Algebra and logic for access control

Matthew Collinson, David Pym

HP Laboratories

HPL-2008-75R1

Keyword(s):
Access control, process algebra, bunched Logic; systems modelling

Abstract:

The access control problem in computer security is fundamentally concerned with the ability of system entities to see, make use of, or alter various system resources. As such, many access control situations are essentially problems of concurrency. We give an account of fundamental situations in access-control in distributed systems using a resource-based process calculus and a hybrid of Hennessy-Milner and resource logic. This yields a consistent account of operational behaviour and logical reasoning for access control, that includes an analysis of co-signing, roles and chains-of-trust.

External Posting Date: May 21, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: May 21, 2009 [Fulltext]

Algebra and logic for access control

Matthew Collinson and David Pym
HP Labs, Filton Road, Stoke Gifford, Bristol BS34 8QZ, U.K.
Email: matthew.collinson@hp.com; david.pym@hp.com

April 15, 2009

Abstract

The access control problem in computer security is fundamentally concerned with the ability of system entities to see, make use of, or alter various system resources. As such, many access control situations are essentially problems of concurrency. We give an account of fundamental situations in access-control in distributed systems using a resource-based process calculus and a hybrid of Hennessy-Milner and resource logic. This yields a consistent account of operational behaviour and logical reasoning for access control, that includes an analysis of co-signing, roles and chains-of-trust.

1 Introduction

Access control is one of the fundamental issues in information security.

In computer systems of almost all levels of scale, certain behaviours will be desirable and certain others undesirable. A great many of these behaviours involve the ability of entities (users, programs, processes) in the system to access some other entity (resource, program, process). The denial of undesirable, and the permission of desirable behaviours is thus reduced to the *access control problem*.

Access control is, of course, one of the main issues in computer security, with work in the area extending back over many years (for example by Lampson [23] and by Saltzer and Schroeder [43]) and gaining ever greater prominence until the present day. One strand of work in this area concerns the development of logical languages, sometimes called *security languages*, for reasoning about and making access control decisions, as exemplified in work by Abadi et al. [1, 3, 25] and DeTreville [16].

Important challenges for logical security languages are the representation of *co-signing*, of *roles* and of *chains-of-trust*. The security language in [3] introduces novel connectives for co-sigining and roles and this is sufficient to allow formal inferences to be made, in particular for chains-of-trust. However, the mathematical semantics of such languages, where they exist, are not transparently related to operational behaviour. Furthermore, there is a hint in [3] that such connectives correspond to concurrent behaviour — as indeed they must.

The lack of a suitable semantics means that, given an existing system in which we care about access control, it is difficult to see how the existing security languages can be used to capture access control behaviour *in a provably sound way*. Of course, by design, access-control systems can be implemented which closely conform with policies and protocols embodied in security languages.

In this paper we show that process calculus can be used to give a semantics for a security language, thus giving a meaningful account of (suitably defined) connectives for security languages, including both co-signing and roles. Chain-of-trust arguments arise naturally from the underlying semantics.

The modelling framework we propose is based on resources, processes, and modal bunched logic developed by Pym, Tofts and Collinson [38, 39, 15, 14]. The present paper presents an application of that earlier work to give an account of security languages and security-related problems. The basic idea is that resources R and processes, in the sense of (synchronous) process algebra, E co-evolve,

$$R, E \xrightarrow{a} R', E',$$

according to the specification of a partial function, $\mu : (a, R) \mapsto R'$, that determines how an action a evolves E to E' and R to R'. The base case of the operational semantics is given by action prefix:

$$\frac{}{R, a : E \xrightarrow{a} R', E} \qquad (\mu(a, R) = R') \ .$$

The theory of this calculus of resources and processes (**SCRP**) has been explored in detail in [38, 39, 15] and [14], but a brief review of the process definitions is included.

In the security literature, the entitites which act within systems are often referred to as *principals*. A key step for us is the representation of principals as processes. A similar approach has been taken by a number of authors, particularly regarding security protocols, see for example [5, 44, 45].

The **SCRP** calculus comes with a *Hennessy-Milner (modal) logic* [19] called **MBI** for the specification and verification of properties of systems (resource-process states). **MBI** is simultaneously a *resource logic* in the sense of bunched logic, **BI**, and its cousin Separation Logic [21, 32, 40].

A key aspect of **SCRP** and **MBI** is the relationship between concurrent composition and multiplicative ('separating') conjunction. This exploits an underlying resource semantics, based on that of BI, in which resources carry monoidal structure:

$$R, E \models \phi_1 * \phi_2 \qquad \text{iff} \qquad R_1, E_1 \models \phi_1 \quad \text{and} \quad R_2, E_2 \models \phi_2$$

for some R_1, R_2 and E_1, E_2 such that $R = R_1 \circ R_2$ and $R, E \approx R, E_1 \times E_2$.

In our setting, the multiplicative conjunction can be used, for example, to describe a co-signing requirement for resource access. The principals who must co-sign are E_1 and E_2. The resources R_1 and R_2 represent, respectively, E_1 and E_2's separate access rights, together with the shared resource to be accessed. The composite resource $R_1 \circ R_2$ represents the appropriate combination of access rights and shared resource. Each ϕ_i can be used as (a proxy for) some certificate that each E_i holds (in order, say, to sign). A more detailed illustration of this is found in Example 3 in Section 6.

In order to describe the roles of principals of the form 'E in the role of F', we introduce an additional binary process constructor, $\propto$, into SCRP. Thus, a resource-process state

$$R, E \propto F.$$

represents a principal E in the role F, toegether with resources R. Note that the role F is itself represented by a process and that it is intended to have fewer abilities than E. Along with this construct comes a logical modality, given by the following forcing definition:

$$R, G \models \{E\}\phi \quad \text{iff} \quad \exists\ F \text{ s.t. } R, G \approx R, E \propto F \text{ and } F \lessapprox E \text{ and } R, F \models \phi.$$

Here $\lessapprox$ (respectively $\approx$) is the notion of simulation (respectively bisimulation). Thus a logical assertion $\{E\}\phi$, read '*E says* ϕ', is used to describe properties that may hold of some role of a process, but not of the process itself: for example, often the reduced 'user' role of some 'administrator' process has additional safety properties. Example 6 in Section 6 is of this kind.

The intention of this work follows the tradition of using process calculus as a modelling tool. This paper is intended to serve as a foundation for the modelling of certain existing security situations. In practice, this will take place in the Demos2k tool (`http://www.demos2k.org`), and a variant thereof *LD2k* [11, 13], which is particularly tailored towards event-modelling and performance analysis in distributed systems, and which is closely related to the calculus we have presented here.

Section 2 contains a brief review of SCRP. Section 3 describes how this is modified to deal with compound principals that include roles. Section 5 gives the associated modal resource logic and some basic results. Section 6 gives a range of examples:

- First, a basic example in which access is performed by a resource guard on behalf of a general agent;
- Second, a similar example, in which a guard authorizes an agent to access a resource;
- Third, joint-access request, in which two agents must both request access, and in so doing must combine their permission resources;
- Fourth, exclusive access, a variant of joint access, in which two agents may mutually exclusively access a resource;
- Fifth, authorization by delegation, in which a guard must consult a second authority with which resides the access control list, so establishing a chain of trust;
- Sixth, reduction to role, in which we have an agent together with a role for that agent which has reduced access rights;
- Last, modelling access with assertion-based control, in which access control decisions are based on a logical language rather than just ACLs (cf. Binder [1]), giving another example of a chain of trust, formed by the trust agents have in others' public statements.

Section 7 describes changes that must be made to the set-up when additional logical power is required in the specification language. Finally, Section 8 contains a discussion of open problems and further directions that we are pursuing.

2 Resources and Processes

Many access control systems live in an environment in which significant events occur simultaneously. Moreover, events of the access system itself may occur concurrently and there may be complex interactions between all parts of the system and the environment. A modelling framework that describes such systems and their environment must be able to capture concurrency in a natural way.

Process calculus, like Milner's CCS [29, 31], or the more general synchronous calculus SCCS [30], is an elegant methodology for dealing with such situations. It provides a precise framework for the construction of models. In particular, it has the important property of *compositionality*: the description of a large system is constructed from those of component subsystems.

SCRP is a form of synchronous process calculus. In contrast to standard process calculi, it has an explicit treatment of resource. **SCRP** was introduced in the papers [38, 39, 15] and later refined in [14]. The calculus presented in this paper is a closely related variant of those calculi.

In this paper, we often use partial functions, writing $exp \downarrow$ and $exp \uparrow$ to mean that an expression exp is, respectively, defined or undefined. We also make use of Kleene equality between expressions: the left-hand side of an equality, $lexp \simeq rexp$, is defined if and only if the right-hand side is defined, and when defined they are equal.

Mild constraints are placed upon the type of resource treated. A *resource monoid*, modelling the composition and comparison of resource elements [33, 36, 37], is a structure

$$\mathbf{R} = (\mathbf{R}, \circ, \mathbf{e}, \sqsubseteq) \ .$$

We do not use a separate notation to distinguish the carrier set $\mathbf{R}$ from the structure. We reserve the letters R, S, T, U, V for resources. The structure has a preorder $\sqsubseteq$, a partial, binary composition $\circ$, and has a distinguished element $\mathbf{e}$. The operation $\circ$ satisfies monoid associativity and commutativity axioms up to Kleene equality. The unit of $\circ$ is $\mathbf{e}$. Composition with this unit is always defined. Therefore, the structure satisfies the unit axiom for a commutative monoid up to actual equality. Resource monoids are further required to satisfy the *bifunctoriality condition*:

$$\text{if } R \sqsubseteq R' \text{ and } S \sqsubseteq S' \text{ and } R' \circ S' \downarrow \text{ then } R \circ S \downarrow \text{ and } R \circ S \sqsubseteq R' \circ S'$$

for all R, R', S, S' in $\mathbf{R}$. For the purposes of this paper, the preorder $\sqsubseteq$ is always taken to be the equality relation.

We assume a commutative monoid, Act, of *actions*. Just as in standard process algebra, these actions correspond to the events of a system. We reserve the letters $a, b, c, \ldots$ for actions. Composition is written by juxtaposition and the unit action is written 1. For the purposes of this paper we assume that the action monoid is generated freely from *atoms*, for which we reserve the letter α.

Assume a (partial) function, called a *modification*, $\mu : \mathsf{Act} \times \mathbf{R} \longrightarrow \mathbf{R}$, satisfying two *coherence* conditions:

1. $\mu(1, R) = R$ for all $R \in \mathbf{R}$;
2. if $\mu(a, R)$, $\mu(b, S)$ and $R \circ S$ are all defined then the Kleene equality $\mu(ab, R \circ S) \simeq \mu(a, R) \circ \mu(b, S)$ holds.

Define a total operation called *hiding*, that takes any resource R and any action a and produces an action $\nu R.a$. Any action a may be written uniquely (up to re-ordering) as a product $a = \prod\{\alpha_i \mid i \in I\}$ for some finite set I. Then we may take

$$\nu R.a = \prod\{\alpha_i \mid i \in I \;\&\; \mu(\alpha_i, R) \uparrow\} \;. \tag{1}$$

Recall that the product of an empty set of actions gives the identity action.

There are six basic forms of process in **SCRP**: zero, prefix, sum, product, hiding, constant. The letters A, B, C, D, E, F, G, H are reserved for processes. A *state* consists of a resource and a process. *Operational behaviour* is given by transitions (binary relations) labelled by actions on the set of states. We detail the forms of process and their operational behaviour within states below. The definition constitutes a structural operational semantics [34].

The *zero* process, $\mathbf{0}$. A state with $\mathbf{0}$ as its process component makes no state transitions.

A *prefix* process is of the form $a : E$ where E is any process and a is any action. The operational rule for this is

$$\frac{}{R, a : E \xrightarrow{a} \mu(a, R), E} \quad (\mu(a, R) \downarrow)$$

where R is any resource. When $\mu(a, R)$ is defined we say that a is *enabled at* R.

A *sum* is of the form $\sum_{i \in I} E_i$, where I is an arbitrary index set and each E_i is a process. We often use the infix notation $E + F$ when the cardinality of the index set is 2. The rule

$$\frac{R, E_i \xrightarrow{a} R', E_i'}{R, \sum_{i \in I} E_i \xrightarrow{a} R', E_i'}$$

gives the operational behaviour for sums.

A *(synchronous) product* is of the form $E \times F$, where E and F are processes. The rule

$$\frac{R, E \xrightarrow{a} R', E' \qquad S, F \xrightarrow{b} S', F'}{R \circ S, E \times F \xrightarrow{ab} R' \circ S', E' \times F'} \quad (R \circ S \downarrow)$$

describes the evolution of states formed from product processes. The idea is that the two component processes should bring together their resources in order to agree a simultaneous step forward.

A *hiding* (or just *hide*) is of the form $\nu R.E$ where E is a process and R is a resource. The operational rule is

$$\frac{R \circ S, E \xrightarrow{a} R' \circ S', E'}{R, \nu S.E \xrightarrow{\nu S.a} R', \nu S'.E'} \quad (\mu(\nu S.a, R) = R') \;.$$

The idea of hiding is that the process $\nu S.E$ carries private, local resource S that is hidden from external processes, and that it may use this resource to evolve into a new process $\nu S'.E'$. Furthermore, the action $\nu S.a$ through which this happens does not exhibit the atomic actions enabled by S.

A *constant* is defined through the use of a recursive definition. They are a way of introducing recursive process terms into the calculus. An alternative is to use fixed points, as described in [38, 39, 15, 14]. A recursive definition takes the form

$$\begin{array}{ccc} X_1 : & = & E_1 \\ \vdots & & \vdots \\ X_n : & = & E_n \end{array}$$

for some integer n, where each X_i is a *process variable* and each E_i may contain any of the $X_1, \ldots X_n$ but no other process variables. We usually write a tuple of processes as E and the ith component as E_i. Each of these systems of simultaneous equations uniquely specifies a canonical (least) solution, that is, a sequence of processes C satisfying the equations. Each C_i is a *constant.* Suppose that each C_i is C_i. The operational rule for such a constant C_i is

$$\frac{R, E_i[\mathsf{E}/\mathsf{C}] \xrightarrow{a} R', G}{R, C_i \xrightarrow{a} R', G},$$

where E are the defining expressions for the sequence C. For example, the special process $\mathbf{1}$, that can only tick, is defined by the equation $X := 1 : X$. We usually abbreviate definition by constants by writing the names of the constants rather than variables, so for example $\mathbf{1} := 1 : \mathbf{1}$.

3 An Extended Calculus for Principals

In this section, we present a process calculus **ACCRP** which is tailored to describing systems formed from principals. Thus the kinds of principals which arise in access contol problems in computer security are to be described as process terms in **SCRP**. Our approach has some common ground with work that uses process calculus for the formal analysis of security protocols [5, 41, 45], but this will not be our main concern.

The tailoring of the calculus reflects the fact that there are certain compound principals which occur time-and-again in the security literature. In particular, there are *conjunctions* of principals, which may perform some access just if both principals do so together — we use the synchronous product for such processes. There are also principals which are formed by adopting *roles* with fewer capabilities. We introduce a dedicated new connective $\propto$ for roles. Further discussions of compound principals may be found in the papers by Abadi et al. [1, 3, 25] which have strongly influenced the present work.

To form the calculus **ACCRP** the grammar of **SCRP** is extended with the role constructor, so that

$$E ::= \ldots \mid E \propto E \ .$$

The operational rule for the constructor introducing a principal 'E in the role of F', where F is a principal with reduced capacity (at a reduced resource), is

$$\frac{R, F \xrightarrow{a} R', F' \quad R \circ S, E \xrightarrow{a} R' \circ S', E'}{R \circ S, E \propto F \xrightarrow{a} R' \circ S', E' \propto F'} \quad (F \lessapprox E) \tag{2}$$

where $\lessapprox$ is defined below. An example of the use of this constructor is given in Example 6 in Section 6.

We define notions of equivalence and inequivalence for states and processes. Define

$$R, E \lessapprox S, F \tag{3}$$

if $R = S$ and, whenever $R, E \xrightarrow{a} R', E'$, there is some F' with $R, F \xrightarrow{a} R', F'$ and $R', E' \lessapprox R', F'$. In such circumstances, we say that S, F *simulates* R, E.

Write $R, E \approx R, F$ iff $R, E \lessapprox R, F$ and $R, F \lessapprox R, E$ both hold. and say that R, E is *(locally) bisimilar* to R, F. We write $E \lessapprox F$ or $E \approx F$, when, respectively, $R, E \lessapprox R, F$ or $R, E \approx R, F$ for all resources R.

We note that $\lessapprox$ is defined by mutual recursion with processes since the role constructor uses $\lessapprox$ as a side-condition. This is somewhat unusual for a process calculus. In order to make this work, we do not allow process variables to occur inside role constructors (note also that they are not required conceptually). That is, we restrict the form of the recursive equations used to define constants.

Introduce a syntactic complexity measure, $h(E)$, of the height of the tower of $\propto$ connectives used to define each process E. Take $h(E) = 0$ for $E = \mathbf{0}$ or any process variable. We take $h(a : E) = h(E)$ and $h(E + F) = h(E \times F) = \max\{h(E), h(F)\}$ and $h(\mathsf{C}_i) = \max\{h(E_1), \ldots, h(E_n)\}$, where $E_1, \ldots E_n$ are the components of E defining the constants $\mathsf{C} := \mathsf{E}$. Finally, take $h(E \propto F) = \max\{h(E), h(F)\} + 1$. Notice that the processes in the side-conditions of the role rule have lower complexity than the role that is the source of the transition.

Transitions do not increase process height: the proof is the evident induction on derivations.

Lemma 1. If $R, E \xrightarrow{a} R', E'$, then $h(E') \leq h(E)$,

Therefore the definition of simulation can be re-stated in a stratified way. Thus, to show $R, E \lessapprox R, F$ we need only compare transitions into processes into the same or lower strata. Hence the mutual recursion between transitions and simulation is well-defined.

4 Algebraic and Dynamical Properties

A number of simple properties of **ACCRP**-systems hold. As these systems are defined through the use of structural operational semantics, a critical proof technique is the use of induction on the structure of derivations of state-transitions. The first property to observe is that the evolution of resource is completely determined by the choice of action.

Lemma 2. If $R, E \xrightarrow{a} R', E'$ then $R' = \mu(a, R)$.

Proof. Induction on the structure of derivations. The base case is where E is a prefix process; then, for any R, R', E' as in the statement of the lemma, we have $R' = \mu(a, R)$. The induction hypothesis is that all shorter derivations satisfy the statement of the theorem. Take, for example, the role definition in rule (2) above. The induction hypothesis gives $R' \circ S' = \mu(a, R \circ S)$ using the right-hand premise, but this is precisely the required property for the conclusion. We omit the other cases, as they are equally straightforward, but note that the product case relies upon the second coherence condition on modifications, and that the hiding case uses the side-condition on the hiding rule. □

The local bisimulation relation is an equivalence.

Proposition 1. The relation $\approx$ on processes is an equivalence relation, and, for all E, F, G:

1. if $E \approx F$ then $E \propto G \approx F \propto G$.

Proof. The proof of all parts of this follow from the definition of $\approx$ and by applying the standard methods for bisimulation relations. □

Processes satisfy a number of other equalities and inequalities, including the following:

Proposition 2. The constructor $\propto$ has the following properties with respect to bisimulation:

1. $E \propto F \lessapprox E$;

2. $E \propto E \approx E$;

3. If $E \lessapprox F$ then $E \propto G \lessapprox F \propto G$;

Proof. Again, the proofs of these points are direct uses of the definition of simulation. □

The above results tells us that we have a system that formally reconstructs the following natural properties of roles: any agent acting in one of its roles is less powerful (has fewer or equal capabilitites) than the original agent; an agent E in the role E is as powerful as the agent E; if E is less powerful than F then every role of E is less powerful than the corresponding role of F.

5 Logic

In Hennessy-Milner logic [19, 46] a forcing relation is used to relate CCS processes to assertions of their properties, with the judgement $E \models \phi$ being read as 'process E has property ϕ'. The language of propositions typically contains the classical propositional connectives, $\wedge$, $\vee$, and $\neg$, together with the classical action modalities $\langle a \rangle$ and $[a]$. In our setting, the corresponding judgement, $R, E \models \phi$, says that property ϕ holds of process E in the presence of resources R; that is, of the system model R, E.

In our synchronous setting, we are able to provide an analysis of various structural aspects of processes. In particular, we obtain essentially the following logical characterization of the synchronous product: $R, E \models \phi_1 * \phi_2$ iff $R_1, E_1 \models \phi_1$ and $R_2, E_2 \models \phi_2$ for some R_1, R_2 and E_1, E_2 such that $R = R_1 \circ R_2$ and $R, E \approx R, E_1 \times E_2$. This characterization stands in contrast to the situation for CCS [46], in which $E \mid F \models \phi$ iff $E \models \phi / F$ where the definition of ϕ / F involves 'distributing the process though the formula'. We also obtain a characterization of hiding in terms of the multiplicative existential quantifier (see below), which exploits also the presence of resources in the forcing judgement. Both of these structural characterizations are exploited in the access-control examples presented in Section 6.

In general, the logic MBI, introduced in [38, 39, 15], admits a range of connectives and quantifiers, including multiplicative modalities. For the present paper, however, we introduce a logic **MBIa** for reasoning about properties of **ACCRP** systems from which we omit, for technical reasons, the multiplicative implication, $-\!\!*$, and the multiplicative modalities. This logic is able to express some interesting aspects of access control properties.

Assume a countable set, ActVar of *action variables*, ranged over by x, and a constant symbol a for each action a of **ACCRP**. Let $\mathsf{A} = \mathsf{ActVar} \cup \mathsf{Act}$ and let a range over this set. We assume a given set of *relation* symbols on actions, each with a given arity. Atomic formulae φ consist of all instances of relations, that is, if p is a relation symbol of arity n and $\mathsf{a}_1, \ldots, \mathsf{a}_n \in \mathsf{A}$, then $p(\mathsf{a}_1, \ldots, \mathsf{a}_n)$ is an atomic formula.

The formulae of the language **MBIa** are defined by the grammar

$$\begin{aligned} \phi := \quad & \bot \mid \top \mid \varphi \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \to \phi \mid I \mid \phi * \phi \\ & \mid \langle \mathsf{a} \rangle \phi \mid [\mathsf{a}]\phi \mid \{E\}\phi \mid \exists x.\phi \mid \exists_\nu x.\phi \mid \forall x.\phi \mid \forall_\nu x.\phi \ , \end{aligned}$$

for $\mathsf{a} \in \mathsf{A}$ and processes E. The connectives $\top$, $\neg$, $\vee$, $\wedge$ and $\to$ are the connectives true, negation, disjunction, conjunction and implication of classical logic. The connectives $\langle \mathsf{a} \rangle \phi$ and $[\mathsf{a}]\phi$ are classical modal connectives, intended to express properties that hold after, respectively, some or any, instance of a from a state. The connectives I and $*$ are known as the *unit* and *multiplicative conjunction*, often pronounced *star*. The connective $\exists$ is classical existential quantification. The connective $\exists_\nu$ is the *multiplicative existential* quantifier. The modality $\{E\}\phi$ is read E *says* ϕ, and is intended to express the fact that a process may indirectly witness a fact through the use of a role of E. The *sentences* are just the formulae without free variables. For any formula ϕ, let $\phi[a_1/x_n, \ldots, a_n/x_n]$ be the formula formed by replacing each occurrence of each variable x_i by the term a_i. More generally, one may want to allow function symbols on actions, compound action terms, equalities between such terms and further logical operators.

A valuation $\mathcal{V}$ for the language above is fixed by choosing an $(n+1)$-ary relation $\mathcal{V}(p)$ (taking n actions and one state) for each relation symbol of arity n. Each set $\mathcal{V}(p)$ must be closed under the relation $\approx$. An *assignment*, η, is a function from ActVar to Act. For any η, let $\eta[a/x]$ be the

$$R, G, \eta \vDash \varphi(x_1, \dots x_n) \text{ iff } (\eta(x_1), \dots, \eta(x_n), R, G) \in \mathcal{V}(\varphi)$$

$$R, E, \eta \vDash \bot \text{ never}$$

$$R, E, \eta \vDash \top \text{ always}$$

$$R, E, \eta \vDash I \text{ iff } R = \mathbf{e} \text{ and } R, E \approx R, \mathbf{1}$$

$$R, E, \eta \vDash \neg\phi \text{ iff } R, E, \eta \vDash \phi \text{ does not hold}$$

$$R, E, \eta \vDash \phi \wedge \psi \text{ iff } R, E, \eta \vDash \phi \text{ and } R, E, \eta \vDash \psi$$

$$R, E, \eta \vDash \phi \vee \psi \text{ iff } R, E, \eta \vDash \phi \text{ or } R, E, \eta \vDash \psi$$

$$R, E, \eta \vDash \phi \to \psi \text{ iff } R, E, \eta \vDash \phi \text{ implies } R, E, \eta \vDash \psi$$

$$R, E, \eta \vDash \phi_1 * \phi_2 \text{ iff } \exists R_1, R_2, E_1, E_2.\ R = R_1 \circ R_2\ ,\ R, E \approx R, E_1 \times E_2\ ,\ R_1, E_1, \eta \vDash \phi_1\ ,\ R_2, E_2, \eta \vDash \phi_2$$

$$R, E, \eta \vDash [\mathsf{a}]\phi \text{ iff } \forall R', E'.\ R, E \xrightarrow{\mathsf{a}} R', E' \text{ implies } R', E', \eta \vDash \phi$$

$$R, E, \eta \vDash \langle\mathsf{a}\rangle\phi \text{ iff } \exists R', E'.\ R, E \xrightarrow{\mathsf{a}} R', E' \text{ and } R', E', \eta \vDash \phi$$

$$R, G, \eta \vDash \{E\}\phi \text{ iff } \exists F.\ R, G \approx R, E \propto F \text{ and } F \lessapprox E \text{ and } R, F, \eta \vDash \phi$$

$$R, E, \eta \vDash \exists x.\phi \text{ iff } \exists a.\ R, E, \eta \vDash \phi[a/x]$$

$$R, E, \eta \vDash \forall x.\phi \text{ iff } \forall a.\ R, E, \eta \vDash \phi[a/x]$$

$$R, E, \eta \vDash \exists_\nu x.\phi \text{ iff } \exists S, F, a.\ R \circ S\downarrow \text{ and } \mu(a, S)\downarrow \text{ and } R, E \approx R, \nu S.F \text{ and } R \circ S, F, \eta \vDash \phi[a/x]$$

$$R, E, \eta \vDash \forall_\nu x.\phi \text{ iff } \forall S, F, a.\ R \circ S\downarrow \text{ and } \mu(a, S)\downarrow \text{ and } R, E \approx R, \nu S.F \text{ implies } R \circ S, F, \eta \vDash \phi[a/x]$$

Figure 1: Interpretation of Logical Formulae

asssignment that is identical η, except that $\eta(x) = a$. A valuation is extended to an interpretation of formulae by means of a forcing relation $\vDash$, as in Figure 1.

All of the clauses, except for $\{E\}$, in Figure 1 have been previously studied in the context of **SCRP**. In particular, notice how $*$ specifies that a state is (up to bisimilarity) a synchronous product with suitably sub-divided resource, and how $\exists_\nu$ specifies a hiding. In a similar way, $\{E\}\phi$ specifies a state that is a role (up to bisimilarity) and that the role it takes satisfies ϕ. Thus the role F of E witnesses ϕ for G (even when E itself does not). Some examples of the use of these exotic connectives are given in the sequel.

The Hennessy-Milner-style result given below holds. This shows that algebraically equivalent processes satisfy the same logical specifications.

Theorem 1. If $R, E \approx R, F$ and $R, E, \eta \vDash \phi$ then $R, F, \eta \vDash \phi$.

Proof. The proof is by induction on the structure of ϕ. The base of the induction is assumed because the interpretation of atomic formulae is assumed to be closed under $\approx$. Most of the other steps are contained in the proof of the analogous result in [38, 39]. Now consider the step for $\{E\}\phi$. Suppose that $R, E \approx R, F$ and $R, E, \eta \vDash \{G\}\phi$. Then there is some H with $R, E \approx R, G \propto H$ and $R, H, \eta \vDash \phi$. Since $\approx$ is transitive, we also have $R, F \approx R, G \propto H$, and so $R, F, \eta \vDash \{G\}\phi$. □

The converse to the above theorem holds, so that logically equivalent states are algebraically equivalent. The proof for the analogous result in [38, 39, 15] suffices.

Theorem 2. $R, E \approx R, F$ whenever $R, E, \eta \vDash \phi$ iff $R, F, \eta \vDash \phi$ for all ϕ.

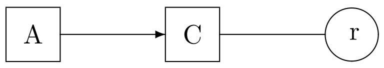

Figure 2: A guarded resource

A number of important reasoning principles are justified in this context. The first of these reveals part of the intended meaning, that if some process has a property then any process that uses it as a role also has a version of that property, but guarded by a use of the says modality.

Proposition 3.

1. If $R, G, \eta \vDash \phi$ and $G \lessapprox F$ then $R, F \propto G, \eta \vDash \{F\}\phi$.
2. If $R, G, \eta \vDash \{E\}\phi$ and $E \approx F$ then $R, G, \eta \vDash \{F\}\phi$ holds.
3. $R, E, \eta \vDash \phi$ iff $R, E \propto E, \eta \vDash \phi$.
4. If $R, E, \eta \vDash \phi$ then $R, E, \eta \vDash \{E\}\phi$.

Proof.

1. If $R, G, \eta \vDash \phi$ and $G \lessapprox F$, then by definition of the interpretation $R, F \propto G, \eta \vDash \{F\}\phi$, since $R, F \propto G \approx R, F \propto G$.
2. If $R, G, \eta \vDash \{E\}\phi$ then there is some H such that $R, G \approx R, E \propto H$ and $R, H, \eta \vDash \phi$. If $E \approx F$ then $E \propto H \approx F \propto H$ by Proposition 1. Therefore $R, G, \eta \vDash \{F\}\phi$.
3. By Proposition 2 we have $E \approx E \propto E$ and so the third point holds by Theorem 1.
4. The fourth point follows from the third and the interpretation of 'says'. □

Many other semantic reasoning principles hold; see [14] for a discussion and a deductive system, in the absence of roles and 'says'.

6 Examples

The examples of this section are intended to illustrate some of the most common access control situations, how they may be *modelled* in resource-based process algebra, and which logical specifications they satisfy. They are *not* intended to give complete formal renderings of existing protocols, although we believe that this could certainly be done for many protocols, as **SCRP** has at least the same expressive power as CSP [20, 45, 41]. Similarly, we do *not* claim that our logical language is the only logical language that can express each of the properties below. Rather our goal is to describe, specify and reason about models of systems in which security concerns are critical. Here we demonstrate that the foregoing calculus is a practical semantic foundation for such work, with sufficient richness to capture structural properties of composite agents, for example co-signing and roles, but also with well-specified operational behaviour.

In each of the following examples we specify a modification on atomic actions only. The fact that this extends uniquely to a modification (on all actions) is a consequence of a mild generalization of a result proved in [14]. The generalization allows one to work with resource monoids that are not required to be total, and such that cancellation exists only as a partial function ($\forall R, S, T.\ R \circ S = R \circ T \Longrightarrow S = T$).

Let 2 be the resource monoid $\{0, 1\}$ with unit 0 and composition + with $0 + n = n$, $1 + 1 \uparrow$. This kind of resource is often used to represent a semaphore. We use these, and the positive integers, as channels to communicate and moderate interaction between processes: the approach in modelling languages like Demos2k is essentially the same.

Example 1. (Access by proxy). Consider the situation in Figure 2, with a principal A attempting to access a resource r via a guard C. Assume that the way this works is that A makes a request to access r and then C either implements this or does not depending on whether it believes A has the right to perform this act.

This can be modelled through the use of a synchronous product

$$A \times C$$

where A and C are defined by the equations,

$$A = 1 : A + a : A \qquad C = 1 : C + c : C$$

and where a is the access request and c performs that access on behalf of A.

In order for this to make sense we must define an appropriate resource monoid and modification function. These should ensure that certain sequences of actions may occur and certain others may not. In this example, and those that follow, this sequencing will be controlled through the use of semaphores which form part of the resource.

For example, consider the situation in which r is intended to hold an integer and a makes a request to increment r. We could take resources to be triples of of the form

$$\langle m, n, L \rangle,$$

where m is an integer, $n \in 2$ and L is a set of actions of C that are allowed access to r. The integer m represents the contents of r. The integer n represents a resource component (like a semaphore or buffer) used to communicate the access request from A to C. The set L represents an access control list. Indeed, we adopt the informal convention of calling such a set an ACL.

A meaningful choice of resource monoid composition takes pointwise addition on the first two components and non-overlapping disjoint union for the third component. That is,

$$\langle m, n, L \rangle \circ \langle m', n', L' \rangle = \begin{cases} \langle m + m', n + n', L \cup L' \rangle & \text{if } L \cap L' = \emptyset \\ \uparrow & \text{otherwise} \end{cases}$$

for all suitable m, m', n, n', L, L'. The unit of this monoid is $\langle 0, 0, \emptyset \rangle$.

Let the set of atomic actions contain just the actions a and c. We choose the modification so that

$$\mu(a, \langle m, n, L \rangle) = \langle m, 1, L \rangle$$

$$\mu(c, \langle m, n, L \rangle) = \begin{cases} \langle m + 1, 0, L \rangle & \text{if } n = 1, c \in L \\ \uparrow & \text{otherwise.} \end{cases}$$

We find a sequence of access events of the form

$$\begin{array}{rcl} \langle 0, 0, [c] \rangle, A \times C & \xrightarrow{a} & \langle 0, 1, [c] \rangle, A \times C \\ & \xrightarrow{c} & \langle 1, 0, [c] \rangle, A \times C \end{array}$$

for example. On the other hand, the increment of r only takes place after a request has been issued and the corresponding action of C found in the list L. Thus we see that C increments r on behalf of A.

Note that the choices of resource, resource composition and modification function are tied to significant aspects of operational behaviour, and that here we are just taking a simple example.

Many variations on this first simple example are easily expressed:

1. In the above example, an access c is not necessarily granted immediately after any request a, or indeed before any other request a. This is a simple choice, and it is easy to modify this to give more sophisticated interactions.

2. In the above example, accesses come in pairs $\langle a, c \rangle$ consisting of the request a and the requested action c. If two different agents wish to perform the same access upon r, but they have different permissions, we must have two distinct actions (a_1 and a_2, say, with corresponding c_1 and c_2) representing the two different access requests. For some situations, an alternative would be to use an ACL containing the access requests a_i, for then the same c may result from both a_i.

3. The resource and modification can be changed so that the action c may occur arbitrarily often after a single instance of a. An additional agent (and action) which stops access may then be added, if desired.

4. The modification can be changed so that permission to access can only be exercised once: the μ can be chosen to remove c from L following an action of c.

5. If L is a multiset, then μ can be chosen so that units of permission are consumed. Again, c removes an instance of c from L. It may be natural then to have an additional agent that creates permissions (by adding to the ACL).

6. A blacklisting approach is easily modelled by taking L to be a 'blocked-list': this is achieved by changing the modification so that μ is defined at c when $c \notin L$, instead of $c \in L$.

Example 2. (Direct access). This example modifies the previous one so that A itself performs an action upon r after requesting and receiving permission from C.

We consider a system with process component $A \times C$ again, but with

$$\begin{array}{rcl} A & = & 1 : A + a : A' \\ A' & = & 1 : A' + i : A \\ C & = & 1 : C + c : C \end{array}$$

where this time a is the access request sent to C, c is the response sent back to A and i is the incrementation action on r.

Resources are taken to be of the form $\langle m, n, p, L \rangle$ where $m \in \mathbb{N}$, $n, p \in 2$ and L is a list of actions. The component p is used to represent the signal from C to A. Composition of resources is defined pointwise using the resource monoids defined above.

We choose the modification with:

$$\mu(a, \langle m, n, p, L \rangle) = \langle m, 1, p, L \rangle$$

$$\mu(c, \langle m, n, p, L \rangle) = \begin{cases} \langle m, 0, 1, L \rangle & \text{if } n = 1 \text{ and } c \in L \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(i, \langle m, n, p, L \rangle) = \begin{cases} \langle m+1, n, 0, L \rangle & p = 1 \\ \uparrow & \text{otherwise.} \end{cases}$$

Then we find, for example, that with $c \in L$, the system $\langle 0, 0, 0, L \rangle, A \times C$ makes transition sequences

$$\ldots \xrightarrow{a} \ldots \xrightarrow{c} \ldots \xrightarrow{i} \ldots,$$

where each access i must be preceded by some response c and that must be preceded by some request a.

It is straightforward to extend the above to situations in which there are multiple agents $A_1, \ldots, A_n$ attempting to acccess multiple resources via multiple guards. We take a synchronous product of all the agents A_i and all the guards. More interesting situations arise when there are assumptions about concurrent accesses.

Example 3. (Joint-access requests). In this example, there are two principals (processes) A_1 and A_2 that can only access some basic resource r via some guard process C after they have both made requests.

We take the process part of the system to be $A_1 \times A_2 \times C$ with

$$\begin{array}{rclcrcl} A_1 & = & 1 : A_1 + a_1 : A_1' & \qquad & A_1' & = & 1 : A_1' + i : A_1 \\ A_2 & = & 1 : A_2 + a_2 : A_2 & & C & = & 1 : C + c : C \end{array}$$

and resources of the form $\langle m, n_1, n_2, p, L \rangle$, where m is an integer, $n_1, n_2, p \in 2$ and L is an ACL represented by a set of actions. Take composition of resource to be pointwise addition. Take each access request a_j to increment n_j (undefined if $n_j = 1$); c to increment p and set both $n_j = 0$ just when $n_1 = n_2 = 1$ (otherwise undefined) and $c \in L$; let the access i resulting from a_1, a_2 increment m and set $p = 0$, when $p = 1$ (otherwise undefined). More precisely,

$$\mu(a_1, \langle m, n_1, n_2, p, L\rangle) = \langle m, n_1 + 1, n_2, p, L\rangle$$

$$\mu(a_2, \langle m, n_1, n_2, p, L\rangle) = \langle m, n_1, n_2 + 1, p, L\rangle$$

$$\mu(c, \langle m, n_1, n_2, p, L\rangle) = \begin{cases} \langle m, 0, 0, 1, L\rangle & \text{if } n_1 = n_2 = 1 \text{ and } c \in L \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(i, \langle m, n_1, n_2, p, L\rangle = \begin{cases} \langle m+1, n_1, n_2, 0, L\rangle & \text{if } p = 1 \\ \uparrow & \text{otherwise} \end{cases}$$

for all $m \in \mathbb{N}$ and $n_1, n_2, p \in 2$.

Then we have transition sequences of the following forms:

$$\ldots \xrightarrow{a_1} \ldots \xrightarrow{a_2} \ldots \xrightarrow{c} \ldots \xrightarrow{i} \ldots$$

$$\ldots \xrightarrow{a_2} \ldots \xrightarrow{a_1} \ldots \xrightarrow{c} \ldots \xrightarrow{i} \ldots$$

$$\ldots \xrightarrow{a_1 a_2} \ldots \xrightarrow{c} \ldots \xrightarrow{i} \ldots$$

amongst the possible system behaviours.

Let ϕ_j be the property that the resource-component at n_j is 1. From the point of view of **MBIa**, we have that the judgement relation

$$\langle m, 1, 1, p, L\rangle, A_1 \times A_2, \eta \vDash \phi_1 * \phi_2$$

holds for any m, p, η, because $\langle m, 1, 0, p, L\rangle, A_1, \eta \vDash \phi_1$ and $\langle 0, 0, 1, 0, \emptyset\rangle, A_2, \eta \vDash \phi_2$ both hold and

$$\langle m, 1, 0, p, L\rangle \circ \langle 0, 0, 1, 0, \emptyset\rangle = \langle m, 1, 1, p, L\rangle\ .$$

Thus, this judgement expresses the fact that both access requests have been made. From this it may be inferred that the response c may grant permission for the joint requests a_j to perform the access action i. To summarize: the $*$ connective describes co-signing situations in a particularly natural way.

There are important variants of the joint-access pattern.

1. This kind of example can be further refined so that the use of $*$ also requires that two signatories must hold disjoint permissions.

2. In the example above access can be granted after the two requests are made in any order, or simultaneously. This can be modified so that access is only granted if the two agents make their request simultaneously. Indeed, it is a particular version of a concurrent *handshaking* situation, as described in [38, 39, 15].

3. The two authorizing agents must both give authorization in some chosen sequence. This can be captured in **SCRP** by a specific use of *resource-transfer* as exposed in [38, 39, 15].

4. These examples extend to situations requiring agreement between multiple parties.

Example 4. (Exclusive access). We suppose that we are in a situation in which we have two agents A_1 and A_2 that both wish to access r via C, but that only one of the agents A_i must be able to access r at any time. This is a classic concurrent mutual-exclusion situation and is modelled in **SCRP** through the use of a resource that can only be used by one process at a time.

We take resources to be of the form

$$\langle m, n_1, n_2, p_1, p_2, q, L\rangle \ ,$$

where m is the integer-valued content of r, the $n_i, p_i, q \in 2$ and L is an ACL. We take the composition of resources to be defined pointwise with the operations indicated above.

We choose atomic actions a_1, a_2, c_1, c_2, i and a modification as follows:

$$\mu(a_1, \langle m, n_1, n_2, p_1, p_2, q, L\rangle) = \langle m, 1, n_2, p_1, p_2, q, L\rangle$$

$$\mu(a_2, \langle m, n_1, n_2, p_1, p_2, q, L\rangle) = \langle m, n_1, 1, p_1, p_2, q, L\rangle$$

$$\mu(c_1, \langle m, n_1, n_2, p_1, p_2, q, L\rangle) = \begin{cases} \langle m, 0, n_2, 1, p_2, q, L\rangle & \text{if } n_1 = 1 \text{ and } c_1 \in L \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(c_2, \langle m, n_1, n_2, p_1, p_2, q, L\rangle) = \begin{cases} \langle m, n_1, 0, p_1, 1, q, L\rangle & \text{if } n_2 = 1 \text{ and } c_2 \in L \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(i, \langle m, n_1, n_2, p_1, p_2, q, L\rangle) = \begin{cases} \langle m, n_1, n_2, p_1, p_2, q, L\rangle & \text{if } q = 1 \text{ and } (p_1 = 1 \text{ or } p_2 = 1) \\ \uparrow & \text{otherwise.} \end{cases}$$

Define the processes

$$\begin{array}{rcllll} A_1 & = & 1 : A_1 & + & a_1 : A_1' & \\ A_1' & = & 1 : A_1' & + & i : A_1 & \\ A_2 & = & 1 : A_2 & + & a_2 : A_2' & \\ A_2' & = & 1 : A_2' & + & i : A_2 & \\ C & = & 1 : C & + & c_1 : C & + \quad c_2 : C \end{array}$$

where a_1, a_2 are the respective requests by A_1 and A_2 to perform i.

Consider any system

$$\langle m, n_1, n_2, p_1, p_2, q, 1\rangle, A_1 \times A_2 \times C$$

There are transition sequences of the three forms

$$\begin{array}{c} \ldots \xrightarrow{a_1} \ldots \xrightarrow{c_1} \ldots \xrightarrow{i} \ldots \\ \ldots \xrightarrow{a_1} \ldots \xrightarrow{c_1} \ldots \xrightarrow{i} \ldots \\ \ldots \xrightarrow{a_1} \ldots \xrightarrow{a_2} \ldots \xrightarrow{c_1} \ldots \xrightarrow{c_2} \ldots \xrightarrow{i} \ldots \xrightarrow{i} \end{array}$$

amongst others, but not of the form

$$\ldots \xrightarrow{ii} \ldots$$

because $q \in 2$. That is, requests a_1 and a_2 can be made, possibly simultaneously, the responses c_1 and c_2 can come back, possibly simultaneously, and incrementations i can be made, but *not* simultaneously.

Example 5. (Authorization by delegation). Consider a situation in which the guard C must now consult some other principal B who owns an ACL, L, that says that the requested access should be granted. This is a simple version of the situation described in Figure 3.

Such a situation can be modelled using resources of the form $\langle m, n, p, q, k, L\rangle$ where m is an integer, n, p, q, k are copies of the semaphore 2, and L is a list of actions. Composition is defined pointwise from the operations on components considered in the previous examples. Let $L_0 = \{a\}$ and consider the resource $R_{L_0} = \langle 0, 0, 0, 0, 0, L_0\rangle$.

We use agents

$$\begin{array}{rcllll} A & = & 1 : A & + & a : A_0 & \\ A_0 & = & 1 : A_0 & + & i : A & \\ B & = & 1 : B & + & bb' : B & \\ C & = & 1 : C & + & c : C & + \quad d : C \end{array}$$

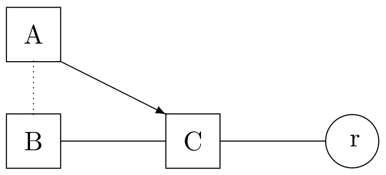

Figure 3: Guarded resource with delegation

in a product

$$A \times C \times \nu R_{L_0}.B$$

featuring a hiding. The modification is defined by:

$$\mu(a, \langle m,n,p,q,k,L\rangle) = \langle m,1,p,q,k,L\rangle$$

$$\mu(b, \langle m,n,p,q,k,L\rangle) = \begin{cases} \langle m,n,0,1,k,L\rangle & \text{if } p = 1 \text{ and } a \notin L \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(b', \langle m,n,p,q,k,L\rangle) = \begin{cases} \langle m,n,p,q,k,L\rangle & \text{if } a \in L \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(c, \langle m,n,p,q,k,L\rangle) = \begin{cases} \langle m,n,1,q,k,L\rangle & \text{if } n = 1 \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(d, \langle m,n,p,q,k,L\rangle) = \begin{cases} \langle m,n,p,0,1,L\rangle & \text{if } q = 1 \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(i, \langle m,n,p,q,k,L\rangle) = \begin{cases} \langle m+1,n,p,q,0,L\rangle & \text{if } k = 1 \\ \uparrow & \text{otherwise.} \end{cases}$$

Thus: a is an access request made by A to B, using the channel n; c represents C asking B, using the channel p, if the request should be granted; b' represents B consulting its private ACL, L_0; b is the signal from B to C, using the channel q, that the access should be granted; d is the signal from C to A, using the channel k, that the access has been granted; i is the actual incrementation action that takes place, given that all the above have happened.

With the folowing resources,

$$\begin{array}{ll} R_0 = \langle 0,0,0,0,0,\emptyset\rangle & R_1 = \langle 0,1,0,0,0,\emptyset\rangle \\ R_2 = \langle 0,0,1,0,0,\emptyset\rangle & R_3 = \langle 0,0,0,1,0,\emptyset\rangle \\ R_4 = \langle 0,0,0,0,1,\emptyset\rangle & R_5 = \langle 1,0,0,0,0,\emptyset\rangle \end{array}$$

we have, for example, the system evolutions

$$\begin{array}{ll} & R_0, A \times C \times \nu R_{L_0}.B \\ \xrightarrow{a} & R_1, A_0 \times C \times \nu R_{L_0}.B \\ \xrightarrow{c} & R_2, A_0 \times C \times \nu R_{L_0}.B \\ \xrightarrow{b} & R_3, A_0 \times C \times \nu R_{L_0}.B \\ \xrightarrow{d} & R_4, A_0 \times C \times \nu R_{L_0}.B \\ \xrightarrow{i} & R_5, A \times C \times \nu R_{L_0}.B \end{array}$$

making use of

$$\frac{R_2 \circ R_{L_0}, B \xrightarrow{bb'} R_3 \circ R_{L_0}, B}{R_2, \nu R_{L_0}.B \xrightarrow{b} R_3, \nu R_{L_0}.B}$$

to give the b-transition.

Let ϕ be the assertion 'the q component of the resource is $1'$, The relation

$$R_2, \nu R_{L_0}.B, \eta \vDash \langle b \rangle \phi$$

specifies that the system $R_2, \nu R_{L_0}.B$ can signal to C that the access should be granted. In more detail, this happens because

$$R_2, \nu R_{L_0}.B, \eta \vDash \exists_\nu x.\langle bx \rangle \phi$$

which holds because the private ACL L_0 can be consulted by B using the hidden action b'.

It is a straightforward matter to extend the preceding example to longer chains of trust: each delegation is specified in the logic by a suitable $\exists_\nu$ satisfaction statement. Statements may then be chained together using the definition of the satisfaction relation to show that appropriate access decisions can be taken.

Example 6. (Reduction to a role). Consider a situation where some process C guards two basic resources r_1 and r_2, a process A and a role B. The process A can make an access requests a_j to increment the value m_j stored at r_j for both $j = 1$ and $j = 2$. However, in the role B it can only make the access request a_1.

Take resources of the form $\langle m_1, m_2, p_1, p_2, q_1, q_2, L \rangle$ where the $m_j \in \mathbb{N}$, the $p_j, q_j \in 2$ and L is a set of actions (representing an ACL containing permitted responses of C). Let

$$R = \langle 0, 0, 0, 0, 0, 0, \{c_1, c_2\} \rangle S = \langle 0, 0, 0, 0, 0, 0, \{c_1\} \rangle \mathbf{e} = \langle 0, 0, 0, 0, 0, 0, \emptyset \rangle$$

be resources.

Consider the process terms:

$$\begin{array}{rcl} A & = & \nu R.A' \\ B & = & \nu S.A' \\ A' & = & 1 : A' + a_1 : A'' + a_2 : A'' \\ A'' & = & 1 : A' + i_1 : A + i_2 : A \\ C & = & 1 : C + c_1 : C + c_2 : C \end{array}$$

Take μ as follows:

$$\mu(a_1, \langle m_1, m_2, p_1, p_2, q_1, q_2, L \rangle) = \quad \langle m_1, m_2, 1, p_2, q_1, q_2, L \rangle$$

$$\mu(a_2, \langle m_1, m_2, p_1, p_2, q_1, q_2, L \rangle) = \quad \langle m_1, m_2, p_1, 1, q_1, q_2, L \rangle$$

$$\mu(c_1, \langle m_1, m_2, p_1, p_2, q_1, q_2, L \rangle) = \begin{cases} \langle m_1, m_2, p_1, p_2, 1, q_2, L \rangle & \text{if } p_1 = 1 \text{ and } c_1 \in L \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(c_2, \langle m_1, m_2, p_1, p_2, q_1, q_2, L \rangle) = \begin{cases} \langle m_1, m_2, p_1, p_2, q_1, 1, L \rangle & \text{if } p_2 = 1 \text{ and } c_2 \in L \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(i_1, \langle m_1, m_2, p_1, p_2, q_1, q_2, L \rangle) = \begin{cases} \langle m_1 + 1, m_2, p_1, p_2, 0, q_2, L \rangle & \text{if } q_1 = 1 \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mu(i_2, \langle m_1, m_2, p_1, p_2, q_1, q_2, L \rangle) = \begin{cases} \langle m_1, m_2 + 1, p_1, p_2, q_1, 0, L \rangle & \text{if } q_2 = 1 \\ \uparrow & \text{otherwise} \end{cases}$$

The system $\mathbf{e}, A$ has transition sequences of the form

$$\begin{array}{c} \ldots \xrightarrow{a_1} \ldots \xrightarrow{c_1} \ldots \xrightarrow{i_1} \ldots \\ \ldots \xrightarrow{a_2} \ldots \xrightarrow{c_2} \ldots \xrightarrow{i_2} \ldots \\ \ldots \xrightarrow{d} \ldots \xrightarrow{a_1} \ldots \xrightarrow{c_1} \ldots \xrightarrow{i_1} \ldots \end{array}$$

amongst others, but not of the form

$$\dots \xrightarrow{d} \dots \xrightarrow{a_2} \dots \xrightarrow{c_2} \dots \xrightarrow{i_2} \dots$$

because the permissions associated with B do not include those for the $\langle a_2, c_2 \rangle$ request-response pair. Similarly, the system $\mathbf{e}, A \propto B$ has

$$\dots \xrightarrow{a_1} \dots \xrightarrow{c_1} \dots \xrightarrow{i_1} \dots$$

as a possible behaviour, but not

$$\dots \xrightarrow{a_2} \dots \xrightarrow{c_2} \dots \xrightarrow{i_2} \dots.$$

We may express logically the fact that $A \propto B$ cannot perform all the accesses that A can, given resource $\mathbf{e}$. Given $\phi := \langle a_1 \rangle \langle c_1 \rangle \langle i_1 \rangle \top$ we find that

$$\mathbf{e}, A, \eta \vDash \phi \qquad \mathbf{e}, B, \eta \vDash \neg\phi \qquad \mathbf{e}, A \propto B, \eta \vDash \{A\}\neg\phi$$

hold.

Note that we can often use simple resource assertions instead of complex modal assertions: for example, it is often enough to specify that sufficient resource is present to enable an action to fire instead of specifying that the action can fire using a modality.

The following example if of a significantly different type. In the foregoing examples the systems we modelled were all based on variants of agents with ACLs. The logical language we had is then used to make assertions about systems. In contrast, in the following example, a security-language is used to govern access decisions. Thus there are logical formulae *in* the models, as well as *about* the models.

Example 7. (Modelling access with assertion-based control). Consider a system with 3 agents. The first of these wishes to perform some operation. However, it only perfoms this action when it believes that it should; it receives such information from the second agent, and trusts this information; the second agent, in turn, receives the information from the third agent, which it trusts. This is a simple chain-of-trust. The first agent does not explicitly trust the third, however it does so implicitly.

We assume the existence of a simple language of access assertions

$$\begin{array}{rcl} p_0 & ::= & \mathsf{mayAcc}(A_1, i, r_1) \\ p & ::= & p_0 \mid \mathsf{states}(C_k, p_0) \mid \neg p_0 \end{array}$$

with $1 \leq k \leq 3$. That is, for this example, we are only concerned with the ability of A_1 (defined below) to access some resource r_1 with the operation i. This is easily generalized by extending p_0 with many atoms. This language should not be confused with the Hennessy-Milner logic described earlier. Let $\mathcal{L}$ be the set of propositions of this language.

We assume the existence of basic actions with approxiamte intended meanings:

c_j^p	C_j states p
b_j^p	update belief of C_j with p, in the light of trusted statement that p
a	request to do some operation (access) on r_1
d	unlock the requested operation
i	do the operation requested

for $1 \leq j \leq 3$ and p, as above. In addition, for slightly technical reasons, there are also actions d' and ${c'_2}^{p_0}$ present. These are further explained by the modification.

Define a set of formulae of $\mathcal{L}$ to be *consistent* when it does not contain both p and $\neg p$, for any p. Reserve the letters capital Γ, Δ for such sets. The set of sets of consistent formulae is a resource monoid with the operation

$$\Gamma \circ \Delta = \begin{cases} \Gamma \cup \Delta & \text{provided } \Gamma \cup \Delta \text{ is consistent} \\ \uparrow & \text{otherwise} \end{cases}$$

for any Γ and Δ. We write the unit as $\emptyset$. When this composite is defined we say that Γ and Δ are consistent.

The resources for the system we wish to describe are then of the form $\langle m, n, k, \Gamma\rangle$, where $m \in \mathbb{N}$, p, $k \in 2$ and Γ is a consistent set of formulae. The composition operation acts pointwise on the components of these quadruples, with the component operations indicated above. The unit is $\mathbf{e} = \langle 0, 0, 0, \emptyset\rangle$. As usual, the letters R, S stand for such resources. Define the abbreviation $R \circ \Delta = \langle m, n, k, \Gamma \circ \Delta\rangle$ for any resource $R = \langle m, n, k, \Gamma\rangle$ and consistent set of formulae Δ. Similarly, we write $\phi \in R$ as an abbreviation for $\phi \in \Gamma$, where Γ is the resource component of R.

The modification is specified by:

$$\begin{array}{rcl}
\mu(a^p, \langle m, n, k, \Gamma\rangle) & = & \langle m, 1, k, \Gamma\rangle \\[2ex]
\mu(b_1^p, R) & = & \left\{\begin{array}{ll} R \circ [p] & \text{if } \mathsf{states}(C_2, p) \in R \\ \uparrow & \text{otherwise}\end{array}\right. \\[2ex]
\mu(b_2^p, R) & = & \left\{\begin{array}{ll} R \circ [p] & \text{if } \mathsf{states}(C_3, p) \in R \\ \uparrow & \text{otherwise}\end{array}\right. \\[2ex]
\mu(c_2^p, R) & = & \left\{\begin{array}{ll} R \circ [\mathsf{states}(C_2, p)] & \text{if } n = 1 \\ \uparrow & \text{otherwise}\end{array}\right. \\[2ex]
\mu({c'_2}^p, R) & = & \left\{\begin{array}{ll} R & \text{if } p \in R \\ \uparrow & \text{otherwise}\end{array}\right. \\[2ex]
\mu(c_3^p, R) & = & R \circ [\mathsf{states}(C_3, p)] \\[2ex]
\mu(d, \langle m, n, k, \Gamma\rangle) & = & \left\{\begin{array}{ll} \langle m, 0, 1, \Gamma\rangle & \text{if } n = 1 \\ \uparrow & \text{otherwise}\end{array}\right. \\[2ex]
\mu(d', R) & = & \left\{\begin{array}{ll} R & \text{if } p \in R \text{ and } n = 0 \\ \uparrow & \text{otherwise}\end{array}\right. \\[2ex]
\mu(i, \langle m, n, k, \Gamma\rangle) & = & \left\{\begin{array}{ll} \langle m+1, 0, 0, \Gamma\rangle & \text{if } k = 1 \\ \uparrow & \text{otherwise}\end{array}\right.
\end{array}$$

on atomic actions.

Define the resources (multisets)

$$S_1 = \emptyset = S_2 \qquad\qquad S'_1 = [p_0] = S'_2.$$

Consider the process $A_1 \times C_1 \times C_2 \times C_3$, where

$$\begin{array}{rcl}
A & = & 1 : A + a : A + i : A \\[2ex]
C_3 & = & 1 : C_3 + c_3^{p_0} : C_3 \\[2ex]
C_1 & = & \nu S_1 . D_1 \\[2ex]
D_1 & = & 1 : D_1 + dd' : D_1 + b_1^{p_0} : D_1 \\[2ex]
C_2 & = & \nu S_2 . D_2 \\[2ex]
D_2 & = & 1 : D_2 + b_2^{p_0} : D_2 + c_2^{p_0} {c'_2}^{p_0}
\end{array}$$

Thus: A_1 requests to perform the operation i using the access a; the guard C_1 must decide whether to allow this access or not, and to do this it consults C_2, which in turn consults C_3. The fact that C_1 trusts C_2, which trusts C_3 is built into the model.

There is a transition sequence:

$$
\begin{array}{ll}
 & \mathbf{e}, A_1 \times \nu S_1.D_1 \times \nu S_2.D_2 \times C_3 \\
\xrightarrow{a} & \\
 & \langle 0,1,0,\emptyset\rangle, A_1 \times \nu S_1.D_1 \times \nu S_2.D_2 \times C_3 \\
\xrightarrow{c_3^{p_0}} & \\
 & \langle 0,1,0,[\mathsf{states}(C_3,p_0)]\rangle, A_1 \times \nu S_1.D_1 \times \nu S_2.D_2 \times C_3 \\
\xrightarrow{b_2^{p_0}} & \\
 & \langle 0,1,0,[\mathsf{states}(C_3,p_0)]\rangle, A_1 \times \nu S_1.D_1 \times \nu S_2'.D_2 \times C_3 \\
\xrightarrow{c_2^{p_0}} & \\
 & \langle 0,1,0,[\mathsf{states}(C_2,p_0),\mathsf{states}(C_3,p_0)]\rangle, A_1 \times \nu S_1.D_1 \times \nu S_2'.D_2 \times C_3 \\
\xrightarrow{b_1^{p_0}} & \\
 & \langle 0,1,0,[\mathsf{states}(C_2,p_0),\mathsf{states}(C_3,p_0)]\rangle, A_1 \times \nu S_1'.D_1 \times \nu S_2'.D_2 \times C_3 \\
\xrightarrow{d} & \\
 & \langle 0,0,1,[\mathsf{states}(C_2,p_0),\mathsf{states}(C_3,p_0)]\rangle, A_1 \times \nu S_1'.D_1 \times \nu S_2'.D_2 \times C_3 \\
\xrightarrow{i} & \\
 & \langle 1,0,0,[\mathsf{states}(C_2,p_0),\mathsf{states}(C_3,p_0)]\rangle, A_1 \times \nu S_1'.D_1 \times \nu S_2'.D_2 \times C_3 \\
\xrightarrow{a} & \\
 & \langle 1,1,0,[\mathsf{states}(C_2,p_0),\mathsf{states}(C_3,p_0)]\rangle, A_1 \times \nu S_1'.D_1 \times \nu S_2'.D_2 \times C_3 \\
\xrightarrow{d} & \\
 & \langle 1,0,1,[\mathsf{states}(C_2,p_0),\mathsf{states}(C_3,p_0)]\rangle, A_1 \times \nu S_1'.D_1 \times \nu S_2'.D_2 \times C_3 \\
\xrightarrow{i} & \\
 & \langle 2,0,0,[\mathsf{states}(C_2,p_0),\mathsf{states}(C_3,p_0)]\rangle, A_1 \times \nu S_1'.D_1 \times \nu S_2'.D_2 \times C_3 \\
\ldots &
\end{array}
$$

for example. Thus belief cascasdes down from C_3 to C_1 and this allows A_1 to perform the operation requested. In the present model belief cannot be revoked, and so the chain-of-trust does not need to be consulted for the second incrementation. The action ${c'_2}^{p_0}$ is used to check if C_2 believes p_0, whilst the action $c_2^{p_0}$ extrudes information back into the global resource by making the public statement $\mathsf{states}(C_2, p_0)$. Similarly, the action d' used to check that C_1 believes p_0, whilst d extrudes information by unlocking the incrementation.

The intended trust relation

$$C_2 \text{ states } p \quad \text{implies} \quad C_1 \text{ believes } p$$

follows from an implication between instances of satisfaction

$$R, C_1 \times C_2 \vDash \mathsf{states}(C_2, p) \quad \text{implies} \quad R, C_1 \vDash \exists_\nu x.\phi$$

in the Hennessy-Milner logic, where $S, F \vDash \phi$ iff $p \in S$, for any S, F.

We note that the use of formulae-as-resources and hiding gives a simple account of agents with private belief, and conjecture that more sophisticated versions of this approach could be of use in many other situations.

These examples can be further extended so that longer chains-of-trust and more complex patters are used for determining accesses.

7 Global Simulation and Multiplicative Implication

The logical calculus **MBIa** of Section 5 above combines Hennessy-Milner-style modal connectives with a separating conjuction in the style of **BI** and Separation Logic. The multiplicative implication, $-\!\!*$, of **BI** was omitted as it complicates the treatment somewhat. Here, we rectify

$$R, E, \eta \vDash \phi \mathrel{-\!\!*} \psi \quad \text{iff} \quad \forall S, F.\ \ S, F, \eta \vDash \phi \ \text{ implies } \ R \circ S, E \times F, \eta \vDash \psi$$

$$R, E, \eta \vDash \langle \mathsf{a} \rangle_\nu \phi \quad \text{iff} \quad \exists S, E'.\ R \circ S, E \xrightarrow{\mathsf{a}} \mu(a, R \circ S), E' \ \text{ and } \ \mu(a, R \circ S), E', \eta \vDash \phi$$

$$R, E, \eta \vDash [\mathsf{a}]_\nu \phi \quad \text{iff} \quad \forall S, E'.\ R \circ S, E \xrightarrow{\mathsf{a}} \mu(a, R \circ S), E' \ \text{ implies } \ \mu(a, R \circ S), E', \eta \vDash \phi$$

Figure 4: Extended Interpretation of Logical Formulae

that omission and also include versions of the multiplicative modalities $\langle a \rangle_\nu$ and $[a]_\nu$ previously considered for the original version of **MBI** [38, 39, 15].

In order to give such a treatment we must make some alterations to the notion of bisimulation, and so to the process calculus. We define a modified process calculus **ACCRPb** together with a new simulation relation $\sim$. The new process calculus is formed by replacing the rule for roles with the rule

$$\frac{R, F \xrightarrow{a} R', F' \quad R \circ S, E \xrightarrow{a} R' \circ S', E'}{R \circ S, E \propto F \xrightarrow{a} R' \circ S', E' \propto F'} \ (F \lesssim E) \tag{4}$$

where $\lesssim$ is the largest relation on processes such that, if $E_1 \sim E_2$ and $R, E_1 \xrightarrow{a} R', F_1$ for any R, R', F_1, then there is some F_2 such that $R, E_2 \xrightarrow{a} R', F_2$ and $E_2 \lesssim F_2$. We call the relation $\sim\, = \,\lesssim \cap \gtrsim$ the *global bisimulation relation.* As with $\approx$, the definition of this relation can be stratified to show that the mutually recursive definition above makes sense. The relation $\sim$ is extended to states by taking $R, E \sim S, F$ just if $R = S$ and $E \sim F$.

The relation $\sim$ is a *congruence* for the process constructors (not just an equivalence) and all of the results of Proposition 2 are retained, but with $\lesssim$ (and $\sim$) replacing $\lessapprox$ (respectively $\approx$) throughout. Indeed, the following strengthening of Proposition 2 holds.

Proposition 4. The constructor $\propto$ has the following properties:

1. $E \propto F \lesssim E$;
2. $E \propto E \sim E$;
3. If $G \lesssim E$ then $G \lesssim E \propto G$;
4. If $E \lesssim F$ then $E \propto G \lesssim F \propto G$;
5. $(E_1 \propto F_1) \times (E_2 \propto F_2) \lesssim (E_1 \times E_2) \propto (F_1 \times F_2)$;
6. $E \propto (F \propto G) \lesssim (E \propto F) \propto G$ provided $F \lesssim E$ and $G \lesssim E \propto F$;
7. $(E \propto F) \propto G \lesssim E \propto (F \propto G)$ provided $G \lesssim F$.

Proof. The proofs are direct uses of the definition of the relation $\lesssim$, and we omit them. □

The logical language **MBIa** is extended as follows

$$\phi ::= \ldots \mid \phi \mathrel{-\!\!*} \phi \mid \langle a \rangle_\nu \phi \mid [a]_\nu \phi$$

to give a new language **MBIb**.

The notion of valuation for the logic is changed so that $\mathcal{V}(p)$ is closed under the relation $\sim$ on states. The interpretation of Figure 1 is modified so that, wherever some relation $R, E \approx R, F$ is expressed, it is replaced by the relation $E \sim F$ on processes. The use of $\lessapprox$ for $\{E\}\phi$ is replaced by a use of $\lesssim$. In addition we extend the interpretation with the clauses in Figure 4.

We then have that Theorem 1 holds with $\sim$ replacing $\approx$ throughout and with all formulae, ϕ, drawn from **MBIb**. On the other-hand counterexamples exist to Theorem 2 when $\sim$ is used in place of $\approx$ (and where μ is a non-trivial modification function). We conjecture that, in general, under reasonable conditions, Theorem 2 using $\sim$ does not hold. The results of Proposition 3 hold with $\lesssim$ (resp. $\sim$) replacing $\lessapprox$ (resp. $\approx$) throughout.

The relation $\approx$ is not used when dealing with **MBIb** since then the essential result Theorem 1 does not hold for many non-trivial modification functions. That is, there can be E and F such that $R, E \approx R, F$ for all R but some ϕ such that $R, E, \eta \vDash \phi$ and $R, F, \eta \vDash \neg\phi$, see [15] for details. Indeed $\sim$ should be used whenever we wish to use a logical language that either features the multiplicative modaltities or that features both the multiplicative implication and the additive modalities. To summarize:

1. For the $(\top, I, \wedge, \vee, \neg, \rightarrow, \langle - \rangle, [-], \exists, \forall, \exists_\nu, \forall_\nu)$-fragment of the logical language the use of the local simulation, $\lessapprox$, throughout will be suitable.

2. For any frament featuring $\langle - \rangle_\nu$ or $[-]_\nu$, or $-\!\!*$ together with either of $\langle - \rangle$, $[-]$ the global simulation, $\lesssim$, should be used throughout.

The new logical connectives can be used as follows:

($-\!\!*$). Imagine a situation in which there is some component E that is intended to plug into certain types of system, and imagine that E comes with resources R. Suppose that we wish to guarantee that whenever R, E is plugged into some suitable system S, F, that the resulting compound system makes no accesses to some resource r. This can be expressed by the proposition $R, E \vDash \phi -\!\!* (\neg\psi)$, where ψ is an appropriate 'access r' proposition, as in the previous examples, and ϕ represents the 'suitability' condition.

($[a]_\nu$). Let ψ be as in the previous example. Suppose that we have some system R, E satisfying $[a]_\nu(\neg\psi)$. The resources R often include the permissions of E (as with the ACLs of the earlier examples) and that resource composition takes the union of permissions. Then the logical formula above guarantees that there is no way for E to access the resource r, no matter how its permissions are extended.

($\langle a \rangle_\nu$). We may often wish to specify a system that cannot make a particular access, for example with $R, E \vDash \neg\langle c \rangle\top$ for some access c, because it lacks permission, but such that it, if granted permission it can make the access, e.g $R, E \vDash \langle c \rangle_\nu \top$.

We leave it to the reader to embed these in more 'realistic' situations if these are not regarded as self-evidently practical.

8 Directions

Mild changes to the rules of **SCRP** can result in calculi with significantly different properties. For example, in [14] it is shown how the algebraic and logical theories can be considerably strengthened from the original version presented in [38, 39, 15]. The simple changes made were small changes to the coherence conditions on modification and side-conditions on the operational rules. Those changes result in the *admissibility* of the following rule:

$$\frac{R, E \xrightarrow{a} R', E'}{R \circ S, E \xrightarrow{a} R' \circ S, E'} \quad (R \circ S \downarrow)$$

for all R, R', S, S', E, E'. That is, this rule emerges as a property of systems. We note that it is similar to the *frame rule* of Separation Logic, [21, 40]. When this rule is not admissible, it may be useful to include it explicitly. In particular, this rule leads to algebraic relations like $E \times 1 \sim E$ and for the role constructor, associativity , $E \propto (F \propto G) \sim (E \propto F) \propto G$, for all E, F and G. On the logical side (using $\sim$), we get logical axioms like $\phi * I \leftrightarrow \phi$. It also leads to $(\{F \propto G\}\phi) \leftrightarrow (\{F\}(\{G\}\phi))$ between the $\{-\}$ modality and the role constructor. Despite these useful consequences, we have not chosen to include this frame rule (implicitly or explicitly), since it may not be applicable to all the situations we wish to model. For example, if resources contain access blacklists, and composition joins those lists by non-overlapping union, as above, then the frame rule above would be undesirable. In such situations, a more subtle treatment using the order of the resource monoid should be used, as with the intuitionistic version of **MBI** in [14].

A good deal of work remains to be done on the model-checking problems for **SCRP**-like calculi. These problems are significantly harder than standard model-checking problems, since they involve searches for resource decompositions, searches across processes and appropriate (bi)simulation checks. However, the possession of such model-checking tools would give more powerful reasoning methods for semantically-justified logical access control, in a manner complementary to that of [3, 16].

Together with B. Monahan we are producing a modelling environment *LD2k* for large-scale distributed systems — summaries of this ongoing work may be found in [11, 13]. The tool is an extension of the existing Demos2k tool (`http://www.demos2k.org`). which has firm foundations in the process algebra SCCS. It is intended that *LD2k* will have a process calculus semantics in a variant of **SCRP**. Access control is a property of interest in many of the models we wish to consider and the work presented herein is a foundational study related to such models. However, in the models we wish to consider resources will typically be physicallly distributed and it will be pragmatic to have *location* play a role as fundamental as that of resource. Location should be a first-class citizen in the underlying process calculus, in the same way that resource is a first-class citizen in **SCRP**. The paper [39] contains some basic ideas about the embodiment of the location concept in a process algebra with transitions of the form $L, R, E \xrightarrow{a} L', R', E'$, where L, L' are locations, R, R' are resources and E, E' are processes. A fuller treatment is contained in [12].

Acknowledgements

We are grateful to Brian Monahan and Jonathan Hayman for their help and suggestions. Guy McCusker suggested the plugging example for $-\!\!*$ in Section 7 in a related discussion. We thank also Chris Tofts for helpful contributions related to this work.

References

[1] M. Abadi. Logic in access control. In *Proc. LICS'03*, pages 228–233, 2003.

[2] M. Abadi. Access control in a core calculus of dependency. In *Computation, Meaninig and Logic: articles dedicated to Gordon Plotkin*, volume 172 of *Electronic Notes in Theoretical Computer Science*, pages 5–31. Elsevier, 2007.

[3] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Progrogramming Languages and Systems*, 4(15):706–734, 1993.

[4] M. Abadi and D. Garg. A modal deconstruction of access control logics. In *FOSSACS 2008*, pages 216–230. Springer-Verlag, 2008.

[5] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. Conf. Comp. Comm. Security*, pages 36–47. ACM Press, 1997.

[6] G. Birtwistle. *Demos — discrete event modelling on Simula*. Macmillan, 1979.

[7] G. Birtwistle, R. Pooley, and C. Tofts. Characterising the structure of simulations using CCS. *Transactions of the Simulation Society*, 10(3):205–236, 1993.

[8] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[9] P. Bonatti, S. De Capitani Di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security*, 5(1), 2002.

[10] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proc. Royal Soc. A*, 426(1871):233–271, 1989.

[11] M. Collinson, B. Monahan, and D. Pym. Located Demos2k — Towards a Tool for Modelling Processes and Distributed Resources. Technical Report HPL-2008-76, Hewlett Packard Laboratories, 2008. Available at `http://library.hp.com/techpubs/2008/HPL-2008-76.html`.

[12] M. Collinson, B. Monahan, and D. Pym. A Logical and Computational Theory of Located Resource. Technical Report HPL-2008-74R1, Hewlett Packard Laboratories, 2008. To appear in the *Journal of Logic and Computation*. Available at `http://library.hp.com/techpubs/2008/HPL-2008-74R1.html`.

[13] M. Collinson, B. Monahan, and D. Pym. An Update to Located Demos2k. Technical Report HPL-2008-205, Hewlett Packard Laboratories, 2008. Available at `http://library.hp.com/techpubs/2008/HPL-2008-205.html`.

[14] M. Collinson and D. Pym. Algebra and logic for resource-based systems modelling. Technical Report HPL-2009-21, Hewlett Packard Laboratories, 2009. To appear in *Mathematical Structures in Computer Science*. Available at `http://library.hp.com/techpubs/2009/HPL-2009-21.html`.

[15] M. Collinson, D. Pym, and C. Tofts. Errata for Formal Aspects of Computing (2006) 18:495–517 and their consequences. *Formal Aspects of Computing*, 19(4):551–554, 2007.

[16] J. DeTreville. Binder, a logic-based security language. In *Proc. 2002 IEEE Symp. Security and Privacy*, pages 105–113, 2002.

[17] D. Dolev and A. Yao. On the security of public key protocols. In *Proc. IEEE 22nd Symp. Found. Comp. Sci.*, pages 350–357, 1981.

[18] R. Focardi, S. Rossi, and A. Sabelfeld. Bridging language-based and process calculi security. In *Proc. FOSSACS'05*, number 3441 in LNCS, pages 299–315. Springer, 2005.

[19] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[20] C. Hoare. *Communicating sequential processes.* Prentice-Hall, 1985.

[21] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. POPL 2001*, pages 14–26. ACM, 2001.

[22] S. Kripke. Semantical analysis of modal logic I. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[23] B. Lampson. Protection. In *Proc. Fifth Princeton Symp. Information Sciences and Systems*, pages 437–443, 1971.

[24] B. Lampson. Computer security in the real world. *IEEE Computer*, 6(37):37–46, 2004.

[25] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 4(10):265–310, 1992.

[26] H. Mantel. On the composition of secure systems. In *Proc. IEEE Symp. Security and Privacy.* IEEE Computer Society, 2002.

[27] W. Mao. *Modern Cryptography: Theory and Practice.* Prentice Hall, 2004.

[28] J. Millen. The interrogator model. In *IEEE Symposium on Security and Privacy*, pages 251–260, 1995.

[29] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[30] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[31] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[32] P. O'Hearn. Resources, concurrency and local reasoning. *Theoretical Computer Science*, 375(1–3):271–307, 2007.

[33] P. O'Hearn and D. Pym. The logic of bunched implications. *Bull. Symb. Logic*, 5(2):215–244, 1999.

[34] G. Plotkin. Structural operational semantics. *Journal of Logic and Algebraic Programming*, 60:17–139, 2004. Original manuscript 1981.

[35] D. Pym. On bunched predicate logic. In *Proc. LICS'99*, pages 183–192. IEEE, 1999.

[36] D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002. Errata at: `http://www.cs.bath.ac.uk/~pym/BI-monograph-errata.pdf`.

[37] D. Pym, P. O'Hearn, and H. Yang. Possible worlds and resources: The semantics of **BI**. *Theoretical Computer Science*, 315(1):257–305, 2004.

[38] D. Pym and C. Tofts. A calculus and logic of resources and processes. *Formal Aspects of Computing*, 18(4):495–517, 2006. Errata in [15].

[39] D. Pym and C. Tofts. Systems Modelling via Resources and Processes: Philosphy, Calculus, Semantics, and Logic. In L. Cardelli, M. Fiore, and G. Winskel, editors, *Computation, Meaning and Logic: articles dedicated to Gordon Plotkin*, volume 107 of *Electronic Notes in Theoretical Computer Science*, pages 545–587. Elsevier, 2007. Errata in [15].

[40] J. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proc. LICS'02*, pages 55–74. IEEE, 2002.

[41] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols.* Addison-Wesley, 2001.

[42] G. Salaün, M. Allemand, and C. Attiogbé. Specification of an access control system with a formalism combining CCS and CASL. In *Proc. IPDPS'02*. IEEE, 2002.

[43] J. Saltzer and M. Shroeder. The protection of information in computer systems. *Proc. IEEE*, 63(9):1278–1308, 1975.

[44] A. Scedrov, J. Mitchell, A. Ramanathan, and V. Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353:118–164, 2006.

[45] S. Schneider. Security properties and CSP. In *IEEE Symp. Security and Privacy*, pages 174–187, 1996.

[46] C. Stirling. *Modal and temporal properties of processes.* Springer, 2001.

[47] C. Tofts. Processes with probabilities, priority and time. *Formal Aspects of Computing*, 6:536–564, 1994.

[48] C. Tofts. Efficiently modelling resource in a process algebra. Technical Report HPL-2003-181, Hewlett-Packard Laboratories, 2003. Available at: `http://www.hpl.hp.com/techreports/2003/HPL-2003-181.pdf`.

[49] C. Tofts. Process algebra as modelling. *Electronic Notes in Theoretical Computer Science*, 162:323–326, 2006. Proceedings of the Workshop “Essays on Algebraic Process Calculi” (APC25).

[50] D. Wijesekera and S. Jajodia. Policy algebras for access control — the predicate case. In *9th ACM Conference on Computer and Communications Security*, pages 171–180, 2002.