Dept. of Computing and Inf. Sc. Technical Report 1997-405

AN ACTION-BASED LOGIC OF CAUSALITY, KNOWLEDGE, PERMISSION AND OBLIGATION [1]

Glenn MacEwen
Department of Computing and Information Science
Queen's University, Kingston, Ontario K7L 3N6 Canada
Xiao Jun Chen
Department of Computer Science, University of Ottawa
Scott Knight
Department of Electrical and Computer Engineering
The Royal Military College of Canada
E-mail: macewen@qucis.queensu.ca
Phone: 613-545-6052 Fax: 613-545-6513

Abstract

The critical properties required of secure systems can vary widely in their nature from application to application. Different properties required within the same system can result in interactions that are difficult to understand. In addition, the integration of systems having different properties can cause unanticipated interactions. The *Critical System Logic* (CSL) addresses these problems by providing a common language for specifying a wide range of security-related properties so that their interactions can be analyzed. CSL extends ACTL, an action-based propositional branching time temporal logic. It contains elements of epistemic logic to reason about knowledge, temporal logic to reason about time and causality, and deontic logic to reason about permission and obligation. Permission and obligation are given semantics determined by role-based access and visibility control. Examples, including those of access control in medical system records and multi-level secrecy, are given to demonstrate the expressiveness of the logic and to argue the utility of formal reasoning in such applications.

Key Words: security, formal methods, requirements specification, temporal logic, deontic logic, knowledge, permission, roles.

[1]This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Università degli Studi di Roma *La Sapienza*. It was initiated by the first two authors during 1995 at the Università di Roma while the first was there on sabbatical.

1 Introduction

An important part of requirements specification for critical systems with security requirements is a precise expression of those security properties that are fundamental and formalizable. The nature of such properties can vary widely over issues of *secrecy*, *integrity*, *legitimate use* and *availability*. Secrecy pertains to the prevention of unauthorized information flows through a system. Integrity pertains to the protection of system state from unauthorized modification. Legitimate use pertains to the prevention of unauthorized use of system resources. Availability pertains to the prevention of unauthorized interference with use of system resources. Security can mean any combination of these concerns.

In addition to having an inherently large scope, security properties can have aspects that overlap onto other domains of behaviour such as fault-tolerance. Consequently, it is highly desirable to have available a formal requirements specification language that is able to express this wide range of properties. Furthermore, if a number of critical properties are specified the danger of inadvertent undesirable interactions arises. It is necessary therefore to be able to analyze such collections of properties and to derive properties from them.

Finally, the integration of systems having security property requirements raises similar problems of inadvertent interactions. Here the problem is often more difficult to address because the security properties of the component systems being integrated may have been specified in different languages using different concepts thus making the analysis of interaction very difficult.

We view the four security property classes, secrecy, integrity, legitimate use and availability, as being the *basic security property classes*, while other property classes such as identification, authentication, access control, data privacy, data integrity and digital signatures are *supporting security mechanisms*. The reason for this is that the first four seem to be more properly part of requirements specification while the latter are properties often specified in ways to enforce the basic properties. For example, in order to correctly enforce a secrecy property with respect to a particular user, that user needs to be identified. And access control is often used as a mechanism to enforce secrecy.

The *Critical System Logic* (CSL) was developed to provide a common language for specifying critical properties of systems, primarily from a computer security viewpoint. It is intended for the specification and analysis of properties over a wide range of security concerns. It is also specifically designed to be at a sufficiently abstract level so that it can be used to express properties that may have been originally specified in different formalisms so that they can be analyzed together. At the same time, the intended domain of application of CSL is much larger than computer security alone. The reason for this, from a computer security perspective, is that security properties can be compromised by behaviour that is not accounted for in formulating the security properties themselves. For example, CSL allows the expression of some aspects of fault-tolerance. Extensions to provide hard-real-time properties remain an objective of future work.

Most current and proposed formal security policy definitions deal with only one of the security policy classes. For example, see [18, 34, 25] for secrecy, [7, 5, 2] for integrity, and [19, 33, 1] for legitimate access. There has been work in integrating secrecy and integrity [24], and secrecy under special (possibly faulty) behaviour [4]. We are not aware of a formal definition of availability. CSL is an attempt to provide a way of expressing at least some aspects of all of these concerns. In this, we have been successful with respect to secrecy, integrity and legitimate access. Some limited properties of availability and fault behaviour are also expressible.

CSL builds on Computation Tree Logic (CTL) which is interpreted on Kripke Structures and is a well-known temporal logic used for state-based logical reasoning [15]. The Action-based version of CTL (ACTL) [29] has similar syntactic and semantic styles as CTL, but it is interpreted on labelled transition systems (LTS's) and thus permits us to express properties in terms of sequences of actions and to work on action-based logical reasoning [30].

In summary, CSL is a formal method intended for requirements specification and analysis of critical systems. It extends ACTL with the following primitive notions: active agents called *subjects*; action names called *commands*; visibility of commands; the enabling and disabling of authorization for subjects to invoke commands; the masking and unmasking of authorization for subjects to view the occurrence of commands; and the backwards temporal operators *Since* and *Last*. The three major security modalities, *permission* (and its dual *obligation*), *causality* and *knowledge*, are derived definitions. *Roles* are introduced to express the time dependent set of access and view authorizations possessed by a subject. The semantics is interpreted on LTS's. The logic is motivated in part by an earlier Security Logic called

SL [17] which provided a vehicle to investigate the relationship between the knowledge and permission modalities. However, that logic was not given a clear semantics. Indeed, the semantics of permission were left completely open.

The set of *subjects* models the active entities in any universe of discourse. These can be users, processes, programs, accounts or whatever other entity it is useful to model for expressing security. The *formulae* of the logic express assertions about the state of the system under specification. Knowledge allows one to specify that a subject *knows* that a formula is true. Causality allows one to specify that a subject can *cause* a formula to be true. The temporal logic allows the truth of a formula to be time dependent. *Obligation* and *permission* are properties specifying an authorization for a formula; the first says that a formula is not authorized to be false, and the latter says that a formula is authorized to be true. Although there have been formal treatments of roles, e.g. [12], we are not aware of an action-based approach, nor of an approach incorporating view authorizations in addition to access authorizations. For a formula to be obligated means that if the formula is not true in some system state then there is some other state which is identical except that the formula is true and which is deemed to be a better state. Obligation and permission are duals; permission for a formula to be true means that it is not obligated to be false.

The remainder of the paper is organized as follows. The next section presents the computational model, which is an extended labelled transition system. Section 3 gives the syntax and semantics of the logic. Section 4 extends the logic with derived operators which include those for causality, knowledge, permission, obligation and roles. Section 5 discusses some of the common problematic properties of deontic logics which do not occur in CSL; it also gives some short examples of specification and of properties that are of general verification interest. Section 6 presents some security applications, and the last section contains suggestions for future work and some comments about related work.

2 Computational Model

CSL is interpreted on labelled transition systems, which have been widely used as operational semantic models of distributed and concurrent systems to formalize the activities and changes of a system [28, 32]. In particular, they offer a simple and convenient model to explicitly describe causality and choice, the two main aspects of interest in distributed systems. Causality refers to the fact that certain events in a distributed system can only occur in a fixed order; for example, a message can be received only after it has been sent. Choice captures the fact that systems can behave in a non-determinate fashion. In other words, at certain points of the computation, the system may choose between alternative events, leading to different behaviours. Formally, a *Labelled Transition System* (LTS) is a tuple $\mathcal{A} = (Q, A, \rightarrow, 0_Q)$ where:

- Q is a set of states;
- A is a finite non-empty set of *actions*;
- $\rightarrow \subseteq Q \times A \times Q$ is the *transition relation*: an element $(r, a, q) \in \rightarrow$ is called a *transition* and is written as $r \stackrel{a}{\rightarrow} q$;
- 0_Q is the initial state.

Typically, Q is a set of states or configurations: states of a database, program states, states of knowledge, configurations of a machine and so on. A transition $r \stackrel{a}{\rightarrow} q$ represents the fact that state r may become q due to or under the influence of a which could be an action, a period of time, a person or whatever. In CSL, we use labelled transition systems to model commands invoked to change the state of some system of interest, which may or may not be a distributed system.

Computation is defined on paths over a LTS. In the following, let $r, q, s, \ldots$ range over Q and $a, b, \ldots$ range over A. Paths over a LTS $\mathcal{A} = (Q, A, \rightarrow, 0_Q)$ are defined as follows:

- A sequence $(q_0, a_0, q_1)(q_1, a_1, q_2) \ldots \in \rightarrow^\infty$ is called a *path* from q_0; a path that cannot be extended, i.e. is infinite or ends in a state without outgoing transitions, is called a *fullpath*; the *empty path* consists of a single state $q \in Q$ and is denoted by q;
- If $\pi = (q_0, a_0, q_1)(q_1, a_1, q_2) \ldots$ we denote the starting state q_0 of the sequence by $\mathrm{first}(\pi)$ and the last state in the sequence (if the sequence is finite) by $\mathrm{last}(\pi)$; if π is an empty path (i.e. $\pi = q$, then $\mathrm{first}(\pi) = \mathrm{last}(\pi) = q$;

- Concatenation of paths is denoted by juxtaposition: $\pi = \rho\theta$; it is only defined if ρ is finite and $\text{last}(\rho) = \text{first}(\theta)$.

- When $\pi = \rho\theta$ we say that θ is a *suffix* of π and that it is a *proper suffix* if $\rho \neq q, q \in Q$.

- When $\sigma = \rho\xi$ we say that ρ is a *prefix* of σ and that it is a *proper prefix* if $\xi \neq q, q \in Q$.

A prefix of a path from 0_Q is called an *initial-path.* If ρ is an initial-path, *path(ρ)* is the set of fullpaths of which ρ is a prefix. In the following, let ρ range over initial-paths, σ range over fullpaths, θ, η range over suffixes of initial-paths, and ξ range over suffixes of fullpaths.

Actions play an important role in computation; in fact, we are interested in reasoning about sequences of actions to capture the behaviour of a system. To define actions, we assume sets of subjects and commands. The set $\mathcal{U}$ of *subjects* includes the initial subject boot. One can think of these as the active entities of interest: processes, processors, computers, humans, terminals, etc. The set $\mathcal{C}$ of *commands* includes the initial command init. One can think of these as the ways in which a subject can receive or issue a communication: interactive terminal commands, operating system commands, application language operations, message types, etc.

2.1 Actions and Security

The logic permits reasoning about static, i.e. mandatory, security policies. The capability of a subject to arbitrarily invoke a command or to view the occurrence of a command can be restricted by specifications given in the language of the logic. Such restrictions limit the evolution of paths, and thus the behaviour of the system under specification. The restrictions on what commands a subject is authorized to invoke or view in any state, given by the specification, form the basis of a mandatory security policy.

In addition, the logic permits reasoning about dynamic policies through roles; one can think of a *role* as the status of some subject as it acts. A subject can have many roles, and can share each with other subjects. A role defines the capability of a subject to act and to view. Examples of roles are: message sender, system administrator, auditor, authorized user, system programmer, trusted system process, I/O processor, etc. A role assigned to a subject confers to the subject a subset of commands that the subject is authorized to invoke, and a subset of commands that the subject is authorized to view. In the computational model, then, a role is simply a set of command authorizations. To capture roles we have four deontic relations, the first two of which are $\Uparrow \subseteq \mathcal{C} \times (\mathcal{U} \times \mathcal{C})$ and $\Downarrow \subseteq \mathcal{C} \times (\mathcal{U} \times \mathcal{C})$. $c \Uparrow (i, c')$ means that immediately following the occurrence of c, i is authorized to invoke command c'. $c \Downarrow (i, c')$ means that immediately following the occurrence of c, all current authorizations for i to invoke command c' are revoked. We call a pair $(c, (i, c')) \in \Uparrow$ an *access-authorization* and a pair $(c, (i, c')) \in \Downarrow$ an *access-forbiddance*. The effect of an access-authorization (access-forbiddance) persists until a corresponding access-forbiddance (access-authorization) occurs. An access-authorization (access-forbiddance) $(c, (i, c'))$ is said to *enable* (*disable*) the (subject,command) pair (i, c').

The final two of the four deontic relations are $\triangle \subseteq \mathcal{C} \times (\mathcal{U} \times (\mathcal{U} \times \mathcal{C}))$ and $\bigtriangledown \subseteq \mathcal{C} \times (\mathcal{U} \times (\mathcal{U} \times \mathcal{C}))$. $c \triangle (i, (j, c'))$ means that immediately following the occurrence of c, i is authorized to view the invocation by j of command c'. $c \bigtriangledown (i, (j, c'))$ means that immediately following the occurrence of c, all current authorizations for i to view the invocation by j of command c' are revoked. We call a pair $(c, (i, (j, c'))) \in \triangle$ a *view-authorization* and a pair $(c, (i, (j, c'))) \in \bigtriangledown$ a *view-forbiddance*. The effect of a view-authorization (view-forbiddance) persists until a corresponding view-forbiddance (view-authorization) occurs. A view-authorization (view-forbiddance) $(c, (i, (j, c')))$ is said to *unmask* (*mask*) the viewing by subject i of j invoking command c'.

These four relations are also intended to capture the deontic interdependence of commands independent from subjects, what we call *safety security*; the occurrence of a certain command can mean that some other command should not occur until after some third command occurs. For example, in some system the occurrence of an off command might mean that an on command should not occur until after a reset command has occurred. Under a safety-based policy an authorized command is authorized for all subjects, and a forbidden command is forbidden for all subjects.

An action is a tuple $a = (S, cm, R, E, D, U, M)$ where

- $S \subseteq \mathcal{U}$, the set of subjects invoking cm; $|S| > 1$ represents a *synchronization* in which all the subjects in S jointly invoke cm,
- $cm \in \mathcal{C}$, the associated command,

- $R \subseteq \mathcal{U}$, the set of subjects that view (receive or observe) cm,
- $E \subseteq \mathcal{U} \times \mathcal{C}$, the set of pairs (i, c) such that $cm \Uparrow (i, c)$,
- $D \subseteq \mathcal{U} \times \mathcal{C}$, the set of pairs (i, c) such that $cm \Downarrow (i, c)$,
- $U \subseteq \mathcal{U} \times (\mathcal{U} \times \mathcal{C})$, the set of pairs $(i, (j, c))$ such that $cm \bigtriangleup (i, (j, c))$,
- $M \subseteq \mathcal{U} \times (\mathcal{U} \times \mathcal{C})$, the set of pairs $(i, (j, c))$ such that $cm \bigtriangledown (i, (j, c))$,

and where E and D, and U and M, are respectively disjoint, and $S \subseteq R$. In addition, if $(i, c) \in E$ then $(i, (i, c)) \in U$ and if $(i, (i, c)) \in M$ then $(i, c) \in D$, since, respectively, a subject invoking a command is aware of that fact so an enabled command should be viewable and a masked command should not be invokable. The initial command init is implicitly authorized.

For CSL we define an *Extended Labelled Transition System* (ELTS) as a tuple $\mathcal{A} = (Q, A, \rightarrow, 0_Q, 0_A)$, where the first four components are the same as for a LTS and 0_A is an initial action of the form $(\{\mathsf{boot}\}, \mathsf{init}, \{\mathsf{boot}\}, E, \{\}, U, \{\})$. E is the set of initially enabled (subject,command) pairs. U is the set of initially unmasked (subject,(subject,command)) pairs. The following two conditions hold: (i) $\mid \{q \mid 0_Q \stackrel{0_A}{\rightarrow} q\} \mid > 0$; (ii) if $0_Q \stackrel{a}{\rightarrow} q$ then $a = 0_A$.

Note that in our setting the notion of the visibility of actions, which has been widely discussed in process theory (see e.g. [28]), is implicitly and statically defined in terms of specific subjects; i.e. any action not visible to subject i is silent with respect to i.

3 The Logic

As with ACTL, we start with a small logic of actions which is embedded within CSL. This logic permits the specification of a class of actions by constraining the components of members of the class. In the following, let $i, j, \ldots$ range $\mathcal{U}$ and $c, d, \ldots$ range over $\mathcal{C}$. We write $a.X$ to denote the component X of action a. Given $\mathcal{U}$ and $\mathcal{C}$, the set of action formulae $\mathcal{L}_A$ is given by the following BNF grammar:

$\chi ::= \mathsf{true} \mid \neg\chi \mid \chi \wedge \chi \mid sb(i) \mid cm(c) \mid rc(i) \mid en(i, c) \mid ds(i, c) \mid um(i, (j, c)) \mid ms(i, (j, c))$

We write false for $\neg\mathsf{true}$. The operators $\neg$ and $\wedge$ have the usual meaning. The mnemonics *sb, cm, rc, en, ds, um* and *ms*, respectively, abbreviate the words *subject, command, receiver, enables, disables, unmasks* and *masks*.

The semantics for the action formulae $\mathcal{L}_A$ is given by means of the satisfaction relation $\models \subseteq A \times \mathcal{L}_A$ inductively defined as follows:

$a \models \mathsf{true}$	always	
$a \models \neg\chi$	iff	$a \not\models \chi$
$a \models \chi \wedge \chi'$	iff	$a \models \chi$ and $a \models \chi'$
$a \models sb(i)$	iff	$i \in a.S$
$a \models cm(c)$	iff	$a.cm = c$
$a \models rc(i)$	iff	$i \in a.R$
$a \models en(i,c)$	iff	$(i,c) \in a.E$
$a \models ds(i,c)$	iff	$(i,c) \in a.D$
$a \models um(i,(j,c))$	iff	$(i,(j,c)) \in a.U$
$a \models ms(i,(j,c))$	iff	$(i,(j,c)) \in a.M$

Example 1: The action formula $(sb(i) \vee sb(j)) \wedge cm(c) \wedge rc(k) \wedge en(k,d) \wedge um(k,(l,d))$ specifies the class of all actions that model command c invoked by either or both of subjects i and j, which are visible to subject k as well as the invokers i,j, which enables command d for subject k, and which unmasks l invoking d for subject k.

Notice that any action formula of the form $cm(c) \wedge cm(d)$, for $c \neq d$, is unsatisfiable, and also that $sb(i) \Rightarrow rc(i)$, $en(i,c) \Rightarrow um(i,(i,c))$ and $ms(i,(i,c)) \Rightarrow ds(i,c)$ are axioms.

We now give the main CSL logic in which the logic of actions is embedded. The language of CSL formulae is given by the following grammar:

$$\phi ::= \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \exists\gamma \mid \forall\gamma \mid \overleftarrow{\mathsf{X}}_\chi\phi \mid \phi_\chi\overleftarrow{\mathsf{U}}_\chi\phi \mid \mathsf{K}_i\phi \mid \mathsf{A}_i^c \mid \mathsf{V}_i^{j,c}$$

$$\gamma ::= \mathsf{X}_\chi\phi \mid \phi_\chi\mathsf{U}_\chi\phi \mid \phi_\chi\mathsf{U}\phi$$

As with action formulae, we write false for $\neg\mathsf{true}$ and the operators $\neg$ and $\wedge$ have the usual meaning. A formula of the form ϕ, called an *initial-path formula*, specifies a state property. A formula of the form γ, called a *fullpath*

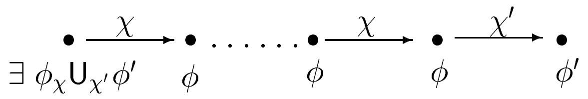

Figure 1: Visualization for *Until* Modality.

formula, specifies a property of future time. $\exists\gamma$ asserts that there exists a future path with the property γ. $\forall\gamma$ asserts that γ holds for all future paths.

$\mathsf{X}_\chi \phi$ (pronounced *Next*) asserts that the next transition involves an action that satisfies χ and that after the action ϕ holds. Figure 1 illustrates the meaning of U, (pronounced *Until*). $\phi_\chi \mathsf{U}_{\chi'} \phi'$ means that ϕ holds now and eventually an action satisfying χ' will occur after which ϕ' will hold, and until that time all actions will satisfy χ and all states will satisfy ϕ. The alternate version of *Until*, $\phi_\chi \mathsf{U} \phi'$, is similar except that until ϕ' holds all actions will satisfy χ and all states will satisfy ϕ; this includes the case that ϕ' holds in the current state.

$\overleftarrow{\mathsf{X}}_\chi \phi$ (pronounced *Last*) is the backwards analog of $\mathsf{X}_\chi \phi$. $\overleftarrow{\mathsf{X}}_\chi \phi$ asserts that the previous transition involved an action that satisfied χ and that before the action ϕ held. $\overleftarrow{\mathsf{U}}$ (pronounced *Since*) is the backwards analog of U. $\phi_\chi \overleftarrow{\mathsf{U}}_{\chi'} \phi'$ means that at some time in the past, an action satisfying χ' has occurred. Before that action, the state satisfied property ϕ' and after that action (since then) all the states satisfied ϕ and all the actions satisfied χ.

$\mathsf{K}_i \phi$ (pronounced i *Knows* ϕ) asserts that subject i knows fact ϕ. This operator is explained in more detail after we show the semantics. A_i^c and $\mathsf{V}_i^{j,c}$ form the basis for permission; A_i^c means that subject i is access-authorized to invoke command c and $\mathsf{V}_i^{j,c}$ means that subject i is view-authorized to view subject j invoking command c.

The formal semantics of CSL formulae is given below. *Satisfaction* of an initial-path formula ϕ (fullpath formula γ) by an initial-path ρ (fullpath σ with prefix ρ), notation $\rho \models_{\mathcal{A}} \phi$ or just $\rho \models \phi$ ($\rho, \sigma \models_{\mathcal{A}} \gamma$ or just $\rho, \sigma \models \gamma$), is given inductively by:

$\rho \models \mathsf{true}$	always
$\rho \models \neg\phi$	iff $\rho \not\models \phi$
$\rho \models \phi \wedge \phi'$	iff $\rho \models \phi$ and $\rho \models \phi'$
$\rho \models \exists\gamma$	iff there exists a fullpath $\sigma \in path(\rho)$ such that $\rho, \sigma \models \gamma$

$\rho \models \forall\gamma$	iff for all fullpaths $\sigma \in path(\rho)$, $\rho, \sigma \models \gamma$
$\rho \models \overleftarrow{\mathsf{X}}_{\chi}\phi$	iff $\rho = \theta(q, a, q')$ and $a \models \chi$ and $\theta \models \phi$
$\rho \models \phi_{\chi}\overleftarrow{\mathsf{U}}_{\chi'}\phi'$	iff $\rho \models \phi$ and there exists $\theta = \theta'(q, a, q')$, prefix of ρ, such that $\theta' \models \phi'$, $a \models \chi'$ and for all $\eta = \eta'(r, b, r')$, prefixes of ρ, of which θ is a proper prefix, we have $\eta' \models \phi$ and $b \models \chi$
$\rho \models \mathsf{K}_i\phi$	iff $\rho \models \mathsf{true}_{\neg rc(i)}\overleftarrow{\mathsf{U}}_{cm(\mathsf{init})}(\neg\exists(\mathsf{true}_{\neg rc(i)}\mathsf{U}\neg\phi))$ or there exists some command c such that $\rho \models \mathsf{true}_{\neg rc(i)}\overleftarrow{\mathsf{U}}_{rc(i)\wedge cm(c)\wedge\neg cm(\mathsf{init})}$ $\mathsf{K}_i(\neg\exists(\mathsf{true}_{\neg rc(i)}\mathsf{U}_{rc(i)\wedge cm(c)}(\exists(\mathsf{true}_{\neg rc(i)}\mathsf{U}\neg\phi))))$
$\rho \models \mathsf{A}_i^c$	iff there exists subject k and command d such that $\rho \models (\mathsf{true}_{\neg ds(i,c)}\overleftarrow{\mathsf{U}}_{en(i,c)\wedge cm(d)\wedge sb(k)}\ \mathsf{true})$ and either $d = \mathsf{init}$ or for every subject k' and command d' $\rho \models \neg(\mathsf{true}_{\neg ds(i,c)}\overleftarrow{\mathsf{U}}_{en(i,c)\wedge cm(d')\wedge sb(k')}\ \neg\mathsf{A}_{k'}^{d'})$
$\rho \models \mathsf{V}_i^{j,c}$	iff there exists subject k and command d such that $\rho \models (\mathsf{true}_{\neg ms(i,(j,c))}\overleftarrow{\mathsf{U}}_{um(i,(j,c))\wedge cm(d)\wedge sb(k)}\ \mathsf{true})$ and either $d = \mathsf{init}$ or for every subject k' and command d' $\rho \models \neg(\mathsf{true}_{\neg ms(i,(j,c))}\overleftarrow{\mathsf{U}}_{um(i,(j,c))\wedge cm(d')\wedge sb(k')}\ \neg\mathsf{A}_{k'}^{d'})$
$\rho, \sigma \models \mathsf{X}_{\chi}\phi$	iff $\sigma = \rho'\xi$ and $\rho' = \rho(q, a, q')$ and $\rho' \models \phi$ and $a \models \chi$
$\rho, \sigma \models \phi_{\chi}\mathsf{U}_{\chi'}\phi'$	iff $\rho \models \phi$ and $\sigma = \rho\xi$ and there exists $\theta = \theta'(q, a, q')$, prefix of ξ, such that $\rho\theta \models \phi'$, $a \models \chi'$ and for all $\eta = \eta'(r, b, r')$, prefixes of ξ, which are proper prefixes of θ, we have $\rho\eta \models \phi$ and $b \models \chi$
$\rho, \sigma \models \phi_{\chi}\mathsf{U}\phi'$	iff $\sigma = \rho\xi$ and there exists a prefix η of ξ such that $\rho\eta \models \phi'$ and for all $\theta = \theta'(r, b, r')$ which are prefixes of η, we have $\rho\theta' \models \phi$ and $b \models \chi$

The *Since* modality $\overleftarrow{\mathsf{U}}$ plays an important role in our logic because it allows us to derive useful information about the past. The following example shows a way of using $\overleftarrow{\mathsf{U}}$ to express the fact that a command c has been seen by subject i.

> **Example 2:** A user sees any message that he receives. An initial-path ρ holds the sequence of message sending actions, and thus also the information about these received messages. For each user i, only some of the messages are visible. Recall that subscripting a modal operator with an action class $rc(i)$ specifies that the

action is visible to i; in the last state of ρ, the string of messages that i has seen can be defined by

(i) $seen_i(\varepsilon) = \mathsf{true}_{\neg rc(i)} \overleftarrow{\mathsf{U}}_{sb(\mathsf{boot})} \mathsf{true}$

(ii) $seen_i(sc) = \mathsf{true}_{\neg rc(i)} \overleftarrow{\mathsf{U}}_{cm(c) \wedge rc(i)} seen_i(s)$

where in (i), ε denotes the empty string: user i sees nothing iff after the initial action (identified by action formula $sb(\mathsf{boot})$), no action is visible to i (expressed by action formula $\neg rc(i)$). (ii) expresses the fact that i has seen a string s of messages concatenated with message c iff there is a state in the past in which i has seen s, from which it received visible message c (expressed by $cm(c) \wedge rc(i)$) and after which no other messages are visible.

It is easy to see that, by means of $\overleftarrow{\mathsf{U}}$ and action formulae, we can also derive other useful information about the past. Such information can be important to the security of behaviours in the future. For example, the access rights of user i depend on how he has logged in. Furthermore, deriving information via $\overleftarrow{\mathsf{U}}$ gives us the ordering of past events. This is important when we need to select among the messages according to the order in which they appeared. For example, if a user first logged in as a system manager then as a normal user, his authorization should be based on the latest login.

Observations with respect to a subject i in an initial-path ρ form the basis both for controlling i's future behaviours, and for obtaining its *knowledge*. Intuitively, the knowledge of a subject i in a given state is determined by all those states that, from i's perspective, appear equivalent in the sense that any one of them could possibly be the actual current state. That is, i cannot distinguish equivalent states using only the partial state observable to it. A subject i knows a fact ϕ, denoted $\mathsf{K}_i\phi$, if ϕ is true in all the states that appear equivalent to i [22].

It is easy to see that a subject i cannot distinguish exactly those states with past paths (histories) having the same observations with respect to i. Therefore, i knows a fact ϕ in a state s iff all the states having the same observations (in the past) w.r.t. i as does s, satisfy ϕ. This can be characterized by using $\overleftarrow{\mathsf{U}}$ to go back to the initial state and then using U to proceed forward to find all the states having these observations, skipping through all actions invisible to i.

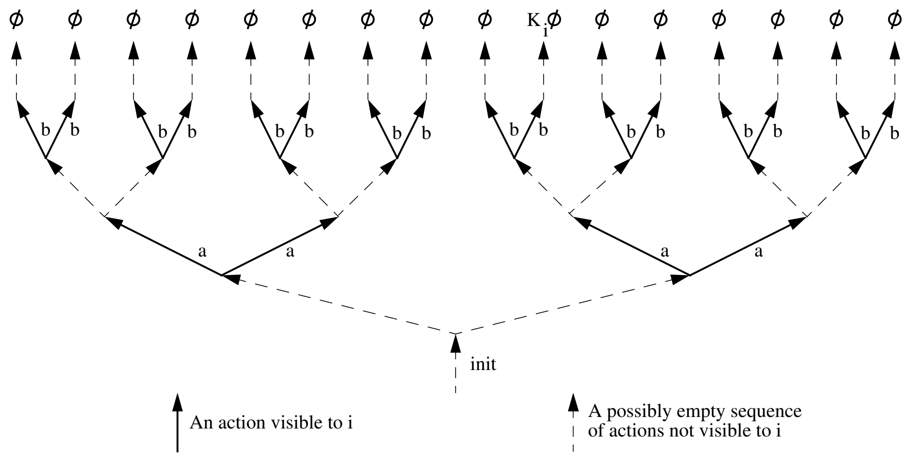

Figure 2: Example to Help Visualize the Knowledge Modality

The semantic definition of $\mathsf{K}_i\phi$ has a recursive style similar to the definition of $seen_i$ in Example 2. Figure 2 shows an example to help visualize and explain the definition of knowledge. The tips of the arrows at the top of the diagram represent all the states having histories with the same subsequence of actions visible to i, namely ab. Actions visible to i are shown with solid arrows and (possibly empty) sequences of actions not visible to i are shown with dotted arrows. The initial action init is not visible. One arrow tip is labelled with $\mathsf{K}_i\phi$ to represent a state for which one wants to check if i knows ϕ. In order to check that fact one must find all the other states with identical visible histories, i.e. those represented by the upper arrow tips, and check that ϕ is satisfied in all of them.

The search for all such equivalent states starts by moving backwards through the history, ignoring invisible actions, until a visible action is found. At this point one records the visible command name, b in the example, and moves forward in all possible suffixes identical to the one just traversed backwards, to ensure that none ends in a state that does not satisfy ϕ. The search also continues backwards recursively to find the next visible action, a in the example, and carries out the same check at that point. The recursive search ends when the initial action init is reached.

Fundamental to security in CSL are the facts that a particular subject i is access-authorized for command c in the current state, and that a particular subject i is view-authorized for the subject-command pair (j, c) in the current state. These two facts, respectively, are denoted by A_i^c and $\mathsf{V}_i^{j,c}$. For A_i^c to hold the (subject,command) pair (i, c) must have been enabled by some access-authorization $(d, (i, c))$. Also, a command in such an authorization must, itself, have been authorized for all invoking subjects when it occurred, and so on back in the current history until the initial command init is reached. To determine whether or not a pair (i, c) has been enabled it is necessary to scan backwards along the history to determine that the last relevant action was an enabling action. The scan then continues recursively to determine that all the (subject,command) pairs in the enabling actions in the relevant sequence were themselves enabled. The search for unmasking is analogous.

4 Derived Operators

The usual derived operators of temporal logics can be defined. For example, for the two quantifications of *Eventually* we write

$$\forall \mathsf{F}\phi \quad \stackrel{def}{\equiv} \quad \forall(\mathsf{true}_{\mathsf{true}}\mathsf{U}\phi)$$

and

$$\exists \mathsf{F}\phi \quad \stackrel{def}{\equiv} \quad \exists(\mathsf{true}_{\mathsf{true}}\mathsf{U}\phi)$$

The two quantifications of *Always* are then defined by

$$\forall \mathsf{G}\phi \quad \stackrel{def}{\equiv} \quad \neg\exists\mathsf{F}\neg\phi$$

and

$$\exists \mathsf{G}\phi \quad \stackrel{def}{\equiv} \quad \neg\forall\mathsf{F}\neg\phi$$

Similarly, the analogous *Backwards Eventually* and *Backwards Always* operators can be defined

$$\overleftarrow{\mathsf{F}}\,\phi \quad \stackrel{def}{\equiv} \quad \mathsf{true}_{\mathsf{true}}\overleftarrow{\mathsf{U}}_{\mathsf{true}}\phi$$

$$\overleftarrow{\mathsf{G}}\,\phi \quad \stackrel{def}{\equiv} \quad \neg\,\overleftarrow{\mathsf{F}}\,\neg\phi$$

The important notion of causality is easily introduced via a defined *Caused* operator. A subject i is said to have caused the path formula γ, denoted $\mathsf{C}_i\gamma$, when γ holds on all future paths and i reached the current state, via some action, from a state in which some different action could have led to a state in which γ does not hold on some future path. Put another way, it is necessary that γ holds after i's action and, if it had not been i that acted, γ might not hold.

$$\mathsf{C}_i\gamma \quad \stackrel{def}{\equiv} \quad \forall\gamma \ \wedge \ \overleftarrow{\mathsf{X}}_{sb(i)}\, \exists\mathsf{X}_{\mathsf{true}}\neg\forall\gamma$$

The permission modality P on a state formula depends on the authorizations and forbiddances in the history of the current state, and is based on the structure of the formula. In developing the various definitions we use the usual definition of obligation,

$$\mathsf{O}\phi \quad \stackrel{def}{\equiv} \quad \neg\mathsf{P}\neg\phi$$

which can be used to assist the intuition in developing the definitions below. First, constants are dealt with in a straightforward way:

$$\mathsf{P}\ \mathsf{true} \quad \stackrel{def}{\equiv} \quad \mathsf{true}$$

$$\mathsf{P}\ \neg\mathsf{true} \quad \stackrel{def}{\equiv} \quad \mathsf{false}$$

Conjunction also offers no difficulty:

$$\mathsf{P}(\phi \wedge \phi') \quad \stackrel{def}{\equiv} \quad \mathsf{P}\phi \wedge \mathsf{P}\phi'$$

$$\mathsf{P}\neg(\phi \wedge \phi') \quad \stackrel{def}{\equiv} \quad \mathsf{P}\neg\phi \vee \mathsf{P}\neg\phi'$$

For convenience in defining quantification over future paths we use an auxiliary operator $\overleftarrow{\mathsf{A}}$ expressing the fact that the previous action was authorized:

$$\overleftarrow{\mathsf{A}} \ \stackrel{def}{\equiv} \ \bigwedge_{i\in\mathcal{U},c\in\mathcal{C}} (\overleftarrow{\mathsf{X}}_{sb(i)\wedge cm(c)}\ \mathsf{true} \Rightarrow \overleftarrow{\mathsf{X}}_{\mathsf{true}}\ \mathsf{A}_i^c)$$
$$\bigwedge_{i,j\in\mathcal{U},c\in\mathcal{C}} (\overleftarrow{\mathsf{X}}_{rc(i)\wedge sb(j)\wedge cm(c)}\ \mathsf{true} \Rightarrow \overleftarrow{\mathsf{X}}_{\mathsf{true}}\ \mathsf{V}_i^{j,c})$$

Using this shorthand notation we have:

$$\mathsf{P}\exists\mathsf{X}_\chi\phi \stackrel{def}{\equiv} \exists\mathsf{X}_\chi(\mathsf{P}\phi \wedge \overleftarrow{\mathsf{A}})$$

$$\mathsf{P}\neg\exists\mathsf{X}_\chi\phi \stackrel{def}{\equiv} \neg\exists\mathsf{X}_\chi(\neg\mathsf{P}\neg\phi \wedge \overleftarrow{\mathsf{A}})$$

$$\mathsf{P}\forall\mathsf{X}_\chi\phi \stackrel{def}{\equiv} \forall\mathsf{X}_\chi(\mathsf{P}\phi \wedge \overleftarrow{\mathsf{A}})$$

$$\mathsf{P}\neg\forall\mathsf{X}_\chi\phi \stackrel{def}{\equiv} \neg\forall\mathsf{X}_\chi(\neg\mathsf{P}\neg\phi \wedge \overleftarrow{\mathsf{A}})$$

$$\mathsf{P}\exists(\phi_\chi\mathsf{U}_{\chi'}\phi') \stackrel{def}{\equiv} \mathsf{P}\phi \wedge (\mathsf{P}\exists\mathsf{X}_{\chi'}\phi' \vee \mathsf{P}\exists\mathsf{X}_\chi\exists(\phi_\chi\mathsf{U}_{\chi'}\phi'))$$

$$\mathsf{P}\neg\exists(\phi_\chi\mathsf{U}_{\chi'}\phi') \stackrel{def}{\equiv} \mathsf{P}\neg\phi \vee (\mathsf{P}\neg\exists\mathsf{X}_{\chi'}\phi' \wedge \mathsf{P}\neg\exists\mathsf{X}_\chi\exists(\phi_\chi\mathsf{U}_{\chi'}\phi'))$$

$$\begin{aligned}\mathsf{P}\forall(\phi_\chi\mathsf{U}_{\chi'}\phi') \stackrel{def}{\equiv}\ & \mathsf{P}\phi \wedge \\ & \mathsf{P}\neg\exists\mathsf{X}_{\neg\chi\wedge\neg\chi'}\mathsf{true} \wedge \mathsf{P}\neg\exists\mathsf{X}_{\mathsf{true}}(\neg\phi' \wedge \neg\forall(\phi_\chi\mathsf{U}_{\chi'}\phi')) \wedge \\ & \mathsf{P}\neg\exists\mathsf{X}_{\neg\chi\wedge\chi'}\neg\phi' \wedge \mathsf{P}\neg\exists\mathsf{X}_{\chi\wedge\neg\chi'}\neg\forall(\phi_\chi\mathsf{U}_{\chi'}\phi')\end{aligned}$$

$$\begin{aligned}\mathsf{P}\neg\forall(\phi_\chi\mathsf{U}_{\chi'}\phi') \stackrel{def}{\equiv}\ & \mathsf{P}\neg\phi \vee \\ & \mathsf{P}\exists\mathsf{X}_{\neg\chi\wedge\neg\chi'}\mathsf{true} \vee \mathsf{P}\exists\mathsf{X}_{\mathsf{true}}(\neg\phi' \wedge \neg\forall(\phi_\chi\mathsf{U}_{\chi'}\phi')) \vee \\ & \mathsf{P}\exists\mathsf{X}_{\neg\chi\wedge\chi'}\neg\phi' \vee \mathsf{P}\exists\mathsf{X}_{\chi\wedge\neg\chi'}\neg\forall(\phi_\chi\mathsf{U}_{\chi'}\phi')\end{aligned}$$

$$\mathsf{P}\exists(\phi_\chi\mathsf{U}\phi') \stackrel{def}{\equiv} \mathsf{P}\phi' \vee (\mathsf{P}\phi \wedge \mathsf{P}\exists\mathsf{X}_\chi\exists(\phi_\chi\mathsf{U}\phi'))$$

$$\mathsf{P}\neg\exists(\phi_\chi\mathsf{U}\phi') \stackrel{def}{\equiv} \mathsf{P}\neg\phi' \wedge (\mathsf{P}\neg\phi \vee \mathsf{P}\neg\exists\mathsf{X}_\chi\exists(\phi_\chi\mathsf{U}\phi'))$$

$$\mathsf{P}\forall(\phi_\chi\mathsf{U}\phi') \stackrel{def}{\equiv} \mathsf{P}\phi' \vee (\mathsf{P}\phi \wedge \mathsf{P}\neg\exists\mathsf{X}_{\neg\chi}\mathsf{true} \wedge \mathsf{P}\neg\exists\mathsf{X}_{\mathsf{true}}\neg\forall(\phi_\chi\mathsf{U}\phi'))$$

$$\mathsf{P}\neg\forall(\phi_\chi\mathsf{U}\phi') \stackrel{def}{\equiv} \mathsf{P}\neg\phi' \wedge (\mathsf{P}\neg\phi \vee \mathsf{P}\exists\mathsf{X}_{\neg\chi}\mathsf{true} \vee \mathsf{P}\exists\mathsf{X}_{\mathsf{true}}\neg\forall(\phi_\chi\mathsf{U}\phi'))$$

Permission on the *Last* operator is analogous to permission on quantifications of *Next*.

$$\mathsf{P}\overleftarrow{\mathsf{X}}_\chi\phi \stackrel{def}{\equiv} \overleftarrow{\mathsf{X}}_\chi\mathsf{P}\phi \wedge \overleftarrow{\mathsf{A}}$$

$$\mathsf{P}\neg \overleftarrow{\mathsf{X}}_\chi\, \phi \;\stackrel{def}{\equiv}\; \neg \overleftarrow{\mathsf{X}}_\chi \neg\mathsf{P}\neg\phi \;\vee\; \neg \overleftarrow{\mathsf{A}}$$

And permission on the *Backwards Until* operator is analogous to permission on quantifications of *Until*.

$$\mathsf{P}(\phi_\chi \overleftarrow{\mathsf{U}}_{\chi'}\, \phi') \;\stackrel{def}{\equiv}\; \mathsf{P}\phi \;\wedge\; (\mathsf{P}\, \overleftarrow{\mathsf{X}}_{\chi'}\, \phi' \;\vee\; \mathsf{P}\, \overleftarrow{\mathsf{X}}_\chi\, (\phi_\chi \overleftarrow{\mathsf{U}}_{\chi'}\, \phi'))$$

$$\mathsf{P}\neg(\phi_\chi \overleftarrow{\mathsf{U}}_{\chi'}\, \phi') \;\stackrel{def}{\equiv}\; \mathsf{P}\neg\phi \;\vee\; (\mathsf{P}\neg \overleftarrow{\mathsf{X}}_{\chi'}\, \phi' \;\wedge\; \mathsf{P}\neg \overleftarrow{\mathsf{X}}_\chi\, (\phi_\chi \overleftarrow{\mathsf{U}}_{\chi'}\, \phi'))$$

4.1 Derived Role Operators

A subject's access and view privileges are characterized by the commands and (subject,command) pairs for which it is role-authorized. This is captured in CSL with the A_i^c and $\mathsf{V}_i^{j,c}$ operators. A role is associated with a set of access and view authorizations. This association is specified by defining a *role operator* that takes the name of the role. A role operator is subscripted by a subject name to denote that the indentified subject is acting with the defined role authorizations. For example, for a role called "Bigrole" that authorizes access to commands $c_1, c_2, \ldots, c_n$ and the viewing of invocations $(j, c_1), (j, c_2), \ldots, (j, c_m)$ one defines:

$$\mathsf{Bigrole}_i \;\stackrel{def}{\equiv}\; \mathsf{A}_i^{c_1} \;\wedge\; \mathsf{A}_i^{c_2} \;\wedge\; \ldots \mathsf{A}_i^{c_n} \;\wedge\; \mathsf{V}_i^{j,c_1} \;\wedge\; \mathsf{V}_i^{j,c_2} \;\wedge\; \ldots \mathsf{V}_i^{j,c_m}$$

Often roles and commands are subject-relative. For example, $\mathsf{send}(r)$ might be the command "send message to subject r". The role, called "Sender", that authorizes the sending of messages to any recipient is specified via the role operator Sender_i:

$$\mathsf{Sender}_i \;\stackrel{def}{\equiv}\; \bigwedge_{r\in\mathcal{U}} \mathsf{A}_i^{\mathsf{send}(r)}$$

A more restricted role, called "Sender(r)", that authorizes the sending of messages only to r is specified via the role operator $\mathsf{Sender}(r)_i$:

$$\mathsf{Sender}(r)_i \;\stackrel{def}{\equiv}\; \mathsf{A}_i^{\mathsf{send}(r)}$$

5 Properties and Expressivity

CSL encounters the standard problems and paradoxes that commonly occur with deontic logics [37, 16, 11]. For example, we have the following properties:

- $\mathsf{P}(\phi \wedge \phi') \Rightarrow \mathsf{P}\phi \wedge \mathsf{P}\phi'$ This CSL property is known as the Free Choice Paradox, regarded as problematic because it seems to reduce permission for a joint action to individual permissions for the component actions thereby losing the aspect of joint action. It also appears in the following form.

  $$\mathsf{P}\forall\mathsf{X}_{\chi\wedge\chi'}\mathsf{true} \Rightarrow \mathsf{P}\forall\mathsf{X}_{\chi}\mathsf{true} \ \wedge \ \mathsf{P}\forall\mathsf{X}_{\chi'}\mathsf{true}$$

- $\mathsf{P}(\phi \vee \phi') \Rightarrow \mathsf{P}\phi \wedge \mathsf{P}\phi'$, although intuitively desirable, does not normally hold in deontic logics because unacceptable inferences would result [37]. It also does not hold in CSL.

However, some other problematic properties seem to be avoided in CSL. For example:

- $\mathsf{O}\phi \Rightarrow \mathsf{O}(\phi \vee \phi')$ This is known as Ross's Paradox, generally regarded as problematic because it seems to allow an agent obligated for some future action to deduce a weaker arbitrary obligation. For example, in CSL a formula of the following form is valid.

  $$\mathsf{O}\forall\mathsf{F}\text{ "mailed letter" } \Rightarrow \mathsf{O}(\forall\mathsf{F}\text{ "mailed letter" } \vee \forall\mathsf{F}\text{ "burned letter" })$$

  However the following, which seems much less of a problem, is not valid.

  $$\mathsf{O}\forall\mathsf{F}\text{ "mailed letter" } \Rightarrow \mathsf{O}\forall\mathsf{F}(\text{ "mailed letter" } \vee \text{ "burned letter" })$$

- if $\phi \Rightarrow \phi'$ then $\mathsf{O}\phi \Rightarrow \mathsf{O}\phi'$ This is the Good Samaritan Paradox, considered a serious problem for many deontic logics, The usual problematic example offered is that ϕ is "John assists the injured Mary" and ϕ' is "Mary is injured". In CSL, however, one could naturally express this in some form $\exists\mathsf{X}_{\chi}\mathsf{true}$ for ϕ and $\overleftarrow{\mathsf{F}}\,\mathsf{X}_{\chi'}\mathsf{true}$ for ϕ' so that we have

  $$\text{if } \exists\mathsf{X}_{\chi}\mathsf{true} \Rightarrow \overleftarrow{\mathsf{F}}\,\mathsf{X}_{\chi'}\mathsf{true} \text{ then } \mathsf{O}\exists\mathsf{X}_{\chi}\mathsf{true} \Rightarrow \mathsf{O}\,\overleftarrow{\mathsf{F}}\,\mathsf{X}_{\chi'}\mathsf{true}$$

which does not seem to be as problematic as the example suggests. If it is obligated that there is a future path on which John assists the injured Mary then it certainly must be the case that Mary was injured in the past. In other words, it is obligated that Mary was injured in the past.

Finally, Chisolm's Paradox is often offered as a test case for deontic logics as a property that should be expressible:

$$\mathsf{O}\phi \wedge \mathsf{O}(\phi \Rightarrow \phi') \wedge (\neg\phi \Rightarrow \mathsf{O}\neg\phi') \wedge \neg\phi$$

The usual example provided occurs where ϕ is "Mary goes to help her neighbour" and ϕ' is "Mary tells her neighbour that she is coming". In CSL, one could write the following, with $\overleftarrow{\mathsf{X}}_{\chi}\ \mathsf{true}$ representing "Mary helped" and $\overleftarrow{\mathsf{X}}_{\chi'}\ \mathsf{true}$ representing "Mary told".

$$\begin{aligned}&\mathsf{O}\forall\mathsf{F}\ \overleftarrow{\mathsf{X}}_{\chi}\ \mathsf{true} \wedge \mathsf{O}\neg\exists(\overleftarrow{\mathsf{X}}_{\neg\chi'}\ \mathsf{true})_{\mathsf{true}}\mathsf{U}(\overleftarrow{\mathsf{X}}_{\chi}\ \mathsf{true})\ \wedge\\ &\quad(\neg\exists\mathsf{F}\ \overleftarrow{\mathsf{X}}_{\chi}\ \mathsf{true} \Rightarrow \mathsf{O}\neg\exists\mathsf{F}\ \overleftarrow{\mathsf{X}}_{\chi'}\ \mathsf{true}) \wedge \neg\exists\mathsf{F}\ \overleftarrow{\mathsf{X}}_{\chi}\ \mathsf{true}\end{aligned}$$

CSL must be able to specify behavioural properties of systems. For example, to specify that command c can be invoked by any subject $i \in \mathcal{X} \subseteq \mathcal{U}$, one uses the logic of actions to specify:

$$cm(c) \Rightarrow \bigvee_{i\in\mathcal{X}} sb(i)$$

To specify visibility, for example that subject j invoking command c can be seen by any subject $i \in \mathcal{X} \subseteq \mathcal{U}$, one writes:

$$(sb(j) \wedge cm(c)) \Rightarrow \bigwedge_{i\in\mathcal{X}} rc(i)$$

To specify permissions and forbiddances, for example that command c enables all (subject,command) pairs $(i', c') \in \mathcal{X} \subseteq \mathcal{U} \times \mathcal{C}$ and disables all (subject,command) pairs $(i'', c'') \in \mathcal{Y} \subseteq \mathcal{U} \times \mathcal{C}$, one writes:

$$cm(c) \Rightarrow (\bigwedge_{(i',c')\in\mathcal{X}} en(i', c') \bigwedge_{(i'',c'')\in\mathcal{Y}} ds(i'', c''))$$

The Secrecy Property, fundamental to any system enforcing some sort of secrecy, is specified:

$$\mathsf{K}_i\phi \Rightarrow \mathsf{PK}_i\phi$$

The following Integrity Property specifies a form of integrity based on causality. While other notions of integrity exist, this property seems to capture a considerable scope of what is generally regarded as integrity.

$$\mathsf{C}_i\gamma \Rightarrow \mathsf{PC}_i\gamma$$

A generic Legitimate Use Property can be expressed as:

$$\exists \mathsf{X}_{sb(i)\wedge cm(c)}\mathsf{true} \Rightarrow \mathsf{PX}_{sb(i)\wedge cm(c)}\mathsf{true}$$

Availability is not expressible in CSL because it inherently involves real-time; for example, a query might require a response within some time bound. The best we can do is a much weaker property that specifies an eventual response to a query:

$$\overleftarrow{\mathsf{X}}_{cm(query)}\ \mathsf{true} \Rightarrow \forall\mathsf{F}\ \overleftarrow{\mathsf{X}}_{cm(response)}\ \mathsf{true}$$

A secondary goal of CSL is to be able to specify non-security properties such as those of fault tolerance and failure recovery, so that their interaction with security properties can be investigated. Expressing properties of systems in the presence of faults at the requirements level of specification seems inappropriate at first, at least from a purist point of view. However, specifying that there exists a fault that can change the behaviour of a system does not imply that the requirements specifier must identify that fault; that can remain the responsibility of the designer. Rather, the requirements specifier is merely saying that where the designer is unable to produce a system that is tolerant of some expected fault then it is required that the changed behaviour satisfy some property. For example, one may specify the behaviour of a system while it is in a phase of recovery from a failure as follows.

$$\overleftarrow{\mathsf{X}}_{cm(failure)}\ \mathsf{true} \quad \Rightarrow \quad \forall(\phi_\chi \mathsf{U}_{\chi'} \phi')$$

In this example, $failure$ describes some failure action. The specification says that, after such an action, state and action behaviour in the recovery phase must satisfy respectively ϕ and χ until eventually a recovery action satisfying χ' occurs to restore the system to a recovery state that satisfies ϕ'.

Other interesting properties include the Memory Property:

$$\mathsf{K}_i\phi \Rightarrow \forall\mathsf{GK}_i\ \overleftarrow{\mathsf{F}}\ \phi$$

and the Accessability Property:

$$\mathsf{PK}_i\phi \Rightarrow \exists\mathsf{FK}_i\phi$$

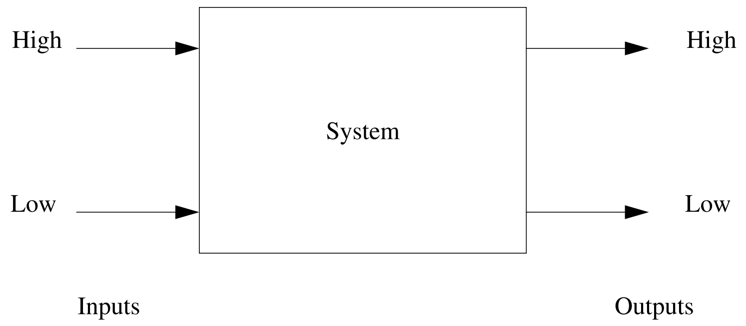

Figure 3: Multi-Level Secrecy System Model

6 Applications

6.1 Mandatory Multilevel Secrecy

A typical simplified system model for multilevel secrecy is shown in Figure 3 in which there are only two users, a *high* user who handles sensitive information, and a *low* user who handles non-sensitive information. Each user invokes a sequence of inputs (input actions) and observes a sequence of outputs (output actions). Input and output sequences and the sequences of different users are interleaved. We associate subject l with the low user, and subject h with the high user. For each input action, an associated subject is specified via the proposition *sb(.)* applied to that action. For each input or output action, some associated subjects are specified via the proposition *rc(.)* applied to that action. The difference between the subjects is that h is view-authorized for high and low actions but l is authorized only for low actions.

Consider, first, subject l. It invokes low inputs, observes both low inputs and low outputs, is enabled for low inputs, and is unmasked for low inputs and outputs. We then have, for all low input commands c and all low output commands c',

$$cm(c) \Rightarrow sb(l)$$

$$cm(c') \Rightarrow rc(l)$$

$$cm(\mathsf{init}) \Rightarrow en(l, c) \ \wedge \ um(l, (l, c)) \ \wedge \ um(l, (l, c'))$$

Subject h is specified analogously, for both low and high actions. For all high input commands d and all high output commands d', and for all low input commands c and all low output commands c',

$$cm(d) \Rightarrow sb(h)$$

$$cm(d') \Rightarrow rc(h)$$

$$\begin{aligned} &cm(\mathsf{init}) \Rightarrow en(h, d) \ \wedge \ um(h, (h, d)) \ \wedge \ um(h, (h, d')) \ \wedge \\ &\quad um(h, (h, c)) \ \wedge \ um(h, (h, c')) \end{aligned}$$

It is required to prove that these assumptions along with the system's behavioural specifications together imply the Secrecy Property. I.e.,

$$\text{Assumptions} \ \wedge \ \text{System Specification} \ \Rightarrow \ \text{Secrecy Property}$$

However, this simple model is too strong; there are system designs that are intuitively secure yet would fail this verification using these simple assumptions. We need to show how such examples can be addressed. On the other hand we also need to examine some examples which attempt to test the ability of a security definition to detect intuitively insecure systems and show how CSL performs. To do this we take three examples from [9], two of which were motivated by other papers in the literature (cited with the example).

6.1.1 Example 3

Consider a system in which a certain low input command c is always echoed as the high output command d [26]. This is clearly secure on an intuitive level because the low subject does not learn anything about the high subject's behaviour. However, the following holds:

$$\overleftarrow{\mathsf{X}}_{sb(l)\wedge cm(c)}\ \mathsf{true} \quad \Rightarrow \quad \mathsf{K}_l \forall \mathsf{F}\ \overleftarrow{\mathsf{X}}_{rc(h)\wedge cm(d)}\ \mathsf{true}$$

which would violate the Secrecy Property because all high outputs are masked for the low subject. The solution is to dynamically unmask the secure high output for the low subject and mask it after it appears:

$$cm(c) \Rightarrow um(l,(h,d))$$

$$cm(d) \Rightarrow ms(l,(h,d))$$

6.1.2 Example 4

The next example attempts to distinguish between insecurities in the system and those in the environment. Suppose that a certain high input command d is always echoed as the low output c. This is clearly insecure on any intuitive level. The question here is the following: If the low subject somehow always manages to produce an input c' immediately after the high input d, then does the fact that the subsequent output c is dependent on c' render the system secure since both are at the low level. The answer is no, because when the input c' occurs the low subject knows that d occurred. Additionally, such a behaviour of the low subject would have to be assumed explicitly thus making it clear in the specification that the violation of secrecy is caused by the environment and not by the system.

6.1.3 Example 5

The next example tests the ability to detect information flows caused by a union of subjects [27]. There are two high subjects h and h' and one low subject l. The system behaviour is to read an input from each of h and h', say c and c', and to write an output to l, say d, that is the exclusive-or of these inputs, i.e. $d = c \oplus c'$. h' writes a sequence of random 0 or 1 inputs. h observes each input c' of h' and transmits an associated sequence of bits to

l as follows: to send i, write $c = i \oplus c'$. l then receives $d = i \oplus c' \oplus c' = i$. Clearly, there is a flow from high to low. However, as shown by Bieber and Cuppens, l cannot deduce any information about h nor about h' because the flow is from the joint subject h and h'. Consequently, to be useful in this situation a definition of secrecy must be able to detect this joint flow and expose the secrecy violation. With CSL applied to this example, we have in the case that l receives a zero:

$$\overleftarrow{\mathsf{X}}_{rc(l)\wedge cm(0)}\ \mathsf{true} \Rightarrow \mathsf{K}_l(\overleftarrow{\mathsf{F}}\overleftarrow{\mathsf{X}}_{sb(h)\wedge cm(0)}\overleftarrow{\mathsf{F}}\overleftarrow{\mathsf{X}}_{sb(h')\wedge cm(0)}\ \mathsf{true}\ \vee \\ \overleftarrow{\mathsf{F}}\overleftarrow{\mathsf{X}}_{sb(h)\wedge cm(1)}\overleftarrow{\mathsf{F}}\overleftarrow{\mathsf{X}}_{sb(h')\wedge cm(1)}\ \mathsf{true})$$

In other words, l on receiving a 0 knows that either the sequence $c = 0; c' = 0$ or the sequence $c = 1; c' = 1$ occurred. It can be shown that to satisfy secrecy l must either have permission to know that the first sequence occurred or have permission to know that the second sequence occurred. In either case, l would need view-authorization for high actions which it would not have for this example.

6.2 Role-Based Access Control

Controlling access to medical records is considered particularly difficult to express formally, so an extensive example of this problem is used to illustrate access control in CSL.

Example 6 Motivated partly by [21], we take the following simplified set of natural language requirements:

(a) A patient may not access his own records.

(b) A primary physician may access his patients' records.

(c) A patient may request the transfer of his record to another primary physician.

(d) A patient must give his permission for his primary physician to share the patient's record with a consulting physician.

(e) A consulting physician may not modify a patient's record.

(f) A consulting physician may not copy or transfer a patient's record.

(g) A nurse, under supervision of the primary physician, may access aspects of the patient's record that are relevant to the nurse's responsibility.

(h) A clerk may have entry responsibility based on a physician's instructions.

(i) A clerk may read all patient's records.

The sort of formal reasoning that one would like to be able to carry out with this application includes the following:

- Given a specification of an abstract system model, along with a specification of the roles involved and their privileges, show that these requirements are satisfied.

Table 1 shows a re-organization of these requirements to make them more amenable to formalization; Requirement (f) of the natural language requirements has been addressed by not providing an operation for copying a patient record, since such an operation is not required. We comment further on the appropriateness of this in the next section.

To formalize the requirements in Table 1, identify the following fixed subsets of the subjects: physicians $\mathcal{P} \subseteq \mathcal{U}$, clerks $\mathcal{K} \subseteq \mathcal{U}$ and for each physician $p \in \mathcal{P}$, staff nurses $\mathcal{N}(p) \subseteq \mathcal{U}$.

The following commands are provided:

$\mathsf{read}(r)$:	Read of record of r, for all subjects r
$\mathsf{cread}(r)$:	Constrained read of record of r, for all subjects r
$\mathsf{write}(r)$:	Write of record of r, for all subjects r
$\mathsf{cwrite}(r)$:	Constrained write of record of r, for all subjects r
$\mathsf{transfer}(r, p)$:	Transfer patient r to primary p, for all subjects r and all physicians $p \in \mathcal{P}$
$\mathsf{grant}(r, p, q)$:	Authorize consultant q to primary p of patient r, for all subjects r and all physicians $p, q \in \mathcal{P}$
$\mathsf{instruct}(k, r)$:	Instruct clerk k about patient r,

1. A person i may read a patient r's record if and only if $i \neq r$ and at least one of the following holds: i is the primary physician of r; i has been granted permission as a consultant to r's current primary physician; i is a clerk.
2. A person i may do a constrained read of a patient r's record if and only if $i \neq r$ and i is a nurse for r's primary physician.
3. A person i may write a patient r's record if and only if $i \neq r$ and at least one of the following holds: i is the primary physician of r; i is a clerk and r's primary physician has instructed him regarding writes to r's record.
4. A person i may do a constrained write of a patient r's record if and only if $i \neq r$ and i is a nurse for r's primary physician.
5. A patient r may effect a transfer to physician p as his primary if and only if $r \neq p$.
6. A patient r may grant permission for his primary physician p to give his record to a consultant q to read if and only if $r \neq q$.
7. A physician p may instruct clerk c to write into patient r's record if and only if $c \neq r$ and p is r's primary physician.

Table 1: Example Medical System Requirements

for all patients r and all clerks $k \in \mathcal{K}$

Notice that the role of "staff nurse to physician p" has been conveniently expressed in the metalogical subsets $\mathcal{N}(p)$, rather than providing a set of explicit such roles. This is possible because there are no commands in the system model associated with such roles, and no dynamic instantiation of such roles.

The *role specifications* comprise two parts: the definition of the role operators, and the specification of role dynamics, i.e. how commands cause the association of roles with subjects. The following are the role operators:

$\mathsf{Patient}(r)_i$: i acts as patient r

$$\mathsf{Patient}(r)_i \stackrel{def}{\equiv} \bigwedge_{p \in \mathcal{P}} \mathsf{A}_i^{\mathsf{transfer}(r,p)}$$

$\mathsf{Client}(p)_i$: i acts as client of physician p

$$\mathsf{Client}(p)_i \stackrel{def}{\equiv} \bigwedge_{q \in \mathcal{P}} \mathsf{A}_i^{\mathsf{grant}(i,p,q)}$$

$\mathsf{Nurse}(r)_i$: i acts as nurse to patient r

$$\mathsf{Nurse}(r)_i \stackrel{def}{\equiv} \mathsf{A}_i^{\mathsf{cread}(r)} \wedge \mathsf{A}_i^{\mathsf{cwrite}(r)}$$

$\mathsf{Primary}(r)_i$: i acts as primary physician to person r

$$\mathsf{Primary}(r)_i \stackrel{def}{\equiv} \mathsf{A}_i^{\mathsf{read}(r)} \wedge \mathsf{A}_i^{\mathsf{write}(r)} \bigwedge_{k \in \mathcal{K}} \mathsf{A}_i^{\mathsf{instruct}(k,r)}$$

$\mathsf{Consultant}(r)_i$: i acts as consultant for patient r

$$\mathsf{Consultant}(r)_i \stackrel{def}{\equiv} \mathsf{A}_i^{\mathsf{read}(r)}$$

Clerk_i: i acts as a clerk

$$\mathsf{Clerk}_i \stackrel{def}{\equiv} \bigwedge_{r \in \mathcal{U}} \mathsf{A}_i^{\mathsf{read}(r)}$$

$\mathsf{Scribe}(r)_i$: i acts as a scribe for the record of patient r

$$\mathsf{Scribe}(r)_i \stackrel{def}{\equiv} \mathsf{Clerk}_i \wedge \mathsf{A}_i^{\mathsf{write}(r)}$$

Now we need to specify role dynamics. It must be made clear that in our approach a role does not determine the ability to invoke a command; rather, the set of commands that a subject is authorized to invoke determines the role that is being exercised by the subject. It might seem that this results in a loss of generality because in many applications a subject may be authorized to invoke a command by virtue of having several roles active at the same time. It may be important to express explicitly which role is being exercised. This would be modelled by providing a command by which a subject declares the role to be associated with a following command. This closely models what is likely to be the case in practice. The following are the specifications of role dynamics (For readability, we have omitted the propositional notation $cm()$ for commands.):

$$\overleftarrow{\mathsf{X}}_{\mathsf{transfer}(r,p)}\ \mathsf{true} \Rightarrow \mathsf{Primary}(r)_p \ \wedge$$
$$\mathsf{Client}(p)_r \ \wedge$$
$$\bigwedge_{n\in\mathcal{N}(p)} \mathsf{Nurse}(r)_n$$
$$\bigwedge_{q\in\mathcal{P},q\neq p} \neg\mathsf{Primary}(r)_q$$
$$\bigwedge_{q\in\mathcal{P},q\neq p} \neg\mathsf{Client}(q)_r$$
$$\bigwedge_{n\in\mathcal{N}(q),q\in\mathcal{P},q\neq p} \neg\mathsf{Nurse}(r)_n$$
$$\bigwedge_{q\in\mathcal{P}} \neg\mathsf{Consultant}(r)_q$$
$$\bigwedge_{k\in\mathcal{K}} \neg\mathsf{Scribe}(r)_k$$

$$\overleftarrow{\mathsf{X}}_{\mathsf{init}}\ \mathsf{true} \Rightarrow \bigwedge_{i\in\mathcal{U}} \mathsf{Patient}(i)_i$$
$$\bigwedge_{k\in\mathcal{K}} \mathsf{Clerk}_k$$

$$\overleftarrow{\mathsf{X}}_{\mathsf{instruct}(k,r)}\ \mathsf{true} \Rightarrow \mathsf{Scribe}(r)_k$$

$$\overleftarrow{\mathsf{X}}_{\mathsf{grant}(r,p,q)}\ \mathsf{true} \Rightarrow \mathsf{Consultant}(r)_q$$

The formal role-based requirements can now be written in detail as:

1. $\exists \mathsf{X}_{sb(i) \wedge \mathsf{read}(r)} \mathsf{true} \Rightarrow$
   $\neg \mathsf{Patient}(r)_i \ \wedge \ (\mathsf{Primary}(r)_i \ \vee \ \mathsf{Consultant}(r)_i \ \vee \ \mathsf{Clerk}_i)$

2. $\exists \mathsf{X}_{sb(i) \wedge \mathsf{cread}(r)} \mathsf{true} \Rightarrow$
   $\neg \mathsf{Patient}(r)_i \ \wedge \ \mathsf{Nurse}(r)_i$

3. $\exists \mathsf{X}_{sb(i) \wedge \mathsf{write}(r)} \mathsf{true} \Rightarrow$
   $\neg \mathsf{Patient}(r)_i \ \wedge \ (\mathsf{Primary}(r)_i \ \vee \ \mathsf{Scribe}(r)_i)$

4. $\exists \mathsf{X}_{sb(i) \wedge \mathsf{cwrite}(r)} \mathsf{true} \Rightarrow$
   $\neg \mathsf{Patient}(r)_i \ \wedge \ \mathsf{Nurse}(r)_i$

5. $\exists \mathsf{X}_{sb(i) \wedge \mathsf{transfer}(r,p)} \mathsf{true} \Rightarrow$
   $\mathsf{Patient}(r)_i \ \wedge \ \neg \mathsf{Patient}(r)_p$

6. $\exists \mathsf{X}_{sb(i) \wedge \mathsf{grant}(r,p,q)} \mathsf{true} \Rightarrow$
   $\mathsf{Patient}(r)_i \ \wedge \ \neg \mathsf{Patient}(r)_q \ \wedge \ \mathsf{Primary}(r)_p$

7. $\exists \mathsf{X}_{sb(i) \wedge \mathsf{instruct}(k,r)} \mathsf{true} \Rightarrow$
   $\neg \mathsf{Patient}(r)_k \ \wedge \ \mathsf{Primary}(r)_i \ \wedge \ \mathsf{Clerk}_k$

Assuming that the system of interest has been specified with a set of *system specifications*, then the verification of interest is:

$$\textit{system specifications} \ \wedge \ \textit{role specifications} \ \Rightarrow \ \textit{role-based requirements}$$

If one proves Causality Security

$$\mathsf{C}_i \phi \Rightarrow \mathsf{PC}_i \phi$$

assuming the system and role specifications, then this subsumes a verification of role-based security, as shown above, along with a proof of safety-based security. This illustrates the fact that Causality Security is a very strong property indeed.

7 Discussion

Inspired by, and similar to the introduction of Hennessy-Milner logic [HM85], the action-based version of the logical description language CTL (ACTL) aims at combining two previously disjoint but complementary approaches for system analysis: action-based behavioural reasoning (causality, choice) and state-based logical reasoning. Such a combination is due to the observation that both of these two areas have their own successful results: operators for transition systems and the issue of behavioural equivalences which are interpreted over LTS's have been well developed to support reasoning with automatic tools; while temporal logics and the associated complexity issue have been thoroughly investigated in the setting of Kripke Structures. As a consequence, adopting ACTL as the basis of CSL and LTS's as the computational model, provides us not only with a way of expressing logical properties in terms of sequences of actions and action-based logical reasoning, but also with the potential to introduce process terms [28, 6, 20] interpreted on LTS's by way of structural rules of operational semantics [32]. The latter, although out of the scope of the present paper, will provide us with a good way of expressing concurrent activities in terms of actions, and with the possibility of working on behavioural reasoning of systems.

Research in deontic modalities in forward-branching backward-linear time is not new [10]. Our contribution is the extension of recent work in the temporal logic of actions with the deontic modalities permission and obligation, as well as that of knowledge. Other recent work on the application of deontic logics to computer security includes [8, 13, 9]. Bieber and Cuppens' work [9] is particularly relevant because it proposes a new definition of secrecy security within a logical framework. This definition, called *Causality*, is ascribed to a specialized "permission to know" modal operator. In addition, their modal logic contains the knowledge operator K_i and the ability to explicitly reference time within propositional variables. CSL has a greater expressibility with respect to the definition of security but less expressibility with respect to explicit time reference.

Informally, Bieber and Cuppens' definition of secrecy is that a subject is permitted to know any information deducible from its inputs. Such a definition is similar to information flow-based definitions of secrecy such as [18, 25, 34] in that what is permitted and forbidden is implicit in the definition. CSL is different from all such definitions in that it requires each action to be explicitly permitted or forbidden via the specification of view-authorizations. Consequently, the policy depends on the application; it is manifested in the specification of view-authorizations along with any other environmental assumptions specified; recall that the verification of secrecy satisfaction is of the form:

$$\text{Assumptions} \wedge \text{System Specification} \Rightarrow \text{Secrecy Property}$$

where the Assumptions embody the view-authorizations and any other environmental assumptions. As a result, CSL does not have an inherent definition of secrecy that can be compared to such flow-based definitions or to Bieber and Cuppens' logic-based definition. Syverson and Gray generalize *Causality* within a probabilistic framework to produce a definition of *Probabilistic Non-Interference* (PNI) [35]. Like *Causality*, PNI incorporates a "permission to know" modal operator rather than a generalized permission operator as in CSL. It is shown that PNI is a necessary and sufficient condition for a system to be free of covert channels. An interesting question is whether or not there is a characterization of such secrecy definitions using view-authorizations.

The backward modalities *Since* and *Last* are introduced and interpreted in a linear sense mainly to express knowledge and causality respectively. A result of this is that efficient model checking is impossible. Indeed, it is shown in [23] that CTL with a linearly interpreted past modality *Since* has the same expressivity as CTL* [14], and thus we cannot expect an efficient model checking tool for CSL since it will cost at least a (single) exponential upper-bound, as does CTL*.

However, we are interested in investigating symbolic model checking for certain important cases, by way of representing infinite tree structures as parameterized finite graphs. Such a parameterization should be based on an investigation into relevant important security properties. We are also currently investigating a syntactic proof system.

Our major reservation concerning the role-based security involves the inability to express properties based on state. For example, not providing a `copy` command in the medical system example to address the "no copy" requirement on consultants can be viewed as moving the system security boundary further out from the state, i.e. the data, of the system model. One could view the boundary as being at the application interface and subjects as modelling human users. A better system model would put the boundary in closer and view the subjects as processes executing on behalf of human users. In this case it is less acceptable to simply say that we will have no copy command since processes can, in general, read from one record and write to another. In this case, the potential information flows in the system model cannot be ignored. Whether such information flows could be addressed by parameterizing commands with data values, e.g. $\mathsf{read}(r, d)$, as in done with languages like CCS [28], has yet to be determined.

Future work also includes an investigation of the inclusion in CSL of the expression of delegation [1]. We also intend to investigate the relationship of CSL to cryptographic protocol logics [36]. Finally, we intend to investigate extensions to CSL to address real-time applications [31, 3].

Acknowledgement: The authors would like to acknowledge Rocco De Nicola for suggesting ACTL as a basis for the logic.

References

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

[2] I.F. Akyildiz and J.V. Luo. Formal modelling of commercial integrity policy in parallel and distributed systems. Technical Report GIT-ICS-91-25, College of Computing, Georgia Institute of Technology, April 1991.

[3] R. Alur, C Courcoubatis, and D. Dill. Model-checking for real-time systems. In *Proceedings of Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.

[4] L. Badger. Providing a flexible security override for trusted systems. In *Proceedings of the Computer Security Foundations Workshop II*, pages 115–121, Franconia, N.H., 1990.

[5] L. Badger. A model for specifying multi-granularity integrity policies. In *Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy*, Oakland, California, May 1989.

[6] J.C.M. Baeton and W.P. Weijland. *Process Algebra.* Cambridge University Press, Cambridge, UK, 1990.

[7] K. J. Biba. Integrity considerations for secure computer systems. Technical Report TR-3153, MITRE Corporation, April 1977.

[8] Pierre Bieber. A definition of secure dependencies using the logic of security. In *The Computer Security Foundations Workshop*, pages 2–11, Franconia NH, 1991.

[9] Pierre Bieber and Frederick Cuppens. A logical view of secure dependencies. *Journal of Computer Security*, 1:99–129, 1992.

[10] M. Byrd. Megarian necessity in forward-branching, backward-linear time. *Midwest Studies in Philosophy*, 3:463–469, February 1978.

[11] H. Castaneda. *The Paradoxes of Deontic Logic*, pages 37–85. D. Reidel Publishing Company (dist. by Kluwer Boston Inc., Hingham, Mass.), 1981. In New Studies in Deontic Logic: Norms, Actions and the Foundations of Ethics.

[12] Frederick Cuppens. Roles and deontic logic. unpublished, 1994.

[13] Frederick Cuppens. A logic analysis of authorized and prohibited information flows. In *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pages 100–109, Oakland, California, May 1993.

[14] E.A. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter 16, pages 996–1072. Elsevier, 1990.

[15] E.A. Emerson and J.Y Halpern. "sometimes" and "not never" revisited. *Journal of the ACM*, 33(1):151–178, 1986.

[16] D. Follesdal and R. Hilpinen. *Deontic Logic: An Introduction*, pages 1–35. D. Reidel Publishing Company (dist. by Humanitites Press, New York), 1971. In Deontic Logic: Introductory and Systematic Readings.

[17] J.I. Glasgow, G.H. MacEwen, and P. Panangaden. A logic for reasoning about security. *ACM Transactions on Computer Systems*, 10(3):226–264, August 1992.

[18] J.A. Goguen and J. Meseguer. Unwinding and inference control. In *IEEE Symposium on Security and Privacy*, pages 75–86, Oakland, CA, April 1984.

[19] M.A. Harrison, W.L. Ruzzo, and J.d. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[20] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[21] M.Y. Hu, S.A. Demurjian, and T.C. Ting. User-role based security profiles for an object-oriented design model. In *Database Security VI: Status and Prospects*, pages 333–348. Elsevier Science Publishers B.V. (North Holland), 1993. IFIP.

[22] S. Kripke. Semantical considerations of modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[23] F. Laroussinie and P. Schnoebelen. A hierarchy of temporal logics with past. In *STACS*, 1994.

[24] T.F. Lunt et al. The seaview security model. *IEEE Transactions on Software Engineering*, SE-16(6), June 1990.

[25] A.D. McCullough. Specifications for multi-level security and a hook-up property. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 161–166, Oakland, CA, April 1987.

[26] J. McLean. Security models and information flow. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 180–187, Oakland, CA, May 1990.

[27] J. K. Millen. Hook-up security for synchronous machines. In *Proceedings of the Computer Security Foundations Workshop*, pages 84–90, Franconia, New Hampshire, June 1990.

[28] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[29] R. De Nicola, A. Fantechi, S. Gnesi, and G. Ristori. An action-based framework for verifying logical and behavioural properties of concurrent systems. *Computer Networks*, 25(7):761–778, February 1993.

[30] R. De Nicola and F.W. Vaandrager. Action versus state- based logics for transition systems. In I. Guessarian, editor, *Proc. Ecole de printemps on semantics of concurrency, LNCS 469*, pages 407–419. Springer-Verlag, 1990.

[31] J.S. Ostroff and W.M. Wonham. A framework for real-time discrete event control. *IEEE Transactions on Automatic Control*, pages 386–397, December 1987.

[32] G. Plotkin. A structural approach to operational semantics. Technical report, Computer Science Dept. Aarhus Univ. Denmark, 1981. DAIMI-FN-19.

[33] R.S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating schemes. *Journal of the ACM*, 35(2):404–432, 1988.

[34] A.D. Sutherland. A model of information. In *9th National Computer Security Conference*, pages 175–183, Gaithersburg, MD, September 1986.

[35] P.F. Syverson and J.W. Gray. The epistemic representation of information flow security in probabilistic systems. In *Proceedings of the Computer Security Foundations Workshop VIII*, pages 152–166, Kenmare, Ireland, 1995.

[36] P.F. Syverson and P.C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of The 1994 IEEE Symposium on Security and Privacy*, pages 14–28, Oakland, California, May 1994.

[37] G.H. von Wright. *An Essay in Deontic Logic and The General Theory of Action*, volume 21 of *Acta Philosophicas Fennica.* North-Holland, 1968.