Model-checking Access Control Policies

Dimitar P. Guelev Mark Ryan Pierre Yves Schobbens

July 9, 2004

Abstract

We present a model of access control which provides fine-grained data-dependent control, can express permissions about permissions, can express delegation, and can describe systems which avoid the root-bottleneck problem. We present a language for describing goals of agents; these goals are typically to read or write the values of some resources. We describe a decision procedure which determines whether a given coalition of agents has the means (possibly indirectly) to achieve its goal. We argue that this question is decidable in the situation of the potential intruders acting in parallel with legitimate users and taking whatever temporary opportunities the actions of the legitimate users present. Our technique can also be used to synthesise finite access control systems, from an appropriately formulated logical theory describing a high-level policy.

Keywords: access control; security model; trust management; analysis; verification; logic-based design.

Introduction

In a world in which computers are ever-more interconnected, *access control systems* are of increasing importance in order to guarantee that resources are accessible by their intended users, and not by other possibly malicious users. Access control systems are used to regulate access to resources such as files, database entries, printers, web pages. They may also be used in less obvious applications, such as to determine whether incoming mail has access to its destination mailbox (spam filtering), or incoming IP packets to their destination computers (firewalls).

We present a model of access control which has among others the following features:

- Access control may be dependent on the data subject to control. This is useful in certain applications, such as the conference paper review system described below, or stateful firewalls, databases, etc. In [7], this is called *conditional authorisation*.
- Delegation of access control is easily expressed. This helps to avoid the root bottleneck, whereby root or the owner of a resource is required in order to make access control changes, and the insecurity caused by investing too much power in a single agent.
- Permissions for coalitions to act jointly can be expressed.

A key feature of our model is that *permissions* are functions of *state variables*, and therefore may change with the state. Because the ability to change the state is itself controlled by permissions, one can, in particular, express *permissions about permissions*. This allows us easily to devolve authority downwards, thus avoiding the root bottleneck, and to express delegation.

A potential problem of sophisticated access control systems, such as those which can be described using our model, is *indirect paths*. It might be that the system denies immediate access to a resource for a certain

agent, but it gives the agent indirect possibilities by allowing it to manipulate permissions. Hence, there could be a sequence of steps which the agent can execute, in order to obtain access to the resource. We are interested in verifying access control systems to check whether such indirect paths exist.

Example 1 Consider a *conference paper review system*. It consists of a set of papers, and a set of agents (which may be authors, programme-committee (PC) members, etc). The following rules apply:

1. The chair appoints agents (if they agree to it) to become PC members. PC members can resign unilaterally.
2. The chair assigns papers for reviewing to PC members.
3. PC members may submit reviews of papers that they have been assigned.
4. A PC member a may read b's review of a paper, if:
   - the paper has not been assigned to a; or
   - the paper has been assigned to a, and she has already submitted her own review.
5. PC members may appoint sub-reviewers for papers which they have been assigned. Sub-reviewers may submit reviews of those papers. The PC member can withdraw the appointment of sub-reviewers.
6. Authors should be excluded from the review process for their papers.

Each of these rules is a read access or a write access by one or more agents to a resource. We formalise this example in the next section, and use it as a running example through the paper. Statements 3 and 4 illustrate the dependency of write access and read access (respectively) on the current state. Statement 5 shows how permissions about permissions are important; here, the PC member has write permission on the data expressing the sub-reviewers' write permission on reviews.

Model checking such an access control system will answer questions such as: *can an author find out who reviewed her paper? Can a reviewer of a paper read someone else's review, before submitting his own?* We answer the second question in Example 10.

Example 2 In health care, access control systems govern an agent's ability to read and change a patient's records [2], whether the agent is the patient, a relative, a treating doctor or nurse, etc. It is desirable for this system to be flexible and allow delegation.

- The patient may delegate readability or writability of certain data (such as her address or telephone number) to a friend.
- The treating doctor may delegate readability or writability of other data (such as treatment details) to a colleague.
- The appointment booking system may allow the patient to book appointments, subject to some restrictions.

Possible indirect paths include: the patient temporarily changes his address in order to obtain an appointment at a certain hospital, and then changes it back again.

The main part of this paper presents a simple language for programming access, a propositional language for specifying access goals, and an accessibility operator which denotes that a given goal is achievable by means of a program in the programming language and can be used to formulate access control policies. We propose axioms which lead to the expressibility of this operator in propositional logic and to decision procedures for it. These procedures allow access control policies to be checked and behaviour that violates them to be proposed as counterexample to imperfect implementations of policies. Furthermore, the propositional expressibility of the accessibility operator entails that implementations of policies formulated with it can be automatically synthesised. We also show that it is decidable whether the execution of a certain program by one coalition provides another coalition with temporary opportunities that are sufficient for the achievement of a certain goal, given that the second coalition can interleave its actions with the actions of the first one.

A Prolog implementation of one of the possible decision procedures for our accessibility operator (together with examples) is available on the web [8].

Structure of the paper. We first define our model of access control formally, show how Example 1 can be encoded in it and point to some properties of our model known to be important from the literature. Then we introduce the simple programming language which expresses the procedures that coalitions of agents can use to access systems and define a class of goals that can be pursued by coalitions of agents. For every concrete system it is decidable whether a coalition can achieve a given goal of this class by running a program. We argue that the techniques developed in detail for the simple programming language can be straighforwardly extended to languages based on high-level access actions. In the concluding section we explain how these techniques lead to algorithms for model checking access control policies on existing systems and synthesising systems which implement given policies.

1 Access control systems

We denote the set of propositional formulas φ built using the variables p from some given vocabulary P by $\mathbf{L}(P)$. We adopt $\Rightarrow$ and $\bot$ as basic in the construction of these formulas and regard $\top$, $\neg$, $\wedge$, $\vee$ and $\Leftrightarrow$ as abbreviations. We denote the set of the variables occurring in a formula $\varphi \in \mathbf{L}(P)$ by $\mathrm{Var}(\varphi)$.

Definition 3 An *access control system* is a tuple $S = \langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$, where P is a set of propositional variables as above, Σ is a set of *agents*, and $\mathtt{r}$ and $\mathtt{w}$ are mappings of type $P \times \mathcal{P}_{\mathit{fin}}(\Sigma) \to \mathbf{L}(P)$, where $\mathcal{P}_{\mathit{fin}}(\Sigma)$ stands for the set of the finite subsets of Σ. The mappings $\mathtt{r}$ and $\mathtt{w}$ are required to satisfy

$$A \subset A' \text{ implies } \vdash \mathtt{r}(p, A) \Rightarrow \mathtt{r}(p, A') \text{ and } \vdash \mathtt{w}(p, A) \Rightarrow \mathtt{w}(p, A'). \tag{1}$$

The requirement (1) reflects that a coalition A has the abilities of all of its subcoalitions A.

The state of an access control system $S = \langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$ is determined by the truth values of the variables $p \in P$, denoted by 0 and 1. States are models for $\mathbf{L}(P)$ as a propositional logic language. We represent the states of S by the subsets of P, $s \subseteq P$ representing the state at which the variables which evaluate to 1 are those in s. We denote the truth value of formula φ at state s by $\varphi(s)$. Truth values of formulas are defined in the usual way.

Given $p \in P$, $A \subset_{\mathit{fin}} \Sigma$ and $s \subseteq P$, coalition A has the right to read or overwrite p at state s iff $\mathtt{r}(p, A)(s) = 1$ or $\mathtt{w}(p, A)(s) = 1$, respectively. The definitions of $\mathtt{r}$ and $\mathtt{w}$ are assumed to be known to all agents $a \in \Sigma$. Agents, however, may lack the permission to access variables in the formulas that $\mathtt{r}$ and $\mathtt{w}$ produce, and therefore be unable to decide what is permitted at certain states.

Example 4 Consider the Conference paper review system again. Let `Papers` and `Agents` be fixed sets, let the function

$$\mathtt{author} : \mathtt{Papers} \times \mathtt{Agents} \to \{\bot, \top\}$$

be fixed, and $c : \mathtt{Agents}$ the constant $c : \mathtt{Agents}$ denote the chairperson of the programme committee. Let P contain the variables

$\mathtt{pcmember}(a)$	a is a PC member
$\mathtt{reviewer}(p, a)$	paper p is assigned to PC member a
$\mathtt{subreviewer}(p, a, b)$	paper p is assigned to sub-reviewer b by PC member a
$\mathtt{submittedReview}(p, a)$	a review of p has been submitted by sub-reviewer a
$\mathtt{review}(p, a)$	the review of p from sub-reviewer a

for each $a \in \mathtt{Agents}$ and $p \in \mathtt{Papers}$. Let $\mathtt{pcmember}(c)$ hold (initially) and $\mathtt{r}$ and $\mathtt{w}$ be defined as follows:

$\texttt{pcmember}(.)$ The set of PC members is known to everyone.

$\texttt{r}(\texttt{pcmember}(a),\ A) \rightleftharpoons \top$.

A PC member may be appointed by a joint action of the chair and the candidate, and may resign unilaterally:

$\texttt{w}(\texttt{pcmember}(a),\ A) \rightleftharpoons \{a, c\} \subseteq A \vee (a \in A \wedge \texttt{pcmember}(a))$.

Here and below we use definition schemata, which become well-formed formulas over P when the agents and coalitions occurring in them become instantiated. In particular, $\texttt{author}(p, a)$ stands for a propositional constant for each pair $p \in \texttt{Papers}$, $a \in \texttt{Agents}$.

$\texttt{reviewer}(.,.)$ Reviewers are known to the PC members, except the authors of the respective paper:

$\texttt{r}(\texttt{reviewer}(p, a), \{x\}) \rightleftharpoons \texttt{pcmember}(x) \wedge \neg\texttt{author}(p, x)$

The chairperson c may assign a paper p to a PC member a who is not an author of p, if a accepts. Both c and a may resign reviewership of p unilaterally, unless a has already assigned p to a sub-reviewer:

$$\texttt{w}(\texttt{reviewer}(p, a), A) \rightleftharpoons \left(\begin{array}{c} (\texttt{pcmember}(a) \wedge \{a, c\} \subseteq A \wedge \neg\texttt{author}(p, a) \wedge \neg\texttt{reviewer}(p, a)) \vee \\ ((a \in A \vee c \in A) \wedge \texttt{reviewer}(p, a) \wedge \neg \bigvee_{b \in \texttt{Agents}} \texttt{subreviewer}(p, a, b)) \end{array} \right)$$

$\texttt{subreviewer}(.,.,.)$ The review status of a paper is known to PC members who are not authors of the paper, and to the respective sub-reviewers:

$\texttt{r}(\texttt{subreviewer}(p, a, b), \{x\}) \rightleftharpoons (\texttt{pcmember}(x) \wedge \neg\texttt{author}(p, x)) \vee x = b$

A reviewer a may assign a paper p to at most one sub-reviewer b, who is not an author of p, and has not been assigned p by another reviewer. (To review p personally, a must become his/her own sub-reviewer.) A reviewer may revoke sub-reviewership, and a sub-reviewer may resign, unless a review has already been submitted:

$$\texttt{w}(\texttt{subreviewer}(p, a, b), A) \rightleftharpoons \left(\begin{array}{c} \left(\begin{array}{c} \{a, b\} \subseteq A \wedge \texttt{reviewer}(p, a) \wedge \neg\texttt{author}(p, b) \wedge \\ \neg \bigvee_{d \in \texttt{Agents}} (\texttt{subreviewer}(p, a, d) \vee \texttt{subreviewer}(p, d, b)) \end{array} \right) \vee \\ (\texttt{subreviewer}(p, a, b) \wedge (a \in A \vee b \in A) \wedge \neg\texttt{submittedReview}(p, b)) \end{array} \right)$$

$\texttt{submittedReview}(.,.)$ Whether a review on a paper has been submitted is known to PC members, except the authors of the paper:

$\texttt{r}(\texttt{submittedReview}(p, a), \{x\}) \rightleftharpoons \texttt{pcmember}(x) \wedge \neg\texttt{author}(p, x)$

A subreviewer may submit a review once. (This makes the current value of $\texttt{review}(p, a)$ final.)

$\texttt{w}(\texttt{submittedReview}(p, a), \{x\}) \rightleftharpoons x = a \wedge \bigvee_{b \in \texttt{Agents}} \texttt{subreviewer}(p, b, x) \wedge \neg\texttt{submittedReview}(p, x)$

$\texttt{review}(.,.)$ PC member a can read reviews of a paper p, provided a is not its author and does not have a review outstanding for p.

$$\texttt{r}(\texttt{review}(p, a), \{x\}) \rightleftharpoons \left(\begin{array}{c} \texttt{pcmember}(x) \wedge \neg\texttt{author}(p, x) \wedge \texttt{submittedReview}(p, a) \wedge \\ (\bigvee_{b \in \texttt{Agents}} \texttt{subreviewer}(p, b, x) \Rightarrow \texttt{submittedReview}(p, x) \vee x = a) \end{array} \right)$$

A sub-reviewer may update the contents of his review until he/she makes it final by setting $\texttt{submittedReview}$ to 1:

$\texttt{w}(\texttt{review}(p, a), \{x\}) \rightleftharpoons x = a \wedge \bigvee_{b \in \texttt{Agents}} \texttt{subreviewer}(p, b, x) \wedge \neg\texttt{submittedReview}(p, x)$

The formulas $\texttt{r}(.,.)$ and $\texttt{w}(.,.)$ in 2-4 which are defined about singleton coalitions extend to bigger coalitions by monotonicity.

The purpose of this example is to illustrate our model and syntax. It becomes clear in Example 10 that the design of the system specified above is not flawless. It admits violating some well-established practices of conference management.

We extend $\texttt{r}$ to a mapping from $\mathbf{L}(P) \times 2^{\Sigma}$ to $\mathbf{L}(P)$ by putting $\texttt{r}(\varphi, A) \rightleftharpoons \bigwedge_{p \text{ occurs in } \varphi} \texttt{r}(p, A)$.

An access control system $\langle P, \Sigma, \texttt{r}, \texttt{w}\rangle$ is *finite*, if P and Σ are finite. In this paper we study finite access control systems. We only consider systems whose resources are sets of boolean variables; for example, the review of a paper was represented as a boolean, which is more crude than the reviews from most conferences.

1.1 Comparison with other models

Several formal models of access control have been published. The influential early work [9] proposed a model for access control with a matrix containing the current rights of each agent on each resource in the modelled system. The actions allowed include creating and destroying agents and resources and updating the matrix of the access rights. The possibility to carry out an action is defined in terms of the rights as described in the matrix. Given the generality of that model, it is not surprising that the problem of whether an agent can gain access to a resource, called the *safety problem*, is not decidable. This can be largely ascribed to the possibility to change the sets of agents and resources in the model. In our model, the sets of agents and resources are fixed.

The formulas $\mathtt{r}(p, A)$ and $\mathtt{w}(p, A)$ may be considered as the values of the cells of an access matrix

	...	Coalition A	...
...	...	...	...
Resource p	...	$\mathtt{r}(p, A)$, $\mathtt{w}(p, A)$	...
...	...	...	...

which for each particular state s of the modelled system corresponds to a matrix of the form from [9] describing the rights of reading and writing at that state. Unlike [9], entries in the matrix are updated by actions specifically for that purpose, whereas in our model coalitions update *general purpose* state variables, which in turn affect the value of the formulas $\mathtt{r}(.,.)$ and $\mathtt{w}(.,.)$. This allows the modelling of *automatic* dependencies between the contents of the access control system, if viewed as a database, and the rights of its users. The special case in which every particular right can be manipulated by a dedicated action can be modelled in our system by choosing a dedicated propositional variable $q_{\mathtt{x},p,A}$ for each triple $\mathtt{x} \in \{\mathtt{r}, \mathtt{w}\}$, $p \in P$ and $A \subseteq \Sigma$ and defining $\mathtt{x}(p, A)$ as $q_{\mathtt{x},p,A}$. Then changing the right $\mathtt{x}$ of coalition A on p can be made independently for each triple $\mathtt{x}, p, A$. In this case, however, special care needs to be taken to avoid infinite digressions like $q_{\mathtt{x},p,A}$, $q_{\mathtt{y},q_{\mathtt{x},p,A},B}$, $q_{\mathtt{z},q_{\mathtt{y},q_{\mathtt{x},p,A},B},C}$, ...

An analysis of formal models is given in [7]. Desirable properties highlighted in the literature include:

- *Conditional authorisations* [7]. Protection requirements may need to depend on the evaluation of conditions. As shown by the example above, this is a central feature of our model.
- *Expressibility of joint action* [10, 1]. Some actions require to be executed jointly by a coalition of agents, such as the appointment of an agent to the programme committee in the example above, which requires the willingness both of the chair and the candidate.
- *Delegation mechanisms*. In particular, permission to delegate a privilege should be independent of the privilege [4]. Delegation mechanisms may be classified according to permanence, transitivity and other criteria [5].
- *Support for open and closed systems* [7]. In open systems, accesses which are not specified as forbidden are allowed. Thus, the default is that actions are allowed. In closed systems, the default is the opposite: actions which are not expressly allowed are forbidden.
- *Expressibility of administrative policies* [7]. Administrative policies specify who may add, delete, or modify the permissions of the access control system. The are "one of the most important, although less understood" aspects of access control, and "usually receive little consideration" [7]. In our model, they are fully integrated, as the conference paper review example shows.
- *Avoidance of root bottleneck.* Called 'separation of duty' in [7], this property refers to the principle that no user should be given enough privilege to misuse the system on their own. Models should facilitate the design of systems having this property.

- *Support for fine-and coarse-grained specifications* [7]. Fine-grained rules may refer to specific individuals and specific objects, and these should be supported. But allowing *only* fine-grained rules would make a model unusable; some coarse-grained mechanisms such as roles must also be supported. Our model supports fine-grained rules. It relies on a higher-level language such as the language of predicates used in the example above to express coarse-grained rules.

Our model satisfies all these properties, except the last one. It is not meant to be a language for users. It represents a low-level model of access control, which we can use to give semantics to higher-level languages such as RBAC [12], OASIS [3], and the calculus of [1].

2 Programs in systems with access control

In this section we introduce a simple language which can be used to program access to systems as we described above. Programs α in it have the syntax

$$\alpha ::= \texttt{skip} \mid p{:=}\varphi \mid \texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \alpha \mid (\alpha;\alpha) \tag{2}$$

and the usual meaning. It can be shown that adding a *loop* statement, e.g. $\texttt{while } \varphi \texttt{ do } \alpha$ to this language would have no effect on its ultimate expressive power. This follows from our choice to model only finite state systems. We do not include loops in (2), because our concern is the mere existence of programs with certain properties.

2.1 Semantics of programs

We define the semantics of programs in (2) as functions from states to states. This can be regarded as a *denotational semantics* for (2), as known from the literature (see, e.g., [11]). The ingredient of this semantics that is specific and most important to our study is a mapping describing executability of programs as the subject to access restrictions. The notation below is introduced to enable the concise definition of the semantics. Let $S = \langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$ be a fixed access control system for the rest of this section.

Definition 5 Substitutions are functions of the type $P \to \mathbf{L}(P)$. We record substitutions f in the form $[f(p)/p : p \in P]$. We often write $[f(p)/p : p \in Q]$ where $Q \subset P$ to denote $\lambda p.\mathbf{if}\ p \in Q\ \mathbf{then}\ f(p)\ \mathbf{else}\ p$. If $Q = \{p_1, \ldots, p_n\}$, then we sometimes denote $[f(p)/p : p \in Q]$ by $[f(p_1)/p_1, \ldots, f(p_n)/p_n]$.

A substitution f is extended to a function of type $\mathbf{L}(P) \to \mathbf{L}(P)$ by the clauses $f(\bot) = \bot$ and $f(\varphi \Rightarrow \psi) = f(\varphi) \Rightarrow f(\psi)$. We omit the parentheses in $f(\varphi)$ for $\varphi \in \mathbf{L}(P)$. Given substitutions f and g, fg denotes $[fg(p)/p : p \in P]$. $\exists p\varphi$ stands for $[\bot/p]\varphi \vee [\top/p]\varphi$. $\forall p\varphi$ stands for $\neg\exists p\neg\varphi$. If $\mathrm{Var}(\varphi) = \{p_1, \ldots, p_n\}$, then $\exists\varphi$ and $\forall\varphi$ stand for $\exists p_1 \ldots \exists p_n\varphi$ and $\forall p_1 \ldots \forall p_n\varphi$, respectively.

Let $\mathbf{P}$ be the set of all programs in P. The function $[\![.]\!] : \mathbf{P} \to (P \to \mathbf{L}(P))$ is defined by the clauses:

$$\begin{aligned}
[\![\texttt{skip}]\!] &= [p/p : p \in P] = [\,] \\
[\![p := \varphi]\!] &= [\varphi/p] \\
[\![\texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \beta]\!] &= [(\varphi \wedge [\![\alpha]\!](p)) \vee (\neg\varphi \wedge [\![\beta]\!](p))/p : p \in P] \\
[\![(\alpha;\beta)]\!] &= [\![\alpha]\!][\![\beta]\!]
\end{aligned}$$

Proposition 6 *If S grants all the access α attempts, then the run of α from state $s \subseteq P$ takes S to state $\{p : ([\![\alpha]\!](p))(s) = 1\}$.*

Every particular step of the execution of a program can be carried out only if the respective coalition has the necessary access rights. E.g., for an assignment $p := \varphi$ to be executed, the coalition needs the right to overwrite p and read the variables occurring in φ. We define this by means of the function $[\![.,.]\!] : 2^{\Sigma} \times \mathbf{P} \to \mathbf{L}(P)$. $[\![A, \alpha]\!]$ evaluates to a formula which expresses whether the coalition A may execute the program α. $[\![.,.]\!]$ is defined by the clauses:

$$\begin{aligned}
[\![A, \texttt{skip}]\!] &= \top \\
[\![A, p := \varphi]\!] &= \mathtt{r}(\varphi, A) \wedge \mathtt{w}(p, A) \\
[\![A, \texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \beta]\!] &= \mathtt{r}(\varphi, A) \wedge (\varphi \Rightarrow [\![A, \alpha]\!]) \wedge (\neg\varphi \Rightarrow [\![A, \beta]\!]) \\
[\![A, (\alpha; \beta)]\!] &= [\![A, \alpha]\!] \wedge [\![\alpha]\!][\![A, \beta]\!]
\end{aligned}$$

Proposition 7 *S will grant coalition $A \subseteq \Sigma$ to execute program α from state s iff $[\![A, \alpha]\!](s) = 1$.*

Despite its ultimate simplicity, the language (2) can describe every deterministic and terminating algorithm for access to a system the considered type, as long as it is assumed that a failed access attempt can only bring general failure, and cannot be used to, e.g., draw conclusions on the state of a system for the purpose of further action. This restriction can be lifted. See the more general setting outlined in Subsections 4.2 and 4.3.

2.2 Programs which obtain access

Let $S = \langle P, \Sigma, \mathtt{r}, \mathtt{w} \rangle$ be a fixed access control system again, and let $\mathbf{P}$ be the set of programs (2) in the vocabulary P. Given a state $s \subseteq P$ and a $p \in P$, the truth values $\mathtt{r}(p, A)(s)$ and $\mathtt{w}(p, A)(s)$ indicate whether A can read and write p, respectively, *in state* s. However, it may be that A currently does not have some permission, but that A can change the state in order to obtain it. In this section we define $\mathtt{R}_A\varphi$ and $\mathtt{W}_A\varphi$, which denote A's ability to read/write ϕ by a possibly lengthy sequence of steps. Such sequences can be encoded as programs of the form (2). The ultimate ability for A to obtain the truth value of $\varphi \in \mathbf{L}(P)$ can be understood as the ability of A to run a program $\alpha \in \mathbf{P}$ that works out the value of φ and copies it into some variable p_0 such that $\mathtt{r}(p_0, A) = \mathtt{w}(p_0, A) = \top$. It can be expressed in terms of $[\![\alpha]\!]$ and $[\![A, \alpha]\!]$ as follows:

$$\mathtt{R}_A\varphi \rightleftharpoons (\exists\alpha \in \mathbf{P})\forall([\![A, \alpha]\!] \wedge ([\![\alpha]\!](p_0) \Leftrightarrow \varphi)) \tag{3}$$

Similarly, the ability of A to drive the system into a state where some $\varphi \in \mathbf{L}(P)$ has a truth value of A's choosing, can be expressed by the formula

$$\mathtt{W}_A\varphi \rightleftharpoons (\exists\alpha_\top, \alpha_\bot \in \mathbf{P})\forall([\![A, \alpha_\top]\!] \wedge [\![A, \alpha_\bot]\!] \wedge [\![\alpha_\top]\!]\varphi \wedge [\![\alpha_\bot]\!]\neg\varphi) \tag{4}$$

The universal closures $\forall$ in (3) and (4) express that α, $\alpha_\top$ and $\alpha_\bot$ are runnable and produce the stated results from all initial states. Note that $\mathtt{R}_A$ and $\mathtt{W}_A$ allow for destructive behaviour of the programs involved. Obtaining the desired goal may involve changing the state, possibly in a way which A cannot undo. In the next section, we consider a more expressive goal language in which we restrict the search to programs which are not destructive.

The formulas (3) and (4) determine the ability of A to execute a program which would achieve the goal of reading or writing φ. Quantifier prefixes like $(\exists\alpha \in \mathbf{P})$ make it hard to evaluate (3) and (4) directly. However, if S is finite, these formulas have purely propositional equivalents, and therefore can be computed mechanically, because there are only finitely many different programs in the vocabulary P modulo semantical equivalence. Of course, the enumerating all these programs in order to evaluate $(\exists\alpha \in \mathbf{P})$ is very inefficient. In Section 3 we treat $\mathtt{R}_A$ and $\mathtt{W}_A$ as special cases of a more general accessibility operator. In Appendix A we describe a way to evaluate this operator, and consequently, $\mathtt{R}_A$ and $\mathtt{W}_A$, without resorting to quantifier prefixes of the form $(\exists\alpha \in \mathbf{P})$, which is more efficient.

3 A general accessibility operator

Extracting information and driving a system into a state with some desired property are only the simplest goals of access. One goal cannot be treated without regard for others, because achieving a goal may have destructive side effects which prevent another goal from being achieved. That is why achieving composite goals sometimes needs to be planned with all their subgoals in mind at the same time. In this section, we consider a language for describing more refined kinds of access. Our language allows us to express boolean combinations of goals. Expressible goals include preserving the truth value of some formulas while reading or setting the truth values of others. Preservation is understood as restoring the original value of the formula in question upon the end of activities, and not necessarily keeping the value constant throughout the run of a program.

The accessibility operator in this language is written in the form $A(\Phi, \psi)$ where A is a coalition, Φ is a list of formulas in $\mathbf{L}(P)$ that A wants to read, and ψ is a *goal formula* with the syntax

$$\psi ::= \bot \mid \top \mid \mathbf{p} \mid \psi \wedge \psi \mid \psi \vee \psi \tag{5}$$

where $\mathbf{p}$ denotes an atomic goal of one of the following forms:

- $\overline{\varphi}'$, where $\varphi \in \mathbf{L}(P)$; this is the goal of making φ true.
- $\overline{\varphi}$, where $\varphi \in \mathbf{L}(P)$; this is the goal of "realising" that φ is true.

$\top$ and $\bot$ stand for a trivial goal, which calls for no action, and an unachievable goal, respectively. The goal $\psi_1 \vee \psi_2$ is regarded as achieved if either ψ_1 or ψ_2 are. The goal $\psi_1 \wedge \psi_2$ is achieved if both ψ_1 and ψ_2 are. Atomic goals of the form $\overline{\varphi}$ may fail even if A manages to obtain the truth value of φ, in case it turns out to be 0. On the other hand a goal of the form $\overline{\varphi} \vee \overline{\neg\varphi}$ can be assumed achieved without any action.

Example 8 The expression $A(\langle p\rangle,\ (\overline{q} \wedge \overline{q}') \vee (\overline{\neg q} \wedge \overline{\neg q}'))$ denotes that A wants to read p and *preserve* the truth value of q. If $\mathtt{r}(p, A) = q$ and $\mathtt{r}(q, A) = \mathtt{w}(q, A) = \top$, then A can achieve its goal by means of the program

$$\mathtt{if}\ q\ \mathtt{then}\ p_0{:=}p\ \mathtt{else}\ (q{:=}\top;\ p_0{:=}p;\ q{:=}\bot)$$

where p_0 is a variable dedicated to storing the value of p. Note that the program restores the value of q after temporarily setting it to 1 in order to gain access to p in the else clause of the conditional statement. The goal described by the simpler expression $A(\langle p\rangle, \top)$, which does not require q to be restored, can be achieved by the simpler program $(q{:=}\top;\ p_0{:=}p)$.

The formula ψ in $A(\Phi, \psi)$ can express an arbitrary relation $R(\overline{p_1}, \ldots, \overline{p_n}; \overline{p_1}', \ldots, \overline{p_n}')$ between the initial values $\overline{p_1}, \ldots, \overline{p_n}$ and the final values $\overline{p_1}', \ldots, \overline{p_n}'$ of the variables $p_1, \ldots, p_n$ of the system as a requirement for A to satisfy. The main difficulty in implementing the relation R in our setting is not in computing R, but to the planning of the actions needed to access the variables.

Example 9 Let $P = \{p_1, p_2, p_3\}$, $A \subseteq \Sigma$, $\mathtt{r}(p_1, A) = \neg p_2$, $\mathtt{w}(p_1, A) = p_2$, $\mathtt{r}(p_2, A) = \mathtt{w}(p_2, A) = \top$, $\mathtt{r}(p_3, A) = p_1$ and $\mathtt{w}(p_3, A) = \neg p_1$. *From any state, can A achieve a state in which the value of p_3 is inverted?* Yes; for example, this program samples the variables in order to determine what it can do, and inverts the value of p_3. A can run it from any state.

```
if p2 then (
    p1:=⊤;
    if p3 then (p1:=⊥; p3:=⊥) else (p1:=⊥; p3:=⊤)
)
else if p1 then
    if p3 then (p2:=⊤; p1:=⊥; p3:=⊥) else (p2:=⊤; p1:=⊥; p3:=⊤)
else (
```

$p2{:=}\top;\ p1{:=}\top;$
$\texttt{if}\ p3\ \texttt{then}\ (p1{:=}\bot;\ p3{:=}\bot)\ \texttt{else}\ (p1{:=}\bot;\ p3{:=}\top)$
)

The program (except for the formatting) was produced by our implementation [8].

In general, the goal $A(\Phi, \psi)$ expresses the ability of the coalition A to execute a program which reads the values of formulas in Φ, while changing the values of formulas in order to make the relation represented by ψ hold. The simple goals expressed by $\mathtt{R}_A\varphi$ and $\mathtt{W}_A\varphi$ can be expressed in this language:

$$\mathtt{R}_A\varphi \Leftrightarrow A(\{\varphi\}, \top), \qquad \mathtt{W}_A\varphi \Leftrightarrow A(\emptyset, \overline{\varphi}') \wedge A(\emptyset, \overline{\neg\varphi}').$$

In Appendix A we show that the possibility (for A) to achieve $A(\Phi, \psi)$ can be decided mechanically and, if $A(\Phi, \psi)$ is achievable, a program which can be used (by A) to achieve it can be synthesised.

To demonstrate this, we add the superscripts V, T, K to goal expressions. $A^{V,T,K}(\Phi, \psi)$ expresses the existence of a program α which A can execute to read the formulas from Φ and enforce the relation represented by ψ, *provided that the initial state s of the system satisfies $s \cap V = T$* and *without going through any of the states in the list of states K*. We use the superscript triple V, T, K to express achievability of *sub*goals which can arise after some action that brings partial knowledge of the state of the system has already been taken. The list K is used to prevent considering moving to states which have already been explored. Now the original form $A(\Phi, \psi)$ can be viewed as the special case $A^{\emptyset,\emptyset,\emptyset}(\Phi, \psi)$, in which nothing is assumed about initial states. Further details are given in Appendix A.

Sometimes goals involve enabling the achievement of further goals. A natural way to formulate and to enable reasoning about such goals is to allow *nested occurrences* of the accessibility operator A in goal formulas ψ:

$$\psi ::= \bot \mid \top \mid \mathbf{p} \mid A(\Phi, \psi) \mid \psi \wedge \psi \mid \psi \vee \psi \tag{6}$$

Example 10 For the conference paper review system, the question of whether reviewer a of paper p can read reviewer b's review before submitting his own, may be written as:

$$\{a, b, c\}(\emptyset, \overline{\mathtt{submitted}(p, b) \wedge \neg\mathtt{submitted}(p, a)}' \wedge \{a\}(\langle \mathtt{review}(p, b)\rangle, \overline{\mathtt{submitted}(p, a)}')).$$

This formula asks: *is it possible for a, b and the chair c to reach a state s in which b has submitted his review of p but a has not yet submitted hers, and from there a may read b's review and then submit hers?* If this formula holds, we can synthesise a program for $\{a, b, c\}$ to enable $\{a\}$ to achieve $(\langle \mathtt{review}(p, b)\rangle, \overline{\mathtt{submitted}(p, a)}')$ from such an s. Surprisingly, the answer is "yes". PC member can a read b's review, then become appointed a subreviewer by c and submit her own review.

If Φ is the empty list $\langle\rangle$, then $A^{V,T,K'}(\Phi, B(\Phi', \psi'))$ means that A can reach a state s in which A's knowledge of s will be sufficient for B to achieve (Φ', ψ'). In case $\Phi' \neq \langle\rangle$, we assume that it is possible to achieve $A^{V,T,K'}(\Phi, B(\Phi', \psi'))$ by (I) *A sharing with B its knowledge* of a reached s described by appropriate V and T upon passing the control to B and then (II) B reading the formulas from Φ *for* A. That is why we have

$$B^{V,T,\{T\}}(\Phi' * \Phi, \psi') \Rightarrow A^{V,T,K'}(\Phi, B(\Phi', \psi'))$$

where $\Phi' * \Phi$ denotes the concatenation of Φ' and Φ. Since K is irrelevant to the description of the knowledge of coalition A on S, it does not appear on the left of $\Rightarrow$ above. Appendix A covers the extended syntax (6).

4 Some generalisations

Here we outline some more general forms of the model of access control described in the previous sections and how our results about this model extend to these forms.

4.1 Concurrent access

Now let coalitions A and B be running programs α and β, respectively. Let the individual steps of α and β be interleaved. Let B have priority over A in the following sense: B can choose to execute as many steps of β as it wishes, then allow the next step of α to be made and regain control. Let β pass control to α for one step by executing the special command `sleep`. Intuitively, this means that B can monitor the behaviour of A to the extent it can read the variables A updates and take advantage of whatever access rights A grants B as a side effect of pursuing its own goals. Let us denote this form of parallel composition of α and β by $\alpha\|\beta$. We define $[\![\alpha\|\beta]\!]$ for α and β of the form $(\gamma;\mathtt{skip})$ for the sake of technical convenience:

$[\![\alpha\|\mathtt{skip}]\!] = [\![\alpha]\!]$;
$[\![\alpha\|(p := \varphi;\beta)]\!] = [\varphi/p][\![(\alpha\|\beta)]\!]$;
$[\![\alpha\|(\mathtt{if}\ \varphi\ \mathtt{then}\ \beta_1\ \mathtt{else}\ \beta_2;\beta_3)]\!] = [(\varphi \wedge [\![\alpha\|(\beta_1;\beta_3)]\!](p)) \vee (\neg\varphi \wedge [\![\alpha\|(\beta_2;\beta_3)]\!](p))/p : p \in P]$;
$[\![\alpha\|((\beta_1;\beta_2);\beta_3)]\!] = [\![\alpha\|(\beta_1;(\beta_2;\beta_3))]\!]$;
$[\![\mathtt{skip}\|(\mathtt{sleep};\beta)]\!] = [\![\beta]\!]$;
$[\![(p := \varphi;\alpha)\|(\mathtt{sleep};\beta)]\!] = [\varphi/p][\![(\alpha\|\beta)]\!]$;
$[\![(\mathtt{if}\ \varphi\ \mathtt{then}\ \alpha_1\ \mathtt{else}\ \alpha_2;\alpha_3)\|(\mathtt{sleep};\beta)]\!] = \left[\begin{array}{l}(\varphi \wedge [\![(\alpha_1;\alpha_3)\|(\mathtt{sleep};\beta)]\!](p)) \vee \\ (\neg\varphi \wedge [\![(\alpha_2;\alpha_3)\|(\mathtt{sleep};\beta)]\!](p))\end{array} /p : p \in P\right]$;
$[\![((\alpha_1;\alpha_2);\alpha_3)\|(\mathtt{sleep};\beta)]\!] = [\![(\alpha_1;(\alpha_2;\alpha_3))\|(\mathtt{sleep};\beta)]\!]$.

The clause $[\![\mathtt{skip}\|(\mathtt{sleep};\beta)]\!] = [\![\beta]\!]$ states that `sleep` has no effect when the program run by A has nothing left to do. To express this about subsequent possible occurrences of `sleep` in β, we extend $[\![.]\!]$ by putting $[\![\mathtt{sleep}]\!] = [\![\mathtt{skip}]\!]$. The executability $[\![A,B,\alpha\|\beta]\!]$ of $\alpha\|\beta$ by A and B can be defined like in the case of programs run by individual coalitions. We skip the definition here.

Now let $p_B \in P$ satisfy $\mathtt{w}(p_B,B) = \top$ and $\mathtt{w}(p_B,A) = \bot$ and assume that B is trying to take whatever opportunities appear while A is executing α, in order to obtain a copy of the truth value of ψ in p_B by executing β. It is natural to assume that B takes the advantage of doing as many things as it wish between every two updates A does. That is why the actions of A and B in the course of their executing α and β respectively are interleaved as in $\alpha\|\beta$. We denote the set of all programs that can possibly have occurrences of `sleep` by $\mathbf{P}_{\mathtt{sleep}}$.

Using $[\![.\|.]\!]$ and $[\![.,.,.\|.]\!]$, we can describe, e.g. what it means for coalition B to be able to read some property φ of the state of the system by means of running program β in parallel with program α being run by coalition A:

$$\mathtt{R}_B(\varphi,A,\alpha) \rightleftharpoons (\exists\beta \in \mathbf{P}_{\mathtt{sleep}})\forall([\![A,B,\alpha\|\beta]\!] \wedge ([\![\alpha\|\beta]\!](p_0) \Leftrightarrow \varphi)) \tag{7}$$

A fixed α implies an upper bound of the number of `sleep` statements that β may need to execute in a sequence in order to let α complete its execution. This entails that, much like in the case of a single coalition accessing the system, there are finitely many β modulo equivalence with respect to their effect on the system, including their interaction with the fixed interleaved α. This means that the quantifier prefix $(\exists\beta \in \mathbf{P}_{\mathtt{sleep}})$ in (7) can be eliminated and, therefore, $\mathtt{R}_B(\varphi,A,\alpha)$ can be calculated. The more efficient approach from Appendix A can be applied to this setting too.

4.2 Access control with arbitrary atomic actions

So far our model allows only simple assignments to boolean variables as the atomic actions. This brings the level of abstraction down and makes some natural things difficult to program. For example, consider the system $S = \langle P,\Sigma,\mathtt{r},\mathtt{w}\rangle$ where $P = \{p_1,p_2\}$ and $\mathtt{w}(p_1,\Sigma) = \mathtt{w}(p_2,\Sigma) = p_1 \wedge p_2$. Then Σ can overwrite each of p_1 and p_2 at state $\{p_1,p_2\}$, but can never change the values of both variables, because once one of the values becomes 0, the writing permission is lost. Hence, there is no way to permit Σ the transition from state $\{p_1,p_2\}$ to state $\emptyset$ without also allowing Σ to change some of the states $\{p_1\}$ and $\{p_2\}$, or even to

leave S in one of these states and never proceed towards $\emptyset$. This means that coalitions cannot be forced to maintain integrity constraints, like, e.g., $p_1(s) = p_2(s)$ for all $s \subseteq 2^P$, keep logs, or be saved from painting themselves into a corner. This restriction can be removed by introducing high-level atomic actions instead of the single variable assignments. In this section we argue that the technique developed for assignments as the atomic transformations on system state generalise to arbitrary atomic actions.

Let our programming language have the atomic statements $\mathtt{a}_1, \dots, \mathtt{a}_k$, each denoting some transformation on the state. Let $[\![\mathtt{a}_i]\!]$ and $[\![A, \mathtt{a}_i]\!]$ denote the substitution which represents the transformation performed by $\mathtt{a}_i$ in the sense of Proposition 6, and the rights required for A to execute $\mathtt{a}_i$ in the sense of Proposition 7, respectively. $[\![\mathtt{a}_i]\!]$ and $[\![A, \mathtt{a}_i]\!]$ were defined in terms of $\mathtt{r}$ and $\mathtt{w}$ for the case of $\mathtt{a}_i$ being assignments in Section 2. Now we assume that these are *given* substitutions and formulas for $\mathtt{a}_1, \dots, \mathtt{a}_k$. Let us replace $\mathtt{w}$ by the mappings $[\![., \mathtt{a}_i]\!] : 2^\Sigma \to \mathbf{L}(P)$, $i = 1, \dots, k$, in systems which control access by atomic actions $\mathtt{a}_1, \dots, \mathtt{a}_k$. Let us retain $\mathtt{r}$, which determines reading permissions. We obtain access control systems of the form

$$\langle P, \Sigma, \mathtt{r}, \lambda A.[\![A, \mathtt{a}_1]\!], \dots, \lambda A.[\![A, \mathtt{a}_k]\!] \rangle.$$

Since for every subsitution $[\![\mathtt{a}_i]\!]$ there is an α_i of the form (2) such that $[\![\mathtt{a}_i]\!] = [\![\alpha_i]\!]$, we can reproduce the results from Sections 2-3 and Appendix A about such systems. This possibility shows that, despite its restrictions, the language (2) and the techniques for it from Sections 2.2-3 have a fundamental role in the assembly of the respective machinery for the analysis of finite access control described in terms of high-level actions.

4.3 General knowledge states

So far we have allowed only pairs of the form V, T to represent the knowledge states of coalitions. A knowledge state can be viewed as a nonempty set of system states. Pairs V, T can represent only some such sets. In general, any set of system states described by a satisfiable formula from $\mathbf{L}(P)$ can be viewed as a knowledge state. This leads to a natural generalisation of $A^{V,T,K}(\dots)$ to the form $A^{\chi,K'}(\dots)$ where K' is a sequence of formulas $\chi_1, \dots, \chi_k$ from $\mathbf{L}(P)$ such that $\vdash \chi_{i+1} \Rightarrow \chi_i$ and $\not\vdash \chi_i \Rightarrow \chi_{i+1}$, $i < k$, and χ_k is χ.

General knowledge states can be used to deal with the assumption that a failed access attempt ony causes the attempting coalition A to learn that the (generally arbitrary) formula $\mathtt{x}(p, A)$ does not hold, where $\mathtt{x} \in \{\mathtt{r}, \mathtt{w}\}$ denotes the attempted action and p is the accessed variable.

Conclusions

We conclude by listing some problems whose solutions can be derived from the techniques developed in this paper.

Model checking (Synthesis of attacks). Given a concrete access control system of the form $\langle P, \Sigma, \mathtt{r}, \mathtt{w} \rangle$ the recursive equation (16) for $A(\Phi, \psi)$ from Appendix A provides an algorithm to calculate the ability of a coalition A to achieve a general goal combining reading and writing variables, and, if there is such ability, to synthesise a program for A to achieve the goal. Hence it can be checked whether the system permits various forms of legitimate access, leak of data or attacks which can be written as goals of the form (Φ, ψ). In Subsection 4.1 we show that the same problem is decidable in the situation of the potential intruders acting in parallel with legitimate users and taking whatever temporary opportunities the actions of legitimate users present.

Synthesis of access control systems. Given a set of propositional variables P, a set of agents Σ and an access control policy formulated as a logical theory about $A(.,.)$ for $A \subseteq \Sigma$ on systems which have their

state described in terms of the variables from P, it can be decided whether an access control system of the form $\langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$ which implements this policy exists and, if so, definitions for its remaining components $\mathtt{r}$ and $\mathtt{w}$ can be proposed. This can be done by developing the equation (16) into full propositional definitions of the instances of $A(.,.)$ involved in the formulation of the policy and establishing the satisfiability of the policy with respect to the applications of $\mathtt{r}$ and $\mathtt{w}$ at the respective states of the system treated as propositional variables. If the policy turns out to define a satisfiable restriction on $\mathtt{r}$ and $\mathtt{w}$, any particular pair of mappings $\mathtt{r}$ and $\mathtt{w}$ which satisfies this restriction can be chosen to complete the access control system in a way which implements the given policy.

In Section 4 we argued that the results from Sections 2-3 can be reproduced for systems of a general form where access is based on an arbitrary set of high-level actions. A representation of the respective access operator $A(.,.)$ like that in Appendix A for the basic case can be assembled from the components used in this basic case. We proposed a way to reason about goals which involve enabling some further goals to be achieved. We also showed how to generalise the form of knowledge states of coalitions used in Sections 2-3 and thus allow to describe that coalitions know arbitrary constraints on the states of systems and that coalitions can learn from failures.

The algorithms which follow from Appendix A are not optimal. Results on the complexity of the problems on the class of all access control systems of the considered form might be practically unrepresentative, because instances of extreme complexity usually have little in common with typical real cases. That is why it would be interesting to describe subclasses which exhibit the kinds of regularity typical for real access control systems first.

References

[1] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[2] R. J. Anderson. *Security in Clinical Information Systems*. British Medical Association, 1996. `www.cl.cam.ac.uk/users/rja14/policy11/policy11.html`.

[3] Jean Bacon, Ken Moody, and Walt Yao. Access control and trust in the use of widely distributed services. *Lecture Notes in Computer Science*, 2218:295+, 2001. Also: *Software Practice and Experience* 33, 2003.

[4] O. Bandmann, M. Dam, and B. Firozabadi. Constrained delegations. In *Proc. IEEE Symposium on Security and Privacy*, pages 131–142, 2002.

[5] E. S. Barka. *Framework for Role-Based Delagation Models*. PhD thesis, George Mason University, 2002.

[6] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[7] Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Access control: principles and solutions. *Software Practice and Experience*, 33:397–421, 2003.

[8] D. P. Guelev. Prolog code supporting "Model-checking access control policies". `http://www.cs.bham.ac.uk/~dpg/mcacp/`, November 2003.

[9] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. On protection in operating systems. In *Proceedings of the fifth symposium on Operating systems principles*, pages 14–24. ACM Press, 1975.

[10] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[11] H. Riis Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. See `http://www.imm.dtu.dk/~riis` for information about how to download a copy of the book and supplementary course material.

[12] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

A Calculating $A^{V,T,K}(\Phi,\psi)$

In this appendix we axiomatise the operator $A^{V,T,K}(\Phi,\psi)$. The axioms we present can be used as reduction rules to decide whether $A^{V,T,K}(\Phi,\psi)$ holds, that is, whether, given an access control system $S=\langle P,\Sigma,\mathtt{r},\mathtt{w}\rangle$, and assuming that $A\subseteq\Sigma$ knows that the current state s of S satisfies $s\cap V=T$, A can obtain the truth values of the formulas from Φ at s and reach a state which is related to s in the way described by ψ. Furthermore, A is supposed to be able to achieve its goal without (repeating) visits to states which match partial state descriptions from the list K. We assume that K is a set of subsets of P and, as long as the variables whose values A knows are all in V, $X\in K$ means that A should avoid states s such that $s\cap V=X$. K becomes obsolete when A learns the values of more variables.

In the rest of this appendix we assume that $\mathrm{Var}(\varphi)\cap V=\emptyset$ for φ from Φ and for φ such that $\overline{\varphi}$ occurs in ψ, because it is always possible to simplify away the dependencies of goals on known variables. We denote the elements of Φ by $\varphi_1,\ldots,\varphi_l$.

The ability of A to achieve a goal (Φ,ψ) is equivalent to the existence of a program α of the form (2) for the actions of A, which A can execute without fail from states satisfying $s\cap V=T$ and achieve (Φ,ψ).

The lemma below shows that, as far as program behaviour is concerned, we only need to consider programs of a special form:

Lemma 11 *Every program α of the form (2) is equivalent to one with the syntax*

$$\alpha ::= \mathtt{skip}\mid \mathtt{if}\ p\ \mathtt{then}\ \alpha\ \mathtt{else}\ \alpha\mid (p{:=}\bot;\alpha)\mid (p{:=}\top;\alpha) \tag{8}$$

Proof: An assignment $p{:=}\varphi$ can be replaced by $\mathtt{if}\ \varphi\ \mathtt{then}\ p:=\top\ \mathtt{else}\ p:=\bot$ to avoid formulas other than $\top$ and $\bot$ on the right of $:=$. Furthermore, testing compound formulas in conditional statements can be replaced by sequences of testings of the variables which occur in these formulas. Finally, using the associativity of $(.;.)$ and the equivalence between $(\mathtt{if}\ \varphi\ \mathtt{then}\ \alpha_1\ \mathtt{else}\ \alpha_2;\beta)$ and $\mathtt{if}\ \varphi\ \mathtt{then}\ (\alpha_1;\beta)\ \mathtt{else}\ (\alpha_2;\beta)$, all sequential compositions can be made start with an assignment. $\dashv$

Let the triple V,T,K describe the knowledge $s\cap V=T$ of A and A's having visited the states described in the list K. Programs α of the form (8) which A can use to achieve a goal (Φ,ψ) can either start with a conditional statement which samples a variable, or with an assignment to a variable.

Sampling variable $p\in P$ could be worthwhile for A only if $p\notin V$ and A can be confident that it is possible iff $\forall\sigma_{V,T}\mathtt{r}(p,A)$ is true, where

$$\sigma_{V,T}\rightleftharpoons[\bot/p:p\in V\setminus T][\top/p:p\in T].$$

Sampling p increases the knowledge V,T of A to $V\cup\{p\},T'$, where T' is either T, or $T\cup\{p\}$, depending on the sampled value. After a sampling step A can forget the list K of partial descriptions of the states to avoid, because now it has track of one more variable which these partial descriptions do not cover. The increased knowledge of A can be used to simplify its goal (Φ,ψ). Φ can be replaced either by $[\bot/p]\Phi$, which is

$$\langle[\bot/p]\varphi_1,\ldots,[\bot/p]\varphi_l\rangle$$

or by $[\top/p]\Phi$, which is defined similarly, depending on the sampled value of p. The goal formula ψ can be simplified to either $\sigma_{p,\bot}\psi$ or $\sigma_{p,\top}\psi$, where

$$\sigma_{p,\tau}\rightleftharpoons[\overline{[\tau/p]\varphi}/\overline{\varphi}:\varphi\in\mathbf{L}(P)]\text{ for }\tau\in\{\bot,\top\}$$

for the same reasons. Hence we have the axiom:

$$\bigvee_{p\notin V}\left(\begin{array}{c}A^{V\cup\{p\},T\cup\{p\},\{T\cup\{p\}\}}([\top/p]\Phi,\sigma_{p,\top}\psi)\wedge\\ A^{V\cup\{p\},T,\{T\}}([\bot/p]\Phi,\sigma_{p,\bot}\psi)\wedge\forall\sigma_{V,T}\mathtt{r}(p,A)\end{array}\right)\Rightarrow A^{V,T,K}(\Phi,\psi) \tag{9}$$

Assigning a variable p can be worthwhile for A either if $p \notin V$, or if $p \in V$, the assignment really changes the state of S and does not take S to a state s such that $V \cap s \in K$ (this would indicate that A has been awarely in that state of S before). Again, A can be confident that the assignment is possible iff $\forall\sigma_{V,T}\mathtt{w}(p, A)$ is true.

Assignment to $p \notin V$ increases the knowledge of A of the current state of S by the variable p. It is safe with respect to the goal (Φ, ψ) only if the formulas from Φ do not depend on p for their truth values, because overwriting a variable whose value has not been sampled in advance can destroy information on the initial values of these formulas. A must observe similar safety with respect to the atomic goals of the form $\overline{\varphi}$ from ψ. Let us use the formula

$$G \rightleftharpoons \bigwedge_{i=1}^{l}([\bot/p]\varphi_i \Leftrightarrow [\top/p]\varphi_i)$$

to express that the truth values of the formulas $\varphi_1, \ldots, \varphi_l$ from Φ do not depend on p, whose so far unchanged value can be lost upon the assignment. Then we have the following axioms about assigning variables $p \notin V$:

$$\bigvee_{p\notin V} A^{V\cup\{p\},T\cup\{p\},\{T\cup\{p\}\}}(\Phi, \overline{G}\wedge[\overline{[\bot/p]\varphi \wedge [\top/p]\varphi}/\overline{\varphi} : \varphi \in \mathbf{L}(P)]\psi)\wedge\forall\sigma_{V,T}\mathtt{w}(p, A) \Rightarrow A^{V,T,K}(\Phi, \psi) \quad (10)$$

$$\bigvee_{p\notin V} A^{V\cup\{p\},T,\{T\}}(\Phi, \overline{G} \wedge [\overline{[\bot/p]\varphi \wedge [\top/p]\varphi}/\overline{\varphi} : \varphi \in \mathbf{L}(P)]\psi) \wedge \forall\sigma_{V,T}\mathtt{w}(p, A) \Rightarrow A^{V,T,K}(\Phi, \psi) \quad (11)$$

Assignment to $p \in V$ changes the knowledge of A of the current state, because the known value of p is changed. Since we assume that variables with known values do not occur in Φ, nor in atomic goals of the form $\overline{\varphi}$ in ψ, we have the following axioms about such assignments:

$$\bigvee_{p\in V} A^{V,T\setminus\{p\},K\cup\{T\setminus\{p\}\}}(\Phi, \psi) \wedge \forall\sigma_{V,T}\mathtt{w}(p, A) \Rightarrow A^{V,T,K}(\Phi, \psi), \text{ for } T \setminus \{p\} \notin K \quad (12)$$

$$\bigvee_{p\in V} A^{V,T\cup\{p\},K\cup\{T\cup\{p\}\}}(\Phi, \psi) \wedge \forall\sigma_{V,T}\mathtt{w}(p, A) \Rightarrow A^{V,T,K}(\Phi, \psi), \text{ for } T \cup \{p\} \notin K \quad (13)$$

Finally, A might be able to recognise that its goal (Φ, ψ) has been achieved. The following axiom states that if φ or its negation is a tautology, then reading its value can be regarded as achieved:

$$(\forall\varphi_i \vee \forall\neg\varphi_i) \wedge A^{V,T,K}(\langle\varphi_1, \ldots, \varphi_{i-1}, \varphi_{i+1}, \ldots, \varphi_l\rangle, \psi) \Rightarrow A^{V,T,K}(\langle\varphi_1, \ldots, \varphi_l\rangle, \psi) \quad (14)$$

Successive simplifications of the formulas from Φ which are obtained upon sampling variables p and applying the substitutions $[\top/p]$ and $[\bot/p]$ occurring in the corresponding axiom 9 should lead A to goals where Φ consists of such tautological formulas, if A can achieve its original goal at all. To realise whether the current state of S is related to its initial state as prescribed by ψ, A should be able to evaluate the formula

$$[\varphi/\overline{\varphi}', \varphi/\overline{\varphi} : \varphi \in \mathbf{L}(P)]\psi.$$

Note that this formula is in $\mathbf{L}(P)$, while ψ has the syntax (5). If applying $\sigma_{V,T}$ to this formula produces a tautology, then A can conclude that ψ has been achieved. Again, successive simplifications of the subgoals of the form $\overline{\varphi}$ in ψ are relied on to enable A to reach a state of S where this holds. We have the axiom:

$$\forall\sigma_{V,T}[\varphi/\overline{\varphi}', \varphi/\overline{\varphi} : \varphi \in \mathbf{L}(P)]\psi \Rightarrow A^{V,T,K}(\langle\rangle, \psi) \quad (15)$$

Note that in each of the axioms above the goal expressions on the left of the main $\Rightarrow$ are simpler than those on the right of the main $\Rightarrow$. This becomes clear if we define an ordering $<$ on the superscripts of goal expressions by putting $\langle V, K\rangle < \langle V', K'\rangle$ iff either $V \supset V'$ or $V = V'$ and $K \supset K'$ and notice that superscripts occurring on the left of $\Rightarrow$ in our axioms are smaller than superscripts occurring on the right. Furthermore, each of the possible cases for appearing in the syntax (8) of programs α has a corresponding axiom which describes the conditions in which an α of the respective form can make its next step and the way this step affects the goal: axioms (14) and (15) describe the cases when A can achieve its goal by doing nothing (`skip`); axiom (9) covers the case when a program with a conditional main statement can do, and axioms (10), (11), (12) and (13) are about programs starting with the four possible forms of assignment, depending on the value assigned, and whether the variable assigned has been read in advance. That is why, if $\mathcal{A}$ denotes the conjunction of the formulas on the left of $\Rightarrow$ in (9)-(15), then

$$A^{V,T,K}(\langle\varphi_1, \ldots, \varphi_l\rangle, \psi) \Leftrightarrow \mathcal{A} \tag{16}$$

defines $A^{V,T,K}(\langle\varphi_1, \ldots, \varphi_l\rangle, \psi)$ by induction on V, K and the length l of the list of formulas to read. The axioms (9)-(15) can used to write the Horn clauses of a logic program to obtain a truth value for $A^{V,T,K}(\Phi, \psi)$ and, if this expression turns out to be true, to synthesize the respective α. Indeed, we have done this [8].

The axioms above, except (15) apply without change to the case which includes subgoals of the form $B(\Phi', \psi')$ (where $B \subset \Sigma$ need not be the same as A.) To include such subgoals, we use that every formula ψ of the syntax (6) with such subgoals has an equivalent in disjunctive normal form of the form $\psi_1 \vee \psi_2$ where all the elementary conjunctions in ψ_1 and none of the elementary conjunctions ψ_2 have occurrences of subgoals of the form $B(\Phi', \psi')$. Then A can achieve (Φ, ψ) if either A achieves (Φ, ψ_2), which means that A should read all the formulas from Φ itself, or if A achieves $(\langle\rangle, \psi_1)$ which includes enabling some other coalitions mentioned in the subgoals of ψ_1 of the form $B(\Phi', \psi')$ to continue and achieve $(\Phi' * \Phi, \psi')$ this way finishing the job of A. Let σ denote the substitution $\sigma_{V,T}[\varphi/\overline{\varphi}', \varphi/\overline{\varphi} : \varphi \in \mathbf{L}(P)]$ for the sake of brevity. Then we have the following axiom:

$$\left(\begin{array}{l} \forall[B^{V,T,\{T\}}(\Phi' * \Phi, \psi')/B(\Phi', \psi') : B \subseteq \Sigma, \Phi' \in (\mathbf{L}(P))^*, \psi \in \mathbf{G}(P)]\sigma\psi_1 \vee \\ \forall\sigma\psi_2 \wedge \left(\bigwedge_{i=1}^{l} \forall\varphi_i \vee \forall\neg\varphi_i\right) \end{array}\right) \Rightarrow A^{V,T,K}(\langle\rangle, \psi_1 \vee \psi_2) \tag{17}$$

where $\mathbf{G}(P)$ denotes the set of the goal formulas with the syntax (6) based on the vocabulary P. This axiom subsumes axioms (15) and (14).

Alternatively, axioms (9)-(15) can be used to calculate $A^{V,T,\{T\}}(\langle\varphi_1, \ldots, \varphi_l\rangle, \psi)$ by model-checking a formula in the propositional μ-calculus (see e.g. [6]). Assume there are no subgoals of the form $B(\Phi', \psi')$ in ψ for the sake of simplicity. Consider a system, whose state space is the set of quadruples V_0, T_0, V, T such that $T_0 \subseteq V_0 \subseteq V \subseteq P$ and $T \subseteq V$, where P is the vocabulary of a fixed access control system as above. A quadruple V_0, T_0, V, T represents a state of knowledge of A which consists of the fact $V_0 \cap s_0 = T_0$ about the initial state s_0 of S and the fact $V \cap s = T$ about the current state s of S. The meaning of V, T is like in (9)-(15). Hence the quadruple V_0, T_0, V, T represents A's knowledge of both the initial and the current state. (The addition V_0, T_0 is not needed in these axioms, because they prescribe to simplify φ in subgoals of the form $\overline{\varphi}$ and in formulas to read from Φ immediately each time the value of a variable becomes known.)

Consider the μ-calculus language with the modalities $\langle\texttt{sample}\ p\rangle$, $\langle p{:=}\bot\rangle$ and $\langle p{:=}\top\rangle$ for each $p \in P$. Let the corresponding accessibility relations $R_{\texttt{sample}\ p}$, $R_{p:=\bot}$ and $R_{p:=\top}$ be defined by the clauses

$$R_{\texttt{sample}\ p}(V_0, T_0, V, T; V_0', T_0', V', T') \leftrightarrow \left(\begin{array}{c} \forall\sigma_{V,T}\mathtt{r}(p, A) \wedge p \notin V \wedge V_0' = V_0 \cup \{p\} \wedge V' = V \cup \{p\} \wedge \\ \left(\begin{array}{l} T_0' = T_0 \setminus \{p\} \wedge T' = T \setminus \{p\} \vee \\ T_0' = T_0 \cup \{p\} \wedge T' = T \cup \{p\} \end{array}\right) \end{array}\right)$$

$$R_{p:=\bot}(V_0,T_0,V,T;V_0',T_0',V',T') \leftrightarrow \forall\sigma_{V,T}\mathtt{w}(p,A) \wedge V_0' = V_0 \wedge T_0' = T_0 \wedge V' = V \cup \{p\} \wedge T' = T \setminus \{p\}$$

$$R_{p:=\top}(V_0,T_0,V,T;V_0',T_0',V',T') \leftrightarrow \forall\sigma_{V,T}\mathtt{w}(p,A) \wedge V_0' = V_0 \wedge T_0' = T_0 \wedge V' = V \cup \{p\} \wedge T' = T \cup \{p\}$$

Each of these accessibility relations represents an action on behalf of A in which A either increases its knowledge or both increases its knowledge and changes the current state. The knowledge of A is sufficient to establish that its goal is already achieved iff the formula

$$\forall[\sigma_{V_0,T_0}\varphi/\overline{\varphi},\sigma_{V,T}\varphi/\overline{\varphi}' : \varphi \in \mathbf{L}(P)]\psi \wedge \bigwedge_{i=1}^{l}(\forall\sigma_{V_0,T_0}\varphi_i \vee \forall\sigma_{V_0,T_0}\neg\varphi_i)$$

is valid. Let the set of states from which A can reach a satisfactory state be $[\![X]\!]$. Then the following implications hold:

$$\langle p{:=}\bot\rangle X \Rightarrow X, \quad \langle p{:=}\top\rangle X \Rightarrow X, \quad \langle \mathtt{sample}\ p\rangle\top \wedge [\mathtt{sample}\ p]X \Rightarrow X, \quad p \in P \tag{18}$$

The $[\cdot]$ in the last formula means that A should be prepared for any outcome of the sampling. The least solution of the system of inclusions (18) is the set of the knowledge states in which A can make a plan to reach a satisfactory state without fail. Hence $A^{V,T,\{T\}}(\langle\varphi_1,\ldots,\varphi_l\rangle,\psi)$ is equivalent to the satisfaction of

$$\mu X.\left(\begin{array}{c}\left(\forall[\sigma_{V_0,T_0}\varphi/\overline{\varphi},\sigma_{V,T}\varphi/\overline{\varphi}' : \varphi \in \mathbf{L}(P)]\psi \wedge \bigwedge\limits_{i=1}^{l}(\forall\sigma_{V_0,T_0}\varphi_i \vee \forall\sigma_{V_0,T_0}\neg\varphi_i)\right) \vee \\ \bigvee\limits_{p\in P}((\langle\mathtt{sample}\ p\rangle\top \wedge [\mathtt{sample}\ p]X) \vee \langle p{:=}\bot\rangle X \vee \langle p{:=}\top\rangle X)\end{array}\right)$$

at state $\emptyset,\emptyset,V,T$.