S'' \in RGU\_trees(\sigma_p, K)$, such that $S' \preceq S''$.
We prove (C) by contradiction; Contrary to (C), assume that there exists $S'' \in RGU\_trees(\sigma_p, K)$, such that $S' \preceq S''$; from this, we see that $K \in Red\_Principals(S'', s' - s)$; clearly, there exists a $R' \in Roles(s)$ such that $K \notin \mathcal{F}_{u_{m+1}}(R')$ and $S'' \in GU\_trees(u_{m+1})$; since $final(\sigma_p) = dual(s')$, it is the case that $S'' \in GU\_trees(s')$; due to the completeness of the fix point algorithm, we see that $K \in Y_{R'}$ at the end of the fix point computation and $K \in \mathcal{F}_{s'}(R')$; hence $K \in \mathcal{F}_{u_{m+1}}(R')$, which is a contradiction. Thus we see that (C) holds for all $K \in Principals(s)$. Hence condition (*) given in the definition of acceptable rp-sequences is satisfied. Hence σ_p is an acceptable rp-sequence for s. Clearly the length of σ_p is bounded by n^3 where n is the size of s. $\square$

In this part of the appendix we give the proof of theorem 2. First we give some results from monadic first order logic that will be used in the proof. We first define the syntax of first order formulas. We assume that we have a set of first order variables denoted by $x, y, ..$, a set of unary predicate symbols represented by $F_1, ...$ and a set of constants. An atomic formula is of the form $x \in F, c \in F$ or of the form $x = y$ or of the form $x = c$ where x, y are first variables, F is a unary predicate symbol and c is a constant. The set of formulas of Monadic First order Logic (MFL) is the smallest set X containing all the atomic formulas, and satisfying the following closure conditions: If $f, g \in X$ then $\neg f$, $f \wedge g$ and $\exists i(f)$ are also in X.

For any first order formula f, let $free\_var(f)$, $predicates(f)$ and $constants(f)$, respectively denote the set of free variables in f, the set of predicates in f and the set of constants in f. A model $\mathcal{I}$ for f is triple $(U_{\mathcal{I}}, \Theta_{\mathcal{I}}, \Phi_{\mathcal{I}})$ where $U_{\mathcal{I}}$ is the universe over which the variables range, $\Theta_{\mathcal{I}} : predicates(f) \rightarrow 2^{U_{\mathcal{I}}}$ and $\Phi_{\mathcal{I}} : constants(f) \rightarrow U_{\mathcal{I}}$ are functions that interpret the predicate symbols and constant symbols. Note that $2^{U_{\mathcal{I}}}$ is the power set of $U_{\mathcal{I}}$. An evaluation ρ for f with respect to the model $\mathcal{I}$ is a function that assigns values from $U_{\mathcal{I}}$ to the variables in $free\_var(f)$. An interpretation for f is a pair $(\mathcal{I}, \rho)$ where $\mathcal{I}$ is a model for f and ρ is evaluation for f with respect to $\mathcal{I}$. The satisfaction of a formula in an interpretation $(\mathcal{I}, \rho)$, denoted by $(\mathcal{I}, \rho) \models f$ is defined inductively in the usual way.

Let $F_0, ..., F_{k-1}$ be the predicate symbols in $predicates(f)$. Let $\mathcal{I}$ be a model. Let $Vec_{\mathcal{I}} : U_{\mathcal{I}} \rightarrow \{0,1\}^k$ be a function that associates a vector $Vec_{\mathcal{I}}(x) = (a_0, ..., a_{k-1})$ with each $x \in U_{\mathcal{I}}$ such that, for every j, $0 \leq j \leq k-1$, $a_j = 1$ iff $x \in \Theta_{\mathcal{I}}(F_j)$; essentially the j^{th} component of the vector $Vec_{\mathcal{I}}(x)$ denotes whether x is in $\Theta_{\mathcal{I}}(F_j)$ or not. For any vector $\vec{a} \in \{0,1\}^k$, let $\#(\mathcal{I}, \vec{a})$ denote the number of distinct $x \in U_{\mathcal{I}}$ such that $Vec_{\mathcal{I}}(x) = \vec{a}$.

Let $\mathcal{I}, \mathcal{J}$ be models for f. Without loss of generality, we assume throughout the paper that for any two quantifiers $\exists i$ and $\exists j$ appearing in f, the variables i, j are different and no variable appears both as a free variable and as a bound variable in f. If f does not satisfy this property, then we can get an equivalent formula that satisfies this property by renaming some of the variables. Now we define a number m as follows. If the symbol $=$ appears in f then m is the sum of the number of variables and the number of distinct constants appearing in f, otherwise m is zero. We say that two models $\mathcal{I}, \mathcal{J}$ for a formula f are equivalent if the following conditions hold.

- For each $\vec{a} \in \{0,1\}^k$, either both $\#(\mathcal{I}, \vec{a})$ and $\#(\mathcal{J}, \vec{a})$ are greater than m, or they are equal.

- For each pair of distinct constant symbols c, d, $\Phi_{\mathcal{I}}(c) = \Phi_{\mathcal{I}}(d)$ iff $\Phi_{\mathcal{J}}(c) = \Phi_{\mathcal{J}}(d)$.

Let $\rho_{\mathcal{I}}, \rho_{\mathcal{J}}$ be evaluations for f with respect to the models $\mathcal{I}, \mathcal{J}$ respectively. We say that they are equivalent if the following conditions hold.

- For every $x \in free\_var(f)$, $Vec_{\mathcal{I}}(\rho_{\mathcal{I}}(x)) = Vec_{\mathcal{J}}(\rho_{\mathcal{J}}(x))$.
- For each pair of distinct variables $x, y \in free\_var(f)$, $\rho_{\mathcal{I}}(x) = \rho_{\mathcal{I}}(y)$ iff $\rho_{\mathcal{J}}(x) = \rho_{\mathcal{J}}(y)$.
- For each variable $x \in free\_var(f)$ and each constant c appearing in f, $\rho_{\mathcal{I}}(x) = \Phi_{\mathcal{I}}(c)$ iff $\rho_{\mathcal{J}}(x) = \Phi_{\mathcal{J}}(c)$.

We say that two interpretations $(\mathcal{I}, \rho_{\mathcal{I}})$, $(\mathcal{J}, \rho_{\mathcal{J}})$ are equivalent if $\mathcal{I}, \mathcal{J}$ are equivalent and $\rho_{\mathcal{I}}, \rho_{\mathcal{J}}$ are equivalent. The following lemma states that if two interpretations are equivalent then they both satisfy f or they both do not satisfy f. The lemma is proved by induction on the structure of f.

Lemma 9 *Let $(\mathcal{I}, \rho_{\mathcal{I}})$ and $(\mathcal{J}, \rho_{\mathcal{J}})$ be equivalent interpretations for a MFL formula f, then $(\mathcal{I}, \rho_{\mathcal{I}}) \models f$ iff $(\mathcal{J}, \rho_{\mathcal{J}}) \models f$.* □

It is to be noted that any FCTL formula f without temporal operators is a MFL formula where the predicate symbols are roles. Any policy s defines an interpretation for such a formula in a natural way, i.e.,the universe is $Principals(s)$ and a role $A.r$ is interpreted to be the set $\mathcal{F}_s(A.r)$. Now we restate and prove theorem 2.

Theorem 2: Given an initial state $\mathcal{P}$, a restriction pair $(\mathcal{G}, \mathcal{S})$, and a FCTL formula of the form $AG(f)$ where f has no temporal operators and has no free variables , then the problem of determining the role restricted dynamic system given by $\mathcal{P}$ and $(\mathcal{G}, \mathcal{S})$ satisfies $AG(f)$ is in the complexity class co-NEXPTIME. (NEXPTIME is the class of languages that can be decided in non-deterministic exponential time).

Proof: Let $\mathcal{P}$, $(\mathcal{G}, \mathcal{S})$ and $AG(f)$ be as given in the statement of the theorem. It is enough if we show that the problem of determining if there is a policy $\mathcal{P}'$ that can be obtained from $\mathcal{P}$ satisfying the restriction rule $(\mathcal{G}, \mathcal{S})$ such that $\mathcal{P}'$ satisfies $\neg f$ is in NEXPTIME.

Our proof more or less directly follows from the corresponding result for role containment given in [23,20]. Let $\mathcal{P}'$ be a policy obtained from $\mathcal{P}$ in conformance with the restriction rule $(\mathcal{G}, \mathcal{S})$ such that $\mathcal{P}'$ satisfies $\neg f$. As before define m as follows. If the equality symbol $=$ is used in f then let m be number of variables plus the number of constants appearing in f, otherwise let m be zero. As given in [23,20], we can assume that all statements added to $\mathcal{P}$ to get $\mathcal{P}'$ are simple member statements.

Let $SigRoles(\mathcal{P}, \mathcal{P}', f)$ be the set $Z_1 \cup Z_2$ where Z_1 is the set of roles that appear both in $\mathcal{P}$ and $\mathcal{P}'$ and Z_2 is the set of roles appearing f. As in [23,20], we define an equivalence relation $\equiv$ on the set $Principals(\mathcal{P}')$. For two principals Y_1, Y_2, $Y_1 \equiv Y_2$ if both $Y_1, Y_2 \in Principals(\mathcal{P})$ and $Y_1 = Y_2$, or for every role $A.r$ in $SigRoles(\mathcal{P}, \mathcal{P}', f)$, $Y_1 \in \mathcal{F}_{\mathcal{P}'}(A.r)$ iff $Y_2 \in \mathcal{F}_{\mathcal{P}'}(A.r)$. The number of equivalence classes of $\equiv$ is bounded by $2^{O(n)}$ where n is the size of $\mathcal{P}$. For each equivalence class C of $\equiv$, let $count(C)$ to be the minimum of $m+1$ and the number of elements in C. In [23,20], another policy $\mathcal{P}''$ is constructed by taking one member from each equivalence class of $\equiv$. Here we construct $\mathcal{P}''$ by taking $count(C)$ members from each equivalence class C of $\equiv$. Let $chosen(C)$ be the members chosen from C. The policy $\mathcal{P}''$ is defined as follows. If $K_1.r \leftarrow K_2$ is a rule in $\mathcal{P}'$ then for each pair (K_1', K_2') of chosen members from the equivalence classes of K_1, K_2 respectively, we have the rule $K_1'.r \leftarrow K_2'$. Note that the size of $\mathcal{P}''$ is at most exponential in the size of $\mathcal{P}$. By using the same proof as in [23,20], it can be shown that for every $K \in principals(\mathcal{P}'')$ and for every role $A.r \in SigRoles(\mathcal{P}, \mathcal{P}', f)$, it is the case that $K \in \mathcal{F}_{\mathcal{P}'}(A.r)$ iff $K \in \mathcal{F}_{\mathcal{P}''}(A.r)$.

Let $\mathcal{I}, \mathcal{J}$ be the models for f as given by $\mathcal{P}', \mathcal{P}''$ respectively, i.e., for any role $A.r$ appearing in f, $\Theta_{\mathcal{I}}(A.r) = \mathcal{F}_{\mathcal{P}'}(A.r)$ and $\Theta_{\mathcal{J}}(A.r) = \mathcal{F}_{\mathcal{P}''}(A.r)$ and $\Phi_{\mathcal{I}}(c) = \Phi_{\mathcal{J}}(d)$. It is not difficult to see that $\mathcal{I}, \mathcal{J}$ are equivalent models. Also, since we assume that $\mathcal{P}'$ satisfies $\neg f$, we see that the first order interpretation $(\mathcal{I}, \emptyset)$ satisfies $\neg f$; here $\emptyset$ is the empty evaluation. As a consequence, from lemma 9, we see that $(\mathcal{J}, \emptyset) \models \neg f$ since $(\mathcal{I}, \emptyset) \models \neg f$.

The above construction shows that if $EF(\neg f)$ is satisfiable then there is a policy $\mathcal{P}''$, reachable from $\mathcal{P}$, whose size is exponential and which satisfies $\neg f$. We can give an algorithm in NEXPTIME that guesses $\mathcal{P}''$ and checks if it satisfies f. Checking if $\mathcal{P}''$ satisfies $\neg f$ can be done in time $O(N^{c'.length(f)})$ where c' is some constant and $length(f)$ is the length of the formula and N is the size of $\mathcal{P}''$ [13]. Since N is exponential in the size of $\mathcal{P}$, we see that this algorithm is in NEXPTIME. □

7.5 Appendix: Checking Monotonic Formulas

In previous subsection 7.4 of the appendix, we defined Monadic First Order Logic. As observed there, every FCTL formula f without temporal operators is a MFL formula. Consider any MFL formula f. Every such formula can be converted into an equivalent formula in which negations apply only to atomic formulas and all double negations are eliminated; This conversion can be done by using D'Morgan's rules and by using the equivalences $\neg\forall x(g) \equiv \exists x(\neg g)$ and $\neg\neg(g) \equiv g$. We call such formulas as normalized formulas. An occurrence of an atomic formula h in a normalized formula is said to be negative if the occurrence is $\neg h$, otherwise it is said to be positive. An atomic formula is said to be a membership atomic formula if it is of the form $x \in F$ or of the form $c \in F$ where x is a first order variable and F is a monadic predicate symbol and c is a constant. A *positive formula* is a normalized formula in which all the quantifiers are existential and all occurrences of membership atomic formulas are positive. A *negative formula* is a normalized formula in which all quantifiers are universal and all occurrences of membership atomic formulas are negative. Note that a negative formula is the negation of a positive formula.

Let $(\mathcal{I}, \rho_{\mathcal{I}}), (\mathcal{J}, \rho_{\mathcal{J}})$ be two interpretations for f. Recall that the interpretation $\mathcal{I}$ is given by the triple $(U_{\mathcal{I}}, \Theta_{\mathcal{I}}, \Phi_{\mathcal{I}})$ where $U_{\mathcal{I}}$ is the universe and $\Theta_{\mathcal{I}}, \Phi_{\mathcal{I}}$ are functions that interpret predicates and constants respectively. Similar is the case for $\mathcal{J}$. We define a partial order $\preceq$ on such interpretations as follows: $(\mathcal{I}, \rho_{\mathcal{I}}) \preceq (\mathcal{J}, \rho_{\mathcal{J}})$ if $U_{\mathcal{I}} \subseteq U_{\mathcal{J}}$ and for every monadic predicate F appearing in f, $\Theta_{\mathcal{I}}(F) \subseteq \Theta_{\mathcal{J}}(F)$ and $\Phi_{\mathcal{I}} = \Phi_{\mathcal{J}}$ and $\rho_{\mathcal{I}} = \rho_{\mathcal{J}}$. Now the following lemma can be easily proved by induction on the length of f.

Lemma 10 *Let f be a MFL formula and $(\mathcal{I}, \rho_{\mathcal{I}})$, $(\mathcal{J}, \rho_{\mathcal{J}})$ be interpretations for f such that $(\mathcal{I}, \rho_{\mathcal{I}}) \preceq (\mathcal{J}, \rho_{\mathcal{J}})$. Then, the following properties hold.*

- *If f is a positive formula and $(\mathcal{I}, \rho_{\mathcal{I}}) \models f$ then $(\mathcal{J}, \rho_{\mathcal{J}}) \models f$.*
- *If f is a negative formula and $(\mathcal{J}, \rho_{\mathcal{J}}) \models f$ then $(\mathcal{I}, \rho_{\mathcal{I}}) \models f$.* □

Let the dynamic system be given by the initial state s and the restriction pair $(\mathcal{G}, \mathcal{S})$. The minimal state reachable from s is defined in subsection 3.4; it is the state obtained from s by deleting all statements that define roles not in $\mathcal{S}$. The interpretation for the formula f induced by the minimal state is smaller than (with respect to the partial order $\preceq$) the one induced by any other reachable state. The proof of the lemma 5 given in subsection 3.4 follows from this observation and lemma 10 given above.

Now we consider the problem of checking $EF(f)$ for the case when f is positive. We assume that all the roles appearing in f also appear in s. Let s and $(\mathcal{G}, \mathcal{S})$ be as given above. In 3.4, we define a maximal state s' by introducing new principals. The state s' satisfies the properties given by the following lemma. The lemma states that if any principal of s belongs to some role of s in any reachable state then that principal belongs to that role in the state s'. Also if any new principal can be in a some role in a reachable state then all members of Γ are members of that role in the state s'.

Lemma 11 *Let $K.l \in Roles(s)$.*

- *For any $K' \in Principals(s)$, if $K' \in \mathcal{F}_t(K.l)$ for some state t reachable from s (note t can be s itself), then $K' \in \mathcal{F}_{s'}(K.l)$.*
- *If there exists a $K' \notin Principals(s)$ such that $K' \in \mathcal{F}_t(K.l)$ for some t reachable from s, then $\Gamma \subseteq \mathcal{F}_{s'}(K.l)$.* □

Suppose that g is any FCTL formula without temporal operators such that all roles appearing in it are from $Roles(s)$. Let t be any state reachable from s. Let ρ, ρ' be evaluations of g that are consistent with t and s' respectively; that is, for every $x \in free\_var(g)$, $\rho(x) \in Principals(t)$ and $\rho'(x) \in Principals(s')$. We say that ρ' is equivalent to ρ if the following conditions hold: for every $x \in free\_var(g)$ if $\rho(x) \in Principals(s)$ then $\rho'(x) = \rho(x)$, otherwise $\rho'(x) \in \Gamma$; for every $x, y \in free\_var(g)$, $\rho(x) = \rho(y)$ iff $\rho'(x) = \rho'(y)$. It is to be noted that for a given ρ, there can be many equivalent evaluations ρ' as given above. The following lemma cane be proved by induction on the length of g.

Lemma 12 *Let g, t, ρ, s', ρ' be as given above. Then $(t, \rho) \models g$ iff $(s', \rho') \models g$.*
$\square$

Lemma 6, given in subsection 3.4, follows from the above lemma.