

Combining Logics for Modelling Security Policies

Chuchang Liu

Patrick McLean

Maris A. Ozols

Information Networks Division
Defence Science and Technology Organisation
PO Box 1500, Edinburgh, SA 5111, Australia
Email: {Chuchang.Liu,Patrick.McLean,Maris.Ozols}@dsto.defence.gov.au

Abstract

This paper considers a logical framework for modelling security policies for information systems. Epistemic, deontic and temporal logics can respectively be used to express what an agent knows or believes, what an agent is permitted or forbidden to know or do, and the dynamic progress of a system over time. In modelling a security policy for a practical system, one may need to combine these logical notions to express statements of the policy, so a combination of logics is considered. We investigate the issues regarding techniques for combining logics, approaches to formalizing security policies based on a combined logic, and strategies applied for reasoning about the security properties required to be satisfied by a policy. Several possible future research directions under this logical framework are discussed.

Keywords: security, security policy, policy languages, logics, combining logics.

1 Introduction

The security policy for a given system specifies the security measures that regulate how sensitive information and other resources are managed, protected and distributed within the system. Such a policy reflects the high-level security goals of the system and it should imply them.

The ingredients for creating a security policy include the system to which the security policy applies, and a language for the policy maker (PM) to express the particular system security policy. In designing a security policy, the PM needs to answer the question:

- What kind of a security policy is required for protecting the system against any threat to its security properties?

While designing a security policy language, the policy language designer (PLD) needs to ask:

- What are the requirements of a policy language in order that it is able to express such policy?

The PM (for a chosen domain) and the PLD must work together if they are to correctly answer the later point.

Natural language (or a subset of natural language) is typically used for expressing security policies. The freedom allowed in natural language means that ambiguity and conflicts in policy often arise; the informal nature of natural language also means that analysing

policy for potential conflicts may be difficult. A more formal security policy language is therefore needed.

Formal security policy for an organisation helps establish standards for IT resource protection by assigning program management responsibilities and providing basic security rules, guidelines, and definitions for everyone in the organisation. Such a policy should be sufficiently clear and comprehensive to be accepted and followed throughout the organisation. It could serve as a basis for the enforcement of more detailed rules and procedures, and help prevent inconsistencies that can introduce risks.

This paper proposes a logical framework for a formal security policy language that meets the following set of criteria:

1. The language must be able to capture all required concepts. This will require the PM and PLD to extract the policy concepts while ignoring information that is neither a definition nor a normative rule.
2. The language must allow reasoning about policy, to highlight policy conflicts.
3. The language must be easy for the PM to work with, and easy for users to understand.

Formal methods can be used in addressing the first two criteria, with collaboration between PM and PLD for the last two. The chosen environment for specifying our policy language is a high security environment, for example, a defence environment. The design of such a language should be a collaborative process involving experts from both policy creation and formal methods.

As is well known, epistemic logic, deontic logic, and temporal logic can respectively be used to express what an agent knows or believes, what an agent is permitted or forbidden to know or do, and when events happen and how states change over time. The notions contained in these logics are fundamental to the formal specification of security policies. Some of the earliest work using deontic logic for the specification of security policies can be found in (Glasgow & Macewen et al. 1992). In (Cholvy & Cuppens 1997), the standard deontic logic is used to represent security policies with the aim of analysing conflicts in the policy specification. Ortalo gives a language for specifying security policies (Ortalo 1998), which is also based on deontic logic. Regarding temporal representation, Peri (Peri 1996) proposes a temporal logic-based language for specification and verification of security policies. He develops a method to describe the set of traces generated by the specification of a policy and proposes a technique for getting a set of rules to verify that the set of traces have the properties specified by the policy.

It is clear that any logical system modeling reactive agents should be a combined system of logics of

knowledge, belief, time, and norms. For modelling security policies, one may need to combine all the epistemic notion, deontic notion and temporal notion together to express statements of a policy. Hence combinations of these logics must be considered in the study of formal representations for security policies. In fact, there has been some work on the development of new logical systems by combination for specifying security policies, for example, in (Cuppens & Saurel 1996) the authors combine a deontic logic to model the concept of permission, obligation and prohibition with a modal logic of action; and Cuppens and Demolmore (Cuppens & Demolmore 1997) propose a modal logical framework that combines epistemic and deontic logics for specifying confidentiality policies. However, there is still a lack of standard formalisation techniques applied for all purposes. We believe that, in order to support the analysis process of various security policies, it is necessary to systematically investigate formal representation techniques, and provide more generic tools for the specification, and reasoning about, such policies.

Our work is motivated first by the need to provide a general methodology to build an appropriate logical basis for formally expressing security policies for computer and information systems; secondly, our purpose is to find sufficient conditions of security and formalise them based on the logics developed for implementing particular security policies in practice. In this paper, we will focus on the issues regarding

- techniques for combining logics, which are particularly related to modelling security policies,
- approaches for formalizing security policies based on a combined logic, and
- strategies for reasoning about the security properties that are specified by a policy.

In this paper, we do not attempt to deeply study all these aspects, but we give some general ideas for achieving our goals. We also identify possible future research directions related to the logical framework we propose. In the next section, we discuss why combining modal logics must be considered for the formalization of security policies. In section 3, we present two techniques for combining logics with the aim of providing some non-complex but strong enough combined logics for use in the specification and analysis of security policies. Section 4 discusses how to express security rules in our logical framework. Section 5 discusses reasoning techniques through an example. Section 6 concludes the paper with a short discussion on future work.

2 Security Policies and Modal Logics

A typical security policy is an *access control policy* (ACP). Such policies are usually represented as rules specifying the conditions under which subjects are permitted to access some IT resources. An ACP may be based on various security models, for example, the DAC (Discretionary Access Control) model that restricts access to objects based on subject, the MAC (Mandatory Access Control) model (also called the Bell-LaPadula model) which is based on fixed regulations mandated by a central authority, and RBAC (Role-Based Access Control) model in which permissions are defined in terms of positions of users, *etc.*

There are many ways to classify security policies, see, for example, (Dulany 2002). We simply classify security policies into two types: *general policy* and *issue-specific policy*. A general policy, sometimes called the *meta-level policy*, is intended to address

the broadest aspects of IT security and to establish a security program. A meta-level policy should have breadth of scope and be very general, for example:

- “*The system should include an defined, consistent and documented access control policy.*”

An issue-specific policy is developed to address the specific activities in a system, e.g., in a password policy, we may have

- “*All user-level passwords must be changed at least every month.*”

In studying security policies for computer and information systems, it is important to provide a fundamental methodology for precisely representing what an agent (a user) knows or believes about a given system, what he is permitted to know or do within the system, and how the permissions, knowledge and beliefs of agents change over time. We are therefore interested in three classes of notions, which play an essential role in the formalization of security policies, as follows:

- epistemic notions that can be applied to represent knowledge and beliefs of agents,
- deontic notions that can be applied to represent obligations, permissions for agents (users of a system), and
- temporal notions that can describe the dynamics of a system and can therefore be applied to represent how the knowledge, beliefs, permissions and other attributes change over time.

These notions have been intensively studied by logicians, and modal logics corresponding to these notions have been extensively applied in the fields of computer science and artificial intelligence (Fagin, Halpern, Moses & Vardi 1995, Galton 1987, Meyer & van der Hoek 1995, Meyer & Wieringa 1993).

To formalize epistemic notions, logicians use the epistemic logic (EL), a logic including statements of knowledge and belief. Assuming we have n agents, say a_1, \dots, a_n , this logic will involve two groups of modal operators:

- \mathbf{K}_{a_i} standing for “agent a_i knows that”, and
- \mathbf{B}_{a_i} standing for “agent a_i believes that”,

where $i = 1, \dots, n$.

We may begin with an existing logic L , e.g., typically propositional logic or first-order logic, and extend it with a new formation rule: If φ is a well-formed formula (wff), so are $\mathbf{K}_{a_i} \varphi$ and $\mathbf{B}_{a_i} \varphi$. Thus, modal operators \mathbf{K}_{a_i} and \mathbf{B}_{a_i} in fact transform wffs into wffs. For instance, suppose that *has(john, k)* is a wff in EL standing for “John has the key k to open the secure room”, then we can have the wff $\mathbf{K}_{bob} \text{has}(\text{john}, k)$ that stands for “Bob knows that John has the key k to open the secure room” and the wff $\mathbf{B}_{mary}(\mathbf{K}_{bob} \text{has}(\text{john}, k))$ that stands for “Mary believes that Bob knows that John has the key k to open the secure room”.

The axiomatisation for the EL, apart from those axioms and rules of inference in the logic L , should also include some axioms and rules related to the modal operators in it (see appendix).

To formalize deontic notions, we use the deontic logic (DL). The deontic logic has a basic operator, denoted by \mathbf{O} , which is used to represent “it is obligatory that”. Two other operators, \mathbf{P} and \mathbf{F} are defined by $\mathbf{P} \varphi = \neg \mathbf{O} \neg \varphi$ and $\mathbf{F} \varphi = \mathbf{O} \neg \varphi$ respectively. $\mathbf{P} \varphi$

stands for “it is permitted that φ ” and $\mathbf{F} \varphi$ for “it is forbidden that φ ”.

Like the EL, the DL can also be built upon a given logic system, and the operators \mathbf{O} , \mathbf{P} and \mathbf{F} will transform wffs into wffs. Consider the same example above, suppose that $has(john, k)$ is a wff in DL, then we may have the wffs $\mathbf{O} has(john, k)$, $\mathbf{P} has(john, k)$ and $\mathbf{F} has(john, k)$ which stand for “it is obligatory that John has the key k to open the secure room”, “it is permitted that John has the key k to open the secure room” and “it is forbidden that John has the key k to open the secure room”, respectively.

To formalize temporal notions, we use temporal logic (TL). Consider a simple linear-time temporal logic, in which the collection of moments in time is the set of natural numbers with its usual ordering relation $<$. This temporal logic offers two operators, **first** and **next**, which refer to the initial moment and the next moment in time respectively (Liu & Orgun 1996).

Like epistemic and deontic modal operators, temporal operators **first** and **next** also transform wffs into wffs. Consider the same example, if $has(john, k)$ is a wff in TL we may have the wff **first** $has(john, k)$ that stands for “initially (i.e., at the initial moment in time) John has the key k to open the secure room” and the wff **next** $\neg has(john, k)$ that stands for “at the next moment in time, it is not the case that John has that key k ”.

The discussion above has shown that, to formalize all these notions involved in security policies, modal logics are implied. We need a single appropriate modal logic that can be applied to deal not only with the obligations of agents, but also with the temporal and epistemic aspects of agents. So we see that combining modal logics is of particular relevance to formalizing rich security policies.

3 Combining Logics

Combining logics is emerging as an active research area in formal methods (Blackburn & de Rijke 1997). In simple terms, the problem of combining logics is: given two logics L_1 and L_2 , how may we combine them into a single logic $L_1 + L_2$ that extends the expressive power of each. Different choices (definitions) of the combining operator (+) may give quite different logic systems. Several combination techniques have been investigated, including *fusion* (Kracht & Wolter 1991), *fibring* (Gabbay 1999), and *product* (Gabbay & Shehtman 1998). In this paper, we outline two techniques, called the *hierarchy combination* and *free-mixture combination*, with the purpose of providing a suitable logical framework for modelling security policies.

3.1 Hierarchy Combination

The *hierarchy combination* is based on a hierarchical structure. We say that (S, \rightarrow) is a hierarchical structure if the partial order relation \rightarrow defined over the set S satisfies the following properties:

- for any $X \in S$, $(X, X) \notin \rightarrow$ (antireflexive);
- for any $X, Y \in S$, if $(X, Y) \in \rightarrow$, then $(Y, X) \notin \rightarrow$ (antisymmetric); and
- for any $X, Y, Z \in S$, if $(X, Y) \in \rightarrow$ and $(Y, Z) \in \rightarrow$, then $(X, Z) \in \rightarrow$ (transitive).

We may write $(X, Y) \in \rightarrow$ as $X \rightarrow Y$ and read it as “ X leads up to Y ” or “ X is a succession of Y ”.

Given a hierarchical structure $(\{L_1, \dots, L_n\}, \rightarrow)$ of modal logics L_1, \dots, L_n , the hierarchy combination

of these logics based on the structure is informally defined as a combination that satisfies the condition:

- For any L_i, L_j ($1 \leq i, j \leq n$), if $L_i \rightarrow L_j$, then the modal operators of L_i can be within the scope of the modal operators of L_j , but the modal operators of L_j are never within the scope of the modal operators of L_i .

We denote the combination (the combined logic) by $(L_1 + \dots + L_n)_{\rightarrow}$.

Taking a simple case, we first consider two modal logics L_1 and L_2 , and show how to obtain the new logic $L = (L_1 + L_2)_{\{L_1 \rightarrow L_2\}}$ by the hierarchy combination technique.

Let \mathcal{L}_{L_1} be the set of all formulae in the logic L_1 . We partition \mathcal{L}_{L_1} into two sets, $\mathcal{L}_{L_1}^{(b)}$ and $\mathcal{L}_{L_1}^{(m)}$. A formula $\varphi \in \mathcal{L}_{L_1}^{(b)}$ is called a *boolean combination* iff it is built up from other formulae by the use of one of the boolean connectives \neg or \wedge or any other connectives defined only in terms of those; it is called a *monolithic formula*, and belongs to $\mathcal{L}_{L_1}^{(m)}$, otherwise. Thus, for any formula $\varphi \in \mathcal{L}_{L_1}$, we must have $\varphi \in \mathcal{L}_{L_1}^{(b)}$ or $\varphi \in \mathcal{L}_{L_1}^{(m)}$, but not both.

The syntax of the combined logic L is recursively defined as follows: Let \mathcal{L}_L be the set of formulae in L , then \mathcal{L}_L is the smallest set such that:

- (1) If $\alpha \in \mathcal{L}_{L_1}^{(m)}$, then $\alpha \in \mathcal{L}_L$;
- (2) If $\alpha \in \mathcal{L}_L$, then $\neg \alpha \in \mathcal{L}_L$;
- (3) If $\alpha, \beta \in \mathcal{L}_L$ then $(\alpha \wedge \beta) \in \mathcal{L}_L$;
- (4) If $\alpha \in \mathcal{L}_L$, then $\mathbf{Q} \alpha \in \mathcal{L}_L$, for any modal operator \mathbf{Q} of L_2 .

From this definition, we can see that the modal operators of L_2 never appear within the scope of a modal operator of L_1 .

For defining the semantics of the combined logic L , the possible world semantics approach based on Kripke structures (Kripke 1963) is adequate. Kripke introduced Kripke structures as a formal model for a possible-world semantics for the modal logic of necessity and possibility. A Kripke structure for n agents is a tuple $\mathcal{M} = (S, R_1, \dots, R_n, \pi)$, where S is a set of states or possible worlds, π is a function that is a truth assignment to atomic formulae for each state $s \in S$, and R_i is a binary relation on the states of S , for $i = 1, \dots, n$. Any modal operator is assumed to be associated with a particular relation R_i . We call (S, R_1, \dots, R_n) a Kripke framework. Therefore, a model of a given modal logic consists of a Kripke framework and an assignment function. Let $\mathcal{K}_1, \mathcal{K}_2$ be a class of Kripke models for the logics L_1 and L_2 , respectively. Consider a Kripke framework (S, R_1, \dots, R_n) of L_2 and a function $h : S \rightarrow \mathcal{K}_1$, mapping states in S to a model in \mathcal{K}_1 . Then we define a model of L as a tuple (S, R_1, \dots, R_n, h) , denoted by \mathcal{M}_L . Thus, the semantics of the logic L can recursively be defined as follows:

- $\mathcal{M}_L, s \models \alpha, \alpha \in \mathcal{L}_{L_1}^{(m)}$ iff $h(s) = \mathcal{M}_{L_1}, \mathcal{M}_{L_1} \models \alpha$.
- $\mathcal{M}_L, s \models \neg \alpha$ iff it is not the case that $\mathcal{M}_L, s \models \alpha$.
- $\mathcal{M}_L, s \models \alpha \wedge \beta$ iff $\mathcal{M}_L, s \models \alpha$ and $\mathcal{M}_L, s \models \beta$.
- $\mathcal{M}_L, s \models \mathbf{Q}\alpha$ iff $\mathcal{M}_L, t \models \alpha$, for all $(s, t) \in R_i$, where \mathbf{Q} is associated with R_i .

In order to obtain the proof system of L , we introduce the notion of standard-monolithic boolean combinations first. We say that a formula α is a standard-monolithic boolean combination if $\alpha \in \mathcal{L}_{L_1}^{(m)}$ or $\alpha = \neg\beta$ where β is a standard-monolithic boolean combination or $\alpha = \beta \wedge \gamma$ where both β and γ are standard-monolithic boolean combinations. Thus the following assertion can inductively be proved based on structures of formulae in \mathcal{L}_{L_1} .

- For any formula $\alpha \in \mathcal{L}_{L_1}$, there exists some standard-monolithic boolean combination α' , such that α and α' are equivalent, i.e. α is valid in L_1 iff α' is valid in L_1 .

Note that it is not the case that any formula of L_1 is a formula of L , but it is easy to show that, for any formula α of L_1 , the standard-monolithic boolean combination α' equivalent to α is a formula of L .

Thus, the proof system of the combined logic L is given by the following axioms and rules of inference:

1. The axioms of L_2 ;
2. The rules of inference for L_2 ;
3. For every formula α in \mathcal{L}_{L_1} , if $\vdash_{L_1} \alpha$ and α' is a standard-monolithic boolean combination equivalent to α , then $\vdash_L \alpha'$.

where $\vdash_{L_1} \alpha$ means that α is a theorem of the system L_1 and $\vdash_L \alpha$ means that α is a theorem of the system L . The third item above is a new rule of inference that is applied to preserve the theoremhood of formulae of the logic L_1 ; we call it the *rule of theoremhood preservation* (**TP**). Therefore, simply, the proof system of the combined logic L consists of all axioms and inference rules of its component logic L_2 together with the rule **TP**, which guarantees that, if a formula is a theorem of the other component logic L_1 , then a standard-monolithic boolean combination equivalent to that formula is a theorem of the combined logic.

Given two logics L_1 and L_2 , the hierarchical relation over $\{L_1, L_2\}$ can be $\{\}$ (the empty set), $\{L_1 \rightarrow L_2\}$ or $\{L_2 \rightarrow L_1\}$. Note that

$$(L_1 + L_2)_{\{L_1 \rightarrow L_2\}} \neq (L_1 + L_2)_{\{L_2 \rightarrow L_1\}},$$

there are therefore three different hierarchy combined logics from the two logics:

$$(L_1 + L_2)_{\{\}}, \\ (L_1 + L_2)_{\{L_1 \rightarrow L_2\}}, \text{ and } (L_1 + L_2)_{\{L_2 \rightarrow L_1\}}.$$

The last two combined logics can be determined in the same way shown above. While the first one, $(L_1 + L_2)_{\{\}}$, is the simplest combination of L_1 and L_2 ; informally, we can view this combination as consisting of all formulae in the two languages and the boolean combinations of these formulae.

There are several advantages within the hierarchy combination technique. Firstly, this combination technique allows us to handle the syntax and semantics of combined logic systems easily. The process of obtaining a combined logic with the hierarchy combination technique has been standardised: we have a standardised pattern for defining the syntax, including four formation rules, which are suitable for any modal logics; the semantics of the combined logic is defined by the standard possible-world semantics approach with Kripke framework; and the proof system given above can be applied for any combined logic obtained from two logics by the hierarchy combination technique without the need for changes.

In the following, we use examples to demonstrate the features of the hierarchy combination technique. All the examples are involved in the logics DL , EL , and TL . A brief introduction to these logics is provided in the appendix of this paper.

Example – $(TL + EL)_{\{EL \rightarrow TL\}}$

The hierarchy combination technique can be viewed as a generalisation or an extension of the temporalisation methodology (Finger & Gabbay 1992). Therefore, the process of obtaining the logic system $(TL + EL)_{\{EL \rightarrow TL\}}$ is a good example to demonstrate how the hierarchy combination technique works. To simplify our discussion, we denote $(TL + EL)_{\{EL \rightarrow TL\}}$ by ETL . Similarly, we will denote $(DL + EL)_{\{DL \rightarrow EL\}}$ by DEL , $(DL + EL)_{\{EL \rightarrow DL\}}$ by EDL , and so on.

Let $\mathcal{L}_{EL}^{(m)}$ be the set of all monolithic formulae in EL , and \mathcal{L}_{ETL} be the set of formulae of the logic system ETL . Then \mathcal{L}_{ETL} is the smallest set such that:

- If $\alpha \in \mathcal{L}_{EL}^{(m)}$, then $\alpha \in \mathcal{L}_{ETL}$;
- If $\alpha \in \mathcal{L}_{ETL}$, then $\neg\alpha \in \mathcal{L}_{ETL}$;
- If $\alpha, \beta \in \mathcal{L}_{ETL}$ then $(\alpha \wedge \beta) \in \mathcal{L}_{ETL}$;
- If $\alpha \in \mathcal{L}_{ETL}$, then **first** $\alpha \in \mathcal{L}_{ETL}$ and **next** $\alpha \in \mathcal{L}_{ETL}$.

From the definition above, we can see that **first** and **next** never appear within the scope of a modal operator \mathbf{K}_{a_i} or \mathbf{B}_{a_i} .

Regarding the definition of the semantics of ETL , let \mathcal{K} be a class of (Kripke) models of EL . Consider a time frame $(\mathbf{C}, <)$, where $\mathbf{C} = (t_0, t_1, t_2, \dots)$, and a function $h : \mathbf{C} \rightarrow \mathcal{K}$, mapping moments in time on the clock \mathbf{C} to a model in the class \mathcal{K} . Then we define a model of ETL as a triple $\langle \mathbf{C}, <, h \rangle$, denoted by \mathcal{M}_{ETL} . Thus, the semantics of the logic ETL is recursively defined as follows:

- $\mathcal{M}_{ETL}, t_i \models \alpha$, $\alpha \in \mathcal{L}_{EL}^{(m)}$ iff $h(t_i) = \mathcal{M}_{EL}$, $\mathcal{M}_{EL} \models \alpha$.
- $\mathcal{M}_{ETL}, t_i \models \neg\alpha$ iff it is not the case that $\mathcal{M}_{ETL}, t_i \models \alpha$.
- $\mathcal{M}_{ETL}, t_i \models \alpha \wedge \beta$ iff $\mathcal{M}_{ETL}, t_i \models \alpha$ and $\mathcal{M}_{ETL}, t_i \models \beta$.
- $\mathcal{M}_{ETL}, t_i \models \mathbf{first} \alpha$ iff $\mathcal{M}_{ETL}, t_0 \models \alpha$.
- $\mathcal{M}_{ETL}, t_i \models \mathbf{next} \alpha$ iff $\mathcal{M}_{ETL}, t_{i+1} \models \alpha$.

We write $\mathcal{K} \models \alpha$ if, for every model \mathcal{M}_{ETL} in \mathcal{K} and the relevant clock $\mathbf{C} \in \mathcal{C}$ and for every time $t \in \mathbf{C}$, it is the case $\mathcal{M}_{ETL}, t \models \alpha$.

The proof system of ETL consists of the following axioms and rules of inference:

- The axioms of TL ;
- The rules of inference for TL ;
- For every formula α in \mathcal{L}_{EL} , if $\vdash_{EL} \alpha$ and α' is a standard-monolithic boolean combination equivalent to α , then $\vdash_{ETL} \alpha'$. (**TP**)

That is, the axiomatisation for ETL consists of all axioms of temporal logic TL and 4 rules of inference, i.e., **US**, **MP**, **TG** and **TP** (see also appendix).

From this example, we can see that, with the standardised process of obtaining a combined logic by the hierarchy combination technique, we do nothing but

simple substitutions to replace those component logics and modal operators in the standard patterns in order to create the new logic ETL.

Another advantage of the hierarchy combination technique is that a complex combination of multiple modal logics can be obtained by the induction method based on the hierarchy combination of two logics. In fact, according to the above discussion, the syntax, semantics, and the proof system for a hierarchy combined logic of any two logics can easily be determined. Then if, by the induction step, we have obtained a combined system of $n - 1$ logics, L_1, \dots, L_{n-1} , by the hierarchy technique, and assume $L^{(n-1)}$ is this combined logic, we can combine logics $L^{(n-1)}$ and L_n to obtain a combination of n logics L_1, \dots, L_{n-1}, L_n .

One of the most important problems in combining logics is what kind of combination method is chosen to obtain a desired logic system. The third advantage is that combined logics obtained by the hierarchy combination technique are generally strong enough for the use in specifying, and the analysis of, security policies for many different application domains. For example, within DEL, we may have

- (1) $\mathbf{K}_{john} (\mathbf{O} \textit{ keeps_secure}(mary, k))$.
- (2) $\mathbf{B}_{john} ((\mathbf{F} \textit{ sees}(bob, d)) \wedge \neg \textit{ has}(bob, k))$.
- (3) $\mathbf{B}_{john} (\forall x (\textit{ sees}(x, d) \rightarrow \textit{ has}(x, k)))$.
- (4) $\mathbf{B}_{john} \textit{ sees}(bob, d)$.

Formula (1) says that John knows that Mary must keep the key k secure; formula (2) says that John believes that Bob is forbidden to see document d and does not have k ; formula (3) means that John believes that one sees d only if he has k ; and (4) means that John believes that bob sees document d . Such formalization could be useful in analysing inconsistency related to a security policy set by the organisation. For example, according to a contradiction derived from these formulae, John may conclude that the security policy (“Mary must keep the key k secure, and Bob is forbidden to see document d ”) has been violated.

3.2 Free-mixture Combination

As discussed above, for logics DL and EL, there are three hierarchy combined logics from them: $(DL + EL)_{\{\}} \subseteq DEL$ and EDL . $(DL + EL)_{\{\}}$ consists of all formulae in both DL and EL and the boolean combinations of these formulae. For example, assume $\mathbf{O} p$ is a formula of DL and $\mathbf{K}_a q$ is a formula of EL, both $\mathbf{O} p$ and $\mathbf{K}_a q$ and boolean combinations of these formulae such as $\mathbf{O} p \wedge \mathbf{K}_a q$ are formulae in this combined logic. However, in $(DL + EL)_{\{\}}$, we do not have formulae such as $\mathbf{K}_a(\mathbf{O} p)$ and $\mathbf{O}(\mathbf{K}_a q)$ in which the modal operators of DL appear within the scope of modal operators of EL or the modal operators of EL appear within the scope of modal operators of DL. The sort of formulae can however appear in DEL or EDL, but neither DEL nor EDL can contain the formula $\mathbf{K}_a(\mathbf{O} p)$ as well as the formula $\mathbf{O}(\mathbf{K}_a q)$.

We therefore introduce the *free-mixture combination*. In the free-mixture combination, formulae of the combined logic can mix operators from the component logics arbitrarily. Let L_1, \dots, L_n be n logics, we denote the combined logic gained from these logics using the free-mixture combination by $L_1 + \dots + L_n$. Thus, DL + EL is the free-mixture combination of logics DL and EL. In this combined logic, we can of course have both the formulae $\mathbf{K}_a(\mathbf{O} p)$ and $\mathbf{O}(\mathbf{K}_a q)$. Due to space limitation, the detailed discussion about the free-mixture combination is omitted.

Thus, with the logics DL and EL, we have obtained four different combined logics. It is obvious

that we may have the following subset relations regarding these logics:

$$\begin{aligned} (DL + EL)_{\{\}} &\subseteq (DL + EL)_{\{DL \rightarrow EL\}}, \\ (DL + EL)_{\{\}} &\subseteq (DL + EL)_{\{EL \rightarrow DL\}}, \\ (DL + EL)_{\{DL \rightarrow EL\}} &\subseteq DL + EL, \text{ and} \\ (DL + EL)_{\{EL \rightarrow DL\}} &\subseteq DL + EL. \end{aligned}$$

Similar discussion can be made for TL and DL, and for TL and EL. For three logics TL, DL and EL, there are more cases of hierarchy combinations.

It seems that the four combined logics originally from DL and EL may form a lattice under the relation \subseteq . An immediate question is: can all logics obtained by combining any n logics with the hierarchy combination and free-mixture combination form a lattice under the relation \subseteq ? If the answer is “yes”, it would be helpful for discussing the properties of these combined logics and their expressiveness and adaptability in modelling security policies.

4 Expressing Security Policy Rules

In this section, we discuss how to express certain security policy rules in our logical framework. We begin with defining terms relating to notions of obligation and prohibition.

Several important keywords that often appear in security policy documents have been recognised by researches and developers, see, for example, RFC 2119 (<http://www.faqs.org/rfcs/rfc2119.html>) proposed by the Network Working Group. These keywords are listed below and to be interpreted as follows:

1. **MUST**: The use of this word means that the relevant item is mandatory.
2. **MUST NOT**: The use of this phrase means that the non-use of the relevant item is mandatory.
3. **SHOULD**: The use of this word means that valid reasons may exist in particular circumstances to ignore the relevant item, but the full implications SHOULD be understood and carefully weighed before not following this item.
4. **SHOULD NOT**: The use of this phrase means that valid reasons may exist in particular circumstances when the relevant item is acceptable or even useful, but the full implications SHOULD be understood and the case carefully weighed before implementing the item.
5. **MAY**: The use of this word means that the relevant item is optional.

For the formalization of security policy rules, we first need to carefully map these words and phrases into corresponding expressions in our logical framework. However, it is not easy to identify which map is better to express the meaning of a word or a phrase. We propose the following definition:

$$\begin{aligned} \text{MUST } p &\stackrel{\text{def}}{=} \mathbf{O} p \\ \text{MUST_NOT } p &\stackrel{\text{def}}{=} \mathbf{O} \neg p \quad (\equiv \mathbf{F} p) \\ \text{SHOULD } p &\stackrel{\text{def}}{=} \mathbf{O} p \vee \text{SHOULD}(\text{GIVE_REASON_FOR } \neg p) \\ \text{SHOULD_NOT } p &\stackrel{\text{def}}{=} \mathbf{O} \neg p \vee \\ &\quad \text{SHOULD}(\text{GIVE_REASON_FOR } p) \\ \text{MAY } p &\stackrel{\text{def}}{=} \neg \mathbf{O} \neg p \quad (\equiv \mathbf{P} p) \end{aligned}$$

SHOULD p and SHOULD_NOT p are defined recursively, and we are further involved in the definition of

GIVE_REASON_FOR p that may depend on a real situation, see examples given below.

In order to show how the formal definition of keywords would be helpful for the formalization of a certain security policy, we first introduce two basic predicates:

$stop_at(x, y, z)$: x stops y at z , and
 $park_in(x, y, z)$: x parks y in z ,

where variables x , y , and z represent a person, a car, and a location, respectively. Now let us consider the following security policy rules:

- “Everyone must stop his car at (before) the STOP sign”,
- “Everyone must not park his car in area A ”,
- “One should stop his car at (before) the STOP sign unless he is a police officer who is carrying out an emergency task”,
- “One should not park his car in area A unless he is a manager”, and
- “A customer may park his car in the visitor car park area (v_park area)”.

With the formal definition of keywords, we have no difficulty in transforming each of these rules into a formula in the logic DL as follows:

- (1) $\forall x (\mathbf{O} \text{ stop_at}(x, x\text{'s car, STOP}))$.
- (2) $\forall x (\mathbf{O} \neg \text{park_in}(x, x\text{'s car, area A}))$.
- (3) $\mathbf{O} \text{ stop_at}(x, x\text{'s car, STOP}) \vee (\text{is_pol_officer}(x) \wedge \text{emergency}(e) \wedge \text{is_carrying_out}(x, e))$.
- (4) $\mathbf{O} \neg \text{park_in}(x, x\text{'s car, area A}) \vee \text{is_manager}(x)$.
- (5) $\text{is_customer}(x) \rightarrow \mathbf{P} \text{ park_in}(x, x\text{'s car, v_park})$.

In formulae (3) and (4), we are in fact involved in giving reason for why it is ‘ p ’ (or ‘not p ’), according to the real situation.

We have discussed the importance of the formal definition of those keywords related to the deontic notion. For formalizing a specific security policy, it is also important to choose an appropriate combined logic. For example, a confidentiality policy regarding the problem of database access is given as follows:

- P1. “A user may access data only if the user plays the role r .”

We assume that the database, denoted by db , consists of a consistent set of sentences of a language. It is not assumed that all sentences in db are true in the world. But db is a set of beliefs. In this view, if $p \in db$, we may write $\mathbf{B}_{db} p$. While “an agent a knows that the database db believes p ” would be expressed by $\mathbf{K}_a(\mathbf{B}_{db} p)$, and “it is permitted that the agent a knows that db believes p ” is expressed by $\mathbf{P}(\mathbf{K}_a(\mathbf{B}_{db} p))$. This indicates that, to express a confidentiality policy specifying what part of the database an agent is permitted to access, it is better to use the logic EDL. We see r as the set of those who play the role r , then, the formula

$$\mathbf{P}(\mathbf{K}_a(\mathbf{B}_{db} \varphi(x))) \rightarrow a \in r$$

would be regarded as the formal representation of P1, where $\varphi(x)$ defines the part of db that the agent a is permitted to access.

In the following, we consider some security policies that may exist in a defence information environment. The policy P2 regards the security of private keys.

- P2. “The owner of a private key must keep it secure. If a private key is compromised (someone other than the owner of the key acquired it), it must be cancelled. The owner must not use his/her private key if it is cancelled.”

In order to formalize such policy, we first define the following predicates:

$is_owner(x, k)$: x is the owner of the key k .
 $private(k)$: k is a private key.
 $keeps_secure(x, k)$: x keeps the key k secure.
 $is_cancelled(k)$: k is cancelled.
 $got(x, k)$: x acquired the key k .
 $uses(x, k)$: x uses k .

We also need to clarify the practical meaning of the policy. In P2, the first sentence means that, if one holds a private key, then it is obligatory that he/she keeps the key secure. For the second and third sentences, we can also give similar explanations. Thus, the policy P2 can be formalized by the following security rules: for any x and any k ,

- R1. $is_owner(x, k) \wedge private(k) \rightarrow \mathbf{O} \text{ keeps_secure}(x, k)$.
- R2. $is_owner(x, k) \wedge private(k) \rightarrow (\exists a \text{ got}(a, k) \wedge a \neq x \rightarrow \mathbf{O} \text{ is_cancelled}(k))$.
- R3. $owner(x, k) \wedge is_cancelled(k) \rightarrow \mathbf{O} \neg \text{uses}(x, k)$.

In the formalisation of security policies, we do not think that one must always follow the maps given above regarding the meanings of keywords to transform a rule into a formal expression. In many cases, it is better to see the requirements of an agent role as obligations. Let us consider a practical security rule related to an IT administration function as follows:

- P3. “An IT security adviser (ITSA) must be appointed to be responsible for the security of the agent’s network.”

In our understanding, this policy means that an organisation with a network must appoint some person as an ITSA responsible for the security of the system. The rule may therefore be formalized as:

- R4. $is_org(o) \wedge is_network(n) \wedge possesses(o, n) \rightarrow \mathbf{O} (\exists a (\text{appoints}(o, a, ITSA) \wedge is_resp(a, n)))$,

where $is_org(o)$ and $is_network(n)$ stand for “ o is an organisation” and “ n is a network”, respectively; $possesses(o, n)$ means that o possesses the network n ; and $\text{appoints}(o, a, ITSA) \wedge is_resp(a, n)$ means that o appoints a as an ITSA to be responsible for the security of n .

In formalizing security policy rules in a specific application domain, there are three research areas worth further study:

- It is necessary to have an accepted mapping scheme for transforming those keywords contained in existing (informal and/or formal) security policy documents into a standard format with various modal operators. This research may include investigating which words should be chosen as keywords and proposing a suitable transform pattern for them. An uncompleted mapping scheme is presented in this paper for discussion, and more work is needed.
- We are also looking for an essential method (including some principles and techniques) for choosing an appropriate combined logic applied for any given application domain, such that we can easily and conveniently handle the formal representations of security rules in the specific domain.

- The formalization also depends on a correct understanding of those informal security rules of a security policy, which would benefit from the standard mapping scheme techniques provided.

The policy formalization process is in fact a collaborative process involving experts from both the design and formal methods areas. Therefore, for the research on the above mentioned fundamental work, cooperation between the policy designer and researchers is necessary.

5 Reasoning about Security Properties

The security policy P4 given below involves the security level of each person and each controlled location in an organisation. The policy fits the format of the MAC policy model. The security level of a person determines whether or not he/she is able to access controlled rooms.

P4. *“Any person whose security level is less than that of a particular room must be kept out from this room; and, if the person must be kept out from the room, then he/she is forbidden to see any document stored in it.”*

We define a function, $security_level(x)$, which gives x 's security level where x is a room or a person. Then this security policy can be formalized by the following rules (rule schemata): For all agent a , room x and document d ,

- R5. $security_level(a) < security_level(x) \rightarrow$
 $\mathbf{O} \textit{ kept_out_from}(a, x).$
R6. $\mathbf{O} \textit{ kept_out_from}(a, x) \wedge \textit{ stored_in}(d, x) \rightarrow$
 $\mathbf{F} \textit{ sees}(a, d).$

Suppose this policy is satisfied through a certain security mechanism provided (the discussion on what kind of mechanism and how it guarantees that the policy is satisfied are not in the scope of this paper), then, based on R5 and R6, we are able to reason about beliefs of agents concerning the security properties of this policy. For example, assume we have

- (1) $\mathbf{K}_{john} (security_level(mary) = l_1 \wedge security_level(room10) = l_2 \wedge l_1 < l_2).$
- (2) $\mathbf{B}_{john} \textit{ stored_in}(f1, room10).$

This indicates that John knows the security levels of Mary and room 10 and he also believes that the file $f1$ is stored in the room 10. Then, for example, it is easy to show that

- (3) $\mathbf{B}_{john} (\mathbf{F} \textit{ sees}(mary, f1)),$

which means that John believes that Mary is forbidden to see the document $f1$. We outline the proof procedure below.

Firstly, from (1) and by R5, we have

- (4) $\mathbf{K}_{john} (\mathbf{O} \textit{ kept_out_from}(mary, room10)).$

By EA3 (see appendix), from (4) we have

- (5) $\mathbf{B}_{john} (\mathbf{O} \textit{ kept_out_from}(mary, room10)).$

Then from (2) and (5), and by R6, we obtain (3), that is exactly what we want to show.

Reasoning about the security properties of a given security policy usually involves:

- Choosing a (combined) modal logic appropriate for the specific application (in the case above, we use the logic DEL),

- Defining functions and predicates, and then formalizing the policy as a set of rules, denoted for example by Θ ,
- Checking the consistency of Θ , and
- Expressing the assumptions as a set of formulae, denoted by Γ .

Once all the preparations are ready, assuming that a security property is expressed as φ , then the reasoning procedure is to construct a proof in the logic framework of the theorem

$$\Theta \cup \Gamma \vdash \varphi.$$

Note that checking the consistency of Θ is an important step. Given the soundness of our logic, if we find that Θ is not consistent, there may be something wrong with the policy itself, and we (as the PM) therefore need to consider how to correct this policy.

With reasoning, the designer of a security policy is able to find vulnerabilities before the policy is released. The evaluator and/or the policy manager can apply reasoning processes to detect conflicts and ambiguities implied in a security policy and analyse accidents related to implementing such policies. As an example, let us consider a case as follows: Assume that, apart from rules R5 and R6 given above, a policy contains a rule

$$a \in r \rightarrow \mathbf{P}(\mathbf{K}_a(\mathbf{B}_{db} \varphi(x)))$$

which means that, if agent a plays role r , then a is permitted to know the database db believes $\varphi(x)$ to be true. Suppose John, as an evaluator, has gained a belief – believing that Mary is forbidden to see the file $f1$ – through the reasoning process as above. We also assume that John believes that Mary has seen $f1$. This indicates that there is a conflict in this policy or this policy has been violated. With reasoning, John finally found the reason is that the policy (as a whole) implies that $mary \in r$ and, therefore, it in fact implies $\mathbf{P}(\mathbf{K}_{mary}(\mathbf{B}_{db} \varphi(x)))$ to be true, while (we assume) the later implies that Mary can see the file $f1$. However, according to R5 and R6, Mary is not allowed to see the file $f1$.

We believe that conflict analysis and refinement are key areas for further study on reasoning about security policies. Therefore, it is important to gain a formal approach to construct a theory specifying a given security policy. Such an approach should be very general, such that it can be applied for a variety of security policies with various combined logics. Investigation of new reasoning techniques under our logical framework is also an important research direction. With the combination of the notations of obligation, knowledge, beliefs, and time into a single combined logic, we may need to combine various reasoning techniques, including non-monotonic reasoning techniques, temporal reasoning techniques *etc.*, together to obtain new techniques for the specific use of reasoning about security policies.

6 Conclusion

A logical framework for modelling security policies has been discussed. This include methods for combining modal logics expressing three classes of notions involved in security policies; expressing security rules using our combined modal logic; and techniques to construct a theory for reasoning about security properties of a given security policy within this logical framework.

Rather than considering complex combinations when combining logics, we maintained simpler ways

for the practical use of modelling security policies. The boolean connectives, \neg and \wedge etc., in all component logics of our new logic system have a unique semantics interpretation; and the hierarchy combination technique allows us to handle the semantics of combined logics easily. In this way, we can obtain a logic that seems strong enough for formalizing security policies in a specific domain.

The modal logical framework, proposed by (Cuppens & Demolmore 1997), combines the epistemic and deontic logics to specify confidentiality policies that regulate user's knowledge. They define, for each role, two modalities denoted by PKB_r and FKB_r . Formulae of the form $\text{PKB}_r p$ (and $\text{FKB}_r p$) are to be read as "any agent who plays the role r is permitted (and forbidden, respectively) to know that the database believes p ". As shown in the previous sections, based on our logical framework, these modalities can be expressed by a combination of deontic and epistemic modal operators in the logic EDL. In addition, our logical framework is more general and flexible, it deals with not only the obligations and knowledge of agents but also the temporal aspects.

Any logic system may have limitations. Recalling the rules specifying the policy P2, it seems that reasoning about security properties of a policy can often be achieved in the logic DEL. However, this is in general impossible. In fact, this policy involves regulating the knowledge of agents: for the rule R1, one may define that $\text{keeps_secure}(x, k)$ if and only if, for all a , $a \neq x \rightarrow \neg \mathbf{K}_a \text{ is_key}(k)$. Thus, the formula $\mathbf{O} \text{ keeps_secure}(x, k)$ may be replaced by $\mathbf{O} (a \neq x \rightarrow \neg \mathbf{K}_a \text{ is_key}(k))$, which is not a formula in DEL. This indicates that a simple combined logic may be used for modelling the class of security policy but it may not be suitable for other classes of policy. Therefore, developing a technique for classifying security policies and then identifying a suitable combined modal logic based on the classification of a particular security policy is a possible future area of research.

Future work should also include further investigation of the techniques for combining logics, the properties of combined logics related to the specific use of modelling security policies, and the strategies for reasoning. Implementing a security policy language based on our logical framework is also an important issue to consider.

Acknowledgements

The work presented in this article has been supported in part by an Australian Research Council (ARC) Discovery Project grant.

References

- Blackburn, P. & de Rijke, M. (1997), Why combine logics. *Studia Logica*, 59(1):5-27.
- Cholvy, L. & Cuppens, F. (1997), Analyzing consistency of security policies. In *IEEE Symposium on Security and Privacy (S&P97)*, Oakland, CA, IEEE Press.
- Cuppens, F. & Demolombe, R. (1997), A modal logical framework for security policies. In *10th International Symposium ISMIS'97*, LNAI 1325, Springer.
- Cuppens, F. & Saurel, C. (1996), Specifying a security policy: A case study. In *9th Computer Security Foundation Workshop*, County Kerry, Ireland. IEEE Computer Society Press.

- Dulany, K. M. (2002), Security, It's Not Just Technical. <http://www.sans.org/rr/papers/50/499.pdf>
- Fagin, R., Halpern, J. Y., Moses, Y. & Vardi, M. Y. editors. (1995), *Reasoning about Knowledge*. MIT Press, Cambridge (Mass.).
- Finger, M. & Gabbay, D. M. (1992), Adding a temporal dimension to a logic system. *Journal of Logic, Language and Information*, 1:221-237.
- Gabbay, D. M. & Shehtman, V. (1998), Products of modal logics, part 1. *Logic Journal of the IGPL*, 6(1):71-146.
- Gabbay, D. M. (1999), *Fibring Logics*. Oxford University Press, Oxford.
- Galton, A. editor (1987), *Temporal Logics and Their Applications*. Academic Press.
- Glasgow, J. & Macewen, G. et al.(1992), A logic for reasoning about security. *ACM Transactions on Computer Systems (TOCS)*, 10(3):226-264.
- Kracht, M. & Wolter, F. (1991), Properties of independently axiomatizable bimodal logics. *The Journal of Symbolic Logic*, 56(4):1469-1485.
- Kripke, S. (1963), Semantical Considerations of Modal Logic. *Acta Philosophica Fennica*, 16:83-94.
- Liu, C. & Orgun, M. A. (1996), Dealing with multiple granularity of time in temporal logic programming. *Journal of Symbolic Computation*, 22:699-720.
- Meyer, John-Jules Ch. & van der Hoek, Wiebe editors (1995), *Epistemic Logic for AI and Computer Science*. Cambridge University Press, Cambridge.
- Meyer, John-Jules Ch. & Wieringa, R. J. editors (1993), *Deontic Logic in Computer Science*. John Wiley, Chichester.
- Ortalo, R. (1998), A flexible method for information system security policy specification. *The 5th European Symposium on Research in Computer Security (ESORICS 98)*, Louvain-la-Neuve, Belgium, Springer-Verlag.
- Peri, R. V. (1996), Specification and Verification of Security Policies. PhD thesis, University of Virginia.

Appendix

A Epistemic Logic (EL)

Epistemic logic (EL) is a logic concerning statements about knowledge and beliefs. It has been extensively applied in computer science (Meyer & van der Hoek 1995, Fagin, Halpern, Moses & Vardi 1995). We consider the simple propositional epistemic logic. EL has two group of modal operators:

- \mathbf{K}_{a_i} intended to stand for "agent a_i knows that", where $i = 1, \dots, n$, and
- \mathbf{B}_{a_i} intended to stand for "agent a_i believes that", where $i = 1, \dots, n$.

Here we assume that $\mathcal{A} = \{a_1, \dots, a_n\}$ is the set of agents involved in a specific system throughout this paper. Note that any of these modal operators cannot be defined in terms of one another.

Let \mathcal{P} be a denumerably infinite set of propositional symbols. We define that \mathcal{L}_{EL} as the set of formulae in the EL is the smallest set such that:

- $\mathcal{P} \subset \mathcal{L}_{EL}$;
- If A and B are in \mathcal{L}_{EL} , so are $\neg A$ and $A \wedge B$;
- If A is in \mathcal{L}_{EL} , then $\mathbf{K}_{a_i} A$ and $\mathbf{B}_{a_i} A$ are in \mathcal{L}_{EL} .

The last formation rule above make operators \mathbf{K}_{a_i} and \mathbf{B}_{a_i} to transform a well-formed formula to another well-formed formula.

In EL, other connectives \vee , \rightarrow and \leftrightarrow are derived from the primitive connectives.

The definition of the semantics for the logic EL would be referred as the possible-world semantics (Kripke 1963). In such a definition, we have

$$\mathcal{M}, s \models \mathbf{K}_{a_i} \varphi \text{ iff, for all } t \text{ such that } (s, t) \in R_i, \mathcal{M}, t \models \varphi.$$

which means that $\mathbf{K}_{a_i} \varphi$ holds at s in the model \mathcal{M} iff φ holds at t in the model \mathcal{M} , for all t such that $(s, t) \in R_i$, where R_i is the *possibility relation* associated with agent a_i .

We omit the detailed discussion about the semantics of the logic EL, but give a minimal axiomatic system for it below, which contains the following axioms (axiom schemata):

- EA0. all classical tautologies.
- EA1. $\mathbf{K}_{a_i}(A \rightarrow B) \rightarrow (\mathbf{K}_{a_i} A \rightarrow \mathbf{K}_{a_i} B)$.
- EA2. $\mathbf{B}_{a_i}(A \rightarrow B) \rightarrow (\mathbf{B}_{a_i} A \rightarrow \mathbf{B}_{a_i} B)$.
- EA3. $\mathbf{K}_{a_i} A \rightarrow \mathbf{B}_{a_i} A$.
- EA4. $\mathbf{K}_{a_i} A \rightarrow A$.

The rules of inference are:

- MP. From A and $A \rightarrow B$ infer B . (Modus Ponens)
- EG. From $\vdash A$ infer $\vdash \mathbf{K}_{a_i} A$. (E-Generalisation)

By rule EG and axiom EA3, it is easy to show that, if A is a theorem, then $\mathbf{B}_{a_i} A$ must also be a theorem.

B Deontic Logic (DL)

Deontic logic (DL) has been extensively applied in computer science (Meyer & Wieringa 1993). This logic has a basic operator, denoted by \mathbf{O} , which is used to represent “it is obligatory that”. We consider a first-order deontic logic that simply extends classical first-order logic with the special modal operator \mathbf{O} . In the DL, terms and formulae are defined as usual, but it has a particular formation rule related to application of the operator \mathbf{O} . Formally, let \mathcal{L}_{DL} be the set of formulae of the logic DL, then \mathcal{L}_{DL} is defined as the smallest set such that:

- All atoms of first-order logic are in \mathcal{L}_{DL} ;
- If $\varphi \in \mathcal{L}_{DL}$ and $\psi \in \mathcal{L}_{DL}$, then $\neg\varphi \in \mathcal{L}_{DL}$ and $\varphi \wedge \psi \in \mathcal{L}_{DL}$;
- If $\varphi(x) \in \mathcal{L}_{DL}$ where x is a free variable, then $\forall x\varphi(x) \in \mathcal{L}_{DL}$;
- If φ is in \mathcal{L}_{DL} , so is $\mathbf{O} \varphi$.

Other connectives, \vee , \rightarrow , and \leftrightarrow , and the qualifier \exists are defined based on the primitives \neg , \wedge and \forall in the usual manner. In addition, two other operators, \mathbf{P} and \mathbf{F} are introduced based on the operator \mathbf{O} :

$$\begin{aligned} \mathbf{P} \varphi &= \neg \mathbf{O} \neg\varphi, \text{ and} \\ \mathbf{F} \varphi &= \mathbf{O} \neg\varphi. \end{aligned}$$

Here $\mathbf{P} \varphi$ stands for “it is permitted that φ ” and $\mathbf{F} \varphi$ for “it is forbidden that φ ”.

In general, a formula with the operator \mathbf{O} , \mathbf{P} , or \mathbf{F} is about what is required, which is usually related to the use of those English auxiliary verbs, such as ‘must’, ‘must not’, ‘ought to’, ‘have to’, ‘should’, ‘may’ *etc* in modal sentences. For example, we may write $\mathbf{F}reads(john, d)$ and $\mathbf{P}reads(bob, d)$ to represent

- (1) “John must not read the document d ”, and
- (2) “Bob may read the document d ”,

respectively. The sort of formulae are with respect to the actual world and given about what is required by a certain law(s) or contract. Therefore, the definition of the semantics for the logic DL would also be referred to as the possible-world semantics.

The DL has the following axioms:

- DA0. all axioms of the classical first-order logic.
- DA1. $\mathbf{O}(\varphi \rightarrow \psi) \rightarrow (\mathbf{O} \varphi \rightarrow \mathbf{O} \psi)$.
- DA2. $\mathbf{O} \varphi \rightarrow \mathbf{P} \varphi$.
- DA3. $\forall x (\mathbf{O} \varphi(x)) \leftrightarrow \mathbf{O} (\forall x \varphi(x))$.

The rules of inference for this logic include:

- DR1. From φ and $\varphi \rightarrow \psi$ infer ψ . (Modus Ponens)
- DR2. From $\forall x \varphi(x)$ infer $\varphi(y)$. (\forall -Instantiation)
- DR3. From $\varphi(x)$ infer $\forall x \varphi(x)$. (\forall -Generalisation)
- DR4. From $\vdash \varphi$ infer $\vdash \mathbf{O} \varphi$. (\mathbf{O} -Generalisation)

C Temporal Logic (TL)

We consider a very simple linear-time temporal logic, in which the collection of moments in time is the set of natural numbers (\mathcal{N}) with its usual ordering relation $<$. The syntax of this logic is defined as follows:

Let \mathcal{P} be a denumerably infinite set of propositional symbols. We define that \mathcal{L}_{TL} as the set of temporal propositional formulae is the smallest set such that:

- $\mathcal{P} \subset \mathcal{L}_{TL}$;
- If A is in \mathcal{L}_{TL} , then $\neg A$, **first** A and **next** A are in \mathcal{L}_{TL} ; and
- If A and B are in \mathcal{L}_{TL} , so $A \wedge B$ is, too.

Connectives \vee , \rightarrow and \leftrightarrow are derived from the primitive connectives as usual.

We define that the global clock is the increasing sequence of natural numbers, i.e., $\langle 0, 1, 2, \dots \rangle$, and a local clock is an infinite subsequence of the global clock. A *time frame* is denoted by $(\mathbf{C}, <)$, where $\mathbf{C} = \langle t_0, t_1, t_2, \dots \rangle$ is a local clock and $<$ is the ordinary binary relation over \mathbf{C} . A *model* \mathcal{M} is a triple $\langle \mathbf{C}, <, v \rangle$, where $(\mathbf{C}, <)$ is a time frame and v is a total function, called a valuation, from \mathbf{C} to the power set of \mathcal{P} , that is, for any moment in time $t_i \in \mathbf{C}$, $v(t_i) \subseteq \mathcal{P}$. We write $\mathcal{M}, t_i \models A$ to mean that the formula A holds over model \mathcal{M} at time t_i , and it is defined recursively as follows:

- $\mathcal{M}, t_i \models p, p \in \mathcal{P}$ iff $p \in v(t_i)$.
- $\mathcal{M}, t_i \models \neg A$ iff it is not the case that $\mathcal{M}, t_i \models A$.
- $\mathcal{M}, t_i \models A \wedge B$ iff $\mathcal{M}, t_i \models A$ and $\mathcal{M}, t_i \models B$.
- $\mathcal{M}, t_i \models \mathbf{first} A$ iff $\mathcal{M}, t_0 \models A$.
- $\mathcal{M}, t_i \models \mathbf{next} A$ iff $\mathcal{M}, t_{i+1} \models A$.

A formula A is *valid* over a class \mathcal{C} of clocks, indicated by $\mathcal{C} \models A$, if for any model $\mathcal{M} = \langle \mathbf{C}, <, v \rangle$ where $\mathbf{C} \in \mathcal{C}$ and for any $t \in \mathbf{C}$, we have $\mathcal{M}, t \models A$. If Σ is a set of formulae, we write $\mathcal{C} \models \Sigma$ to indicate that $\mathcal{C} \models A$ for every $A \in \Sigma$. Therefore, for different classes we may have different sets of valid formulae.

A minimal axiomatic system for the propositional temporal logic over a class \mathcal{C} contains the following axioms (axiom schemata):

TA0. all classical tautologies.

TA1. $\nabla(\mathbf{first} A) \leftrightarrow \mathbf{first} A$.

TA2. $\nabla(\neg A) \leftrightarrow \neg(\nabla A)$.

TA3. $\nabla(A \wedge B) \leftrightarrow (\nabla A) \wedge (\nabla B)$.

where ∇ stands for **first** or **next** in any axiom schema. The rules of inference are:

US. From $\vdash A(p)$ infer $\vdash A(p \setminus B)$
(Uniform Substitution)

MP. From $\vdash A$ and $\vdash A \rightarrow B$ infer $\vdash B$.
(Modus Ponens)

TG. From $\vdash A$ infer $\vdash \mathbf{first} A$ and $\vdash \mathbf{next} A$.
(T-Generalisation)

In rule **US**, p is the propositional symbol and B is any formula and $A(p \setminus B)$ is a formula resulting from substituting all appearances of p in A by B .